2

Developing Real-Time Emergency Management Applications: Methodology for a Novel Programming Model Approach

Gabriele Mencagli and Marco Vanneschi
Department of Computer Science, University of Pisa, L. Bruno Pontecorvo, Pisa
Italy

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 *Grid and 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.

In this context time-critical applications, notably emergency management systems, are composed of complex sets of application components specialized for executing specific computations, which are able to cooperate in such a way as to perform a global goal in a distributed manner. Since the last years the scientific community has been involved in facing with the programming issues of distributed systems, aimed at the definition of applications featuring an increasing complexity in the number of distributed components, in the spatial distribution and cooperation between interested parties and in their degree of heterogeneity.

Over the last decade the research trend in distributed computing has been focused on a crucial objective. The wide-ranging composition of distributed platforms in terms of different classes of computing nodes and network technologies, the strong diffusion of applications that require real-time elaborations and online compute-intensive processing as in the case of emergency management systems, lead to a pronounced tendency of systems towards properties like self-managing, self-organization, self-controlling and strictly speaking ***adaptivity***.

Adaptivity implies the development, deployment, execution and management of applications that, in general, are dynamic in nature. Dynamicity concerns the number and the specific identification of cooperating components, the deployment and composition of the most suitable versions of software components on processing and networking resources and services, i.e., both the quantity and the quality of the application components to achieve the needed Quality of Service (QoS). In time-critical applications the QoS specification can dynamically vary during the execution, according to the user intentions and the

information produced by sensors and services, as well as according to the monitored state and performance of networks and nodes.

The general reference point for this kind of systems is the Grid paradigm which, by definition, aims to enable the access, selection and aggregation of a variety of distributed and heterogeneous resources and services. However, though notable advancements have been achieved in recent years, current Grid technology is not yet able to supply the needed software tools with the features of high adaptivity, ubiquity, proactivity, self-organization, scalability and performance, interoperability, as well as fault tolerance and security, of the emerging applications.

For this reason in this chapter we will study a methodology for designing high-performance computations able to exploit the heterogeneity and dynamicity of distributed environments by expressing adaptivity and QoS-awareness directly at the application level. An effective approach needs to address issues like *QoS predictability* of different application configurations as well as the predictability of reconfiguration costs. Moreover adaptation strategies need to be developed assuring properties like the stability degree of a reconfiguration decision and the execution *optimality* (i.e. select reconfigurations accounting proper trade-offs among different QoS objectives). In this chapter we will present the basic points of a novel approach that lays the foundations for future programming model environments for time-critical applications such as emergency management systems.

The organization of this chapter is the following. In Section 2 we will compare the existing research works for developing adaptive systems in critical environments, highlighting their drawbacks and inefficiencies. In Section 3, in order to clarify the application scenarios that we are considering, we will present an emergency management system in which the run-time selection of proper application configuration parameters is of great importance for meeting the desired QoS constraints. In Section 4 we will describe the basic points of our approach in terms of how compute-intensive operations can be programmed, how they can be dynamically modified and how adaptation strategies can be expressed. In Section 5 our approach will be contextualize to the definition of an adaptive parallel module, which is a building block for composing complex and distributed adaptive computations. Finally in Section 6 we will describe a set of experimental results that show the viability of our approach and in Section 7 we will give the concluding remarks of this chapter.

2. Related works

When a distributed application is executed on a dynamic execution environment, where the number and features of nodes and network facilities but also the desired QoS level may vary significantly and in unpredictable ways, we need to design systems which are able to modify their actual *configuration* for maintaining and respecting the desired objectives. For application configuration we intend a specific identification of application components, their mapping onto the available computing resources, the specification of their internal behaviors (e.g. if they exploit a sequential or a parallel elaboration) and the way in which the computations are performed (i.e. for a sequential component the selected algorithm whereas for a parallel computation the exploited parallelism scheme). The actual system configuration can be modified by exploiting dynamic reconfiguration activities classified in four general

categories (Arshad et al., 2007; Gomes et al., 2007; Hillman & Warren, 2004; Tsai et al., 2007; Vanneschi & Veraldi, 2007):

- **Geometrical Changes**: affect the mapping between the internal structure of an application component and the system resources on which it is currently executed. Such class of changes consist in migrating a component in response to specific conditions, e.g. when a system resource fails or a new node is included in the system. These reconfigurations can be useful for load-balancing or for improving the service reliability;
- **Structural Changes**: affect the internal structure of an application component. A notable case is that of a distributed component featuring a parallel implementation in terms of multiple concurrent processes. In this situation structural changes consist in modifying the number of executed processes. This occurs when an insufficient number of nodes are available or if the task scheduler decides to change the set of processors allocated to a specific component;
- **Implementation Changes**: are modifications of the behavior of an application component, which changes completely its internal algorithm but preserving the elaboration semantics and the input and output interfaces with the other application components;
- **Interface Changes**: are intensive modifications of the component behavior, which completely changes the set of its provided external operations and services.

The existing programming models face the reconfiguration support development in two different ways. The first solution consists in putting everything concerning the adaptive behavior of a distributed application, like reconfiguration implementations, inside the run-time support system in such a way as to completely hide these aspects from the programmer standpoint. This *transparent-approach* requires a deep knowledge of the application structures so that the reconfiguration code can be automatically extracted by a static process of compilation of the program. In this scenario the programmer is freed from directly programming the reconfiguration phase but it is only involved in defining, by means of proper directives or programming constructs, the reconfigurations that are admissible (e.g. the compatibility between different versions of a component). On the other hand other programming models adopt a completely different approach to adaptivity, in which the programmer is directly involved in defining the reconfiguration activities. For each possible reconfiguration the programmer must provide the implementation that performs the dynamic reconfiguration, and it is also responsible for performing these activities in a fully correct and consistent manner.

Based on the guidelines provided above, in the rest of this section we will introduce some interesting programming models and frameworks for distributed environments. In the first part we will introduce frameworks for developing adaptive distributed applications for mobile and pervasive systems, whereas in the second part we will concentrate on programming models and tools for adaptive high-performance computations.

2.1 Exploiting adaptive behaviors in mobile and pervasive environments

Some research works provide programming frameworks for mobile applications that adapt their behavior in response to the actual level of resource availability (especially of communication networks). For instance in **Odyssey** (Noble et al., 1997) mobile applications exploit run-time reconfigurations which are noticed by the final users as a

change in the application execution quality. The Odyssey operating system is responsible for exploiting periodically resource monitoring activities and for interacting with mobile applications raising or lowering their corresponding quality level. In this approach all reconfiguration actions are automatically triggered by the run-time system without any direct user intervention. Nevertheless the most important drawback of this approach is the quite limited definition of the quality concept: in many cases it only consists in the quality of the visualized data but this assumption can be restrictive when we consider more complex applications involving an intensive cooperation between computation, communication and visualization phases.

In past researches, adaptivity consists in migrating specific parts of a distributed computation for optimizing performance and energy parameters as well as for reliability reasons. As an example in **Aura** (Garlan et al., 2002) adaptivity is expressed introducing the abstract concept of *task*: i.e. a specific work that a user has submitted to the system which can be completed by several distinct applications suitable for different classes of computing resources (e.g. workstations and smartphones). The framework is responsible for exploiting migration activities in a fully transparent way w.r.t the final users. Unfortunately this approach has been developed for very simple user tasks (e.g. writing a document or preparing a presentation). On the other hand, if we consider more complex mobile applications (e.g. that involve compute-intensive computations), transferring a partially computed task to a different platform can be a critical issue concerning both performance and consistency issues that have not been analyzed in these frameworks.

Recently the execution of high-performance parallel applications on mobile computing platforms have been addressed in some preliminary research works. A relevant example is **MB++** (Lillethun et al., 2007), a framework for developing compute-intensive programs for mobile and pervasive environments. In this approach one of the most important shortfalls is the quite limited utilization of mobile nodes, which are limited to pre-processing or post-processing activities whereas compute-intensive elaborations are executed only on HPC resources. In many time-critical scenarios, such as emergency management systems, we often require the possibility to execute forecasting models and decision support systems on a distributed set of localized mobile resources, equipped with a sufficient computational power (e.g. multicore smartphones).

2.2 Existing programming models for high-performance adaptive applications

Parallel computations are a particular class of tightly coupled distributed applications composed of several cooperating parallel modules. Adaptivity for parallel applications executed on traditional HPC architectures is an important feature, especially for real-time processing in which the presence of strong real-time deadlines and performance constraints require the capability of automatically adapting the application configuration for quickly responding to platform changes. Emergency management systems (e.g. disaster prediction and management, risk mitigation of floods and earthquakes) are notable examples in which the behavior of a time-critical processing needs to be periodically monitored and adapted throughout the execution.

One of the most widespread programming models for parallel applications is based on the **MPI** (Snir et al., 1995) (Message Passing Interface) communication library. In this case parallel

programs are expressed in an un-structured way by describing the behavior of a set of distributed processes cooperating by using communication channels. Last implementations of the MPI library provide some form of support to dynamic reconfigurations. MPI2 (Lusk, 2002) provides a mechanism for instantiating new processes at run-time: this feature can be exploited in such a way as to perform structural changes of a parallel program, for instance it is possible to increase the parallelism degree (e.g. the number of executed processes) achieving in this way an expected better performance. In general the management of adaptivity issues in MPI is completely left to the programmer, which is heavily involved in ensuring and maintaining the consistency and the correctness of the computation during and after the reconfiguration phase.

In order to partially solve the complexity issues of un-structured parallel programming, some notable research works have been proposed as the **ASSIST** (Vanneschi, 2002) programming environment. ASSIST is a general parallel programming framework for several classes of computing architectures, from shared-memory platforms as SMP and NUMA multi-processors to distributed-memory multicomputers as cluster of workstations and large-scale Grids (Coppola et al., 2007a). The most important novelty of this approach is the structured methodology for expressing parallel computations, which are instances of well-known parallelism schemes (e.g. task-parallel programs as task-farm or pipeline and data-parallel computations as map, reduce or communication stencils). This approach is known to the scientific community as *Structured Parallel Programming* (Cole, 2004).

In ASSIST a run-time support to dynamic reconfigurations (Aldinucci et al., 2005; Vanneschi & Veraldi, 2007) is rendered by exploiting a transparent-approach to adaptivity. A dynamic reconfiguration is a change involving a specific application component by modifying: (i) the mapping between execution processes and underlying computing resources (i.e. geometrical changes); (ii) by increasing or decreasing the number of execution processes of a component (i.e. structural changes). Some papers (Coppola et al., 2007b; Danelutto et al., 2007) describe how adaptivity is exploited in the ASSIST framework. Reconfigurations are *performance-oriented* (Aldinucci et al., 2006): dynamic changes of application components are triggered by the run-time support in presence of QoS violations of a pre-defined performance contract.

Another interesting work is the **Behavioural Skeleton** (Aldinucci et al., 2008) approach. Adaptivity for distributed high-performance computations is exploited by means of the Grid Component Model (Ahumada et al., 2007; Mathias et al., 2008) (GCM) and the structured parallel programming paradigm (which the authors also call skeleton-based programming). In GCM a self-adaptive component is composed of two main parts: the Membrane which is an abstract unit responsible for controlling the adaptive behavior of the component, and the Content composed of a set of processes performing the corresponding functional logic of the computation. These entities can also be other GCM components (i.e. inner components): therefore the GCM model makes it possible a natural hierarchical nesting between several self-adaptive components.

This approach is characterized by very interesting run-time mechanisms for controlling multiple non-functional concerns (e.g. it is possible to simultaneously control different parameters as performance and security objectives). In this case the solution proposed in (Aldinucci et al., 2009) provides multiple autonomic managers for a single component,

each one controlling a specific non-functional concern by using a set of event-condition-action (ECA) policy rules. Different policies can lead to some conflicting decisions: in this case the authors propose distributed consensus-based solutions.

In this section we have presented the actual state of the art concerning self-adaptive systems for mobile applications and for traditional high-performance computing problems. From our point of view there is not yet a unified approach for programming adaptive high-performance computations executed on highly heterogeneous and dynamic execution environments, such as pervasive and mobile grid infrastructures. Some research works focus on HPC computations in real-time environments, but in these approaches the "mobile part" of application definition is essentially missing. Other research works achieve the necessary expressiveness to define mobile adaptive applications, but they do not face on compute-intensive real-time computations.

3. Application scenarios

In this chapter we are interested in describing the design and the developing issues of time-critical systems as emergency management applications. They are a notable example of systems that exploit highly heterogeneous distributed computing platforms, composed of traditional high-performance architectures as well as mobile nodes and sensors, providing system features and services according to precise QoS constraints. Such QoS is typically related to the performance at which computing results or communication facilities are provided to users and/or to the service availability and reliability w.r.t. software and hardware failures.

A main issue in executing this class of applications on heterogeneous platforms is hence given by the chance of defining proper programming models and run-time supports aiming at enabling the definition and dynamic satisfaction of QoS constraints. This issue can be seen as even more complex by considering the strong dynamic variability of computing and communication services provided by these platforms, also given by the mobility of some of its nodes and their geographical distribution.

Emergency management systems include data- and compute-intensive processing e.g. forecasting and decision support models) not only for off-line centralized activities, but also for on-line and decentralized activities. Consider the execution of software components performing a forecasting model, which is a critical compute-intensive computation to be executed respecting precise operational real-time deadlines. In "normal" conditions (e.g. concerning the network availability and the emergency scenario itself), we could be able to execute these components on a centralized server, exploiting its processing power to achieve the highest performance possible. Critical conditions in the application scenario (e.g. an emergency detection) can lead to different user requirements (e.g. increasing the performance to complete the forecasting computation within a given, new deadline). Moreover, changes in network conditions (e.g. of the interconnection network with HPC centralized resources) or in computing resource availability can lead to the necessity to execute proper versions of the application components directly on localized mobile resources which are available to the users (e.g. civil protection personnel, rescuers and stakeholders).

In such cases, compute-intensive elaborations that need to be periodically executed for monitoring, predicting and taking response actions during an emergency phase, can be alternatively executed on different or additional computing resources, including sets of distributed mobile resources running properly developed versions of these computations. In other words, in these scenarios it is important to assure the *service continuity*, adapting the application to different user requirements but also to the so-called *execution context*, which corresponds to the actual conditions of the both the surrounding environment and the computing and communication platform. So the key issues in the definition of high-performance parallel programming paradigms, models, and frameworks to design and develop these kinds of complex and dynamic applications, is adaptivity (possibility to adapt the application behavior changing component versions and the platforms on which they are executed) and ***Context Awareness*** (knowledge about the current condition of the reference environment).

The general reference point for these kinds of applications is the Grid paradigm (Berman et al., 2003) which, by definition, aims to enabling the access, selection and aggregation of a variety of distributed and heterogeneous resources and services. However, though notable advancements have been achieved in recent years, current Grid technology is not yet able to supply the needed software tools to match the high-performance feature with high adaptivity, ubiquity, proactivity, self-organization, scalability, interoperability, as well as fault tolerance and security, of the emerging applications running on a very large number of fixed and mobile nodes connected by various kinds of networks.

We claim that a ***high-level programming model*** is the only viable solution to design and develop such kind of distributed applications. A novel approach needs to provide a proper combination of high-performance programming models and pervasive and mobile computing frameworks, in such a way to express a QoS-driven adaptive behavior for critical high-performance applications on heterogeneous contexts. In the following section the main features of a novel approach will be discussed in more detail.

4. Features and requirements of a novel approach

A novel approach needs to be characterized by a strong synergy between two different research fields: *Pervasive and Mobile Computing* (Hansmann et al., 2003) and *Grid Computing* (Berman et al., 2003). Pervasive and Mobile Computing is centered upon the creation of systems characterized by a multitude of different computing and communication resources, whose integration aims at offering seamless services to the users according to their current and time-varying needs and intentions. On the other hand Grid Computing focuses on the efficient execution of compute-intensive processes by using geographically distributed sets of computing resources.

To merge these two areas, an effective integration must provide a proper combination of high-performance programming models and mobile computing frameworks in such a way as to express a QoS-driven adaptive behavior for distributed applications like emergency management systems. In this section we will highlight the main guidelines that need to be followed for exploiting such integration. They can be summarized in three different points: (i) how compute-intensive processing for heterogeneous environments can be programmed

and their performance formally analyzed; (ii) how a distributed application can change its QoS behavior at run-time; (iii) when it is necessary to execute an application reconfiguration.

4.1 Structured Parallel Programming

First of all we require a structured methodology for expressing parallel and distributed programs in such a way as to make the programming effort less costly and less time-consuming and to have a predictable QoS behavior for such applications. This means that we need sufficient high-level abstractions featuring a large-degree of programmability, compositionality and *performance portability*. Portability plays a central role in heterogeneous contexts: portable parallel applications are able to achieve acceptable performance results on different computing platforms, trying to exploit in the best way as possible the underlying physical aspects of the target machine. This can be done by tuning proper application parameters (e.g. parallelism degree and task granularity) statically, during the compiling phase, or dynamically through run-time reconfiguration processes.

For these reasons ***Structured Parallel Programming*** (SPP) (Cole, 2004) has been proposed as an effective and attractive approach to parallel programming featuring interesting properties in terms of high-level programmability and performance portability. According to the SPP methodology a parallel computation is expressed by using well-known abstract parallelism schemes for which parametric implementations of communication and computation patterns are known. In fact the experience in parallel programming suggests that parallel programs make use of a limited number of parallelism patterns exhibiting regular structures, both concerning data organization and partitioning or replication of functions. In this way we can identify several parallelism paradigms as *data parallelism* schemes (e.g. map, reduce, parallel prefix and communication stencils) and *task parallelism* structures (e.g. pipeline, task-farm and data-flow). Furthermore the performance behavior of these parallelism schemes has been studied by exploiting formal analysis (i.e. *performance models*) based on queueing theory and queueing network fundamental results, thus rendering the performance modeling of this class of computations usable also from automatic tools as compilers (e.g. for statically deciding the best application configuration) and from run-time supports (e.g. for providing efficient fault-tolerance (Bertolli, 2009) mechanisms but also for deciding the execution of dynamic reconfigurations (Vanneschi & Veraldi, 2007)).

The SPP methodology is based on the concept of parallelism schemes that exhibit the following features: (i) they are characterized by constrains in the parallel computation structure; (ii) they have a precise semantics; (iii) they are characterized by a specific performance model; (iv) they can be composed each other to form complex graph computations. A central property of these schemes is the complete knowledge of their computation structure, e.g. in terms of data and functions replication and/or partitioning. For instance task-farm paradigm exploits replication only, applied to functions and to non-modifiable data. Pipeline exploits a partition of the sequential elaboration in a sequence of multiple successive phases each one using a set of replicated (non-modifiable) or partitioned (modifiable) data. On the opposite direction data-parallel schemes correspond to the replication of the same functionality and to the partitioning of data, so that distinct parallel units are able to apply the same operations to different data partitions in parallel.

Performance of parallel computations can be measured in terms of throughput, i.e. completed tasks per time unit, and computation latency, i.e. the average time needed to execute the computation on just one input task. Parallelism schemes can have different impacts on these two performance measurements. Task-farm and pipeline, though able to increase the throughput of the computation, also tend to increase the latency compared to the sequential implementation due to communication overhead. Data-parallel and data-flow tend to improve both the computation latency and the throughput, but they can be hampered by load-balancing issues. Finally parallelism schemes are also characterized by a different degree of memory utilization. Task-farm tends to increase the memory capacity, the others, being based on data partitioning, tend to decrease it.

As far as composability is concerned, parallel distributed applications can be represented as graphs of modules. Each module can exploit a sequential or a parallel computation based on a specific parallelism scheme (*intra-module parallelism*), whereas modules can be composed in general complex graph structures (*inter-module parallelism*).

4.2 Dynamic reconfigurations of parallel programs

Previously in this chapter we have introduced the concept of application configuration. Any run-time activity that modifies the parallelism degree, the parallel version (e.g. parallelism scheme and parallelized sequential algorithm) or the execution platform of a module is indicated as a ***dynamic reconfiguration process***.

Although there are specific situations in which the "best" configuration for each module can be statically selected during the system design phase (e.g optimizing performance parameters and memory and energy usage), in dynamic execution conditions this best configuration might not be identifiable without run-time information. As an example consider the following situations:

- in dynamic environments, the degree of availability of computing and network resources can dynamically vary also in unexpected ways;
- for compute-intensive elaborations (as forecasting models and simulations), precise user requirements can be requested. For instance when an emergency scenario is detected, the decision support model that assists civil protection personnel could require to be executed until a maximum completion time;
- for irregular parallel problems, characterized by computations whose size, distribution and complexity depends on the properties of input data, a run-time support to dynamic reconfigurations is an unavoidable feature that needs to be provided.

For structured parallel computations we can classify the set of adaptation processes involving a single application module in two categories namely non-functional and functional reconfigurations.

Non-functional Reconfigurations are adaptation processes involving the run-time modification of some implementation aspects of a parallel module of the application graph. For implementation aspects we intend that:

- it is possible to modify the current parallelism degree of a module, e.g. increasing the number of implementation processes in such a way as to achieve a better performance level (e.g. response time for each request or the completion time of the entire computation);

- the run-time support can modify the mapping (i.e. geometrical changes) between implementation processes of a parallel module and underlying physical computing resources on which they are executed;
- the run-time support can provide another important class of geometrical changes involving the run-time *data re-distribution* among the set of worker processes of a parallel module. Such changes are an effective approach for addressing potential workload un-balancing problems of data-parallel parallelizations.

The most important aspect characterizing non-functional reconfigurations is that in every case these processes do not modify the sequential algorithm performed by the parallel module, neither the exploited parallelization scheme.

Another important and, in some sense, intuitive concept is that the same problem can usually be solved in a parallel fashion by exploiting several different parallelization schemes. For instance let us consider a simple example in which a parallel module periodically receives a sequence of input tasks each one consisting in a pair of a matrix and a vector. For each task the module performs a matrix vector product and the result vector is transmitted to a destination module. According to the way in which we perform data distribution among parallel entities, we can parallelize this problem in several ways. A task-farm scheme consists in replicating the entire matrix-vector computation on multiple independent workers that apply the computation on a different set of scheduled input tasks. In this way we are able to improve the throughput of the computation but not the computation latency. For this simple example several data-parallel schemes can also be adopted. For each input task, if we partition the input matrix among a set of parallel processes and we replicate the input vector, we are able to exploit a classic data-parallel *map* scheme in which each process exploits the matrix-vector computation on its partition without requiring inter-process data exchange and synchronizations. Moreover, if the input vector instead of being replicated is partitioned among processes, we are able to exploit a data-parallel approach with a *communication stencil*. In both the cases the data-parallel approach improve latency of the computation as well as its memory occupation w.r.t the task-farm implementation. Moreover the stencil version requires a higher number of exchanged data during the computation.

The previous scenario suggests that we can use this important feature of structured parallel programming in such a way as to effectively deal with highly dynamic and heterogeneous execution environments. For each application module several alternative versions can be provided, and ***Functional Reconfigurations*** are implementation changes that consist in changing at run-time the version which is currently executed by a parallel module. Versions can have a different but compatible semantics: they can exploit different sequential algorithms, parallelization schemes or optimizations, but preserving the module interfaces in such a way that the selection of a specific version does not modify the global application behavior. Each version of the same parallel module is optimized for being executed whenever certain execution conditions are satisfied (e.g. depending on the actual levels of communication bandwidth, power supply, available memory) or on the presence of specific classes of computing resources.

Another important advantage of this methodology w.r.t non-structured parallel programming models is that it renders feasible a transparent approach to application reconfigurations (see Section 2). As demonstrated in (Aldinucci et al., 2006; Vanneschi, 2002; Vanneschi

& Veraldi, 2007), the reconfiguration code that performs non-functional reconfigurations (as parallelism degree variations) can be automatically extracted from the knowledge of the parallelism scheme that needs to be modified. In this case the programmer is only involved in providing the implementation of the high-level functions executed by a set of worker processes, and their number and the interconnections for task reception and result transmission are completely managed by the run-time support system. Moreover, for functional reconfigurations, in (Bertolli et al., 2011) formal consistent reconfiguration protocols have been presented for switching between different structured parallelism schemes.

4.3 Exploiting adaptation strategies for parallel programs

An adaptive application is a system which evolves over time changing its initial configuration in response to external events. Providing a run-time support to dynamic reconfiguration activities is a first and essential requirement for developing adaptive high-performance applications. Although many research works (Aldinucci et al., 2008; 2005; 2006; Coppola et al., 2007a; Vanneschi & Veraldi, 2007) focus on specific implementation issues describing several optimizations for reducing reconfiguration costs in terms of performance degradations for certain parallelization schemes and their compositions, they do not pay sufficient attention to the decision process that triggers the execution of these reconfigurations. We refer to this process as an ***Adaptation Strategy***, that is a plan or a set of rules used by the system for achieving the specific objectives of its adaptive behavior.

Especially for emergency management systems several constraints need to be satisfied throughout the execution. The system must offer its functionalities to the users according to different QoS notions:

- in many adaptive systems we need to control the execution progress preserving observed metrics within user-defined ranges. Classic examples are: e.g. maintaining the mean response time of the system within a *window of tolerance*, i.e. between a maximum and a minimum acceptable threshold over the execution. In this case we refer to this objective as a **threshold specification** problem;
- alternatively we might require to maintain some quantitative execution metrics as closer as possible to a set of desired reference values. For instance it could be necessary to maintain the mean service time of a computation equal to a specified target value desired by the final user. We refer to this approach as a **set-point regulation** problem;
- more often we need to control several execution measurements of the system, as the performance, energy and memory requirements, number of utilized computing and network resources. In this situation we need to find a control law such that a certain optimality criterion is achieved. This problem can be casted into a mathematical fashion introducing an objective function $F(x_1, x_2, \ldots, x_n)$ of n input parameters whose value should be minimized or maximized. Inputs are system controllable features, and the adaptation strategy is aimed at optimizing this multi-variable function during the system execution. This formulation is known as an **utility/cost optimization** problem.

In different research fields (e.g. Autonomic Computing (Huebscher & McCann, 2008) and Control Theory) general adaptive systems have been described in terms of a reference control architecture (see Figure 1) composed of two main parts and proper interaction phases between them. The two parts are:

- the target controlled system (namely ***plant***) that we want to control. It takes *functional inputs* and generates *functional outputs* according to the semantics of its computation. However, for controlling the behavior of a system, other relationships among non-functional inputs and outputs are also important. System execution can be influenced by a specific set of manipulated *control inputs*, whose values strictly affect the way in which the system execution is exploited. Based on the desired control objectives, the quality of system operation is evaluated by measuring non-functional outputs also called *observed outputs* (e.g. the current level of some performance metrics);
- a ***controller***, which is an independent entity able to monitor and affect the system operational conditions by taking and analyzing observed outputs and promptly decide the corresponding set of control inputs.

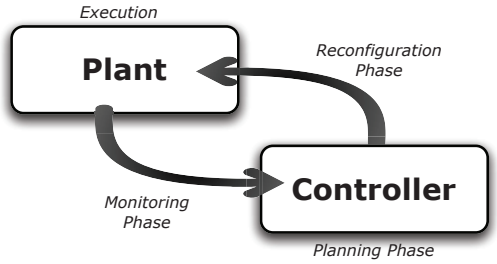

Fig. 1. Control-loop scheme of an Adaptive system: the cycle evolves according to a monitoring phase, a planning phase and a reconfiguration phase.

Interactions between these entities follows a classic closed-loop feedback scheme composed of different phases. *Monitoring phase* involves capturing current properties and measurements of the system which are effective for identifying when the execution of a dynamic reconfiguration can be useful for achieving the desired objectives. *Planning phase* consists in a set of concrete actions aimed to select a new set of control inputs that are the best response to the current observed outputs of the system. Finally the *Reconfiguration phase* applies the decided set of control inputs to the controlled system. In this part of the chapter we are interested in understanding how the planning phase can be exploited and what methodologies can be used. The following two sections will present two different approaches that can be used for addressing the adaptation strategy specification for adaptive systems.

4.3.1 Adaptation strategies expressed by logic rules

A flexible methodology for expressing adaptation strategies, which has been exploited in Pervasive and Mobile scenarios due to its intrinsic simplicity and programmability, consists in defining adaptation processes as reactions to specific system situations. As an example if the available energy level of a mobile device is lower than a specified threshold (e.g. 25 %), the execution of a mobile application can switch to a low-energy version which preserves the battery duration employing a limited utilization of the device display. Therefore a straightforward solution for expressing adaptation strategies consists in providing a mapping between events and execution situations and corresponding reconfigurations (i.e. situation-action pairs) as a finite set of imperative logic rules. System administrators and

designers encode management guidelines as a set of rules that represent the adaptation policy of the system.

Policy rules are a form of guidance used to determine decisions and corresponding actions on the system execution. They have been introduced especially in the field of intelligent agents Russell & Norvig (2003), where abstract entities are able to perceive their environmental conditions and act on the basis of this information in order to maximize their objectives. In this vision an adaptive system is in a specific internal state at every given moment of time, and a set of logic rules may cause an action to be taken an therefore a transition to a different internal state of the system.

Logic rules can be expressed according to several paradigms. Although many techniques exist, with specific modifications and features depending on the particular application context, a general approach is based on **Event-Condition-Action** (ECA) policy rules for programming reactive behaviors. ECA rules have the following basic syntax: *when event if condition then action;*. Informally the abstract semantics is: the occurrence of the event triggers the rule evaluation, the condition is checked in order to ensure that the system is in a specific internal state. If this condition holds, the corresponding action is enforced. Notice that though the state that will be reached by applying the rule is not explicitly expressed, the policy programmer knows the desired effect of the selected action.

If multiple rules are fired simultaneously, the entire set of corresponding actions will be performed at the same time. This may lead to potential conflicting actions (e.g. resulting in conflicting post-adaptation situations). The design of proper techniques for efficiently identifying and resolve such conflicts has been an intensively studied research issue. Although the large research effort, the general problem of avoiding conflicting rules is hard to be solved, especially for complex systems in which the consequence and the side-effects of a dynamic reconfiguration involve the long-term steady-state behavior of the system (i.e. in this case a rule can conflict with rules that may be triggered on different events in future). This suggests that, although highly programmable and user-oriented, *adaptation strategies based on logic rules are difficult to be applied to complex applications as emergency management systems*. In fact in this case it is extremely important to have a reasonable expectation about the consequences of reconfiguration actions on the long-term behavior of the systems (e.g. future performance level after the completion of a reconfiguration process), that can be difficult to be evaluated and represented through situation-action pairs.

4.3.2 Adaptation strategies based on control-theoretic techniques

Besides Artificial Intelligence and Reactive Systems, the concept of automated operations is an important research field in disciplines as mechanical and electrical engineering for developing autonomous systems able to respond to changing workloads and expected run-time conditions. The methodology based on Control Theory (Hellerstein et al., 2004) foundations has been intensively exploited for design controllers and feedback systems in many industrial plants and mechanical infrastructures. These solutions provide powerful mechanisms for dealing with unpredictable changes, uncertainties, and system disturbances.

In many research works several control-theoretic modelings of computing and networking systems have been proposed. Control-based approaches are often a solid solution to

solve network problems like congestion and flow control, and rate adaptation of queueing networks. This leads to an increasingly important research area in which the adaptive system is the computing network itself, i.e. Autonomic Networking (Mortier & Kiciman, 2006), whose interconnection facilities are able to automatically detect, diagnose and repair failures, as well as to adapt the underlying network configuration and optimize its performance and quality of service.

Nevertheless the exploitation of control theory methodologies to computing systems is rarely used in practise. The formal design of controllers needs the ability to precisely quantify the reconfiguration effects on the system evolution. Hence, *system modeling* is a prerequisite for applying such techniques. This model needs to be expressed in an input-output form concerning how control inputs and disturbances affect the behavior of observed measurements. Such relationships between model variables can be determined through *first-principle models*, based on the actual physics laws which govern the evolution of the system, or exploiting *empirical models* in which the relationships are extracted through static techniques (e.g. least-square techniques) on properly defined experimental data and observations.

We claim that a relevant example in which first-principle models can be used consists in the performance modeling of structured parallel computations. For these structured parallelization schemes and their composition in computation graphs, performance models provide a quite acceptable performance predictions of system steady-state behavior, which is often a sufficient guideline to define powerful control-theoretic strategies for controlling parallel applications. When it is possible, these approaches are amenable to provide properties like the *accuracy* of an adaptation strategy (how precisely control objectives are satisfied), *settling time* (how long a reconfiguration lasts until the steady-state behavior of the system is reached) and the *stability degree* of a reconfiguration (how long a system configuration represent a "good choice" for achieving the desired system objectives).

For these reasons control theory techniques, though their applicability is not always straightforward, represent a valuable research direction for controlling time-critical applications as emergency management systems. Based on the main guidelines described in this section, in the rest of this chapter we will introduce in more detail our approach.

5. Design principles of a novel programming model approach

As stated in Section 4.1 our approach is based on a structured methodology in which distributed and parallel applications can be represented as directed graphs of modules interconnected through streams of data [1]. Whereas the structure of the graph can be arbitrary (e.g. cyclic client-server interactions or acyclic graphs of multiple computation phases), the internal parallelism inside each module is expressed by means of structured parallelism schemes.

The adaptive behavior is expressed in the possibility of each parallel module to perform dynamic reconfiguration activities triggered by their own *control logic*. This means that adaptivity is exploited through a set of independent entities that cooperate for implementing the distributed *functional logic* of the application and also for negotiating reconfigurations

[1] A stream of data is a sequence, possibly of unlimited length, of typed data structures.

taken by their control logics. In this section we will introduce the basic structure and the formalization of the concept of adaptive parallel module.

5.1 Structure and modeling of an adaptive parallel module

The core element of our approach is the concept of adaptive parallel module (shortly ***ParMod***), an independent and active unit featuring a parallel elaboration and an adaptation strategy for responding to different sources of dynamicity. From an abstract point of view a ParMod can be structured in two interconnected parts:

- an **Operating Part** is responsible for performing a parallel computation expressed according to a certain parallelism scheme of the SPP framework (e.g. task-farm and data-parallel are relevant examples). Without loss of generality we assume a stream-based computation in which a set of input data streams from other parallel modules are received by the operating part. The parallel elaboration is activated according to a non-deterministic selection of input elements or, based on a data-flow semantics, waiting for the reception of an element from each input interfaces of the module. Result externalization (if it is necessary) is exploited onto output data streams (output interfaces) to other parallel modules;
- a **Control Part** is an autonomous entity able to observe the operating part execution and modify its behavior exploiting reconfiguration activities.

For each ParMod we suppose the presence of multiple alternative configurations of its operating part that differ in the parallelism degree, the structured parallelism scheme or in the execution platform on which they can be deployed. The only constraint that we impose is that every configuration must respect the same input and output interfaces of the ParMod (i.e. admissible reconfigurations are geometrical, structural or implementation changes, see Section 2), in such a way that a local reconfiguration involving a single parallel module does not modify the interfaces provided to other application modules. Therefore each ParMod features a multi-modal behavior defined as follows:

Definition 5.1. (Multi-modal behavior of ParMod Operating Part). At each point of time the operating part can behave according to a certain active configuration belonging to a specific set C of alternatives:

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

This set represents a ***finite and discrete set of statically known alternative configurations*** of the parallel module.

The adaptive behavior of a parallel module is exploited through a periodical information exchange between operating and control part. *Observed outputs* are QoS measurements (e.g. memory occupation, energy consumption, number of completed tasks) that are periodically monitored by the control part. *Control inputs* are proper commands that modify the current operating part configuration exploiting non-functional or functional reconfiguration activities. Observed outputs and control inputs exchange can be exploited periodically, at fixed points equally spaced (in this case we speak about a *time-driven* controller) or whenever specified events are satisfied (as in the case of *event-driven* controllers).

Our ParMod control model is based on a time-driven control part that takes reconfiguration decisions every ***control step*** of duration τ. In other words at the beginning of each control step

the control part acquires observed outputs from operating part, executes a specific control algorithm to decide a corresponding set of control inputs that will be communicated to the operating part.

5.1.1 A Hybrid modeling of the operating part behavior

In order to apply a formal methodology for developing ParMod adaptation strategies, we need to model the QoS temporal evolution of a parallel computation by using a mathematical and formal approach. This means that we need proper mathematical equations that express the relationship among observed output measurements and current control inputs. As we have seen in Section 4.3.2, the structured parallel programming paradigm is a basic point in order to establish such modeling. Relevant examples are:

- structured parallelism schemes feature a sort of predictability of their steady-state performance level. Fixed the parallelism degree and the execution platform, several performance measurements as the mean service time, the mean queue length and the computation latency can be predicted and formally analyzed according to analytical models based on queueing theory and queueing network results (Vanneschi, 2002);
- memory utilization models can also be derived for well-known parallelism schemes by exploiting the specific behavior of different parallelization patterns in terms of function and data replication or partitioning (Bertolli et al., 2010);
- structured parallelism schemes have a precise semantics in terms of computations performed by each parallel unit, the size of exchanged messages and the frequency of certain activities as calculation and message transmission. Such knowledge can be a starting point in order to define models for measuring the power consumption of a parallel module, especially when its execution is mapped onto energy-limited resources as mobile devices.

A formal model that describes the predictability of QoS parameters involves the following set of variables:

- ***state variables*** represent system measurements that are useful to maintain during successive control steps. These variables can describe the global energy consumption caused by the parallel execution of a ParMod, the number of queued tasks, the global number of completed input stream elements. We refer with the term $\mathbf{x}(k)$ as the current value of the internal state of the operating part model at the beginning of the control step k. State variables are directly monitored at each control step by the control part;
- ***disturbance variables*** are uncontrolled exogenous signals that can affect the relationship between control inputs and the observed state variables modeling the actual QoS of the computation. Uncontrolled means that the control part is not able to decide and fix their values, but they are determined by environmental or external decisions outside of ParMod control. Relevant examples are the mean calculation time of a sequential function on a certain target architecture, the communication latency and the mean inter-arrival time of requests to the system. We refer with the term $\mathbf{d}(k)$ as the value assumed by disturbances throughout the k-th control step;
- ***control inputs***, $\mathbf{u}(k)$, indicate the ParMod configuration that will be exploited throughout the k-th control step.

According to the multi-modal behavior of the operating part and the QoS predictability of different ParMod configurations, we can identify two classes of transitions that characterize the ParMod execution:

- *continuous transitions*: when a configuration has been fixed, the evolution of continuous-valued QoS state parameters can be predicted by applying a specific mathematical model corresponding to the currently used configuration;
- *discrete transitions*: by executing reconfiguration activities, the current configuration can be changed passing from a configuration C_i to a different alternative configuration C_j, shortly $C_i \rightarrow C_j$ with $i \neq j$.

The presence of continuous transitions (of continuous variables) and discrete transitions (of alternative configurations) suggests to model the operating part of a ParMod as an *hybrid system* (der & Schumacher, 1999) in which these two dynamics are formally modeled in a unique and refined mathematical structure.

For each configuration $C_i \in C$ the temporal evolution of observed outputs and of the next state variables can be expressed by a model in a state-space form as shown in (2).

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

The semantics of the next state expression provides that, if the values of expected measured disturbances and the current model state for control step k are known, by applying a function ϕ_i we can predict the values assumed by state variables at the beginning of the next control step $k+1$.

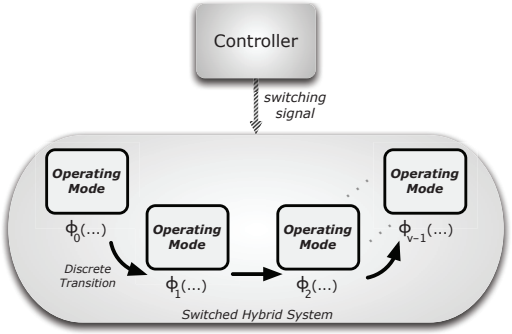

Fig. 2. The operating part evolves according to continuous transitions (i.e. concerning QoS variables predicted through the model of the current configuration), and discrete transitions (reconfigurations).

Two considerations are important at this point. First of all the function ϕ_i can be obtained by applying statistical methods or, as we have seen, by using a higher level knowledge of the structure of the computation behavior in order to extract first-principle relations. The second point is that each model ϕ_i is strictly coupled with a configuration of the ParMod. In some cases a unique mathematical model can be parametrically provided for multiple configurations (e.g. as for different parallelism degrees), but there are also situations in which configurations can require a completely different modeling of the state variables dynamics

(e.g. as for the case of parallel versions expressed by means of different parallelism schemes as task-farm and data-parallel ones).

This variable-structure behavior of the operating part is typical of a large class of hybrid systems featuring a limited set of alternative operating modes (i.e. configurations in our case). Such class consists in the so-called ***Switched Hybrid Systems*** Liberzon (2003) (see Figure 2) in which control inputs from the controller modify the actual configuration of the operating part, and future QoS measurements can be predicted following the evolution rule strictly coupled with the new configuration. In conclusion the entire operating part model is given by:

$$\begin{aligned} &\mathbf{x}(k+1) = \Phi\Big(\mathbf{x}(k), \mathbf{d}(k), \mathbf{u}(k)\Big) \\ &= \text{ if } \Big(\pi\big(\mathbf{u}(k)\big) = C_i\Big) \text{ then } \phi_i\big(\mathbf{x}(k), \mathbf{d}(k)\big) \end{aligned} \tag{3}$$

in which function Φ is a piecewise-defined function that provides a different definition for each alternative configuration of the parallel module, whereas π is a bijective function that maps each possible value of control inputs onto corresponding configuration indices.

5.2 A Control-theoretic adaptation strategy for ParMods

In this section we will propose the exploitation of a control-theoretic technique for controlling the QoS behavior of an adaptive parallel module. We point out that in the context of time-critical applications, like emergency management systems, a classic reactive approach in which the system reacts to well-identified circumstances in pre-programmed way (as in classic environments for adaptive HPC applications) is not an effective adaptation strategy. The criticity of QoS requirements for these applications imposes that reconfiguration decisions must be selected taking into account the history of the system, trying to respond in a pro-active fashion to the user requirements and to the future behavior of the surrounding execution environments. In this case we need adaptation strategies that are more similar to the ones studied in pervasive systems (see Section 2), in which taking actions in advance to potentially critical future events is a central point of this approach.

The *predictive control* approach that we propose for adaptive parallel applications is based on the following points:

- the presence of (empirical or first-principle) mathematical models that can be used to estimate future QoS behavior of the computation in function of a planned sequence of control inputs (i.e. a *reconfiguration plane*) and future predictions of disturbances;
- the presence of an objective function describing the control aims which drive the selection of the optimal control input sequence;
- the specification of boundary conditions on state and control input variables.

This formulation of the predictive control problem is known to control theorists as ***Optimal Control*** (Bertsekas, 1995). Optimal control is the process of determining control and state trajectories for a system over a certain period of time in order to optimize a properly defined objective function. A typical representation of an objective function is given below:

$$\max\ U(k) = \sum_{i=k}^{k+h-1} L\Big(\mathbf{x}(i+1), \mathbf{u}(i)\Big) \tag{4}$$

The function represents an aggregate utility (or cost) which depends on the desirability of future system internal states and control inputs taken for a horizon of N successive control steps (e.g. ideally the whole execution duration). Expressing control preferences among different future states and control input trajectories is an expressive way to define powerful control strategies for structured parallel computations featuring a predictable behavior. Objective functions can be tuned to satisfy the specifications of the system, the control objectives as well as to find feasible trade-offs between conflicting requirements (e.g. optimization problems can exploit an intrinsic multi-variable nature including performance, power and memory measurements and specific constraints on state and control variables).

If we suppose to know the exact trajectory assumed by disturbance inputs for the whole execution duration, and if the system model is sufficiently accurate and precise, we can statically define an optimal reconfiguration plane that optimizes the ParMod execution. Of course this assumption is not always feasible for the following reasons:

- disturbance inputs are variables whose behavior (e.g. average values) can not be statically known a priori but they may depend on uncontrollable factors (as the actual conditions of the underlying execution platform);
- the system model can be effected by perturbations and unmodeled dynamics (e.g. due to unmeasured disturbances) that limit the quality of the future QoS estimations.

For the previous reasons applying the optimal reconfiguration plane step-by-step in an open-loop fashion is not a viable approach. Nevertheless several sub-optimal approaches are available in order to iteratively apply optimization problem solutions in a closed-loop fashion. In the next section one of this technique, namely the ***Model-based Predictive Control***, will be introduced.

5.2.1 Model-based predictive control of structured parallel computations

Model-based predictive control (Garcia et al., 1989) (shortly **MPC**) is a repetitive procedure that combines the advantages of long-term planning (feedforward control based on system predictions over a future horizon) with the advantages of a feedback control using actual system and disturbance measurements. At the beginning of each control step k the values of the model state variables and of past disturbances are measured by the control part. At this point a limited future time horizon (i.e. a *prediction horizon*) of h consecutive control steps is considered, and a prediction of disturbances for this short time interval is exploited through proper statistical techniques. The predicted disturbance trajectory is composed by:

$$D_k^{k+h-1} = \{\hat{\mathbf{d}}(k|k), \hat{\mathbf{d}}(k+1|k), \ldots, \hat{\mathbf{d}}(k+h-1|k)\} \tag{5}$$

where the syntax $\hat{\mathbf{d}}(k+i|k)$ means that the estimated value of disturbance inputs for the step $k+i$ is predicted using the current knowledge at control step k.

These predictions are used to plan a sequence of optimal reconfiguration decisions for each control step of the prediction horizon. The optimization problem (4) is solved for the limited time horizon finding an optimal control input trajectory:

$$U_k^{k+h-1} = \{\mathbf{u}(k|k), \mathbf{u}(k+1|k), \ldots, \mathbf{u}(k+h-1|k)\} \tag{6}$$

Instead of applying the optimal reconfiguration plane step-by-step, the uncertainty of disturbance estimation (which increases going deeper in the prediction horizon) and the potential inaccuracy of the future QoS predictions, suggest a more effective approach based on an iterative procedure. Only the first control decision of the optimal trajectory is transmitted from the control part to the operating part at the beginning of the current step k while the rest are discarded. This process is repeated at the beginning of the next control step when: (i) the new system state is available; (ii) values of past disturbances for the previous step have been measured. The effect of this control algorithm is to move the prediction horizon towards the future following the so-called ***receding or rolling horizon*** technique.

Based on the QoS predictability of structured parallelism schemes, the benefits of the predictive approach are evident. Suppose to have a ParMod control objective formulated as a threshold specification problem: e.g. we need to maintain the observed mean throughput (completed tasks per time unit) of a parallel computation within an acceptable region of values (e.g. between a maximum and a minimum threshold established by the user). Although an initial configuration (e.g. parallelism degree) can be identified during the design phase such that this QoS requirement is met, future modifications of the initial conditions modeled as disturbances (e.g. time-varying task grain) can prevent this configuration to be no longer acceptable. Especially for time-critical applications the utilization of a predictive strategy can be a valuable solution for two main reasons:

- selecting in advance reconfiguration actions can be crucial in order to anticipate undesired behaviors and promptly mitigate the future variability of disturbances;
- typically reconfigurations involve a cost on the execution. After a reconfiguration a ParMod can incur some dead-time, e.g. for completing reconfiguration protocols (see (Bertolli et al., 2011)) or rather the cost for changing the number of currently used computational resources (e.g. as in Cloud Computing environments).

Despite MPC is a largely used approach in many real-world scenarios (e.g. for controlling chemical, mechanical and industrial plants), it is considered a compute-intensive technique especially for hybrid systems (as a ParMod) where the set of control inputs is discrete. In this case the on-line optimization problem requires to explore a search space whose size grows exponentially with the length of the prediction horizon. The search space can be represented as a complete tree (i.e. the ParMod ***evolution tree***) with a depth equal to the prediction horizon length h and an arity that coincides with the number of ParMod configurations ν. The number of explored nodes is given by $\sum_{i=0}^{h} \nu^i$ thus exponential in the prediction horizon length. If not properly addressed with specific techniques (e.g. branch&bound approaches) or heuristics, the combinatorial optimization problem that needs to be solved at each control step implies an exhaustive search by testing among all the feasible combinations of reconfigurations, thus potentially limiting its viability to systems with long sampling intervals. Nevertheless in practical scenarios, since the number of ParMod configurations is often sufficiently limited and prediction horizons are normally short due to disturbance prediction errors, this approach can be exploited in practise without complex search space reduction techniques.

6. Experiments discussion

In this section we will report a summary of experimental results that we have presented in a more comprehensive way in our past publications. Our test-bed application describes

a limited but interesting part of a complex emergency management system for flood predictions, that we have already introduced in Section 3. The application graph is depicted in Figure 3. The system is responsible for providing short-term forecasting results of a flood

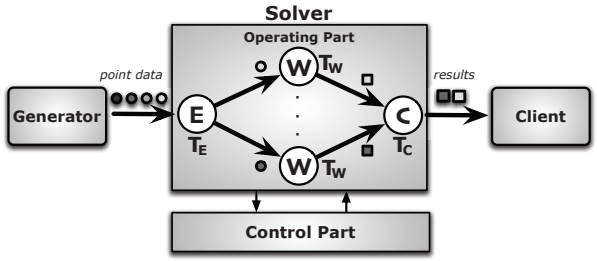

Fig. 3. Flood Emergency Management Application: this test-bed scenario considers a limited set of application modules.

scenario in a real-time fashion to a set of interested users (e.g. emergency stakeholders or rescuers). The graph is composed of a ***Generator*** module that periodically produces 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 an adaptive ***Solver*** ParMod, that numerically solves a system of differential equations describing the flow behavior at the surface. Results evaluation is exploited by a ***Client*** component that performs post-processing and visualization activities on the mobile devices of the users.

An example of flood forecasting model is the TUFLOW hydrodynamic model Charteris et al. (2001), which is based on mass and momentum partial differential equations to describe the flow variation at the surface. Their discrete resolution requires, for each time slice, the resolution of a very large number of sparse tri-diagonal linear systems. For their resolution several highly scalable techniques have been developed over the last decades. In (Bertolli, Buono, Mencagli & Vanneschi, 2009) we have described two sequential variants of the *cyclic reduction* algorithm (Hockney & Jesshope, 1988).

We have provided two distinct parallel versions of the Solver ParMod: the first variant of the cyclic reduction, which requires a smaller number of operations than the second one, is executed inside a task-farm scheme, in which an emitter entity schedules input tasks (points) among a set of workers that execute in parallel the computation on different stream elements. The second version consists in a data-parallel version of the second variant. Although this algorithm requires a greater number of operations, data dependencies and also a higher memory utilization, it is amenable to be implemented in parallel through data-parallel techniques, in which a set of workers cooperatively compute the calculation on a partition of the same input data-structure. Moreover, in order to implement the data dependencies imposed by algorithm semantics, workers communicate with each other following a statically known communication stencil whose form varies between different iterations of the computation.

Such parallelizations respond to different QoS requirements: task-farm is able to increase the throughput of the computation in terms of number of completed points per time unit,

whereas, the data-parallel scheme, can also improve the computation latency for completing a single task elaboration.

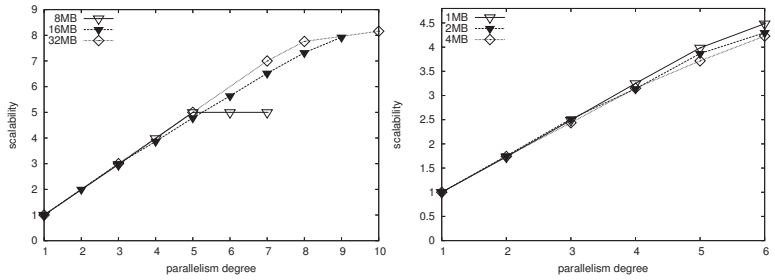

Fig. 4. Scalability of different versions of the flood forecasting model: task-farm on a cluster (left), data-parallel on a multicore architecture (right).

The two versions of the Solver ParMod are suitable for the execution on different computing architectures. The task-farm structure does not require intensive data exchanges between processes, therefore it can be efficiently implemented on a distributed-memory architecture (e.g. a cluster of production workstations). The data-parallel approach is instead amenable of being executed on a shared-memory architecture as a multicore platform. Scalability of the two versions is depicted in Figure 4, showing how different parallel versions achieve near optimal scalability results on highly different computing architectures.

Since the task-farm is based on replication of input data of different stream elements whereas the data-parallel applies a partitioning of the input data of the same task, we expect a different memory utilization of the two parallelization approaches. As described in (Bertolli et al., 2010), models describing the actual memory utilization of a structured parallel computation in function of its configuration parameters, as the parallelism degree and the parallelism scheme, provide accurate results as we can observe in the experiments depicted in Figure 5.

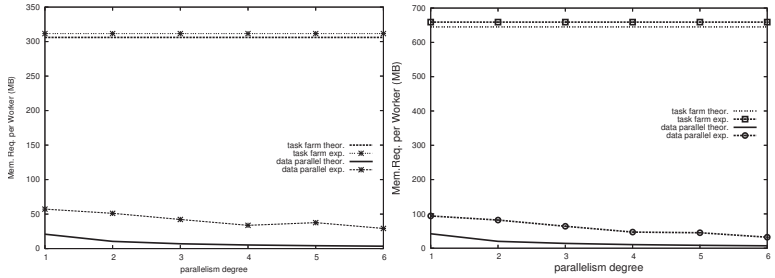

Fig. 5. Memory utilization per worker with tasks of size 16 MB (left) and 32 MB (right): comparison between task-farm and data-parallel schemes.

The experienced memory behavior follows the predicted values: i.e. for the task-farm scheme, based on a pure replication, the worker memory usage is independent on the parallelism degree; for the data-parallel scheme, based on a partitioning of input data, increasing the parallelism degree results on smaller partitions and thus on a lower memory usage per worker. This demonstrates how a proper versioning of a parallel module is useful not only for exploiting in the best way as possible different architectures on which the computation can be

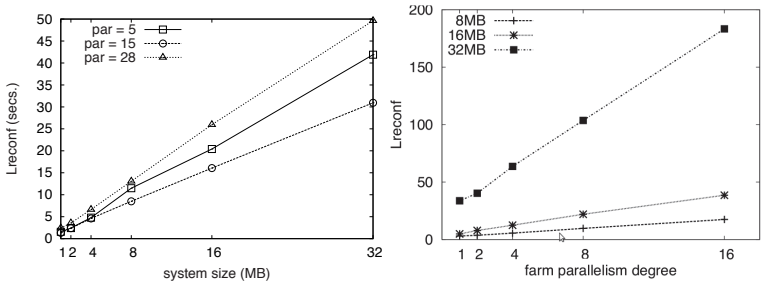

Fig. 6. L_{reconf} for the rollforward (left) and the rollback (right) switching form a source version (a task-farm) to a target version (a data-parallel) of the Solver ParMod. For the rollforward protocol L_{reconf} is dominated by the time spent for completing all the pending tasks in the source version, before starting the target one. For the rollback protocol L_{reconf} indicates the time necessary for re-executing all the rolled-back tasks.

executed, but also the memory utilization of a parallel program can be adapted in response to the current memory availability of the underlying execution platform.

In (Bertolli, Mencagli & Vanneschi, 2009) we have discussed adaptation strategies that modify the Solver behavior in response to the actual availability of network and computing resources. Especially for functional reconfigurations, in which the ParMod computation is migrated from a source to a target environment changing the parallelism version, we need to precisely estimate the overhead induced by these activities over the execution, in order to establish if a reconfiguration action is useful for the execution or not. In (Bertolli et al., 2011) we have presented different reconfiguration protocols for exploiting functional reconfigurations. A ***rollforward protocol*** consists in switching from the source to the target version only when all the pending tasks in execution in the source version of the ParMod have been completely processed. On the other hand a ***rollback technique*** minimizes the intervention of the source version on the protocol: i.e. when a reconfiguration is started, the target version starts its execution immediately, re-executing possible tasks that have been completed (or they are in execution) in the source version of the ParMod. Also in this case, based on the structured parallel programming paradigm, we are able to quantify the overhead induced by different protocols and choosing the best one for each specific case. Figure 6 depicts the time spent for applying a reconfiguration (i.e. *reconfiguration latency* L_{reconf}) in function of the task size.

Finally the exploitation of advanced control-theoretic techniques for controlling the Solver computation has been described in (Mencagli & Vanneschi, 2011) for a Cloud Computing scenario. We have analyzed the problem of executing the task-farm version of the Solver ParMod on a remote cloud architecture on which the cost of execution of a parallel computation is proportional to the number of virtualized resources (e.g. CPU) effectively used. Moreover, in order to discourage too many resource re-organizations, we have supposed the existence of a business model in which a fixed cost should be paid each time a new resource request is submitted to the cloud system.

In this scenario the design of effective adaptation strategies for optimizing the Solver execution are of great importance. For this application we have supposed a QoS objective with a twofold nature. First of all we need to complete the forecasting computation in the minimum

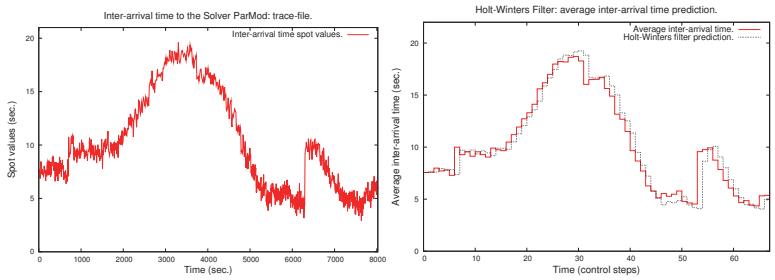

Fig. 7. Trace-file of the inter-arrival time experienced during the execution (left) and the statistical predictions obtained with an Holt-Winters filtering (right).

completion time as possible, such that emergency stakeholders can effectively plan proper response actions in advance to potentially dangerous events. Second we need to minimize the operational cost of the Solver execution on the cloud environment.

For this reason we have adopted the model-based predictive control strategy for the Solver ParMod, starting from a hybrid modeling of interesting QoS measurements of this computation (e.g. number of completed tasks and current operational cost). Due to a time-varying network availability, we have assumed a dynamic workload consisting in a variable mean inter-arrival time[2] of tasks from the Generator module. In Figure 7(left) is depicted the inter-arrival time of tasks experienced during an execution scenario in which the network that interconnects the Generator and the Solver ParMods alternates congestion phases and high-reliability phases (through the exploitation of an advanced network simulation tool, see (Mencagli & Vanneschi, 2011) for further details). In our modeling we have supposed the mean inter-arrival time as a disturbance input that needs to be predicted over a limited horizon. To do this we have exploited a *Holt-Winters filter* which is able to capture non-stationarity processes (e.g. trends and level shifts) of the underlying time-series of past inter-arrival time observations, achieving good prediction results as shown in Figure 7(right).

The task-farm scheme is a parallelization pattern featuring a clear and well-defined performance model: i.e. fixing the task-farm configuration in terms of mean inter-arrival time, service times of the emitter, the collector functionalities, and the number of parallel workers, we are able to analyze the task-farm performance behavior as a queueing network and evaluating the mean inter-departure time of results from this parallel computation.

This performance modeling can be exploited in order to predict how the number of computed tasks evolves during the execution. In (Mencagli & Vanneschi, 2011) the MPC approach has been applied providing an utility function that describes a proper trade-off between completed tasks and operational cost. Limited prediction horizons (e.g. 1, 2 and 3 control steps) have been applied for this experiment.

Figure 8(left) compares different adaptation strategies: (i) a reactive approach as the one described in (Bertolli, Buono, Mencagli & Vanneschi, 2009), in which the parallelism degree is modified according to the actual performance measurements of the computation; (ii) a

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

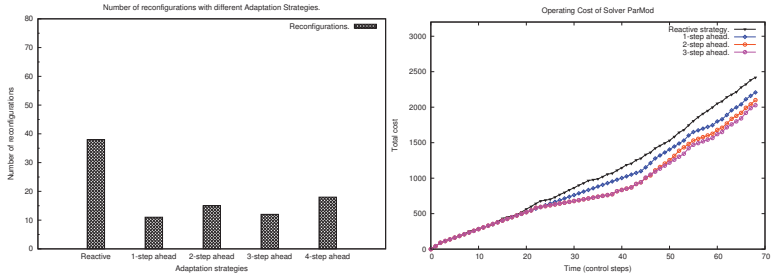

Fig. 8. Number of parallelism degree variations (left) and long-term operational cost of the Solver ParMod execution (right).

model-based predictive control technique that exploits the QoS predictability of the task-farm scheme. As we can observe the predictive approach has a positive impact on the *stability degree* of a ParMod configuration: i.e. for every prediction horizon length, the MPC strategy always features a lower number of reconfigurations than the reactive approach.

Figure 8(right) depicts the long-term operating cost throughout the execution. The importance of having a predictive run-time parallelism degree adaptation is clearly highlighted. W.r.t a purely reactive adaptation, the MPC strategy is able to reduce the operating cost of even the 20% if we exploit a prediction horizon of three steps, thus demonstrating how taking reconfigurations in advance to future workload predictions can be an effective adaptation technique.

7. Conclusion

In this chapter we have provided the design principles of a novel programming model approach for time-critical distributed applications (and notably emergency management systems). Such applications strongly require properties like the predictability of QoS parameters in function of the actual application configuration expressed in terms of identified software components, their interconnections, where they are currently deployed and how their computations are actually exploited. In order to meet these requirements our approach is based on a central point: the employment of the Structured Parallel Programming methodology.

Structured parallelism schemes are based on well-defined parallelization paradigms in terms of data replication or partitioning and function replication. We are able to define multiple alternative versions of the same software component performing critical tasks as forecasting models and simulations, as well as decision support systems. Such versions, besides being characterized by a predictable QoS behavior expressed through an analytical analysis (e.g. Queueing Network models), are also an effective way to deal with heterogeneous set of computing resources where the computation can be alternatively deployed. Moreover, this structured approach is amenable to provide predictable reconfiguration costs (e.g. from migrating the computation between different architectures) and also for being controlled through control-theoretic techniques (as the model-based predictive control procedure), in which the QoS predictability is exploited to plan reconfiguration actions in advance to critical events and optimizing the system execution and its operating cost.

8. References

Ahumada, S., Apvrille, L., Barros, T., Cansado, A., Madelaine, E. & Salageanu, E. (2007). Specifying fractal and gcm components with uml, *SCCC '07: Proceedings of the XXVI International Conference of the Chilean Society of Computer Science*, IEEE Computer Society, Washington, DC, USA, pp. 53–62.

Aldinucci, M., Campa, S., Danelutto, M. & Vanneschi, M. (2008). Behavioural skeletons in gcm: Autonomic management of grid components, *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pp. 54–63.

Aldinucci, M., Coppola, M., Danelutto, M., Vanneschi, M. & Zoccolo, C. (2005). Assist as a research framework for high-performance grid programming environments, *Grid Computing: Software environments and Tools*, Springer, pp. 230–256.

Aldinucci, M., Danelutto, M. & Kilpatrick, P. (2009). Autonomic management of non-functional concerns in distributed & parallel application programming, *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, Washington, DC, USA, pp. 1–12.

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

Arshad, N., Heimbigner, D. & Wolf, A. L. (2007). Deployment and dynamic reconfiguration planning for distributed software systems, *Software Quality Control* 15(3): 265–281.

Berman, F., Fox, G. & Hey, A. J. G. (2003). *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, Inc., New York, NY, USA.

Bertolli, C. (2009). *Fault tolerance for High-Performance applications using structured parallelism models*, VDM Verlag, SaarbrÃijcken, Germany.

Bertolli, C., Buono, D., Mencagli, G. & Vanneschi, M. (2009). Expressing adaptivity and context-awareness in the assistant programming model, *Proceedings of the Third International ICST Conference on Autonomic Computing and Communication Systems*, pp. 38–54.

Bertolli, C., Mencagli, G. & Vanneschi, M. (2009). Adaptivity in risk and emergency management applications on pervasive grids, *Proceeding of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, p. to appear.

Bertolli, C., Mencagli, G. & Vanneschi, M. (2010). Analyzing memory requirements for pervasive grid applications, *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on* 0: 297–301.

Bertolli, C., Mencagli, G. & Vanneschi, M. (2011). Consistent reconfiguration protocols for adaptive high-performance applications, *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pp. 2121 –2126.

Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control, Two Volume Set*, 2nd edn, Athena Scientific.

Charteris, A., Syme, W. & Walden, W. (2001). Urban flood modelling and mapping 2d or not 2d, *Proceedings of the 6th Conference on Hydraulics in Civil Engineering: The State of Hydraulics*, Barton A.C.T: Institution of Engineers, Barton, Australia, pp. 355–363.

Cole, M. (2004). Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming, *Parallel Comput.* 30(3): 389–406.

Coppola, M., Danelutto, M., Tonellotto, N., Vanneschi, M. & Zoccolo, C. (2007a). Execution support of high performance heterogeneous component-based applications on the grid, *Euro-Par'06: Proceedings of the CoreGRID 2006, UNICORE Summit 2006, Petascale Computational Biology and Bioinformatics conference on Parallel processing*, Springer-Verlag, Berlin, Heidelberg, pp. 171–185.

Coppola, M., Danelutto, M., Tonellotto, N., Vanneschi, M. & Zoccolo, C. (2007b). Execution support of high performance heterogeneous component-based applications on the grid, *Euro-Par'06: Proceedings of the CoreGRID 2006, UNICORE Summit 2006, Petascale Computational Biology and Bioinformatics conference on Parallel processing*, Springer-Verlag, Berlin, Heidelberg, pp. 171–185.

Danelutto, M., Vanneschi, M. & Zoccolo, C. (2007). A performance model for stream-based computations, *PDP '07: Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, IEEE Computer Society, Washington, DC, USA, pp. 91–96.

der, S. A. J. v. & Schumacher, J. M. (1999). *Introduction to Hybrid Dynamical Systems*, Springer-Verlag, London, UK.

Garcia, C. E., Prett, D. M. & Morari, M. (1989). Model predictive control: theory and practice a survey, *Automatica* 25: 335–348.
URL: *http://portal.acm.org/citation.cfm?id=72068.72069*

Garlan, D., Siewiorek, D., Smailagic, A. & Steenkiste, P. (2002). Project aura: Toward distraction-free pervasive computing, *IEEE Pervasive Computing* 1(2): 22–31.

Gomes, A. T. A., Batista, T. V., Joolia, A. & Coulson, G. (2007). Architecting dynamic reconfiguration in dependable systems, pp. 237–261.

Hansmann, U., Merk, L., Nicklous, M. S. & Stober, T. (2003). *Pervasive Computing : The Mobile World (Springer Professional Computing)*, Springer.

Hellerstein, J. L., Diao, Y., Parekh, S. & Tilbury, D. M. (2004). *Feedback Control of Computing Systems*, John Wiley & Sons.

Hillman, J. & Warren, I. (2004). An open framework for dynamic reconfiguration, *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society, Washington, DC, USA, pp. 594–603.

Hockney, R. W. & Jesshope, C. R. (1988). *Parallel Computers Two: Architecture, Programming and Algorithms*, IOP Publishing Ltd., Bristol, UK, UK.

Huebscher, M. C. & McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications, *ACM Comput. Surv.* 40(3): 1–28.

Liberzon, D. (2003). *Switching in Systems and Control*, BirkhÃd'user Boston Production, Boston, USA.

Lillethun, D. J., Hilley, D., Horrigan, S. & Ramachandran, U. (2007). Mb++: An integrated architecture for pervasive computing and high-performance computing, *RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE Computer Society, Washington, DC, USA, pp. 241–248.

Lusk, E. (2002). Mpi in 2002: has it been ten years already?, p. 435.

Mathias, E., Baude, F. & Cave, V. (2008). A gcm-based runtime support for parallel grid applications, *CBHPC '08: Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance*, ACM, New York, NY, USA, pp. 1–10.

Mencagli, G. & Vanneschi, M. (2011). Qos-control of structured parallel computations: a predictive control approach, *Technical Report TR-11-14*, Department of Computer Science, University of Pisa, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy.

Mortier, R. & Kiciman, E. (2006). Autonomic network management: some pragmatic considerations, *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, INM '06, ACM, New York, NY, USA, pp. 89–93. URL: *http://doi.acm.org/10.1145/1162638.1162653*

Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J. & Walker, K. R. (1997). Agile application-aware adaptation for mobility, *SIGOPS Oper. Syst. Rev.* 31(5): 276–287.

Russell, S. J. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, Pearson Education.

Snir, M., Otto, S. W., Walker, D. W., Dongarra, J. & Huss-Lederman, S. (1995). *MPI: The Complete Reference*, MIT Press, Cambridge, MA, USA.

Tsai, W. T., Song, W., Chen, Y. & Paul, R. (2007). Dynamic system reconfiguration via service composition for dependable computing, *Proceedings of the 12th Monterey conference on Reliable systems on unreliable networked platforms*, Springer-Verlag, Berlin, Heidelberg, pp. 203–224.

Vanneschi, M. (2002). The programming model of assist, an environment for parallel and distributed portable applications, *Parallel Comput.* 28(12): 1709–1732.

Vanneschi, M. & Veraldi, L. (2007). Dynamicity in distributed applications: issues, problems and the assist approach, *Parallel Comput.* 33(12): 822–845.

Emergency Management
Edited by Dr. Burak Eksioglu

ISBN 978-953-307-989-9
Hard cover, 90 pages
Publisher InTech
Published online 27, January, 2012
Published in print edition January, 2012

After the large-scale disasters that we have witnessed in the recent past, it has become apparent that complex and coordinated emergency management systems are required for efficient and effective relief efforts. Such management systems can only be developed by involving many scientists and practitioners from multiple fields. Thus, this book on emergency management discusses various issues, such as the impact of human behavior, development of hardware and software architectures, cyber security concerns, dynamic process of guiding evacuees and routing vehicles, supply allocation, and vehicle routing problems in preparing for, and responding to large scale emergencies. The book is designed to be useful to students, researchers and engineers in all academic areas, but particularly for those in the fields of computer science, operations research, and human factor. We also hope that this book will become a useful reference for practitioners.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821