Assurance of Self-adaptive Controllers for the Cloud

Alessio Gambi[1], Giovanni Toffetti[1], and Mauro Pezzè[1,2]

[1] University of Lugano 6904, Lugano, Switzerland
[2] University of Milano Bicocca 20100, Milan, Italy
{alessio.gambi,toffettg,mauro.pezze}@usi.ch

Abstract. In this paper we discuss the assurance of self-adaptive controllers for the Cloud, and we propose a taxonomy of controllers based on the supported assurance level. Self-adaptive systems for the Cloud are commonly built by means of controllers that aim to guarantee the required quality of service while containing costs, through a careful allocation of resources. Controllers determine the allocation of resources at runtime, based on the inputs and the status of the system, and referring to some knowledge, usually represented as adaptation rules or models. Assuring the reliability of self-adaptive controllers account to assuring that the adaptation rules or models represent well the system evolution. In this paper, we identify different categories of control models based on the assurance approaches. We introduce two main dimensions that characterize control models. The dimensions refer to the flexibility and scope of the system adaptability, and to the accuracy of the assurance results. We group control models in three main classes that depend on the kind of supported assurance that may be checked either at design or runtime. Controllers that support assurance of the control models at design time privilege reliability over adaptability. They usually represent the system at a high granularity level and come with high costs. Controllers that support assurance of the control models at runtime privilege adaptability over reliability. They represent the system at a finer granularity level and come with reduced costs. Controllers that combine different models may balance verification at design and runtime and find a good trade off between reliability, adaptability, granularity and costs.

1 Introduction

The Cloud paradigm allows for a more efficient use of computing resources, by decoupling software applications from their execution environment. The Cloud infrastructure disconnects applications from the execution environments by introducing a stack of abstraction layers that isolate the execution infrastructure –Infrastructure as a Service (IaaS)– the overall platform –Platform as a Service (PaaS)– and the provided services –Software as a Service (SaaS)–[1].

In this chapter, we focus on the IaaS layer that takes care of allocating resources to applications. Applications shall guarantee the qualities specified by

J. Cámara et al. (Eds.): Assurances for Self-Adaptive Systems, LNCS 7740, pp. 311–339, 2013.

their service level agreements (SLA), which usually associate penalties to SLA violations. At the same time resources come with costs, and providers aim to minimize costs, in order to increase competitiveness and profit [19]. To cope with an unpredictable set of combinations of application requests, service level agreements and usage patterns, the IaaS layers implement self-adaptive controllers, that is, controllers that adapt to different scenarios of applications to be executed, service requirements and traffic conditions.

1.1 An Example of Dynamic Resource Provisioning in the Cloud

To decide how to efficiently allocate resources, self-adaptive controllers refer to some knowledge that is provided in the form of either rules or models. Self-adaptive controllers use rules or models to evaluate different strategies, and chose the best possible tactic to cope with a degrading quality of service in the presence of varying traffic conditions, while avoiding indirect interferences between applications. Dually, self-adaptive controllers use rules or models to chose the right strategy to cope with increasing costs due to overallocated resources when traffic and application conditions change.

We exemplify the problem of devising controllers for dynamic resource allocation in the Cloud, by reporting a brief experience in managing an elastic application based on the *Sun Grid Engine (SGE)* middleware. SGE follows a standard Grid computing architecture with a singleton master node and a set of executor nodes: The master receives jobs that are dispatched to the executors that run them. The middleware has been deployed in a Cloud infrastructure, where virtual execution nodes can be allocated dynamically.

In Figure 1, we report the results of different runs of the system subject to an identical workload but different controllers. The workload lasts thirty minutes and fluctuates in time. It consists of video conversion jobs that execute in six seconds on average. The SLA of the application consists of a single SLO over the response time, and specifies that the system must complete the jobs in less than two minutes in average. The goal of the controller is to dynamically adjust the number of executor VMs to respect the SLO while minimizing costs that depend on the number of used VMs.

The plot depicts the workload in terms of requests per period, number of active executors, number of jobs in the system, and measured response time for two different runs. The continuous red line represents the system when controlled by a self-adaptive Kriging-based controller as described in [27]. We can see that the number of executors changes over time and the response time is always below the 2-minute threshold, marked with the dotted blue line in the figure.

The dashed green line represents the behaviour of a state-of-the-art static rule-based controller as presented by Rodero-Merino et al. in [22]. In the experiment, we set the scaling up rule threshold for the queue length to 15 jobs per executor: each time the ratio between the number of jobs in the queue divided by the number of active executors exceeds this threshold, the system spawns a new executor VM instance. We set the scaling down threshold to 5 jobs per executor. We can see that SLA is violated in correspondence of the second peak in the

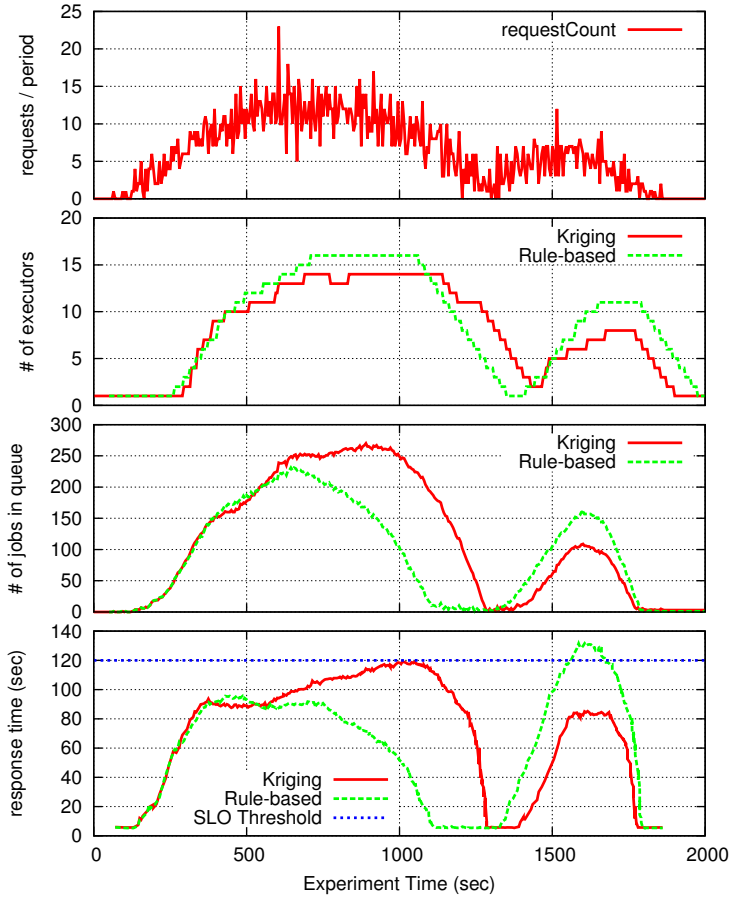


Fig. 1. Configuration, queue, and response time evolution for the SGE using Kriging-based and rule-based control

workload. This suggests that, albeit the rule-based system seems to scale up quickly enough in correspondence of the first workload peak, although using more resources in comparison to the Kriging-based controller, it does not cope well wit the second peak.

The example gives an intuitive idea of the type of actions and decisions that a cloud controller is required to take. The example is deliberately simple, since it has only one controlled variable, i.e., the number of executor nodes, and considers a single type of request. Typical Cloud-based applications are much more complex, since they combine different types of components and services, and hence have a large configuration space. Moreover, the range of served requests typically involves different sets of components causing possible software or hardware contentions that eventually impact on the performance of the applications. To this end, the application *workload mix*, that is the number and type of

incoming requests, typically has a considerable effect on the application response time.

The assurance of self-adaptive controllers requires examining a potentially unlimited set of unpredictable configurations that arise when the system adapts to different execution conditions. While designing self-adaptive controllers is a hard and challenging job, the assurance of self-adaptive controllers is an even harder task, since assurance techniques must cope with infinitely many unpredictable configurations.

In this chapter, we identify two main dimensions of this problem, the target levels of assurance and adaptability, and we propose a classification of self adaptive controllers induced by these two dimensions. We argue that rules and models defined at design time privilege assurance over adaptability, being statically verifiable, but incapable of dealing with situations not foreseen at design time. On the other hand, rules and models that can change at runtime privilege adaptability over assurance, being able to deal with unpredictable situations, but verifiable only under certain conditions. We identify combinations of design and runtime elements that reach a good compromise between assurance and adaptability, and we distinguish some outliers that come from particular choices or uses.

This chapter is organized as follows. Section 2 discusses the many dimensions of the problem and introduces an assurance-base taxonomy for self-adaptive controllers. Section 3 overviews the main approaches based on rules or models defined at design time. Section 4 presents the main approaches where models and rules are adapted at runtime. Section 5 discusses combinations of different kinds of approaches. Section 6 indicates the main research directions in the field.

2 Assurance and Adaptability

When assigning resources to applications, the Cloud shall solve the dilemma of allocating as many resources as possible, to reduce the violations of the service level agreement, while allocating as few resources as possible, to increase the profit and optimize the resource usage. The variety of available resources with different characteristics and costs, the variability and unpredictability of workload conditions, and the different effects of various configurations of resource allocations make the problem extremely hard if not impossible to solve algorithmically at design time. Self-adaptive controllers aim to identify suitable allocations of resources at runtime, based on some knowledge encapsulated in the controllers in the form of rules or models.

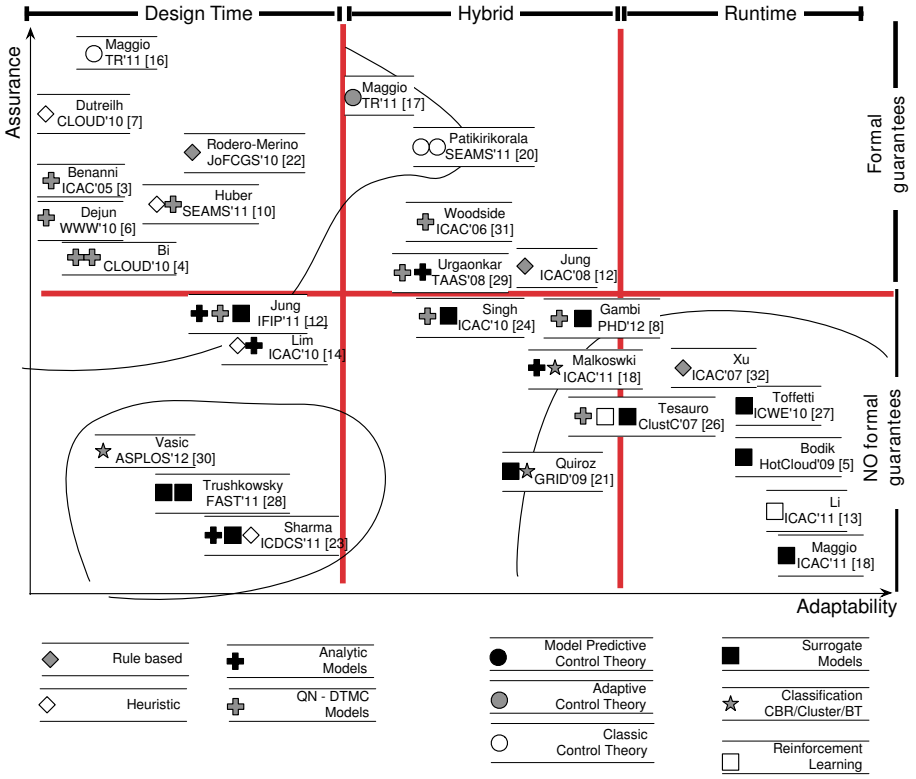
The problem of building efficient self-adaptive controllers has been the target of several research projects, which resulted in many approaches that differ for the models used to capture the knowledge of the system and for the strategies to identify a suitable configuration while reacting to changes of workload conditions. The models of system behavior that are used in self-adaptive controllers span from simple rules to complex analytic or surrogate models. Some approaches require models to be defined and tuned at design time, others define

and tune models at runtime, depending on the nature of the models, the complexity of the system and the reliability requirements of the controllers. Rules and analytic models, like queuing networks, must be defined and tuned at design time and hardly adapt to unpredictable configurations, while surrogate models, like Kriging models, may be defined and tuned at runtime and can adapt to emerging scenarios. Defining proper analytic models for complex systems may require a considerable effort, but their reliability can be assured analytically or experimentally before deployment. Surrogate models can be tuned dynamically according to the system behavior either at testing or runtime independently from the complexity of the modeled system, but their reliability can be assured only on the basis of some hypothesis on the behavior of the system that may not be statically verifiable.

In this paper we argue that there is no best model for self-adaptive controllers, rather different models can be used for different purposes and under different conditions. We followed a systematic process to identify a set of representative approaches in literature that we use here as a reference. The first step of the process consisted of surveying the papers published in the main journals (ACM and IEEE Transactions) and conferences (ICAC, ICSE, Cloud, HotCloud, ACDC, GRID, ICDCS, ASPLOS) relevant for Cloud-computing and distributed or adaptive systems, and identifying the main proposals to deal with scaling or resource-provisioning controllers. Among these approaches, we identified a set of common dimensions (for example, the type of models used, the control logic, the artificial intelligence technique) that we used for clustering the different work. For each cluster we identified a representative by choosing the approach that is more directly applicable to cloud-computing controllers and that is supported by experimental results, and we use the representative to indicate the set of proposals.

We propose a taxonomy of controllers built along two main axes: assurance and adaptability. *Assurance* refers to the possibility of measuring the predictability of the runtime behavior, and thus to the level of reliability of the system. *Adaptability* refers to the capability of the model to cope with changing and evolving configurations, and thus to the level of flexibility and learning capabilities of the system. We chose these two criteria since they provide a clear representation of the main trade-off one has to face when choosing a controller approach. On one hand it is desirable for a controller to be adaptable, to complement the information available at design time with monitoring data from the running system, to adapt to the newly gathered information, and to continuously learn the most appropriate control behavior. On the other hand, a continuously adapted control behavior might practically diverge from the control desired at design time violating some important assumptions or requirements and providing no formal guarantees on its properties.

Assurance. We estimated the *assurance* level of the different approaches by considering the formality of the control model, the hypothesis on the system behavior and the correspondence between the system and the model:



The surveyed approaches represented in the space defined by *adaptability* and *assurance* dimensions. Different symbols represent different categories of approaches.

Fig. 2. A taxonomy of approaches for self-adaptive controllers

Formality of the Control Model. This accounts for the amount of formal guarantees that the approach gives on the model or control. For instance, in traditional control theory, under proper assumptions, the controlled system can be guaranteed to converge to an equilibrium point within a given range of oscillations.

Hypothesis on the System Behavior. This accounts for the set of assumptions on the system behavior that need to be valid for the approach to correctly model the system. For instance, control theory assumes system linearity, while Kriging models assume a certain degree of smoothness of the modeled system.

Correspondence between the System and the Model. This accounts for the (timely) coherence between the current state of the system and its runtime models.

Adaptability. We estimated the *adaptability* of the different approaches by considering three main aspects:

Support for Adaptation. This accounts for the amount of details of the model that are fixed at *design-time*, the amount of details that can be partially adapted or extended by combining additional models (*hybrid*), and the amount of detailed that the controller can learn from system monitoring data collected at *runtime*.

Degree of Adaptation. This accounts for the number of parameters and the degrees of freedom that can be changed in a model, spanning from fixed models, to linear-, polynomial-, regression models all the way to non-parametric models (for example, splines).

Frequency of Adaptation. This accounts for the costs of retraining in terms of speed and frequency, and measures how well a controller can use new monitoring information.

Figure 2 illustrates the two-dimensional space induced by the metrics we chose for classification. The adaptability axis is partitioned in three sections that correspond to approaches that are implemented at design time and have no provision for being updated, hybrid approaches that typically combine several models with limited adaptation capability, and solutions supporting runtime update of the control logic / models, respectively. Among these partitions, controllers with higher degrees of adaptation, or higher frequency of adaptation are moved to the right. The assurance axis is partitioned in two sections corresponding to the approaches that provide formal guarantees on the controller behavior, and the one that do not come with formal proof mechanisms.

Figure 2 presents a dispersion graph of the approaches organized along the two dimensions. In the figure, the approaches are indicated by means of the first author of the paper that bet represents the approach, the publication venue, and the reference number. We specify the nature of the proposed approaches by means of symbols that are explained in the labels.

The regions roughly identified in the figure emphasize easily identifiable similarities in the considered approaches which can be coarsely classified as being either analytic, black box, rule based, or hybrid.

Analytic. Analytic approaches tend to privilege reliability over adaptability. They rely on models that can be statically analyzed to formally prove stability and other important properties of the controllers, and thus they usually provide a high assurance level. However, they must be defined at design time, typically come with high costs in terms of modeling and system knowledge, and are only applicable under strong assumptions (for example, at least 'local' system linearity in control theory), and thus come with reduced adaptability. These models tend to accumulate in the top left corner of the figure.

Black Box. Approaches that use black box models tend to privilege adaptability over reliability. They rely on surrogate models built from monitoring data that can be automatically updated at runtime, while the system is running. Thus, they can adapt to situations that arise only at runtime resulting in high adaptability. However, they provide little support for analysis and formal assurance, and rely mostly on assumptions about the system behavior that may not

be always easy to check statically and enforce dynamically. Thus they provide a lower assurance level than analytic approaches. In the figure these models occupy two distinct regions, the bottom right and the bottom left areas. The bottom right region corresponds to approaches based on black box models initialized and updated at runtime that can achieve high adaptability albeit at the price of a lesser assurance level. The bottom left region corresponds to approaches based on black box models that are used in a framework where they are initialized and updated at design time and not at runtime, thus giving up the adaptability that derive from the possibility of adapting to emerging scenarios at runtime, without gaining much in terms of assurance level.

Rule Based. Approaches based on rules appear in different flavors and span all over the taxonomy space depending on whether rules are fixed at design time and do not change, or new rules are discovered and possibly applied after approval, or the rule set is updated at runtime.

Hybrid. Approaches that combine design time and runtime techniques aim to conjugate the adaptability of models used at runtime with the assurance gained with models used at design time. In the figure, they occupy the middle region.

In the following sections, we illustrate the different classes of the approaches considering the adaptability axis. Section 3 discusses static (design-time) approaches, Section 4 presents approaches that support runtime updates, and Section 5 overviews hybrid approaches that combine both solutions. Each section is organized by clustering approaches that have some commonality (for example, in the type of control logic, in the nature of the models). For each cluster we outline a relevant set of sample approaches, and discuss in detail few representative ones, stressing the relations and trade-offs between assurance level and adaptability.

3 Static Approaches and Design Time Assurance

We use the term *static approaches* to indicate approaches based on models of the controlled system that are defined and verified at design time and not modified, updated, or tuned in production, after system deployment. We refer to these models as static models to stress the design nature of their construction and analysis. These models privilege design time assurance over runtime adaptability: The models of the system identified at design time are not changed during runtime activities to preserve the validity of the design time proofs used to identify system *viability zones* that are defined as states in which the system operation is not compromised [25], along with the main properties of the system behavior.

The most popular static approaches are based on either analytical models or rules and thresholds, while only some static approaches are based based on black-box and surrogate models defined and tuned before deployment. Analytical models can be divided in (1) *time invariant* models that describe the system steady state behavior, and (2) *time varying* models that describe the evolution and the transitory phases of the system behavior. Time invariant models are mathematical relations derived by analyzing queuing networks, Petri nets

and Markov chains, time varying models are typically dynamic state-equations commonly used in classic control theory.

Rules and threshold are constructs building upon logical expressions that are used to trigger control actions. Controllers implemented by means of rules and thresholds are amenable to being transformed into formal representations and proven to satisfy, at least to some extent, predictable behavioral properties. For example, one can prove the absence of conflicts and contradictions in sets of rules, and the termination of the rule triggering process.

Black-box and surrogate models are derived from empirical data obtained by executing the systems. They capture the relationships between input and output variables and are commonly used by controllers as oracles, that is, to predict possible system behaviors in terms of the same input/output variables. Although black-box and surrogate models are amenable to runtime adaptation, in some cases they are used as the core of static approaches. These approaches do not consider the additional adaptability that derive from these models, rather, they focus on the cost of deriving accurate analytical models that decreases dramatically when referring to black-box and analytic models.

Being defined at design time and not updated at runtime, static approaches rely heavily on the correctness and completeness of the knowledge encoded in their models. This implies an extensive degree of experience and understanding of the system behavior, and generally high costs in terms of model identification and/or synthesis. Moreover, formal approaches, in particular classic control theory, are generally applicable under strong assumptions in terms of linearity, monotonicity, and reliability of the controlled system, thus significantly reducing the system viability zones. Under these assumptions, feedback-loop approaches can be proven robust to a certain degree of error in the estimation of the model parameters, hence they can provide effective control actions (for example, maintaining a performance metric within a certain range) even without self-tuning at runtime. However, both errors in parameter estimation and wrong assumptions on system behavior properties can consistently reduce the efficiency of the control in terms of possible SLO violations or resource assignment.

In the rest of this section, we provide representative examples for each class of static approaches that are and can be used inside Cloud IaaS controllers.

3.1 Approaches Based on Control Theory

Cloud controllers based on control theory adapt classic control theory techniques in the context of computing systems aiming to design adaptive, robust, and stable systems [17]. Here we provide only basic information about classic control theory. Interested readers can refer to the classic book by Hellerstein et al. for additional details on control theory approaches to managing computing systems [9].

Some approaches adapt standard control techniques, such as proportional, integral and derivative techniques, to synthesize self-adaptive controllers for the Cloud. These controllers rely on simple models and provide formal guarantees on the behavior of the system, but rely on very strong assumptions that are not verified in highly varying environments. Other approaches try to extend the

scope of applicability of controllers based on classic control theory, by proposing complex parametric models where part of the parameters are unknown at design time. These approaches can still provide a limited set of formal guarantees depending on the assumption of proper on-line estimation methods.

An interesting example of approaches that apply classic control theory for designing Cloud controllers is the control theoretic solution that Maggio et al. propose to deal with self-optimization problems [16]. This solution is developed in the context of a common framework for monitoring system performances and adjust the allocation of resources to applications in order to guarantee a predefined service level.

Classic Control. Maggio et al. start with a simple (stateless and linear) model of the system that assumes a monotonic relation between allocated resources and application target performance. Then, they synthesize a Deadbeat controller, which is a common choice in the context of standard control theory, to track the target performance signal. They study the transient behavior of this controller by setting different parameters to estimate the approximation of the input signal, and to decide if the controller can effectively regulate the system despite its variations. When they cannot determine the suitability of the current model, they further refine the model by introducing more complex techniques and see if the new models are conclusive.

Advanced Control. Maggio et al. improve the basic deadbeat controller by adding an identification block that supports the online estimations of the unknown system parameters. They extend the controller by means of a Kalman filter and a Recursive Least Square algorithm. They observe that adopting more complex solutions, i.e., solutions that use more parameters to describe the relationship between the controlled and target variable, increases the difficulty to prove suitable control properties.

Multi Model Adaptive Control. A good example of multi model controllers is the approach of Patikirikorala et al. who use a particular instantiation of Multi-Model Switching and Tuning (MMST) adaptive control [20]. In a nutshell, they define several (fixed) linear models that describe the system behavior in different working conditions, and synthesize different single-model controllers following the classic control theory. The overall controller monitors the system variables, computes an error metric for each of the models, and enable only the single-model controller that correspond to the minimum predicted error.

Approaches based on control theory provide high guarantees in terms of assurance: stability and dynamic properties of the controller and controlled system can be proved in rigorous mathematical terms. Accurate system identification further increases the control reliability.

Classic single-model approaches require simpler proofs and provide a clear separation between system regions where the control offers formal guarantees (viability regions) and where not. Multi-model approaches are more flexible and are meant to cover wider viability regions. In these approaches, each controller

behaves as in single-model approaches, but no formal guarantees are offered when the overall control logic switches them on and off.

The high assurance level of classic controllers is based on strict assumptions of the behavior of the controlled system that may not be always demonstrated in practice.

3.2 Threshold and Rule Based Approaches

Threshold Based. Threshold-based policies are popular in current industrial applications, as they are simple and intuitive to understand. They are applied by defining lower and upper limits on target metrics that, when crossed, trigger reconfiguration actions. Threshold-based controllers are used in many companies such as Amazon[1], RightScale[2] and Scalr[3]. In many controllers currently used in industrial applications, upper and lower bounds are defined by the customers referring to low level metrics, such as CPU usage. Customers define also the corresponding reconfiguration actions. Dutreilh et al. [7] experiment with static threshold-based policies that rely on high level *SLA* metrics, such as response time. They define control policies based on upper and lower thresholds, fixed amounts of virtual machines to be either allocated or deallocated, and pairs of "inertia" durations that support scaling up and scaling down periods. For instance, if the response time exceeds the upper bound, the controller allocates virtual machines, and inhibits itself for the corresponding inertia interval. If the response time drops below the lower bound, the controller deallocates virtual machines, and again inhibits itself for an inertia interval.

Rule Based. Rule based approaches extend threshold solutions by considering different types of events, and allowing rules to trigger other rules following the common ECA (event, condition, action) paradigm. An interesting rule based approach is *Claudia*, a rule-based controller for virtualized services proposed by Rodero-Merino et al. [22]. Claudia is more general than most rule based approaches, since it considers service life-cycle events and user defined variables, called Key performance indicators (KPIs), as well as business level metrics that holistically describe the status of a complete virtualized service and eventually triggers rules that reconfigure the system. Claudia combines three different types of rules, all defined by Cloud users: *scaling rules* that resemble traditional threshold based approaches and change the number of allocated virtual machines, *reconfiguration rules* that act at deployment time and choose the size and type of virtual machine to be deployed, and *business rules* that constrain the automatic scaling behavior with respect to running costs by limiting for instance the total number of running virtual machines. Business rules consider also Cloud federation concerns, for instance by *migrating* virtual machines from one Cloud provider to another. Claudia monitors the virtual service execution

[1] `http://aws.amazon.com/autoscaling/`
[2] `http://www.rightscale.com/`
[3] `http://scalr.net/`

and periodically tries to fire user defined rules that eventually trigger suitable control actions.

Rule and threshold based approaches can be verified, at least to some extent, formally. Verifying threshold based controllers is generally simpler than verifying rule based controllers, because threshold based controllers presents a less complex set of constraints, while full-fledged rule-based systems may have several conflicting and interdependent rules. Rules are commonly set manually under the assumption that they are able to capture main phenomena and characteristics of the system. They remain stable during the runtime and rely on the assumption that the system evolves only as predicted. Their event and condition clauses clearly identify the system viability zones that, however, remain fixed and may not are able to capture unplanned systems evolution.

3.3 Approaches Based on Analytic Models

Controllers based on analytic models use utility functions that combine system performance metrics and business considerations, like revenue and resource usage cost, to find desirable system configurations. They compute system performance metrics (typically the response time) through either different queuing models or analytical representation of queuing models. Analytical approaches differ each other mainly in terms of the chosen queuing formalism, the complexity and accuracy of the models, and the policy adopted to solve the optimization problem.

Single QN. Benanni and Menascé [3] combine queueing models and combinatorial search to dynamically allocate resources to application environments. They associate a local controller to each application environment and use queueing network models (open models for transactional systems, closed model for batch processing system) to predict the performance metrics of the application environment. Each local controller computes a utility measure by combining performance metrics, service level agreements and penalty functions, and sends the computed utility to a global controller that uses a combinatorial search algorithm to identify the final resources allocation of the entire data center, i.e., of all the application environments. While exploring the configuration space, the global controller interacts with the local controllers by suggesting potential new resource allocations, and the local controllers respond with updated values of local utilities. The queueing networks and their parameters are defined at design time.

Architecture Level Performance Model. Huber et al. [10] introduce an architecture level descriptive model, from which they derive performance models that are used by self-adaptive controllers to predict the system behavior. When the input workload changes, the control algorithm adds resources (virtual machines) to the system in order to eliminate all actual and predicted service level agreement violations, then the controller removes the resources that are underutilized. Huber et al. assume the availability of the architecture level models and estimate the parameters of the model at design time, for example, resource usages, routing/calling probabilities, service times, and possible usage scenarios

and user classes. Once the model is in place, an automatic procedure transforms it into a predictive performance model, i.e., a queuing network, that is used on-demand by the control algorithm.

Mixed Queueing Networks. Bi et al. [4] use an mixed queueing model to simulate multi-tier applications and define a non-linear optimization problem over it to dynamically decide on the system at per-tier. The mixed model combines an M/M/c queuing model, for the *front-end*, with several M/M/1 queueing models for the *per-layer* virtual machines. The optimizer uses the model to calculate the number of resources to provision at the each tier according to the target end-to-end response time for the tier that is assumed to be agreed with customers.

Multiple QNs. Dejun et al. [6] address Web applications modeled as acyclic compositions of services using a what-if-analysis and negotiation among the composed services. Each service estimates its own performance variation in the case of changes of the allocated resource or workload, and pass the estimate up through the invocation tree to produce local decisions that are incrementally aggregated all the way up to a root controller. Dejun et al. model the performance of each single service as a M/M/n/PS queue, and consider performance variations for configuration changes of a single VM (+1 or -1 machine per service). The controller adjust the predictions by estimating the service time for the queuing networks at runtime using a feedback control loop: A threshold on the prediction error of the system performance triggers a new estimation of the service time by using the latest measured response time. Dejun et al. show the effectiveness of the proposed controller for different types of compositions, and claim that a service level agreement for the front-end service allows for finer control of the composed services than service level agreement thresholds on each component.

The approaches based on different queuing networks that we review above are based on the assumption that the chosen queuing model provides a sufficiently accurate representation of the considered system (plus its workload, processors, architecture, and bottlenecks) and that the system is inherently *stable*, that is, it does not present emerging or unpredicted behaviors. They also make several assumptions on the system behavior, its relevant components, and the statistical properties of the workload. Finally, all queuing models require a set of parameters that are estimated at design time with no provision for adjustment at runtime.

Simple models require to estimate few parameters, hence they are easier to configure, but may not be very accurate. Complex models can be more accurate with respect to the system structure, but require to estimate more parameters, and thus demand extra time, and effort. More complex models do not always result in higher accuracy, because of the simplifying assumptions that enable analytically solutions. Richer models (for instance, layered versus plain queuing networks, or mix-aware versus mixed oblivious solutions) typically offer more adaptability, that is more possibilities of closer adaptation to the system, albeit at the cost of higher complexity. In principle, they should be able to provide

better assurance (at least in terms of a closer resembling representation of the modeled system); in practice the high number of parameters that need to be correctly estimated might attenuate the expected improvement.

3.4 Approaches Based on Black Box and Surrogate Models

The use of analytic modes is limited by the need of accurately defining a model structure and computing many parameters. Black box and surrogate models address this problem by generating the models from data gathered during system executions, and thus obtaining models that correspond to the system by construction. We will see in the next section that black box and surrogate models can be derived and adjusted at runtime, thus increasing the flexibility and adaptability of the controllers. Here we survey the main models that are proposed to be tuned before the systems deployment, typically at testing time and during model training.

Case Based and Clustering. Vasic et al. [30] use a workload signature based on Hardware Performance Counters (HPC) to cluster workloads, and associate the identified clusters to previously measured appropriate resource allocations. A proxy collects and computes workload signatures both in the training and control phase, by replicating a fraction of the entire application workload that is directed to a profiler for sampling and measurement. At runtime, a profiler computes the incoming workload signature, a classifier associates it to a class, and the controller applies the recorded resource allocation in a single control action. The controller uses a metric of the *certainty* of the association of the workload to a cluster as an indication of the need to trigger a new training of the classifier. The approach also measures the *interference* of other applications running in the same infrastructure, where interference is defined as the ratio of the performance achieved in production w.r.t. the performance for the same workload signature and resource assignment measured at training time. Substantially, the system works as a cache for previously observed combinations of workload signatures and resource allocations. In case of a cache *miss*, the default policy is to bring the controlled system to its maximum resource allocation to prevent service level objective breaches. Threats to the validity and assurance of the control come mainly from the choice and appropriateness of the metrics used to compute the signature and cluster the workloads. In their experiments, the authors report that the clustering typically results in only few workload classes while applications can normally have very large workload and configuration spaces.

Multiple Surrogates. Trushkowsky at al. [28] address the on-line reconfiguration of storage systems in response to workload changes under stringent performance requirements. The controller manages SCADS [2], a key-value store that offers eventual consistency and an easy partitioning of the key-value stores, hence natively supports replication and elasticity. Two specific issues make the problem hard: 1) to scale a data-intensive system, data items must be moved or copied across instances, impacting negatively on service performance, and 2) high percentiles of response time have much higher variance (and therefore are

much harder to control) than the center of the distribution (average response time). The latter issue can cause oscillations in classical closed-loop control. Trushkowsky at al. tackle these issues by introducing a model predictive control: The controller refers to a model of the system and its current state to compute the optimal sequence of control actions, executes the first action of the sequence, and then recomputes the optimal sequence of control actions to chose the next actions. The system performance model coupled with workload statistics can predict whether a server is likely to meet its service level objectives. In the case of predicted violations, the controller either spawns new server instances or activates "hot" standby ones, and copies or migrates data bins using a copy-duration model for planning. Trushkowsky at al. builds the server performance models using statistical machine learning (SML) on data obtained through *off-line* controlled experimental runs with steady state workloads. They claim that simple changes in workload will not affect the accuracy of these models, that however can degrade over time if the application or the underlying data change consistently, for instance when an individual request returns more data than during training. In this case, the off-line models would have to be rebuilt in production. The approach uses a linear classification model with logistic regression to predict whether a given workload mix (get and put operations) and intensity are likely to cause service level objective violations. The copy-duration model is obtained off-line through the linear regression of samples gathered by running copy operations on servers under different workload rates.

Multiple Surrogate Models. Sharma et al. [23] present Kingfisher, a cost-aware system to scale elastic applications. Kingfisher relies on several models that capture capacity, costs and reaction time concerns, and a linear optimization to solve the cost- transition-time- aware control problems. Pricing models, elasticity mechanism models (migration, replication, etc. on the different platforms) and server capacity (for different resource allocations) are all obtained empirically at design time. The controller uses the models at runtime to solve the integer linear program that accounts for both infrastructure and transition costs and derives appropriate control actions. Kingfisher assumes that applications have a multi-tier software architecture where each tier has its own quality of service requirements that must be met by provisioning sufficient capacity. Moreover, Kingfisher assumes the availability of a perfect forecaster that is defined as forecaster that knows the workload in advance, as well as the perfect estimation of the per-tier peak-workloads. To solve the integer linear program in reasonable time and for not trivial applications, Kingfisher employs a greedy heuristic with a bounded worst case.

Black-box surrogate models provide an assurance level lower than analytic models, because the quality of these approaches depends on properties that are difficult to formalize and measure at design time. For example, the quality of such models depends largely on the amount and quality of the data collected from the system runs, their distribution in the input/output features space, the training/fitting procedures adopted. Being dependent only on data collected at design time, these models may not reflect accurately the real production environment.

Designers must assume that the drift between testing and staging environment is negligible. Moreover, they assume that the system behavior remains stable at runtime. In a sense, these approaches leverages black-box surrogate model because of they capability to learn relations between system variables that are unknown or too complex to estimate precisely; however, they do not leverage this inner capability outside the design time activity, thus limiting the adaptability of the controllers.

3.5 Approaches Based on Heterogenous Models

Some approaches address the limitations of the different techniques by suitably combining heterogeneous static techniques.

Regression. Lim et al. [15] combine a *cloud controller* that manages the compute infrastructure with an *application controller* that manages the resource assignments for each application to satisfy some a given performance goal. For the application controller, they propose to use a classical integral control to add and remove virtual machines based on average virtual machine CPU utilization. Integral control can be proved stable for a continuous control signal. Unfortunately the interface between the cloud and application controller consists of a coarse grained actuator, since one cannot add a fraction of a virtual machine, thus causing possible oscillations of the controlled system. To mitigate this effect, Lim et al. use a proportional thresholding technique, where higher and lower thresholds become smaller as system size increases, rather than a fixed target value for CPU utilization. They also use linear regression to model the relationship between the CPU utilization and the cluster size.

Surrogate Model Analytic and Heuristics. Jung et al. [12] focus on controllers that take into account the costs of system adaptation actions considering both the applications (for example the horizontal scaling) and the infrastructure (for example the live migration of virtual machines and virtual machine CPU allocation) concerns. Thus, they differ from most cloud providers that maintain a separation of concerns, hiding infrastructural control decisions from cloud clients.

The controller relies on an *estimator* that uses 1) automatic off-line experimentation to build a *cost table* quantifying performance degradation for each type of control action and workload, 2) layered queue networks (LQN) to predict the performance of each system configuration given a workload (LQN parameters are estimated offline), and 3) an ARMA filter to estimate the duration for which the current workload will remain stable (i.e., within a band B). The estimate of the duration of the stability of the workload is used to decide whether expensive (long term) control actions are worth or not. The controller searches through the graph of all possible control actions to find the sequence of control actions that maximizes a utility function that takes into account benefits and penalties expressed in terms of the application service level objectives, as well as the relative impact of all the control actions.

Approaches that combine heterogeneous models increase the adaptability with respect to the single static approaches, at the price of reducing the provided assurance level.

4 Dynamic Approaches and Runtime Assurance

We use the term dynamic approaches to indicate approaches based on models of the controlled system that are tuned at runtime, after system deployment. We refer to these models as dynamic models to stress the runtime nature of their construction. These approaches privilege runtime adaptability over assurance. They produce an accurate representation of the modeled system by characterizing the system behavior through metrics collected from the actual system execution. Typically, they build an initial version of the model at staging-time from a set of training samples, and then update the model continuously at runtime while the system is running. Thus, they can adapt to unforeseen changes and behaviors.

Different classes of dynamic approaches are characterized by the type of surrogate model they adopt, ranging from several forms of regression (for example, linear, quadratic, LOESS and Kriging) to machine learning techniques (for example, neural networks and reinforcement learning). Different model types imply different query and update strategies, to account for instance for the possibility of incrementally updating a model or the time and computational complexity of a complete re-training.

Dynamic approaches typically model either a single or a combination of system performance metrics as a function of some endogenous system properties that represent the system *configuration* in terms of both system characteristics, for example, number of allocated VM instances, CPU cores or threads, and exogenous factors, such as for instance the workload applied to the system or the influence of other services co-located in the same infrastructure. The approaches surveyed in this section often differ in terms of the considered metrics and the characterization of the workload features (for instance, workload intensity or service mix), but share some important features. They are highly flexible, and adapt easily to the measured system behavior, because of the ability to learn from and adapt to emerging behaviors, and this is extremely important for instance when the configuration or workload space is too vast for extensive exploration at staging time. They impose few requirements on the experience and knowledge of the modeled system functioning and behavior, because the samples required for training the models come from externally measurable system features.

The adaptive nature of dynamic approaches is both their main feature and their Achille's heal: The ability of constantly change and adapt makes them particularly well suited to highly dynamic systems and, at the same time, makes it hard to assure the quality of the resulting system, and to estimate the correct partition of model training effort between staging time (*bootstraping*) and on-line learning.

In the following, we survey the main approaches of this category distinguishing between approaches based on surrogate models and approaches based on machine learning.

4.1 Approaches Based on Surrogate Models

Surrogate models are build from sample executions of the modeled system, and describe the behavior of a system in terms of relations between input and output features that can be continuously updated with monitoring data. They are commonly used to describe either the steady-state behavior or the mid-to-long time horizon behavior projections. Self-adaptive controllers use surrogate models to predict the close future, given both the current and the estimated values of the input features. Some controllers use surrogate models also to support optimization procedures that explore the system configuration space to find the most suitable system configurations. Less commonly, surrogate models are used to provide model predictive control, where models are used to simulate and track the evolution of the system state under possible control actions, in order to plan for the most suitable ones. Such control strategy aims to maximize the control utility over a *receding* time horizon.

Splines and LOESS Regression. Bodik et al. [5] use statistical machine learning, and in particular smoothing splines and local regression (LOESS), to build performance models of the controlled system. Bodik et al.'s models represent response time as a function of workload intensity and system configuration. The controllers increase the robustness and the adaptivity to changes, by means of *model management* techniques (i.e., online training and change point detection) that update the model, track its quality and eventually rebuild it from scratch. The models are trained with data obtained online from the production environment. The controllers are conservative: They start from the maximum allowed allocation of resources, and decrease the allocation, while incrementally learning optimal configurations, to minimize service disruptions in exploration mode. Building models entirely from online samples simplifies the training phase, but may result in slow convergence of the models to new and possibly temporary system behaviors.

Kriging Models. In a recent work, we proposed an autonomic controller for horizontal scalability based on performance models of the controlled system. For each service level objective, the controller builds a different Kriging model that represents the objective metric (for instance the response time or the throughput) as a function of the number and types of virtual machine instances (system configuration) and a representation of the workload intensity and mix. Kriging models, also called Gaussian Process Regressions (GPRs), approximate target functions by means of a *spatial* correlation of samples. They extend traditional linear regression with a statistical framework that allows them to predict the value of the target function in un-sampled locations together with a confidence measure. In the Kriging based approach, training samples are collected by measuring the system behavior first at staging time to build an initial version of the models, and then in production to continuously updated them. As more samples are used, the accuracy of the model improves, while the uncertainty decreases, and the time to build the model increases. To avoid the collection of unmanageable sets of samples, many of which do not provide additional information

to the model, the controller filters out old samples belonging to the same configuration. Kriging based approaches pair predictions with confidence measures that support the implementation of robust control policies (by tuning the desired risk of violations, impacting on assurance), and drive the exploration and exploitation decisions when learning the system behavior, thus improving over other regression mechanisms.

The assurance of surrogate-based self-adaptive controllers relates to (1) the ability of the models to accurately represent system behavior also in the presence of noisy and missing data, that is, when the quality of data interpolation and regression decreases, (2) the speed of convergence of the learning process, and (3) the accuracy of the quantification of the uncertainty of the model predictions. For example, models that are updated online and frequently, that do not need a large training set, and that can provide an accurate measure of confidence for their predictions, provide higher assurance than models that are updated infrequently and cannot provide any confidence interval for their predictions. Controllers that continuously monitor the quality (accuracy, prediction error, etc.) of the models, and account for completely rebuilding of the models whenever necessary can adapt faster to emerging system behaviors than controllers that merely update the models with new data.

4.2 Approaches Based on Machine Learning

Machine learning techniques are commonly divided in model-*based* and model-*free* techniques, depending on the use of models. Both classes of techniques are exploited to build self adaptive controllers. The most popular control solutions that refer to model based techniques use artificial neural networks (ANN), while popular model-free techniques use reinforcement learning (RL) and clustering applied to the discovery of control rules. In model based solutions, the accuracy of the results depend on crucial choices such as the model structure and the training data, thus no a priori guarantees can be enforced. In model-free solutions, the level of assurance depends on the learning rate, the instability/evolution pace of the controlled system and the size of the action-configuration space, which is proportional to the amount of possible control actions.

Artificial Neural Networks. Artificial neural networks use training samples to build a model of system dynamics that can predict the system reaction to different inputs. The structure of the network and the quality of the training data are critical to the performance [17] and must decided by the designers off-line. After the initial supervised training that sets all the internal parameters, artificial neural networks can be updated on-line by a back-propagation procedure based on punishment-reward concepts. Maggio et al. [16] implement a neural network to control the amount of resources allocated to process at the OS level that guarantees a given service level. Although the context is different from Cloud computing (where artificial neural network based controllers have not been used yet) the basic concepts of modeling and dynamically allocate resources are similar.

Reinforcement Learning. Controllers based on reinforcement learning learn directly optimal control policies, that is, the best set of actions to apply whenever the system enter a state, and thus do not require a model of the system. At runtime, reinforcement learning-based controllers adopt a trial-and-error learning strategy and apply (at least at the beginning) random actions. Effects on the system, and controllers utility function, resort either in action reward (if the actions increased the utility) or punishment (if the actions damages the system, thus lowering the utility). Reinforcement learning solutions suffer from poor scalability in the action-state space and from long convergence rates. To alleviate these limitations, Li and Venugopal [13] propose a distributed implementation of reinforcement learning based self-adaptive controllers. In this solution, each processing node, i.e, a virtual machine, is an independent entity and incorporates a local controller that runs a Q-Learning algorithm. Local updates to the model are pushed to the other controllers via a distributed hash table, so that collaborating controllers can learn the state-action-reward model quickly. At run time, each controller takes scaling decisions based on the shared model and aims to maximize the reward function while model updates are continuously published.

Clustering for Fuzzy Rules. Xu et al. [32] propose a dynamic approach based on fuzzy rules. The controller applies fuzzy modeling to learn the relationship between workload and resource demand, and uses clustering to update the rules at runtime. The controller is organized in two levels: at the lower level, local controllers are associated to physical resources and decide about the resource needs of the virtual applications deployed on the node; at the higher level, a global controller receives all the resource needs from the local controllers, and solves the resulting global optimization problem to decide the final resource allocation. Local controllers estimate the needs of resources at regular intervals for each virtualized application by means of fuzzy inferences: Each controller receives workload data, *fuzzifies* them, triggers the fuzzy rules, and finally produces the output *crispy*. At the same time, controllers analyze the monitoring data to derive new fuzzy rules, adapt existing ones, and remove not optimal rules from the knowledge base. Fuzzy rules are obtained by filtering and clustering raw monitoring data as they reach the controller: Data are filtered if they refer to mappings that lead to service level agreement violations, and then are clustered based on the density of surrounding data points. Eventually, a single rule is associated with each cluster. The controller defines an initial set of clusters, thus a set of fuzzy rules, offline using data from staging experiments, and updates the fuzzy rules at runtime, by adapting the size and number of clusters.

Control solutions based on machine learning techniques are in principle the most flexible solutions, since they can learn any kind of relations either directly modeling the system or capturing the optimal control policy. However, this extreme adaptability come at the cost of the impossibility of proving stability, convergence or any other properties important for control purposes. In particular, artificial neural network and reinforcement learning solutions do not provide any automatic means to evaluate the goodness of their fit nor their predictions, and designers have to either believe in them or not, and employ some external

mechanism to track their error and retrain (whenever it is possible) the model. Machine learning solutions can be useful where the complexity of the controlled system makes it infeasible the construction of any satisfactory analytical solution or white box model, or when designers have no a-priori information about the behavior of the controlled system.

5 Hybrid Approaches and Combined Design and Runtime Assurance

We use the term hybrid approaches to indicate approaches that combine static and dynamic techniques to benefit from high adaptability while guaranteeing high assurance level. Static and dynamic approaches can be combined in many different ways, resulting in different blends of assurance and adaptability.

We distinguish two classes of hybrid approaches depending on the adopted merging strategy: (1) approaches that augment static solutions with learning and (self-)adapting capabilities, and (2) approaches that complement dynamic solutions with static models to modulate the effects of learning on the control behavior. Approaches that augment static technique with learning capabilities aim to increase the level of adaptability while maintaining a high assurance level. Approaches that combine dynamic solution with static model aim to improve assurance while limiting the loss along the adaptability dimension.

5.1 Static Approaches Augmented with Learning Capabilities

Static approaches augmented with learning and self-adapting capabilities increase the adaptability of the underlying static technique while maintaining high assurance level. They try to augment the size of viability zones by allowing the control solutions to deal with unseen and emerging behaviors that may differ from the design time assumptions. Augmented static solutions either relying on static models with online parameters tuning, or on static models that are rebuilt while the system is running.

Static Model with Online Parameter Tuning Approaches that augment static models with online parameter tuning are based on a core static model whose parameters are updated at runtime using monitoring data. The update and learning processes proceed in parallel with the controller activities.

LQN and Kalman Filter. Woodside, Zheng and Litoiu [31] propose a model-based feed-forward solution centered around a layered queueing network model of the system whose parameters are recomputed online. The controller uses the model to predict the system performance depending on the monitored workload, and to optimize the system configuration, in terms of server configurations and resources allocation. The controller adapts the parameters at runtime by means of an Extended Kalman Filter: The filter keeps updating the parameters until the residual error is below a threshold, and in this way the controller relies always on an accurate model of the system. The tracking filter greatly improves

the robustness of the control algorithm in the presence of parameter drift. Moreover, extended Kalman filters have proven optimal properties when the relations between variables are linear, and are expected to have near-optimal properties if non-linearities are involved depending on the local properties around the operating point.

QN and Regression. Urgaonkar et al. [29] propose a dynamic capacity provisioning model for multi-tier Internet applications that uses queuing networks to determine the provisioning of resources for each tier of the application. Differently from the previous work, Urgaonkar et al. use online monitoring data to estimate the session arrival rate, the session duration and other parameters that are fed to the queuing network for the predictions. The proposed controller combines predictive and reactive methods to determine when to provision resources, to cater for respectively long-term / cyclic and short-term / unpredicted variations in the application workload. The controller computes long-term provisioning with the queuing network model where each tier is modeled as G/G/1 queue, and tiers are linked with replication factors to describe how the workload demand is distributed inside the controlled system. The controller deals with short-term variations of the load by means of a *sentry* component that implements admission control policies.

QN and Clustering. Singh et al. [24] use a queueing network to model the system as well, but they rely on mix-aware provisioning techniques to handle non-stationarity in workload mix and volumes. The controller employes k-means clustering to automatically classify the workload mix and uses the queuing model to predict the server capacity and support configuration optimization. The "workload class" is the parameter estimated online by the controller that keeps the model up to date. The initial clustering is computed off-line following an iterative and empirical process. On-line clusters are adjusted (split/merge) whenever the error of the estimated cluster predicted mean service time is greater than a given threshold with respect to the mean service time monitored by the mix-determiner. Maximum number and size of clusters are specified beforehand. Similar to the other approaches in this group, Singh et al. assume that the system can be modeled as a pipeline of independent tiers, for which per-tier service level agreement can be defined, and per-tier demands can be derived from the incoming one. The clustering allows to determine precisely the different types of requests in the workload improving the accuracy of results form the queuing model.

These approaches share the adoption of an analytic system representation (either queuing networks or layered queuing networks) whose structure remains unchanged. They achieve a limited degree of adaptability by estimating one or more parameters online. This caters for situations in which there is a need for minor adjustments with respect to design time expectations on system behavior, but the limits imposed by the initial queuing network model prevent the control from adapting to any possible emergent behavior related for instance to non-modeled system bottlenecks.

Model Update. Approaches that rebuild the model at runtime use a control model that is modified or completely rebuilt at system runtime, either periodically at a fixed time interval or whenever some indicative metric (for instance, a prediction error) crosses an acceptable threshold value.

Rule Base with Bayesian Classification. Jung et al. [11] propose a rule based approach, where rules are automatically generated by means of a machine learning process. The controller uses the rules as in static approaches: It monitors system variables, for instance, the workload, and triggers the control rules when necessary, e.g. when there is a match with the workload intensity, to change the system configuration. The controlled systems are modeled by means of layered queuing network models whose parameters are estimated offline, at design time. The controller then uses the layered queuing network models in a two step discovery process that is carried out at runtime in parallel with the control loop: (i) The controller randomly chooses a set of input workloads, searches for the corresponding optimal system configurations, and encodes the results as rules; (ii) It interpolates all the data via a Bayesian classification algorithm that derives a decision tree saved as policy. The decision tree covers the whole configuration space – thus not only the rules obtained in step (i) – and the learning algorithm can be configured to prune or merge, similar subtrees and configurations to simplify the rule set. The process produces a new decision tree with a finite number of leaves, i.e., system configurations, with a high degree of predictability and verifiability because the new tree encodes all the possible system configurations a priori enabling further decision to be taken at *business* level. The optimization procedure is based on a heuristic gradient search and considers both the system configuration, i.e. the number of replicas for each virtual machine, and their placement on a set of physical nodes. It assumes that the system utility monotonically decreases as resources are deallocated. The quality of the control, measured in terms of utility, depends on three critical factors: the model accuracy, the number of workloads considered during the optimization, and the "compactness" of the decision-tree, as more compact trees are less accurate.

QRS Model and Clustering. Quiroz et al. [21] propose a decentralized online clustering approach to detect patterns and trends in resource demands for jobs in Grid systems, and use this information to optimize the provisioning of virtual resources. The control is fully decentralized: In each control window, the clustering algorithm analyzes the incoming jobs and produces a number of target virtual machine classes. The number of virtual machines that must be provisioned is proportional to the volume of the incoming job for each class. Jobs are either assigned to the available virtual machines as they arrive or wait for the right virtual machine to start. Eventually, each local controller triggers the creation of new virtual machines to process the waiting jobs. Controllers rely on a model to estimate application service time based on Quadratic Response Surface Model (QRSM). The QRSM is fitted using long-term application performance monitoring data, and it is used at runtime to provide feedback about the quality of the clustering, i.e., the appropriateness of requested resources for the

incoming jobs and their ability to meet QoS constraints. Given the actual resources, the QRS model is used to predict the response time for the estimated workload. This prediction is compared against the quality of service requirement and the controller uses the model to adjust the class attributes computed by the clustering. The controller then uses the quality measures obtained from the QRSM as an oracle to re-trigger the evaluation of the clusters with the decentralized online clustering algorithm.

Approaches that rebuild the model at runtime combine some static model, either a linear queuing network or a response surface model, with a learning/discovery mechanism that changes the control logic, rule- or clustering-based, respectively. The static model is derived from design time knowledge or long-term application historical data, and provides the fixed reference for the derivation/optimization and evaluation of the runtime-generated control logic. In a sense, the static model defines fixed boundaries for the controller behavior that make it predictable (i.e., within the boundaries), while the runtime adaptation aims at optimizing the control with respect to the measured system behavior.

5.2 Dynamic Approaches with Static Fall-Back

The lack of assurance of purely dynamic models derives from the dynamic nature of the data used to build and tune the models, that cannot be statically verified by definition. It is difficult to provide assurance proofs of controller behavior for approaches based mainly on runtime measurements, that heavily depend on the availability and quality of the data collected at runtime.

Dynamic approaches with static fall-back aim to improve the accuracy of these models by complementing the dynamic model with analytic approaches that might provide a reasonable alternative in the presence of low quality of the prediction based on the runtime model.

Case Based and Analytic. Malkowski et al. [18] propose a *multi-model* controller that combines the horizontal scale controller originally developed by Lim et al. [14] as the static approach with an *empirical model* obtained from runtime monitoring data. The empirical model uses a throughput vector space (i.e., a list of throughput values, one for each application interaction-type) to represent the combination of configuration, workload, and performances achieved by the system in a 30-seconds time frame. The empirical model is used to find the smallest (cheapest) system configuration that satisfies the service level objectives, and is located within a threshold value in terms of Euclidean distance in the throughput vector space with respect to the predicted workload. In other terms, it selects the smallest configuration among the set of *visited* configurations that were able to withstand a predicted workload intensity (or a comparably "close" intensity) without violating service level objectives. The controller switches to the static approach when the empirical model cannot find a visited configuration within the distance threshold.

Kriging and Analytic. In our recent work, we proposed a similar approach where the controller uses an analytical formulation, derived from a queue network

model, as the static approach, and a Kriging model that interpolates monitoring data in [8]. Differently from the previous approach, we use an interpolation of data, i.e., the Kriging model, instead of raw monitoring data, to drive the control decisions, and use the same model to switch to the queuing network. In fact, the controller leverages the unique ability of Kriging models to provide a confidence measure along with performance prediction: If the confidence of the prediction is too low then the value is discarded and the controller resorts to the queuing network.

Reinforcement Learning with ANN and Queuing Networks. Tesauro et al. [26] employ both static and dynamic techniques in an approach that combines the strengths of reinforcement learning, artificial neural networks and queuing networks. A static technique is used when the reinforcement learning is in learning mode; in this period, the controller resorts to the queuing networks to control the system and collect training data at runtime. To speed up the learning, the data are used to train a non-linear function approximation, in this case an artificial neural network, of the Q-function that is used to obtain the learning rewards. Tesauro and co-authors represent the Q-function using neural networks instead of traditional look up tables to encode the state-action rewards. Artificial neural networks interpolate the collected samples and reduce the need of large training set, improving the controller scalability. By combining reinforcement learning and queuing networks, controllers can avoid poor performance during the training activities, because all the data are collected using the queuing network policy that provides an acceptable quality level of the control with respect to the random control actions of the reinforcement learning exploration. At the same time, controllers can improve their accuracy because steady-state queuing models are unable to take dynamical effects into account while the reinforcement learning can take into account dynamic effects such as transients and switching delays. Once the reinforcement learning is ready, the controller releases the queuing network and adopts it. In this period, parameters of the queue network are continuously updated based on measurements of system behavior, and when necessary, the controller can switch back to it and start the learning process again.

Hybrid approaches achieve high adaptability by using empirically obtained data to model emerging (and possibly unexpected) behavior such as for instance I/O bottlenecks (typically not modeled in analytic approaches / layered queuing networks for cloud controllers) or cross-layer interdependencies. High adaptability aims at a more precise representation of the system behavior geared towards the realization of *more efficient* (i.e., in terms of resource usage, service level objective violations) controllers. To compensate for the dynamic nature of black box models, and their dependency on possibly high varying runtime monitoring data, analytic approaches (typically queuing networks) are used as a safety net to constrain the controller actions. In a sense, analytic models provide the base controller behavior upon which dynamic solutions can improve.

6 Conclusions

In this chapter we presented the current state of the art of self-adaptive controllers for the Cloud Infrastructure as a Service. IaaS self-adaptive controllers dynamically assign resources to services aiming to strike the balance between under-provisioning (by minimizing service level agreement violations) and over-provisioning (by reducing resource assignments) in response to service workload variations. To efficiently allocate resources, self-adaptive controllers refer to some knowledge about the system characteristics and behavior that is typically encoded in terms of models or rules.

While designing self-adaptive controllers is a challenging task in itself, providing assurance guarantees on self-adaptive controllers behavior is even harder, because IaaS controllers typically face unpredictable environmental conditions (for instance, workloads and co-located services interference) and a very large space of configurations.

The different models used by the controllers come with different levels of assurance and adaptability. The inherent problem with adaptability is that it may give rise to undesirable emergent properties, impede the ability of administrators to understand system behaviors, and possibly reduce the predictability of the controlled system. However, controller adaptability may be required for instance in the case of systems too complex to be modeled analytically or whose environmental conditions and configuration spaces are too vast to be effectively explored before application deployment.

As it often happens, there is no "best" solution for IaaS controllers, the choice of the most suitable approach depends very much on the requirements posed on the control. We considered the trade-off between assurance and adaptability by classifying state-of-the-art approaches as belonging to either static (non-adaptable), dynamic (fully adaptable), or hybrid classes (partially adaptable). Static approaches privilege assurance over adaptability, being statically verifiable, but are less effective in dealing with situations not foreseen at design time. Dynamic approaches keep learning and adapting to the measured system behavior, hence offer models closer to the actual system behavior, typically resulting in a more precise (and efficient) control. However, this may come at the cost of reduced predictability. Hybrid approaches try to compensate the weakness of each of the previous classes by complementing static approaches with some degree of adaptability, and dynamic approaches with a fall-back static control.

We expect future research in this area to concentrate even more on hybrid approaches. The challenge is to be able to closely match system behavior, by continuously gathering performance measurements and adapting system models, while not completely giving up on formal proofs and guarantees on controller actions. To this end, some dynamic approaches, in particular the ones based only on the latest measured system behavior, might incur in the risk of learning the behavior of the system while in unperceived abnormal working conditions (for instance, false positives in monitoring VMs running and working state), thus building invalid models and consequently implementing the wrong control decisions. Hybrid approaches take advantage of the fall-back to less precise but

typically more stable design-time models, and offer a safety net for situations in which the monitoring infrastructure cannot reveal malfunctions. Moreover, multi-model controllers can utilize possible discrepancies in the predictions from their different models to implement simple warning mechanisms (for example, requiring human intervention or interpretation of possible malfunctions) or various model-update policies, for instance with change-point detection. It is therefore our opinion that, independently of the levels of assurance or adaptability required by an application, multi-model solutions can typically offer a deeper insight on the controlled system behavior.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Communication of ACM 53(4), 50–58 (2010)
2. Armbrust, M., Fox, A., Patterson, D., Lanham, N., Trushkowsky, B., Trutna, J., Oh, H.: Scads: Scale-independent storage for social computing applications. In: CIDR (2009)
3. Bennani, M.N., Menascé, D.A.: Resource allocation for autonomic data centers using analytic performance models. In: Proceedings of the International Conference on Automatic Computing, ICAC 2005, pp. 229–240 (2005)
4. Bi, J., Zhu, Z., Tian, R., Wang, Q.: Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: Proceedings of International Conference on Cloud Computing, CLOUD 2010, pp. 370–377 (July 2010)
5. Bodik, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of the Conference on Hot Topics in Cloud Computing, HotCloud 2009, pp. 75–80 (2009)
6. Dejun, J., Guillaume, P., Chi-Hung, C.: Autonomous resource provisioning for multi-service web applications. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, pp. 471–480 (2010)
7. Dutreilh, X., Rivierre, N., Moreau, A., Malenfant, J., Truck, I.: From data center resource allocation to control theory and back. In: Proceedings of International Conference on Cloud Computing, CLOUD 2010, pp. 410–417 (July 2010)
8. Gambi, A.: Kriging-based Self-Adaptive Controllers for the Cloud. PhD thesis, University of Lugano (2012)
9. Hellerstein, J., Diao, Y., Parekh, S., Tilbury, D.: Feedback Control of Computing Systems. Wiley (September 2004)
10. Huber, N., Brosig, F., Kounev, S.: Model-based Self-Adaptive Resource Allocation in Virtualized Environments. In: SEAMS 2011: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Honolulu, HI, USA (2011); Acceptance Rate (Full Paper): 27% (21/76)
11. Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., Pu, C.: Generating adaptation policies for multi-tier applications in consolidated server environments. In: Proocedings of the International Conference on Autonomic Computing and Communications, ICAC 2008, pp. 23–32 (June 2008)

12. Jung, G., Joshi, K.R., Hiltunen, M.A., Schlichting, R.D., Pu, C.: A Cost-Sensitive Adaptation Engine for Server Consolidation of Multitier Applications. In: Bacon, J.M., Cooper, B.F. (eds.) Middleware 2009. LNCS, vol. 5896, pp. 163–183. Springer, Heidelberg (2009)
13. Li, H., Venugopal, S.: Using reinforcement learning for controlling an elastic web application hosting platform. In: Proceedings of the International Conference on Autonomic Computing, ICAC 2011, pp. 205–208 (2011)
14. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: Proceeding of the International Conference on Autonomic Computing, ICAC 2010, pp. 1–10 (2010)
15. Lim, H.C., Babu, S., Chase, J.S., Parekh, S.S.: Automated control in cloud computing: challenges and opportunities. In: Proceedings of the Workshop on Automated Control for Datacenters and Clouds, ACDC 2009, pp. 13–18 (2009)
16. Maggio, M., Hoffmann, H., Papadopoulos, A., Panerati, J., Santambrogio, M., Agarwal, A., Leva, A.: Comparison of decision making strategies for self-optimization in autonomic computing systems. ACM Transactions on Autonomous and Adaptive Systems (to appear)
17. Maggio, M., Hoffmann, H., Santambrogio, M.D., Agarwal, A., Leva, A.: Decision making in autonomic computing systems: comparison of approaches and techniques. In: Proceedings of the International Conference on Autonomic Computing, ICAC 2011, pp. 201–204 (2011)
18. Malkowski, S.J., Hedwig, M., Li, J., Pu, C., Neumann, D.: Automated control for elastic n-tier workloads based on empirical modeling. In: Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC 2011, pp. 131–140 (June 2011)
19. Menasce, D.A., Almeida, V.: Capacity Planning for Web Services: metrics, models, and methods. Prentice Hall (2001)
20. Patikirikorala, T., Colman, A., Han, J., Wang, L.: A multi-model framework to implement self-managing control systems for qos management. In: Proceeding of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, pp. 218–227 (2011)
21. Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., Sharma, N.: Towards autonomic workload provisioning for enterprise grids and clouds. In: Proceedings of the IEEE/ACM International Conference on Grid Computing, GRID 2009, pp. 50–57 (2009)
22. Rodero-Merino, L., Gonzalez, L.M.V., Gil, V., Galán, F., Fontán, J., Montero, R.S., Llorente, I.M.: From infrastructure delivery to service management in clouds. Future Generation Computer Systems 26(8), 1226–1240 (2010)
23. Sharma, U., Shenoy, P., Sahu, S., Shaikh, A.: A cost-aware elasticity provisioning system for the cloud. In: Proceedings of International Conference on Distributed Computing Systems, ICDCS 2011, pp. 559–570 (June 2011)
24. Singh, R., Sharma, U., Cecchet, E., Shenoy, P.: Autonomic mix-aware provisioning for non-stationary data center workloads. In: Proceeding of the International Conference on Autonomic Computing, ICAC 2010, pp. 21–30 (2010)
25. Tamura, G., Villegas, N.M., Muller, H., Sousa, J.P., Becker, B., Karsai, G., Mankovskii, S., Pezze, M., Schafer, W., Tahvildari, L., Wong, K.: Towards practical run-time verification and validation of self-adaptive software systems. In: de Lemos, R., Giese, H., Müller, H., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. Dagstuhl Seminar Proceedings (2011)

26. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing 10(3), 287–299 (2007)
27. Toffetti, G., Gambi, A., Pezzè, M., Pautasso, C.: Engineering Autonomic Controllers for Virtualized Web Applications. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 66–80. Springer, Heidelberg (2010)
28. Trushkowsky, B., Bodik, P., Fox, A., Franklin, M.J., Jordan, M.I., Patterson, D.A.: The scads director: scaling a distributed storage system under stringent performance requirements. In: Proceedings of the USENIX Conference on File and Stroage Technologies, FAST 2011, pp. 12–26 (2011)
29. Urgaonkar, B., Shenoy, P.J., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems 3(1), 1–39 (2008)
30. Vasic, N., Novakovic, D., Miucin, S., Kostic, D., Bianchini, R.: DejaVu: Accelerating resource allocation in virtualized environments. In: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, p. 13 (March 2012)
31. Woodside, M., Zheng, T., Litoiu, M.: Service system resource management based on a tracked layered performance model. In: Proocedings of the International Conference on Autonomic Computing and Communications, ICAC 2006, pp. 175–184 (2006)
32. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: On the use of fuzzy modeling in virtualized data center management. In: Proceeding of the International Conference on Autonomic Computing, ICAC 2007, pp. 25–35 (2007)