Approaches and Techniques for Decision Making In Autonomic Computing Systems

Bhargav P Rajyagor1
1Asst. Professor
1Master of Computer Application (MCA)
1Gujarat Technological University - Chaparda

Abstract---Increasingly, applications need to be able to self-reconfigure in response to changing requirements and environmental conditions. Autonomic computing has been proposed as a means for automating software maintenance tasks. As the complexity of adaptive and autonomic systems grows, designing and managing the set of reconfiguration rules becomes increasingly challenging and may produce inconsistencies. Autonomic computing systems adapt themselves thousands of times a second, to accomplish their goal despite changing environmental conditions and demands. The literature reports many decision mechanisms, but in most realizations a single one is applied. This paper compares some state of the art decision making approaches, applied to a self-optimizing autonomic system that allocates resources to a software application providing performance feedback at run-time, via the Application Heartbeat framework. The investigated decision mechanisms range from heuristics to control theory and machine learning: results are compared by means of case studies using standard benchmarks.

Keywords: Decision mechanisms, Comparison, Design approaches, Algorithms, Design, Performance

I. INTRODUCTION

Autonomic computing is a promising research area to confront the complexity of modern applications and involves many exciting challenges [4], one of which is the establishment of systematic and reproducible design processes. In the literature, the autonomic paradigm is characterized by the presence of three distinct phases: sensing, deciding, and acting; notable examples are the Monitor, Analyze, Plan and Execute (MAPE) or Observe, Decide, Act (ODA) loops [1].

In both cases, the “decide”, or equivalently the “analyze and plan”, phase is responsible for providing and enforcing the desired system properties. Thus, the design of the decision phase is essential for obtaining the desired self-configuring, self-healing, self-optimizing, and self-protecting characteristics [5]. The design of closed-loop autonomic systems shows an impressive convergence with control engineering, which to date has matured. Possible reasons are the difficulty of creating a system model (a set of equations if the control theory is to be used) and obtaining the necessary online measurements (e.g., providing application level instrumentation if applications need controlling).

In contrast, machine learning (ML) techniques may not allow the formal guarantees of control theory, but require little or no explicit modeling. Instead, ML approaches capture complex relationships online, automatically learning the interactions and relationships between components.

Thus, in practice, many autonomic systems will combine techniques, e.g., enhancing feedback control solutions with machine learning mechanisms, and vice versa. In order to combine these techniques in a principled manner, this paper begins an investigation comparing decision making processes for self-optimizing autonomic computing systems, covering heuristic, control, and machine learning methods. Its contributions are (a) the proposal of a set of reference problems for comparing performance and applicability of the mentioned solutions, and (b) the synthesis, development, implementation and testing of said techniques.

II. DESIGN FOR SELF-OPTIMIZATION

Several techniques have been used to synthesize decision mechanisms for self-optimizing computing systems. As a notable example, [8] addresses the problem of how to apply a genetic algorithm. The proposed case study is the dynamic reconfiguration of an overlay network for distributing data to a collection of remote data mirrors. The algorithm balances the competing goals of minimizing costs, and maximizing reliability and performance.

In this work, we detail a single problem, whose generality allows us to draw some considerations about decision making processes. Suppose we want to build a self-optimizing operating system, and one of its tasks is to assign resources to running applications. Notice that the word resources may assume different meanings.

In a single device, an application may receive computational units or memory, while in a cloud infrastructure, a resource can be a server devoted to responding to some requests. Proposals to address the management of a single resource have been published, but proposals to manage multiple interacting ones are rarer. Automatically, the number of ways the system capabilities can be assigned to different applications grows exponentially with the number of resources under control, and the optimal allocation of one resource depends on the allocated amounts of other resources, requiring coordination.

Shared resources are redistributed among applications at fixed decision-making intervals, allowing the system to respond to dynamic changes, based on an Artificial Neural Network that learns to approximate the application performance from sensor data. The focus of this paper is the comparison of different techniques to design a self-optimizing system, covering heuristic, machine learning and control-theoretical approaches. A representative problem will be used as a reference example. The goal is to manage the performance of software applications.
instrumented to emit their performance level via the Application Heartbeat framework [3].

III. TECHNIQUES
This section presents an overview of some common techniques for making decisions in a self-optimizing system.

Heuristic solutions start from a guess about application needs and adjust this guess. Heuristic solutions are designed for computational performance or simplicity at the potential cost of accuracy or precision. Such solutions generally cannot be proven to converge to the optimum or desired value.

A notable example is the greedy approach in [3] that optimizes resource allocation and energy management in a hosting center. This system controls server allocation and routing requests based on an economic model, where customers bid for resources as a function of service volume and quality.

Standard control-based solutions employ canonical models – two examples being discrete-time linear models and discrete event systems – and apply standard control techniques such as Proportional Integral (PI) controllers, Proportional Integral and Derivative (PID) controllers, optimal controllers, Petri nets.

Assuming the model to be correct, some properties may be enforced, among which stability and convergence time are probably the most important ones, thereby providing formal performance guarantees. As an ex- ample, Pan et al. [24] propose two PI controllers for guaranteeing proportional delay differentiation and absolute delay in the database connection pool for web application servers. The model used is a first order linear time-invariant system and the PI controllers are design with the Root Locus method. Such techniques may not however be enough in the case of heavily varying environment or workload conditions.

Advanced control-based solutions require complex models, with some unknown parameters (e.g., the machine work- load) that may be estimated online, to provide Adaptive Control (AC).

AC requires an identification mechanism and the ability to adjust controller parameters on the fly. Another advanced control strategy is Model Predictive Control (MPC) where the controller selects the next actions based on the prediction of the future system reactions. The over- head of sophisticated control solutions is greater than that of standard controls; however, one may still be able to formally analyze parameter-varying systems and prove stability, obtaining formal guarantees even in the case of unknown operating conditions.

For example, [7] proposes an approach based on model identification, to adjust the CPU percent- age dedicated to the execution of a web server. A first-order auto-regressive model is used for identification purposes. A PI control structure is presented together with an adaptive controller. The recursive least squares method is used to estimate the model parameters.

Model-based machine learning solutions require the definition of a framework in which to learn system behavior and adjust tuning points online. Neural Networks (NN) are often useful to build a model of the world for control purposes. NN solutions may be used to predict the system reaction to different inputs and, given some training samples, to build a model. The structure of the network and the quality of the training data are critical to performance. The accuracy of the results depend on these crucial choices, and thus no a priori guarantees can be enforced. Another model-based family of techniques is Genetic Algorithms (GA).

Using a genetic algorithm requires selecting a suitable representation for encoding candidate solutions (in other words, a model). In addition, some standard operators (crossover and mutation) must be defined and a mathematical function must be provided to rate candidate solutions and select among them. The overhead of both neural networks and genetic algorithms may in principle be very significant.

Model-free machine learning solutions do not require a model of the system. A notable example is Reinforcement Learning (RL), even if a recent research trend is to complement RL solution with a model definition. RL agents face three major challenges. The first challenge is how to assign credits to actions, the second is how to balance exploration versus exploitation and the third is generalization.

IV. MACHINE LEARNING FRAMEWORKS
Machine learning techniques are often employed as decision mechanisms for a variety of systems. Machine learning allows computers to evolve behaviors based on empirical data, for example from sensor data.

During the years, a number of techniques have been proposed to address both specific and broad issues. As done for classical control-theoretical frameworks, we focus our exploration on well-established, standard decision mechanisms. Therefore, we will explore the implementation of a neural network and a reinforcement learning algorithm, as examples of model based and model free techniques, respectively.

The neural network has to produce the next step control out- puts from the current situation, with the purpose of reducing the error between the measured heart rate and the desired one. Every time we have a new sample, we feed that into the network and update its weights according to the gradient of the error we are experiencing.

![Fig. 1: Neural Network topology.](image)

The network is composed by four different input sources, corresponding to the desired heart rate, the actual heart rate and the two control inputs: number of cores and frequency. With three neurons in the (single) hidden layer and two output neurons we learn the relationship between the in- puts and the (possibly optimal) control strategy. It is worth stressing that we didn’t train the network before launching the experiments and the network itself is trained online, up- dating the weights according to the experienced
error with a gradient descent method. The network topology is shown in Figure 1.

The activation function used for the hidden layer is the atan, while the output layer uses a linear combination; the learning rate is set to 0.6. In general, to find the best solution, a study on the influence of these two parameters should be conducted; moreover, one may in principle train the network before updating the weights online. These two tasks are deferred to future investigations.

A. Genetic algorithms

Genetic algorithms are another class of popular and well-assessed machine learning techniques.

The employment of a genetic algorithm requires selecting a suitable representation scheme for encoding candidate solutions, to define some standard operators, crossover and mutation, and to encode a mathematical function to rate the obtained solutions and to select among them. A possible solution for the proposed case is to synthesize a solution as a couple \{cores, frequency\} and to use as crossover operator the choice of the number of cores from the first solution and the frequency from the second pair.

A mutation operator could be an increase or decrease in one of the two quantities. The fitness function for the proposed problem could be some function of the desired heart rate and the expected speedup given to the application by the encoded solution.

Notice that this straightforward idea for GA is not how- ever of particular interest for the proposed problem. Usually, GAs are used when it is hard to explore the space of possible solutions (either because that space is infinite or for other reasons). In the proposed scenario, the number of possible solutions would be 56 and the algorithm would just choose the most suitable solution, without evolution. In this case it makes no sense to implement a GA and pay its overhead. However, there may be different ways to encode the solution that would exploit the potential of the strategy.

B. Reinforcement Learning

As for reinforcement learning solutions, we implement a SARSA (State-Action-Reward-State-Action) algorithm for the problem at hand [13]. The algorithm learns a decision policy for the system. A SARSA implementation interacts with the environment and updates the policy based on actions taken, known as an on-policy learning algorithm.

It can be shown that under certain boundary conditions SARSA will converge to the optimal policy if all state-action pairs are visited infinitely often.

V. CONCLUSIONS

In this paper, we proposed a comparison of some state-of-the-art techniques for building decision making mechanisms in autonomic computing systems, focusing on self-optimization. The decision making mechanism needs to provide the necessary resources (and possibly no more) to meet the requirements.

Thus, we conclude that for specific systems with a narrow range of target applications developers are probably best off testing a number of decision mechanisms, but for systems with a broad or unknown range of target applications, adaptive control may provide the best general solution.

REFERENCES