Automatic Adaptation of SOA Systems Supported by Machine Learning

Kornel Skałkowski and Krzysztof Zieliński

AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunication, Department of Computer Science, al. Mickiewicza 30, 30-059 Kraków, Poland
{skalkow,kz}@agh.edu.pl

Abstract. Recent advances in the development of information systems have led to increased complexity and cost in terms of the required maintenance and management. On the other hand, systems built in accordance with modern architectural paradigms, such as Service Oriented Architecture (SOA), posses features enabling extensive adaptation, not present in traditional systems. Automatic adaptation mechanisms can be used to facilitate system management. The goal of this work is to show that automatic adaptation can be effectively implemented in SOA systems using machine learning algorithms. The presented concept relies on a combination of clustering and reinforcement learning algorithms. The paper discusses assumptions which are necessary to apply machine learning algorithms to automatic adaptation of SOA systems, and presents a machine learning-based management framework prototype. Possible benefits and disadvantages of the presented approach are discussed and the approach itself is validated with a representative case study.

Keywords: SOA, adaptive manager, machine learning.

1 Introduction

The growing complexity of modern IT systems hinders effective administration, resulting in increased maintenance costs. Geographical distribution of services, dynamic workflow enactment and on-demand service selection improve the systems' scalability and flexibility but do not foster their overall manageability. It should be noted, however, that contemporary architectural paradigms such as Service Oriented Architecture (SOA) [1] or Internet of Things (IoT) [2], provide sophisticated adaptation features [3,4]. Flexibility in terms of service/sensor coupling, instant binding or semantic message routing can be used to modify information flow between system components during runtime, affecting processing speed. Such extensive adaptation opportunities are characteristic of modern design approaches and can be leveraged to solve problems associated with system management and administration through automatic or semi-automatic adaptation.

Issues involved in the adaptation process of enterprise systems are addressed by the well-known MAPE adaptation pattern [5], which introduces four elements

L.M. Camarinha-Matos, S. Tomic, and P. Graça (Eds.): DoCEIS 2013, IFIP AICT 394, pp. 61–68, 2013.

(Monitoring, Analysis, Processing, Execution) necessary in every adaptation framework. This paper presents a new approach to implementation of the Analysis and Processing elements of the MAPE pattern based on a combination of two types of machine learning methods. A clustering algorithm is used to provide automatic recognition of similar system states and grouping them into subsets (called clusters), based on information provided by the Monitoring element interface (e.g. regarding a system load or observed bottlenecks). The goal of further processing is then to find a mapping between the clusters and adaptation actions provided by the Execution element interface (e.g. a service replication, routing changes or resources allocation). These actions should be assigned to clusters in such a way that execution of actions attributed to a cluster to which a current system state has been assigned increases the overall system QoS (Quality of Service). In order to find a mapping which satisfies this condition, a reinforcement learning algorithm has been devised. The paper explains how such a combination of machine learning methods can effectively and flexibly implement the MAPE pattern in service-based systems and discusses assumptions which have to be met by an adaptable system in order to be applicable to our solution. The proposed approach to MAPE is evaluated on the basis of a proof-of-concept implementation.

The paper is organized as follows: in Section 2 the relationship between the proposed approach and IoT architectures is discussed. Section 3 briefly presents related approaches to MAPE implementation. In Section 4 a machine learning-based approach to implementation of the MAPE pattern is elucidated. Section 5 shows how the concept has been implemented in a prototype framework and which algorithms have been chosen. Section 6 discusses evaluation results while Section 7 concludes the paper and discusses future work.

2 Relationship to Internet of Things

Adaptation issues are widely present in various aspects of IoT systems. The vast quantities of objects involved in such systems, huge amounts of information produced, chaotic working environments and the need for autonomous control make efficient and flexible adaptation a crucial part of many IoT solutions. Implementation of the MAPE pattern in IoT architectures requires dedicated monitoring and execution layers which can cope with such issues. Since the approach presented in this paper does not impose any specific monitoring and management framework, it can be applied to IoT infrastructures as well as to other manageable systems.

3 Related Work

Existing approaches to implementation of the MAPE pattern are based on rule/policy engines, decision theory or fuzzy logic. The use of machine learning techniques for SOA system adaptation is only partially covered in existing papers. An approach to context-based adaptation in production systems based on data mining techniques has been proposed in the Self-Learning project [6], which bases in part on learning and

adapter modules. Nevertheless, recent publications released by this project do not clearly point to any particular data mining algorithms and do not present any evaluation results. Other existing papers focus on machine learning-based selection of web services [7,8] and reliability assessment in SOA systems [9]. Although these approaches can partially solve the issue of automatic management of service-based systems, they do not constitute a complete implementation of the MAPE pattern.

4 Machine Learning-Based Approach to MAPE Pattern Implementation

Most existing machine learning algorithms operate on sets of n-dimensional real valued vectors $x \in \mathbf{R}^n$. Unsupervised learning methods, i.e. clustering algorithms, operate directly on such sets, whereas in the case of supervised learning methods or reinforcement learning algorithms an additional value, y, is assigned to every vector and interpreted as the "correct answer" to it. The goal of the analysis and processing elements of the MAPE pattern is to find out which action offered by the execution layer should be invoked in a specific system state. To achieve this goal using machine learning methods we have to represent system state as an n-dimensional vector, while the action suitable for a given system state is equivalent to the "correct answer" value. Based on these observations, the system state at point t is represented as $x^{(t)} \in \mathbf{R}^n$, whereas the set of all observed system states at various points in time ($x^{(t1)}$, $x^{(t2)}$, ..., $x^{(tm)}$) will be called the system state space **X**. It is important to stress that vectors $x^{(t)}$ should contain all available information about the system which should be taken into account during management, including the working context and current configuration. Representing the state of a system in the form of a vector of real values may seem somewhat constraining, yet even those parameters which are expressed in non-numeric form (e.g. strings or enumerations) can usually be converted to numeric values by applying appropriate mappings. Since the main goal of our approach is to manage a complex system in a way which increases its overall QoS level, the system state vectors are assumed to reflect the QoS experienced by users in some way. Certain system parameters directly reflect QoS (e.g. processing time), whereas in more sophisticated cases the QoS level can be calculated with evaluation function e(x).

The management interface which constitutes the execution element of the MAPE pattern, is assumed to be represented as set of values: $\mathbf{A}=\{a_0, a_1, .., a_k\}$, consisting of the available adaptation actions. In order to avoid contradictions and discrepancies during learning we have to assume the action set **A** meets several conditions. First of all, there are *no duplicate actions* in the set, since most learning algorithms use injective functions to produce the "answer value" a_i. Furthermore, we assume that every action a_i can be *repeated any number of times* and the actions are *stateless* (i.e. no action has a different effect when invoked several times in the same system state). Finally, it is necessary for the actions to be *independent of each other*, meaning that no action should require prior execution of any other action. If some actions have to be invoked in a specific sequence, they should be represented as a single action. These assumptions are not challenging and every well designed management

framework usually satisfies them all. In order to make actions comparable to "correct answer" values returned by learning algorithms, the actions should be bijectively mapped to numbers, e.g. simply enumerated. In order to facilitate implementation, both sets (**X** and **A**) are assumed to be fixed for each given adaptable system.

Given the system state and management interface we can precisely define the overall framework goal. Roughly speaking, the framework should perform actions from the space **A** (e.g. service launch or migration) so that the adaptable system provides the best possible QoS level for end users. From a mathematical view point, this problem can be divided into two subproblems. The first subproblem is clustering the system state space **X** into a set of non-empty sets $\{C_1, ..., C_l\}$ which should be characterized by the maximum possible homogeneity of elements within each set (e.g. lowest sum of distances between the elements of C_i) and the maximum possible diversity between sets (e.g. greatest sum of distances between the $\{C_i\}$ sets' centroids). The C_i sets can evolve during system runtime, reflecting changes in the system and its working environment. The second subproblem is mapping the clustered system state space $\{C_i\}$ onto actions: the framework has to find a mapping $\forall_{i=1, ..., l}$ **F**: $C_i \rightarrow (a_j: j=0, ..., k_{(Ci)})$ such that the execution of actions returned by the mapping **F** when the system state belongs to the cluster C_i causes $\sum_i e(x^{(ti)})$ to assume its lowest possible value. The function $e(x^{(t)})$ is the overall system QoS evaluation metric calculated using state vectors $x^{(t)}$ whose values are inversely proportional to the condition of the system. Applying **F** yields a sequence of actions with length $k_{(Ci)}$, sorted from the most appropriate to the least appropriate one (for a given system state) – thus we can say that mapping **F** reflects the adaptable system model. The first subproblem may seem unnecessary as one might claim that actions could be assigned directly to system states $x^{(t)}$. In reality, however, this assumption is only satisfied by very small systems, where **X** can be modeled e.g. as a small finite state machine. In most real systems – especially complex enterprise SOA solutions – this assumption is no longer valid. In such cases the space **X** is usually infinite and multidimensional, so that both elements are essential in order to accomplish the framework objectives.

The first task is a well-known clustering problem, the only major issue being that the clustered space **X** is not known a priori, but is instead constructed during runtime by aggregating $x^{(t)}$ vectors. This issue can be solved using online clustering methods which are designed to cluster data streams. In turn, the second task leads us to the area of reinforcement learning algorithms which are used to teach computer systems how to act in different situations in order to achieve a given goal. The learning mechanism in such algorithms is based on rewards, usually represented as a single real number. In our case the situations are represented as state vectors $x^{(ti)}$ at different points in time t_i, whereas the reward constitutes the system evaluation metric e(x). The reinforcement learning algorithm returns a function, $h_\theta(x)$, called the hypothesis, which provides "correct answer" values for different vectors x. In our approach this function is equivalent to mapping **F** – the returned sequence of actions comprises set **A**, which is calculated on the basis of differences between actions from **A** and the $h_\theta(x)$ function results. The final necessary element is normalization of state vectors $x^{(t)}$. Since most machine learning algorithms require input vectors to have all elements normalized to a common range of values, a normalization function has to be applied to all state vectors prior to clustering. The approach is depicted in Fig. 1.

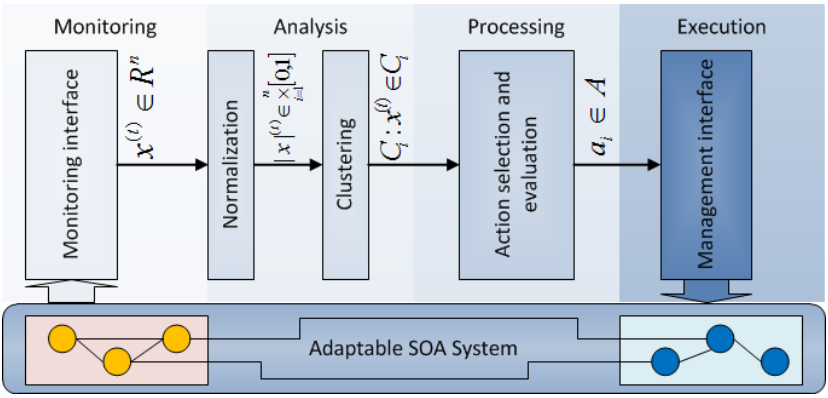

Fig. 1. The machine learning-based approach to the MAPE pattern

The bottom part of Fig. 1 presents an SOA system subjected to adaptation. Above, four elements of the MAPE pattern are shown. The monitoring and management interfaces are assumed to be provided by external frameworks which can be tuned to a specific adaptable system (provided that they meet the assumptions previously stated). The monitoring interface returns vectors $x^{(t)}$ composed of real values representing system states. The management interface exposes **A** – the set of management actions available in the system. The analysis layer consists of two services: the normalizer service, responsible for mapping state vectors to an n-dimensional [0,1] hypercube, and the clustering service which, based on normalized $|x|^{(t)}$ vectors, extracts clusters $\{C_i\}$ representing groups of similar system states. Each cluster represents a pool of system states which significantly diverge from all other states. By assigning vector $|x|^{(t)}$ to an appropriate cluster, the framework can check whether the adaptable system currently belongs to the best possible cluster. This information, along with the current QoS evaluation metric derivative $\partial e(x)/\partial t$ calculated as a differential approximation from several successive observations of e(x), is used by the processing layer to select actions. When $\mathrm{sgn}(\partial e(x)/\partial t) < 0$ no management action is performed because even if the system state is not in the best possible cluster, it is improving and this trend should be maintained. If, however, $\mathrm{sgn}(\partial e(x)/\partial t) \geq 0$ and the system state does not belong to the best cluster, execution of a management action is necessary. In order to do so, a sequence should be returned by the reinforcement learning algorithm. A third case should be distinguished, with $\mathrm{sgn}(\partial e(x)/\partial t) \geq 0$ and the system already assigned to the best cluster. In this case we may not know why the system condition is deteriorating – whether due to brief fluctuations (e.g. caused by a slightly higher load) or more permanent reasons. Thus, a prediction algorithm could be applied to estimate the likely evolution of the system state. Every executed action is evaluated, and, based on the evaluation result, the reinforcement learning algorithm's hypothesis function $h_\theta(x)$ is up- or downregulated in order to improve future decisions. Evaluation bases on observation of system state changes reflected in the QoS evaluation metric derivative $\partial e(x)/\partial t$ over a period of time. If the system state remains poor and shows no signs of improvement ($\mathrm{sgn}(\partial e(x)/\partial t) < 0$), another action from the list returned by the algorithm is executed and evaluated. This process repeats until the system state begins to improve.

The key advantage of the proposed approach is its independence of any specific system model. In contrast to other approaches, e.g. based on policy or rule engines, it does not require any initial configuration or specifications of the adaptable system's model. Moreover, online clustering and reinforcement learning algorithms can dynamically adapt to changes in the model without reconfigurations or restarts. On the other hand, lack of initial knowledge about the managed system means that many incorrect actions can be taken during the startup phase, before the framework learns how to appropriately manage a given system.

5 Approach Implementation and Applied Algorithms

A prototype implementation of the approach described in the previous section is currently being developed. Its most recent version has been implemented as a set of OSGi [10] services providing the above mentioned features. The implementation consists of four services: the normalization service (responsible for state vector normalization), the clustering service (responsible for clustering), the strategy service (responsible for action selection and evaluation) and the evaluation service (providing the e(x) function values).

Since the aim of the implementation is to validate the proposed concept rather than provide sophisticated functionality, the prototype relies on simple machine learning algorithms. Specifically, the clustering service implements a standard k-means algorithm to cluster state vectors collected over a period of time. In the future this algorithm will be swapped for an online clustering algorithm based on the PCA method [11]. As the reinforcement learning algorithm, a simple adaptive gradient descent implementation with a polynomial hypothesis function has been used. The main disadvantage of this algorithm is slow convergence – in the future we intend to apply a more efficient reinforcement learning algorithm.

6 Prototype Evaluation Results – Preliminary Study

The objective of evaluation of the prototype framework was to check whether it properly accomplishes its goals, i.e. invokes appropriate management actions when the overall system QoS level decreases, and to verify if the learning method is appropriate, i.e. whether the hypothesis function properly converges regardless of its initial coefficients. Both goals were evaluated on a load balancing case study in an SOA system. The simulated system consisted of three services. The first service had to invoke either the second or the third service in order to accomplish its functionality. By default, the first service used only the second service – thus the third service remained idle. The response time of the second service was highly dependent on the number of simultaneous invocations. As the number of concurrent requests grew, the service's response time increased noticeably, affecting the overall system QoS. In such cases, the first service was expected to begin using the third service in order to balance load and avoid a decrease in the overall QoS. The simulated services have been implemented using the OSGi technology and deployed in an OSGi monitoring and management

framework provided by the AS3 Studio [12] toolkit. The monitoring interface was configured to monitor two parameters of the system: average processing time (**APT**) and invocation rate (**IR**). These two parameters were passed to the framework prototype as a single vector $x^{(t)}$ = [**APT**, **IR**]. The evaluation function was calculated as e(x) = **APT** + **IR**/2. The management interface exposed two actions: "do-nothing", whose invocation did not affect the simulated system in any way, and "balance-load", which activated load balancing in the first service for a period of time. As a result of the second action, the first service would begin dispatching its requests to both the second and the third service. The evaluation was performed on a computer with an Intel Core 2 Duo 2.80 Ghz CPU and 4 gigabytes of RAM. The hypothesis function was a simple linear polynomial of two variables $h_\theta(x^{(t)}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$, where x_1 = **APT** and x_2 = **IR**. Tab. 1 presents evaluation results from three test runs.

Table 1. Evaluation results of the prototype framework

Processing time speedup	Initial $h_\theta(x)$	Final $h_\theta(x)$	Convergence time	Invalid actions
24%	$0.64+0.69x_1+0.57x_2$	$0.68+0.71x_1+0.59x_2$	0:20 [h]	2
21%	$0.23-0.44x_1+0.01x_2$	$1.06+0.07x_1-0.21x_2$	2:00 [h]	7
19%	$-0.55-0.23x_1-0.03x_2$	$1.12+0.13x_1-0.53x_2$	2:30 [h]	21

Evaluation results confirm that the proposed approach to the MAPE pattern implementation is viable and properly accomplishes the stated goals. Processing time speedup was in the 19% - 25% range, depending on initial coefficients of the hypothesis function. The greatest speedup was observed for near-optimal initial hypothesis coefficients, because in this case the framework almost always executed the "balance-load" action when necessary. In other cases the framework executed a greater number of "do-nothing" actions, before it learned that this action was inappropriate for a high system load state. Convergence time was directly dependent on initial hypothesis coefficients. Better coefficients improved the algorithm's convergence; however in all cases convergence was eventually attained (although with differing final hypothesis coefficients).

7 Conclusions and Further Work

The proof-of-concept evaluation of the approach proposed in the paper shows that machine learning methods can be applied to implementation of the MAPE pattern. Our combination of clustering and reinforcement learning algorithms properly identifies disruptions in system QoS and invokes appropriate management actions. The main advantage of the proposed approach is its independence of any specific system – the framework does not require any *a priori* knowledge about the adaptable system. Flexibility offered by online clustering and reinforcement learning methods means that the approach can be applied to SOA system adaptation as well as to IoT system management. The only evident disadvantage is its potentially long convergence time.

Further development will focus on more advanced algorithms for online data clustering and selection of management actions. The framework effectiveness and scalability will also be evaluated on much more complex case studies and real-world systems, e.g. a telemedicine platform, where maintaining a certain level of QoS is crucial. We also intend to improve the efficiency of our approach by implementing a system state prediction algorithm which could invoke management actions in order to prevent QoS disruptions.

Acknowledgments. The research presented in this paper was partially supported by the European Regional Development Fund programs no. POIG.01.03.01-00-008/08 and UDA-POKL.04.01.01-00-367/08-00.

References

1. Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., Channabasavaiah, K.: S3: A Service-Oriented Reference Architecture. IT Professional 9(3), 10–17 (2007)
2. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A survey. The International Journal of Computer and Telecommunications Networking 54(15), 2787–2805 (2010)
3. Zieliński, K., Szydło, T., Szymacha, R., Kosiński, J., Kosińska, J., Jarząb, M.: Adaptive SOA Solution Stack. IEEE Transactions on Services Computing 5(2), 149–163 (2012)
4. André, F., Daubert, E., Gauvrit, G.: Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures. In: 2010 Fifth International Conference on Internet and Web Applications and Services, ICIW, pp. 309–314 (2010)
5. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
6. Stokic, D., Scholze, S., Barata, J.: Self-learning embedded services for integration of complex, flexible production systems. In: IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, November 7-10, pp. 415–420 (2011) ISSN: 1553-572X
7. Al-Masri, E., Mahmoud, Q.H.: Discovering the best web service: A neural network-based solution. In: IEEE International Conference on Systems, Man and Cybernetics, SMC 2009, October 11-14, pp. 4250–4255 (2009)
8. Mohanty, R., Ravi, V., Patra, M.R.: Web-services classification using intelligent techniques. Expert Systems with Applications 37(7), 5484–5490 (2010) ISSN 0957-4174
9. Challagulla, V.U.B.: A Machine Learning-Based Reliability Assessment Model for Critical Software Systems. In: 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, July 24-27, vol. 1, pp. 79–86 (2007)
10. The OSGi Alliance: OSGi Service Platform Core Specification, Release 4.1 (2007), `http://www.osgi.org/Specifications`
11. McWilliams, B., Montana, G.: Predictive Subspace Clustering. In: 2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA), vol. 1, pp. 247–252 (2011)
12. Żmuda, D., Psiuk, M., Zieliński, K.: Dynamic monitoring framework for the SOA execution environment. Procedia CS 1(1), 125–133 (2010)