Decentralized Planning for Self-adaptation in Multi-cloud Environment

Azlan Ismail[1] and Valeria Cardellini[2]

[1] Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA (UiTM), Malaysia
azlanismail@tmsk.uitm.edu.my

[2] Department of Civil Engineering and Computer Science Engineering
University of Roma "Tor Vergata", Italy
cardellini@ing.uniroma2.it

Abstract. The runtime management of Internet of Things (IoT) oriented applications deployed in multi-clouds is a complex issue due to the highly heterogeneous and dynamic execution environment. To effectively cope with such an environment, the cross-layer and multi-cloud effects should be taken into account and a decentralized self-adaptation is a promising solution to maintain and evolve the applications for quality assurance. An important issue to be tackled towards realizing this solution is the uncertainty effect of the adaptation, which may cause negative impact to the other layers or even clouds. In this paper, we tackle such an issue from the planning perspective, since an inappropriate planning strategy can fail the adaptation outcome. Therefore, we present an architectural model for decentralized self-adaptation to support the cross-layer and multi-cloud environment. We also propose a planning model and method to enable the decentralized decision making. The planning is formulated as a Reinforcement Learning problem and solved using the Q-learning algorithm. Through simulation experiments, we conduct a study to assess the effectiveness and sensitivity of the proposed planning approach. The results show that our approach can potentially reduce the negative impact on the cross-layer and multi-cloud environment.

Key words: Cross-layer self-adaptation, decentralized planning, Markov Decision Process, multi-cloud, reinforcement learning, Q-learning

1 Introduction

The smart environment concept concerns with the implementation of smart city technologies into the urban environment with the goal to benefit and improve people's daily lives. In the context of Internet computing, a smart city can effectively process networked information to improve outcomes on any aspect of city

[⋆] The original publication is available at http://link.springer.com/chapter/10.1007%2F978-3-319-14886-1_9 in *Advances in Service-Oriented and Cloud Computing*, G. Ortiz and C. Tran (eds.), Communications in Computer and Information Science Vol. 508, 2015, pp 76-90.

operations [3]. The key infrastructures to enable smart city technologies lie on Internet of Things (IoT) and Cloud computing. In principle, IoT comprises of three main components; sensors and actuators, data analytic tools, and visualization tools [9]. In a large-scale context, especially with sensor data streams [2], these components should be deployed and managed in a multi-cloud setting.

The layered and multi-cloud environment introduces a compelling challenge in the self-adaptation of IoT-oriented applications [11]. A possible approach to support such mechanism is the adoption of a decentralized control method [21], which is fundamentally based on the MAPE (Monitor, Analyze, Plan, and Execute) reference framework of autonomic computing [12]. MAPE provides a closed control loop capability which can enable an autonomic component to realize self-CHOP (Configuration, Healing, Optimization, Protection) properties as well as emergent self-* properties [18]. The decentralized approach promotes multiple MAPEs and thus a set of adaptation managers to control the adaptation while interacting between each other. Depending on the chosen decentralized pattern, some managers act as sensors and executors, while others analyze and make decisions, or all managers have a complete capability as defined in MAPE. By implementing this way, each manager is able to make and execute its own decision with its own environment. At the same time, it is aware of the other systems operating in the same global environment through coordinated interaction.

The benefits of the decentralized approach include flexibility, performance, and fault tolerance. Indeed, each adaptation component has flexibility in making its own plan and executing it; as a result, the performance of the adaptation mechanism can be speeded up, since the adaptation managers are able to plan and take the necessary actions within their scope (within a layer or a cloud environment). In addition, decentralization provides fault tolerance in the presence of some component failure. However, decentralized self-adaptation implies a communication overhead since the managers should coordinate their actions.

In this paper we present a conceptual decentralized self-adaptation architectural model that operates in a multi-cloud environment. The architecture supports the capability of information sharing pattern among the autonomic adaptation managers, that are responsible to manage the cross-layer as well as the multi-cloud environment.

We also address the decentralized planning problem in realizing the self-adaptation process. In decentralized settings, the adaptation manager will plan the adaptation by considering the information within its environment. However, it can occur that the executed plan impacts negatively on the other layers and cloud providers. Herein, the negative impact refers to the a situation where the predefined Service Level Agreements (SLAs) of a layer or a cloud provider are violated. For instance, rebinding a service at the application layer may increase the resource utilization at the infrastructure layer, thus causing a violation of the resource utilization at a certain period of time. Another example is rescaling the size of the virtual machines which may slow down the application; the slowing down of the application may in its turn affect the application layer, namely the application reliability. Furthermore, the effect of multiple cloud providers that may be involved need to be considered as well; for example, the migration of a processing service from one cloud to another with the goal to locate the service closer to the data sources can affect negatively the traffic load on the receiving cloud and thus its performance. Therefore, a planning solution that is aware of cross-layer and multi-cloud effects is required. To this end, we define a decentralized planning approach that is formulated as a Reinforcement Learning problem and solved using the Q-learning algorithm.

The rest of this paper is organized as follows. We discuss the challenges that require for a decentralized planning for self-adaptation in Section 2. Then, we present a conceptual decentralized self-adaptation architectural model in Section 3. In Section 4, we propose a decentralized planning approach for the self-adaptation. We then report the experimental study to show the effectiveness of the proposed approach in Section 5. In Section 6, we highlight related work. Finally, we conclude and provide hints for future work in Section 7.

2 Challenges for Self-adaptation

2.1 Example Scenario

To motivate our work, we consider as example a traffic management system for an urban environment [14]. This kind of system consists typically of a set of sensing, data analysis and visualization capabilities. The sensing application collects the traffic condition data through sensors dispersed in the urban environment; the data analysis application analyzes and explores the data; finally, the visualization application is used to interpret data in a meaningful form.

In a multi-cloud setting, these applications can be located in different clouds due to non-functional requirements. For instance, the sensing application can be located in a proximate cloud facility that is typically at the network edges (the so called Fog computing [1]), the data analysis application in a distant and resource-rich cloud data center, and the visualization application in a different remote cloud providing high computing power.

In the multi-layer perspective, each type of cloud has more than one layer. In general, a cloud may have an application layer, the middleware layer (platform) to manage the runtime lifecycle of the application, and the infrastructure layer which comprises the virtualized computing, storage and network resources.

During runtime, fault conditions may occur [4] which can cause a specific layer or a cloud to become unresponsive, slow, failed, etc. Such problematic condition requires a self-adaptation mechanism to bring the entire traffic management system back into a healthy condition.

2.2 Motivation

Weyns et al. [21] has proposed several decentralized control patterns which are useful to support the self-adaptation process, i.e., information sharing patterns. Some benefits of the decentralized approach are related to its flexibility and performance. In a decentralized approach, each adaptation component has flexibility in making its own plan and executing the plan. As a result, the performance of the adaptation mechanism can be speeded up, since the adaptation managers are able to plan and take the necessary actions within their scope (within a layer or within a cloud environment).

However, there are issues to be addressed to successfully perform the planning mechanism. In this paper, we focus on the uncertainty impact caused by the future adaptation actions. This means, an adaptation action specifically targeted to a layer may cause a negative impact on other layers or clouds. Herein, the negative impact refers to the a situation where some non-functional requirement is violated. For instance, redeploying an application at the application layer may affect the infrastructure layer in such a way that the resources utilization at a certain period of time is violated. Another example is rescaling the size of the virtual machines which may cause the application to slow down. The slowing down of the application may in its turn affect the application layer, namely the reliability of the application. The uncertainty of the negative impact is caused by the limited knowledge and control obtained by the adaptation managers in producing their adaptation plan. It can be reduced by providing more predicted information to the respective adaptation manager in the planning phase. Therefore, in this paper, we focus on proposing a decentralized planning approach with the cross-layer and multi-cloud impact information.

3 Decentralized Self-adaptation Architecture

In this section, we present the decentralized self-adaptation architecture for the multi-cloud environment in which IoT applications are deployed.

The overall architecture is illustrated in Figure 1 and consists of the layered and multi-cloud models. The layered model within each cloud represents the relationship between the adaptation managers, called Local Adaptation Manager (LAM), and the cloud layers, i.e., application, platform, and virtual machine (infrastructure) layers. This relationship is also viewed as an internal one within a given cloud environment. Each LAM has the capability to monitor, analyze, plan, and execute the proper adaptation actions on the layer it manages. It also has a local storage (LS) to keep the knowledge within its own environment. Whenever needed, LAMs can exchange information between each other. That is, each LAM may request information (or a portion of information) directly to another LAM. As a result, the respective LAM will respond with the required information. This internal communication between LAMs is essential to effectively plan for the adaptation. This kind of setting is called an information sharing pattern [21].

The multi-cloud model represents the relationship between LAMs and the external environment. This is essential since an IoT application can be deployed over multiple clouds. Hence, an adaptation performed by any LAM within its cloud environment can affect another cloud. Whenever needed, each LAM can make a request to the communicator component to get the relevant information from another cloud environment. By using some external information, which is

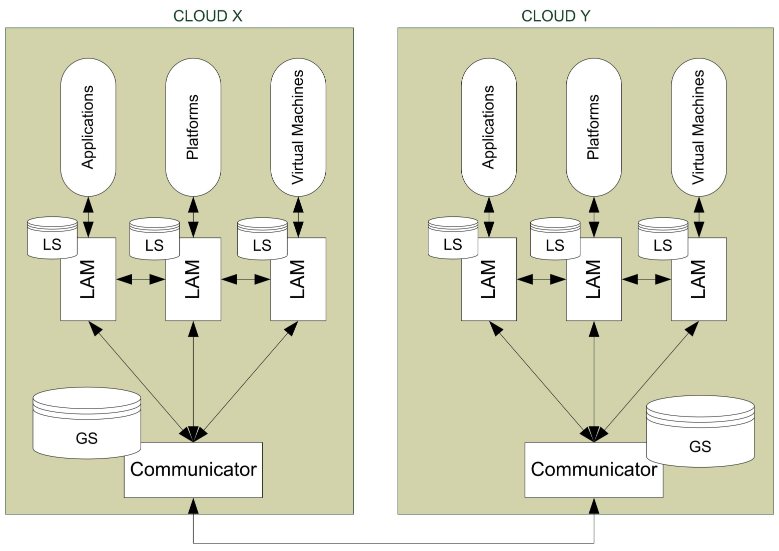


Fig. 1. Decentralized self-adaptation architecture

stored in the global storage (GS), the adaptation plan can be improved. This external information sharing also represents the information sharing pattern.

The communicator in each cloud becomes the agent to support the external communication for requesting and receiving the relevant information about the adaptation impact on the other clouds. The request is triggered by LAM whenever an adaptation is needed. Upon receiving a request from LAM, the communicator submits the request to the respective clouds to obtain the impact information. After receipt of the required response, the communicator returns the information to the respective LAM. In case of receiving a request from another cloud environment, the communicator forwards the request to the respective LAM as well, and waits for a response to be returned to the requesting cloud. By relying on the communicator, each LAM can focus on its core functionalities and let the external communication details be handled by the communicator. The implementation of communicators can be based on the adoption of a message-oriented middleware [8].

The local storage (LS) is used to store several internal information, including the monitoring and analysis data, the adaptation plan and actions, and the cross-layer impact information. One internal information that is exchanged between LAMs during the planning phase is related to the cross-layer impact. It represents the predicted impact level (positive or negative), which is estimated by the affected LAM and caused by the adaptation actions required by the LAM that have planned them to react to some problematic situation. The LAM that initiates the adaptation requests the impact information from another LAM.

The global storage (GS) is used to store external information such as the cloud location and performance. An external information exchanged between clouds during the planning phase is the multi-cloud impact information, which is the estimated impact level caused on the affected cloud by the adaptation actions planned by the problematic cloud. The cloud that initiates the adaptation requests the impact information from the related clouds.

4 Local Adaptation Manager Planning

In this section, we present the design of the planning approach for LAM. We begin by introducing the planning process. Then, we present the model of the planning problem based on Markov Decision Process and the planning algorithm based on Q-learning [20].

4.1 Planning Process in MAPE Cycle

Herein, we define the a normal scenario for the planning process. In the MAPE control loop, the adaptation is needed whenever a SLA violation is detected by the monitoring component. The LAM that triggered the detection will perform the analysis to estimate the impact level based on the pre-defined strategies.

In the initial cycle of adaptation, the analysis only provides the internal impact level information, since the planning has not yet been executed. This internal impact becomes the core input to the planning component. In the planning stage, a pre-processing of Q-value will be executed followed by the learning process. The aim is to identify the appropriate adaptation strategy which can minimize the impact. Once the strategy is determined, it is exchanged with other LAMs and the respective clouds. This is essential to enable other LAMs within the same cloud as well as other clouds to assess the impact in their environment. The exchange mechanism within the same cloud environment may utilize the sharing concept, which means all the estimated impact information will be made public by storing them in the global repository. A synchronization mechanism is needed to ensure the data consistency [5]. Meanwhile, the exchange mechanism between clouds may utilize a message-oriented middleware [8]. Upon receiving the external impact information, the LAM that initiated the adaptation process will replan the strategy aiming at the minimization of the impact level. If the expected strategy is found, then the LAM will notify the others to execute either concurrently or in a coordinated form. Otherwise, re-exchanging the information may be required within a certain period.

In the following section, we focus on modeling the planning for the k-MAPE cycle where k refers to any stage in the cycle.

4.2 Planning Problem Modeling

The planning problem is modeled within the context of a LAM. It consists of a set of elements, namely, the services, the states, the actions, the impact values, the Q-value and the policy. The details are provided as below.

Services The services refer to the set of functionalities associated to the given layer. For instance, a service at platform layer is meant to control the execution of application.

The services are represented by a matrix $LE : L \times E$, where L is a set of layers and E is a set the services for each layer. Therefore, the services managed by a LAM can be represented as $E_l \in LE$, where l is a specific layer and E represents all services associated to layer l. It is assumed that $e_0 \in E_l$ and $e_0 \in E_k$ is associated to the same application.

States The states refer to the condition of a specific service e in a specific layer l. The condition is referred to the non-functional requirement satisfactions. For instance, a state of the application layer's LAM could be "service A's availability is satisfied and its cost is violated".

In this model, the states of service e is a finite number which determines the limit of the adaptation cycle. The states are variables to be assigned with the satisfaction status, such as *satisfied* and *violated*. In the context of multiple LAMs, the states are represented as a matrix $SS : L \times E \times S$ where L is a set of layers, E is a set of services for each layer and S is a set of finite states for each service. Therefore, the states for a LAM can be represented as $S_{lE} \in SS$, where l is a specific layer and E represents all services associated to the layer.

We assume that the states are in a sequence order $(s_0, \ldots, s_n)$ with $s_i \in S_{lE}$, where the first state variable s_0 determines the initial state and the last state variable s_n the final state. The initial state should contain the violated status.

Actions The actions refer to the possible adaptation actions that can be executed for the respective layer. For instance, adding or migrating a new virtual machine instance (infrastructure), updating or redeploying a Web server (platform), and reselecting different component services on the basis of the QoS properties (application).

In the context of multiple LAMs, the actions are represented as a matrix $AA : L \times E \times S$ where L, E, and S have been already defined. Therefore, the set of actions for a LAM can be represented as $A_{lES} \in AA$, where l is a specific layer, E represents all services associated to the layer, and S represents all states for each service E.

Impact Values In this paper, the impact represents the consequence of performing an adaptation. The source of the impact can be either internal or external. The internal source refers to the cross-layer impact within a cloud, while the external source refers to the multi-cloud impact.

The impact level can be of two types: positive or negative. A positive impact means the entire service-based application in the multi-cloud setting can recover from a problematic condition, e.g., a SLA violation, while the negative impact is the opposite condition.

In this work, the impact values represent the impact level (positive or negative) which may be obtained from different sources (internal and external) and is associated to the SLA violation. The values are in the range $[0, 1]$, where 0 means absolutely positive impact and 1 means absolutely negative impact.

The impact values are kept in two matrixes, defined as $IL : L \times L \times E \times A$ for the cross-layer impact and $IC : C \times C \times A$, where $C : L \times E$. We define an impact function $imp(c, l, e, a)$ to compute the total impact of action a in relation to service e at layer l and cloud c. The impact function is formulated as follows:

$$imp(c, l, e, a) = \sum_{l=0,(l,k)\in L}^{l=m} y_{lkea} + \sum_{c=0,(c,z)\in C,c\neq z}^{z=n} x_{cza} \tag{1}$$

In Equation 1, the first term of the sum represents the cross-layer impact and the second term the multi-cloud impact. y_{lkea} is an element of matrix IL and x_{cza} an element of matrix IC. The parameters k and z represent the layer and the cloud that initiates the adaptation, respectively, while l represents the cross-layer relation and c the multi-cloud relation. Meanwhile, e denotes the service and a indicates the action under consideration.

Q-value The Q-value refers to the expected utility value of taking an action in a given state. It is essential to support the Q-learning algorithm [20], in particular to select the optimal action for the next state.

The Q-value is represented as a matrix $Q : L \times E \times S \times A$ where L, E, S and A have been already defined. A single Q-value can be represented as q_{lesa}, which represents the value for the adaptation action a at layer l of service e with state s.

The Q-value is obtained by a Q-function $Q(s_{le}, a_{les})$ which aims at estimating the value of taking action a_{les} at state s_{le}. The formulation is as follows:

$$Q(s_{le}, a_{les}) = q_{lesa} + \partial[imp(c, l, e, a) + \gamma \min_{a'} Q(s'_{le}, a'_{les}) - q_{lesa}] \tag{2}$$

In Equation 2, ∂ and γ are the learning rate and the discount factor respectively, where $0 \leq \partial, \gamma \leq 1$. Setting a higher value for the learning rate results in a quicker learning, while a higher value for the discount rate determines a less worth of future reward than immediate reward.

Policy The policy refers to the generated plan, which comprises a set of appropriate actions from the initial state to the final state. It is generated from the learning process by selecting the best action based on the estimation of Q-value. It is formulated as follows:

$$\pi(s_{lE}) = \arg\min_{a} Q(s_{lE}, a_{lEs}) \tag{3}$$

In Equation 3, the best action is selected by considering the minimum Q-value for each iteration in the Q-learning process. Observe that we take the minimum value rather than the maximum one due to the impact level definition which is part of the Q-value estimation. Technically, a lower Q-value represents a lower impact level, which means a positive impact.

4.3 Planning Algorithm

Algorithm We present the planning algorithm based on Q-learning by assuming that some inputs are made available. The algorithm requires a set of input parameters, namely, the states, the actions, and the impact values related to the respective layer. The algorithm outcome is an optimal policy which contains a set of optimal actions.

```
Require: $(E_l, S_{lE}, A_{lES}, Y_{lLEA} X_{lCEA})$
Ensure: $\pi(s_{lE})$
1: set arbitrary for $Q$, $\partial$ and $\gamma$
2: for each $e \in E_l$ do
3:     for each $s \in S_{lE}$ do
4:         select $a \in A_{lES}$ based on Equation 3
5:         get $l \leftarrow Y_{lLEA}$ and $c \leftarrow X_{lCEA}$
6:         get total impact based on Equation 1
7:         estimate $Q$ based on Equation 2
8:         assign $\pi(s_{lE}) \leftarrow a$
9:     end for
10: end for
```

Algorithm 1: LAM's planning algorithm

The first step is to initialize the Q-value matrix, the learning rate ∂, and the discount factor γ. Then, a set of policies will be generated for each $e \in E$ in the respective layer l. Each service is associated with a finite set of states. Therefore, the policies to be generated are meant for each state of each service. The process of generating a policy begins with selecting an action on the basis of Equation 3. Then, the total impact level is computed according to Equation 1. After that, the Q-value is estimated based on Equation 2. The selected action is then taken as part of the policy. This process is repeated until the policies for all services have been obtained.

5 Evaluation

In this section, we present an initial simulation-based evaluation of the proposed decentralized planning for IoT applications executed in a multi-cloud environment. The goal is to show the effectiveness of the proposed approach in producing a plan which minimizes the overall (cross-layer and multi-cloud) negative impacts. In addition, we present a sensitivity analysis to support our confidence of the simulated data.

5.1 Simulation Setup

We implemented a simplified Java-based multithreaded simulator of the decentralized planning scenario. The main thread acts as communicator, while the other threads represent LAMs and focus only on the planning stage; they receive a set of data inputs to produce the appropriate plans. The core data inputs are the impact and the initial Q-value matrixes, which are randomly generated. Table 1 shows the main simulation parameters and their basic setting. The simulation experiments were conducted on a HP Compaq 6005 PC equipped with 2.80 GHz AMD Phenom®II X4 B93 processor, 4GB of RAM, and Windows 7 Professional.

Table 1. Main simulation parameters and their basic setting

Parameter	Value
Number of clouds	2
Number of layers per cloud	3
Number of services per layer	100 to 500
Number of states per service	5
Number of actions per service	5
Learning rate for Greedy strategy	0.5
Discount factor for Greedy strategy	0.9

5.2 Effectiveness Analysis

Overview We evaluate the effectiveness of the proposed decentralized planning approach by analyzing the average impact level obtained through the optimal plan driven by two strategies, namely *Greedy* and *Random* strategy. The Greedy strategy represents the adaptation plan that is determined by utilizing the cross-layer and multi-cloud impact information as discussed in the previous section. On the other hand, the Random strategy represents a randomly generated adaptation plan and is therefore used for comparison. The average impact level data has been obtained by simulating the multiple LAM planning stages for both strategies.

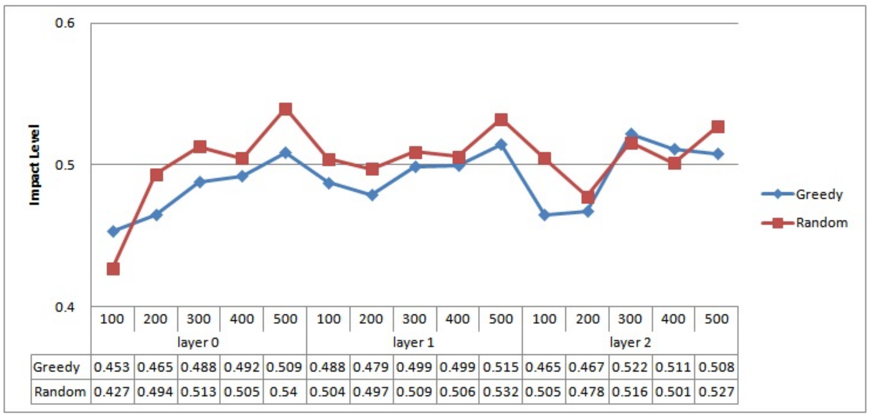
Procedure There were 5 simulation cycles for each strategy, where the cycles differed in the number of services, namely, 100, 200, 300, 400, and 500 services. The other simulation parameters shown in Table 1 were set identically. The simulation begun with the generation of the impact values and initial Q-value matrixes. This information was loaded into the main thread and distributed to the respective threads (LAM's layer). Each thread performed the planning process based on the LAM's planning algorithm to produce the appropriate adaptation plan.

We stored the generated plans for both strategies and for each simulation cycle and used them to calculate the average impact level achieved by each strategy in every simulation cycle. Finally, we normalized the collected average impact level data to obtain a value within the range $[0, 1]$, where 0 represents the most positive impact and 1 the most negative one. The normalization was computed based on the following formula:

$$av_{\phi} = \frac{av - im_{\min}}{im_{\max} - im_{\min}} \tag{4}$$

In Equation 4, av is the average total impact based for n services of a specific layer, $im_{\min}$ is the minimum impact value, $im_{\max}$ is the maximum impact value, and av_{ϕ} is the resulting normalized value.

Results and Findings Figure 2 shows the average impact level values. The x-axis represents the LAM layer as well as the simulation cycle; the y-axis represents the impact level. From the table within Figure 2, we can see that the



	100	200	300	400	500	100	200	300	400	500	100	200	300	400	500
	layer 0					layer 1					layer 2				
Greedy	0.453	0.465	0.488	0.492	0.509	0.488	0.479	0.499	0.499	0.515	0.465	0.467	0.522	0.511	0.508
Random	0.427	0.494	0.513	0.505	0.54	0.504	0.497	0.509	0.506	0.532	0.505	0.478	0.516	0.501	0.527

Fig. 2. Impact level for Greedy and Random strategies with respect to the LAM's layer and number of services

Greedy strategy, which considers multiple impact information, contributes to a significant lower impact when the plan is executed at layer 0 except for 100 services. At layer 1, the generated plan based on the Greedy strategy contributes to a significant lower impact for each simulation cycle. At layer 2, the Greedy strategy contributes to a significant lower impact for 100, 200 and 400 services. Overall, the chart shows that the Greedy strategy contributes to a lower impact level as compared to the Random strategy. Furthermore, most of the impact levels are below 0.5 except for layer 2. These results indicate that the generated plan based on the Greedy strategy can potentially reduce the negative impact on the cross-layer and multi-cloud environment.

5.3 Sensitivity Analysis

Procedure We now present a sensitivity of the planning approach to show that the simulated data were not dependent on the simulation settings used in the previous section and to confirm our confidence on the results. To perform the sensitivity analysis, we have chosen three parameters by focusing only on the effect at layer 0 (due to space constraints) and the Greedy strategy, as reported in Table 2. As shown in the table, a set of base values were identified, together with their low and high values. The base values were used to compute the impact level for the benchmark. The low and high values were used to compute the variation of the impact level. The simulations were carried out for each type of values; base, low and high. The generated plans were recorded and used to calculate the normalized average impact level as in Equation 4.

Table 2. Configuration setting for sensitivity analysis

Factors	Base Value	Low Value	High Value
Number of services	300	100	500
Number of states	5	3	8
Learning rate	0.5	0.1	0.9

Results and Findings Figure 3 shows the average impact level generated for each parameter. The left bars represent the average impact level due to the low values, while the right bars represent the average impact level due to the high values. The line in the middle represents the impact level of the base values.

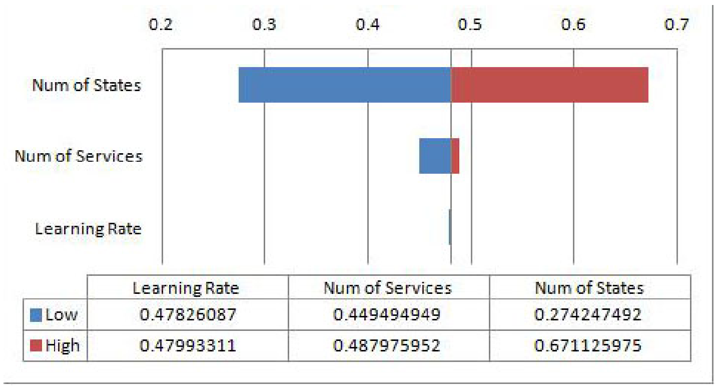
The top bar shows the variation of impact level associated to the number of states. Setting the low value of state number caused the impact level to reduce to 0.27 from the base impact level. Meanwhile, setting the high value of state number caused the impact level to increase to 0.67 from the base impact level. The variation of impact level indicates the values within -0.5 and +0.5 from the base impact level.

Very small variations are shown for the impact level related to the number of services (middle bar) and learning rate (bottom bar). Setting to low value for both parameters caused the impact level to reduce to a value within -0.5 from the base impact level. Meanwhile, setting to high value caused the impact level to increase to a value within +0.5 from the base impact level.

In comparison, a slightly higher variation related to the change of state numbers is expected since the states represent the lifecycle of the adaptation process. The variations shown in Figure 3 also supports our confidence level of 95% that the simulated impact level is expected to variate within the range of -0.5 to +0.5.

6 Related Work

Learning-based approach has been applied in a few works to support the decision making of self-adaptive systems. The work by Elkhodary et al. [7] proposed a learning cycle for understanding the impact caused by an adaptation action



	Learning Rate	Num of Services	Num of States
🟦 Low	0.47826087	0.449494949	0.274247492
🟥 High	0.47993311	0.487975952	0.671125975

Fig. 3. Sensitivity analysis of configuration change for layer 0

against the system's goals. Sykes et al. [19] applied a learning-based approach to learn the appropriate rules for planning at the goal management layer of multi-tier adaptive systems. In comparison, our work is concerned with the application of the reinforcement learning-based approach since it is suitable to deal with incomplete or partial information of the environment.

Kim et al. [13] proposed a reinforcement learning-based approach for planning the reconfiguration architecture of a managed system. Panerati et al. [16] adopted the reinforcement learning-based approach to enable self-adaptive resource allocation. Although the core technique is similar, we consider an additional factor which is the complexity of the learning approach whenever multiple multiple adaptation managers are involved.

The more recent work by Panerati et al. [17] applied a reinforcement learning-based approach to determine the appropriate configuration for coordinating multiple autonomic managers. In this context, we share the same problem motivation as addressed in our previous work [10]. The key idea is to have a centralized or global manager to control the adaptation of individual or local managers. Differently, in this work we apply the reinforcement learning-based approach for the decentralized autonomic managers where there is no centralized controller.

A collaborative reinforcement learning approach has been presented by Dowling et al. [6] to address the optimization problem for decentralized-based self-adaptive systems. This work emphasizes the need of exchanging the information to enable a collaborative learning process of multiple components. Similarly, our approach requires the exchanging concept between adaptation managers. To differentiate with this work, our paper also provides the details on how the individual manager performs the planning by modeling the reward function as a cross-layer and multi-cloud impact level.

We also mention the decentralized self-adaptation mechanism in the cloud context using market-based heuristics [15]. It focused on the service selection problem at the application layer, while we are concerned with a planning problem at different layers in the cloud environment.

Based on the works in literature that we know, we address the gap in providing a reinforcement learning-based planning approach for decentralized self-adaptive systems in a multi-cloud environment.

7 Conclusions and Future Work

In this paper, we presented an architectural model for decentralized self-adaptation of IoT applications operating in a multi-cloud environment. We also proposed a decentralized planning approach that aims at reducing the negative impact of adaptation actions. The evaluation study we conducted supports the fact that the proposed approach is reasonably effective in reducing the negative impact.

Our future work aims mainly at addressing two issues. First, a support for the decentralized planning approach to cater a conflicting problem among LAMs. This issue is essential since each LAM has the capability to make its own decision which can potentially affect the other LAMs. Therefore, an approach is needed to synchronize the generated plans which can thus benefit between each other, especially in the multi-cloud environment. Second, we plan an experimental study to understand the performance differences among decentralized control patterns, in particular the master-slave and information sharing patterns [21]. This understanding can assist in establishing an efficient and effective self-management capability that can be deployed in a real multi-cloud environment.

Acknowledgements This work is supported by the Fundamental Research Grant Scheme (600-RMI/FRGS 5/3 (164/2013)) funded by the Ministry of Higher Education Malaysia (MOHE) and Universiti Teknologi MARA (UiTM), Malaysia.

V. Cardellini also acknowledges the support of the European ICT COST Action IC1304 Autonomous Control for a Reliable Internet of Services (ACROSS).

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proc. of 1st Workshop on Mobile Cloud Computing. pp. 13–16. MCC '12 (2012)
2. Boyle, D., Yates, D., Yeatman, E.: Urban sensor data streams: London 2013. IEEE Internet Computing 17(6), 12–20 (Nov 2013)
3. Celino, I., Kotoulas, S.: Smart cities [guest editors' introduction]. IEEE Internet Computing 17(6), 8–11 (Nov 2013)
4. Chan, K., Bishop, J., Steyn, J., Baresi, L., Guinea, S.: A Fault Taxonomy for Web Service Composition. In: Service-Oriented Computing - ICSOC 2007 Workshops, vol. 4907, pp. 363–375. Springer (2009)
5. De Oliveira, F., Ledoux, T., Sharrock, R.: A framework for the coordination of multiple autonomic managers in cloud environments. In: Proc. of 7th Int'l Conf. on Self-Adaptive and Self-Organizing Systems. pp. 179–188. SASO '13 (Sep 2013)

6. Dowling, J., Cahill, V.: Self-managed decentralised systems using k-components and collaborative reinforcement learning. In: Proc. of 1st ACM SIGSOFT Workshop on Self-managed Systems. pp. 39–43. WOSS '04 (2004)
7. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: A framework for engineering self-tuning self-adaptive software systems. In: Proc. of 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. pp. 7–16. FSE '10 (2010)
8. Fazio, M., Celesti, A., Villari, M.: Design of a message-oriented middleware for cooperating clouds. In: Advances in Service-Oriented and Cloud Computing, CCIS, vol. 393, pp. 25–36. Springer (2013)
9. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems 29(7), 1645–1660 (2013)
10. Ismail, A., Cardellini, V.: Towards self-adaptation planning for complex service-based systems. In: Service-Oriented Computing Workshops, LNCS, vol. 8377, pp. 432–444. Springer (2014)
11. Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M., Hamida, A.: Service-oriented middleware for the future internet: state of the art and research directions. Journal of Internet Services and Applications 2(1), 23–45 (2011)
12. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
13. Kim, D., Park, S.: Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In: Proc. of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 76–85. SEAMS '09 (May 2009)
14. Kostakos, V., Ojala, T., Juntunen, T.: Traffic in the smart city: Exploring city-wide sensing for traffic control center augmentation. IEEE Internet Computing 17(6), 22–29 (2013)
15. Nallur, V., Bahsoon, R.: A decentralized self-adaptation mechanism for service-based applications in the cloud. IEEE Trans. Softw. Eng. 39(5), 591–612 (2013)
16. Panerati, J., Sironi, F., Carminati, M., Maggio, M., Beltrame, G., Gmytrasiewicz, P., Sciuto, D., Santambrogio, M.: On self-adaptive resource allocation through reinforcement learning. In: Proc. of 2013 NASA/ESA Conf. on Adaptive Hardware and Systems. pp. 23–30. AHS '13 (Jun 2013)
17. Panerati, J., Maggio, M., Carminati, M., Sironi, F., Triverio, M., Santambrogio, M.D.: Coordination of independent loops in self-adaptive systems. ACM Trans. Reconfigurable Technol. Syst. 7(2), 12:1–12:16 (2014)
18. Sterritt, R., Parashar, M., Tianfield, H., Unland, R.: A concise introduction to autonomic computing. Adv. Eng. Inform. 19(3), 181–187 (2005)
19. Sykes, D., Corapi, D., Magee, J., Kramer, J., Russo, A., Inoue, K.: Learning revised models for planning in adaptive systems. In: Proc. of 2013 Int'l Conf. on Software Engineering. pp. 63–71. ICSE '13 (2013)
20. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8(3-4), 279–292 (1992)
21. Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Göschka, K.M.: On patterns for decentralized control in self-adaptive systems. In: Software Engineering for Self-Adaptive Systems II, LNCS, vol. 7475, pp. 76–107. Springer (2013)