Learning a Goal-Oriented Model for Energy Efficient Adaptive Applications in Data Centers

Monica Vitali[a,*], Barbara Pernici[a], Una-May O'Reilly[b]

[a] *Politecnico di Milano, piazza Leonardo da Vinci 32, 20133 Milan, Italy*
[b] *Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307*

Abstract

This work has been motivated by the growing demand of energy coming from the Information Technology (IT) sector. We propose a goal-oriented approach where the state of the system is assessed using a set of indicators. These indicators are evaluated against thresholds that are used as goals of our system. We propose a self-adaptive context-aware framework, where we learn both the relations existing between the indicators and the effect of the available actions over the indicators state. The system is also able to respond to changes in the environment, keeping these relations updated to the current situation. Results have shown that the proposed methodology is able to create a network of relations between indicators and to propose an effective set of repair actions to contrast suboptimal states of the data center. The proposed framework is an important tool for assisting the system administrator in the management of a data center oriented towards Energy Efficiency (EE), showing him the connections occurring between the sometimes contrasting goals of the system and suggesting the most likely successful repair action(s) to improve the system state, both in terms of EE and Quality of Service (QoS).

Keywords: Energy efficiency, data centers, adaptive service, goal-oriented adaptation
PACS: 89.20.Ff, 07.05.Mh
2008 MSC: 97R50, 90B50

1. Introduction

The general interest about environmental issues is continuously growing. One of the main contributors to pollution and energy consumption is Information Technology (IT). The impact of technology and its growing rate is evident at the everyday scale, and even more significant at a larger scale.

This work has been motivated by this recent attention towards Green IT and Energy Efficiency (EE) [1][2][3] from different perspectives (usage of resources, applications, and networks), and by the growing demand of energy coming from IT. This demand is comparable with the one of the major countries and it is expected to keep growing in the next years [4]. In this work we focus on efficiency considering the data center level, with a special attention to the contribution of Service Oriented Architectures (SOAs) to energy consumption and performance. In SOAs, applications hosted in the servers are services that can be represented by processes. Each service is decomposed in a set of activities connected in a workflow. Recently, virtualization and cloud computing have changed the way resources are handled in a data center, allowing a more dynamic redistribution of computational resources among the several applications hosted by the data center or the cloud. As a consequence, the waste of resources is (potentially) reduced.

We face the problem of EE by proposing a comprehensive approach. The approach includes the definition of the goals for the organization, the selection of metrics for the assessment of the system state, and the definition of techniques for the improvement of the system. In order to do so, we propose the use of a goal-oriented approach.

The context in which we are going to implement our goal-oriented approach for EE is a data center composed of several servers, each one with different features in terms of amount and kind of components. We focus on the application perspective for improving efficiency. For this reason we are not considering issues related to cooling or other facilities, but we look in details at applications and their IT infrastructure. We consider that the data center uses virtualization for hosting its applications and we consider that each single activity runs on a dedicated Virtual Machine (VM). A monitor-

*Corresponding author, (+39) 0223993424

Email addresses: monica.vitali@polimi.it (Monica Vitali), barbara.pernici@polimi.it (Barbara Pernici), unamay@csail.mit.edu (Una-May O'Reilly)

ing system is available for monitoring the system behavior with sensors that can work at several levels: data center level, server level, Virtual Machine level, and application level. Different kinds of monitoring systems can be employed (e.g. Zabbix[1] and Nagios[2]), as they do not have any influence on the approach.

The purpose of this work is to design a model which allows the described environment to improve its behavior in terms of EE while respecting constraints related to the Quality of Service (QoS) required by the users. The tradeoff between these two perspectives is a challenging topic nowadays and has been faced in many research papers [5][6][7]. However, most of the approaches underestimate the complexity of the environment they describe. A data center, especially modern data centers employing techniques such as virtualization and consolidation, is a dynamic environment where the number of applications and their configuration and deployment are continuously and rapidly changing. Also, external factors can impact on the performance and efficiency of the deployed applications, due to servers overload or to service unavailability. In such an environment, adaptation techniques requiring human intervention can be applied but at the cost of a lack of reactivity in the system. An automatic approach is needed to maximize the performance of the data center and to optimize its efficiency in terms of energy consumption by applying a set of adaptation techniques. In the literature, several techniques for adaptation are described and their effectiveness is proved. Starting from this knowledge we propose a methodology for modeling adaptation in data centers while automatically reacting to its dynamic modifications.

The main contributions of this work can be summarized as follows:

- **Definition and learning of a goal-oriented model for efficiency management** An adaptation approach is proposed to learn a goal-oriented model for efficiency management. The aim of the model is to provide an adaptive environment able to automatically react to suboptimal situations by enacting adaptation strategies that impact directly or indirectly over efficiency and quality. The model is not derived from the knowledge of an expert, but it is build starting from available data using machine learning techniques. It is also able to automatically respond to modifications in a dynamic environment.

[1]http://www.zabbix.com/
[2]http://www.nagios.org/

- **Representation and learning of indicators relations** We express the relations occurring among the various metrics used for monitoring the system state. These relations are very important for predicting the future behavior of the system and for conducting what-if analysis.
- **Adaptive strategy selection** Effects of actions over EE and QoS are automatically learned providing an adaptive and dynamic environment.

In order to realize the described model, we used a goal-driven approach where two layers are defined: a goal layer and a treatment layer. Goals correspond to the desired values for a set of metrics collected through the monitoring system, such as the energy consumption of the data center or of the physical machine and the QoS provided measured in terms of response time or throughput. Relations among metrics are represented using a Bayesian Network (BN) and automatically learned through the employment of machine learning techniques, such as a modification of the Max-Min Hill Climbing Algorithm (MMHC) [8] for learning links and the Maximum a Priori Estimation (MAP) for learning parameters [9]. Results have shown that the proposed methodology is able to learn the whole model from a set of monitored data. Given the model, it is possible to decide which is the best repair action in order to bring the system in a new configuration as near as possible to the optimal one, given a specific context for the system. Repair strategies are contained in the treatment layer and examples are migration, server switch off/on, and VM resources reconfiguration, or combinations of them. The effect of actions over the system are learned using the Multi Armed Bandit Selection (MABS) paradigm [10] and properly adapting the Adaptive Operator Selection (AOS) algorithm [11]. Once learned, the model can be used to prescribe the best adaptation action given a context.

The rest of this work is organized as follows. In Sect. 2 we analyze other approaches using adaptive models to manage EE and/or QoS of a data center. Then we propose our approach in Sect. 3. In Sect. 4 and Sect. 5 we analyze in details the two layers of the proposed model and show how it can be used to improve the system state in Sect. 6. Finally we discuss some experimental results (Sect. 7) and future work (Sect. 8).

2. State of the Art

In this work we focus on EE from the application perspective by adopting the SOA approach, where web services are supported and provided in terms

of Business Processes (BPs). In a SOA, interactions between several services and the combination in which they are used inside a process can be easily modified, by providing a flexible architecture.

One of the issues that have to be faced is the dynamism of the environment where the availability and quality of the offered services can change due to several external and unpredictable factors, such as a modification of the load of a server or the unavailability of a VM. In this case, the system needs to be modified to respond to these modifications in order to respect the contract agreed with the user through the Service Level Agreement (SLA).

The issue of adaptive services has been one of the main goals of the S-Cube Network[3], which developed adaptation concepts and techniques for SOAs including proactive adaptation and predictive monitoring techniques [12]. Another approach is discussed in [13], where authors propose a QoS based Web Service framework, WebQ, which enables the selection of an appropriate service for each of the tasks in the underlying workflow, and dynamically refines existing services by keeping high the performance level. In the present work, we consider adaptation at the application level, considering adaptive actions similar to the one described in [12] and [13], but we propose a more comprehensive approach considering adaptation also at other levels (infrastructure and virtual level adaptation) and by considering EE in conjunction with QoS.

The framework proposed in [6] aims at avoiding Key Performance Indicators (KPIs) violation considering influential factors between metrics. The approach is based on the construction of a dependency tree learned using machine learning techniques. Adaptation actions are described by their positive and negative effects on metrics. To select the best strategy, the first step consists in selecting the set of KPIs for which violation has to be prevented. Once metrics are identified, influential factors are analyzed using the dependency tree, by retrieving the desired state for each of the metrics in order to avoid violations. The next step consists in selecting the set of actions that improve each violated metric without negatively affecting other metrics. The resulted set of actions is combined using a Cartesian combination and ranked considering negative effects over other metrics (the less the better). This work shows the importance of building a model to provide adaptation in SOAs by describing effects of actions over indicators and by stressing the importance of being aware of how indicators influence each

[3]http://www.s-cube-network.eu/

other. The motivations of the described approach are similar to ours and also the methodology has some significant similarities even if different techniques are employed. However, some limitations can be highlighted. First of all, the influential factors existing among indicators and the effects of actions over indicators are statically and manually defined by the user. We think that this is a first step towards adaptation, but not enough in a dynamic environment such as a modern data center. Also, adding a new indicator or a new action requires to build again the model from scratch, while we want to create a mechanism that can automatically react to this kind of modifications. Finally, the approach considers only the application level, without exploring the influence of indicators and repair actions defined at the infrastructure and at the virtualization level, considered in our approach.

A goal-oriented approach for EE has been also proposed in [14], where processes are represented using Goal Requirement Language (GRL) to improve EE. In this approach, EE goals are modeled as non functional requirements. A goal is defined as the objective that the organization wants to achieve which can be a business and a system goal. Goals related to non functional requirements are called soft-goals, while tasks are different strategies that can be used to achieve a goal, which can be decomposed in subtasks. Different kinds of links are used to relate all the elements of the model, expressing the kind and the strength of the relation. This approach allows the authors to deal with performance and energy issues at the same time, even if the main focus is on performance. The approach starts from a context-based model proposed in [15] where three layers are considered for modeling a goal-driven adaptive approach, namely treatment, event, and goal. In this approach the focus is on requirement engineering instead of EE. The main limitation of these approaches is that they do not manage the dynamism of the modeled environment, proposing a static model created by experts.

Artificial intelligence and machine learning techniques can give an important contribution in the management of complex and dynamic environments such as a data center, and an opportunity to implement best practices and improve EE. One possible application is the detection of hidden relations among variables. A recent study conducted by Google [16] employs a neural network framework that learns from monitored data to model plant performance and predict Power Usage Effectiveness (PUE). This approach identifies interactions existing among components, automatically searching for patterns in the monitored data. In the specific case, a neural network is used to predict the outcome of some modifications over the PUE, but the

technique is general and can be used to predict more and different variables enabling what-if analysis for several aspects of the data center.

Another application of artificial intelligence consists in a support to decision making. In [7] authors use Dynamic Fuzzy Q-Learning (DFQL) to optimize energy-performance tradeoff. Q-Learning is a model free reinforcement learning algorithm. It computes state-action qualities as the reward of taking action a in state s, by partitioning continuous states and creating fuzzy rules associated with a set of actions. Each action is associated to a quality factor which is continuously updated. The approach combines exploration and exploitation using double ε-Greedy algorithm. The algorithm considers CPU performance, disk, RAM and network, splitting the work among a local manager (which decides when to migrate a VM) and a global manager (issuing migrations and power state modifications). A similar issue is faced in [17], where authors propose an algorithm for host overload detection by maximizing the mean inter-migration time under the specified QoS goal based on a Markov chain model. In order to be effective, the algorithm needs to work with stationary workloads. The QoS metric is defined as Overload Time Fraction (OTF) measuring the percentage of time in which the host has been overloaded. Only CPU is considered and the state space is discrete and defined in terms of CPU usage intervals. A probability matrix is computed in order to model the transition among states.

Optimizing based on the tradeoff between EE and QoS is also the goal of the approach proposed in [18], using a Constrained Markov Decision Process (CMDP) for adaptation. Repair strategies are limited to frequency scaling and turning on/off of servers. The model defines a tuple $\{X, A, P, c, d\}$ where X is the set of possible states, $A(x)$ is the set of actions available at state X, P is a transition probability from a state to another one applying action a, c is an immediate cost that has to be minimized (energy), and d is a vector of costs related to a set of constraints (QoS). Some assumptions are made when defining the model (e.g., all servers are of the same type and CPU is the bottleneck). The performance of the system is measured using request time and adjustments are activated by the violation of a threshold over the jobs queue length. The algorithm for adaptation tries to improve the situation by increasing CPU frequency. If frequency is higher than a given threshold, a new server is activated. The approach has been evaluated using simulation.

According to what has been discussed, machine learning has proven to be an important resource when dealing with modeling dynamic and complex environments. This gave us a good motivation to employ some of the

techniques described (Decision Support Systems, Reinforcement Learning, Markov Decision Processes) to reach our goal.

In this work, we propose a goal-driven approach to EE considering also a set of QoS metrics as goals of our system. We propose a model able to automatically discover hidden relations among variables of the system at different levels (application, virtual, and physical), and to automatically learn the effect of adaptation actions. Contrary to other approaches discussed in this section, the number of states of the system does not need to be defined a-priori. Also, we consider a non-binding set of metrics for measuring both EE and QoS. Moreover, we consider a more complete set of adaptation actions, modifying the action selection according to the current context of the system. Finally, the proposed approach does not require any constraint about the physical components employed, allowing the usage of heterogeneous servers.

3. A Goal-Oriented Model for Green Data Centers

The model adopted as a basis for our approach is a goal-driven model for EE. It is obtained from a simplification of the approach proposed in [19], following the path started in [20]. In [19], authors present a model for risk assessment where three layers are depicted: a goal layer where the goals and their relations are described, an event layer where events that can impact over the goals are described, and a treatment layer where a set of actions and their ability to mitigate the effect of events are modeled. In [20], the authors apply the model to a different field, by focusing on efficiency and QoS in data centers, focusing on the event layer. Goals are represented by a set of metrics for which the value has to be maintained inside a given interval. When a violation occurs, an event is raised. The adaptation process is composed of two phases: the event creator and the adaptation strategy selector. In the first phase, the state of the system is analyzed in order to identify the violations that can be considered significant. In fact, temporary violations can be ignored since they can automatically recover without enacting any strategy. The aim of the approach proposed in [20] is to identify which events are significant and which are not.

The aim of the proposed approach is to provide a model depicting interactions in the system, which can be used to lead the system towards a better configuration through adaptation. The goal-oriented model, depicted in Fig. 1, is composed of two layers:

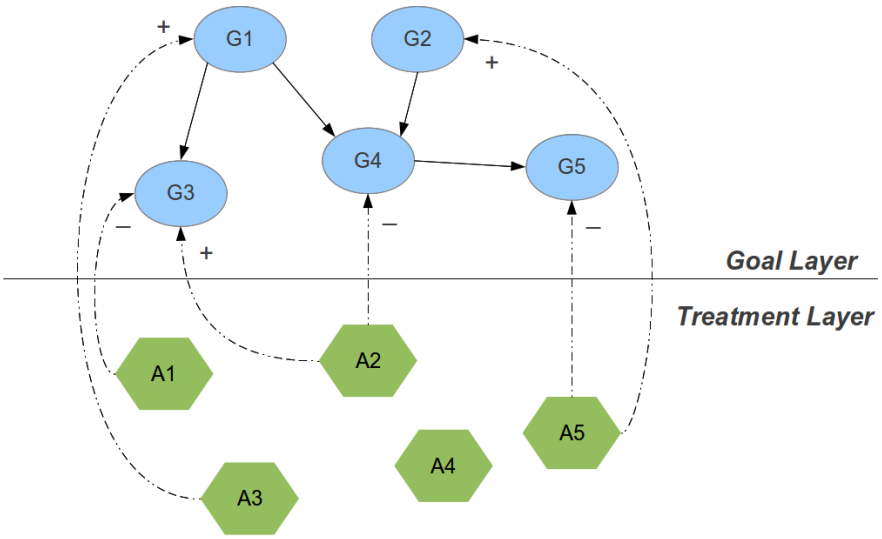


Figure 1: The goal-driven approach for Energy Efficiency

- Goal Layer: this layer is composed by the goals of the system and the relations existing among them;
- Treatment Layer: it is composed of a set of repair actions that can be used to react to suboptimal situations.

Repair strategy are enacted when the violation of a threshold over the indicators value occurs. Even if not explicitly shown, we are referring to the event creator described in [20] to consider only relevant violations, while ignoring temporary ones.

In the proposed approach, goals are represented through satisfaction of a set of metrics used to monitor the system state. In the IEEE standard glossary of software engineering terminology published in 1990 [21], metrics are defined as follows:

> *"A **metric** is a quantitative measure of the degree to which a system, component, or process possesses a given attribute. A **qualitative metric** is a quantitative measure of the degree to which an item possesses a given quality attribute."*

According to this definition, the qualitative metrics are the parameters that have to be computed from measurements of the system behavior. In our

approach, the metrics of interest are related to EE and QoS of the data center. Examples are response time, throughput, energy consumption, VM and host resources utilization. Finally, the term indicator is defined as follows [21]:

> *"An **indicator** is a variable that can be set to a prescribed state based on the results of a process or the occurrence of a specified condition."*

An indicator can be seen as an instance of a metric, applied to a specific aspect of a system and associated with a comparative term used to assess if the monitored value is in line with the prescribed one or not. Examples of indicators in our approach would be CPU utilization of a specific VM or of a specific host, both associated with a range of desirable values. Given a set of available metrics, the system administrator has to be able to choose a subset of them able to lead the system towards its goals. According to this, we define the system state as follows:

Definition 1. *The* STATE OF THE SYSTEM *at a given time t is composed of the values of all the indicators selected to lead the system towards the predefined goals at the time t:*

$$state(t) = \{I_1(t) \dots I_n(t) \dots I_N(t)\} \tag{1}$$

where $I_n(t)$ is the value of indicator n at time t and N is the number of selected metrics.

Once indicators are selected, a monitoring system has to be installed or configured accordingly to this selection. Indicators give relevant information about the actual system state and the desired state, allowing focusing attention over significant aspects that need to be improved. One or more **thresholds** are associated to each indicator:

Definition 2. *A* THRESHOLD *is a constraint or a reference value defined for an indicator, dependent on the pursued goals. A set of thresholds associated to an indicator defines the interval of allowed values for a specific goal.*

Thresholds allow the system administrator to compare the current situation with the optimal one and to understand which aspects should be improved. The satisfaction of these thresholds represents the goal layer of

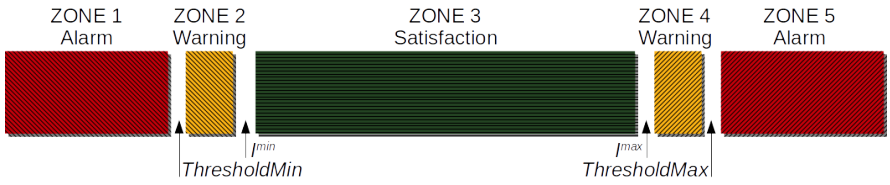


Figure 2: Indicator thresholds intervals with five states

the proposed model. In general, three zones can be defined for each indicator as previously discussed in [22]: (i) the **normal zone** identifies the values that are in the desired range; (ii) the **alarm zone** identifies values out of the desired range; (iii) the **warning zone** identifies values that are still in the normal zone but near to be in the alarm zone. In general, two thresholds need to be defined for each indicator, namely *ThresholdMin* and *ThresholdMax*, defining where the alarm zone starts. The dimension of the warning zone can be defined automatically using Eq. 2:

$$size(Warning(I_n)) = \gamma \cdot \frac{(ThresholdMax - ThresholdMin)}{2} \tag{2}$$

with $\gamma \in [0, 1]$. The value of γ depends on the strategy chosen by the designer. A value near to 1 can be used in order to adopt a proactive approach, where the problem needs to be identified before its actual verification. On the contrary, a small value can be used for a reactive behavior, reacting only when the violation is near to occur. A value from 1 to 5 can be assigned to each of the zones represented in Fig. 2, increasing left to right. This value is used for identifying the state of an indicator.

We can define the context $\mathbb{C}$ of the system as:

Definition 3. *The* CONTEXT $\mathbb{C}$ *is the state of the indicators in a given moment in which the system is evaluated. This state can be of three kinds: satisfied, violated or almost violated, corresponding to the indicators zones normal, alarm and warning.*

Relations exist between different goals and their satisfaction, since indicators are not necessarily independent from each other. That means that a goal satisfaction/violation can have effects on other goals of the system. Relations are expressed through links among goals in the model of Fig. 1.

Whenever a goal is not satisfied, something needs to be done. We indicate these treatments as *repair actions*:

Definition 4. *A* REPAIR ACTION A_l *is a modification of some aspects of the system which can be used in the attempt of improving the system state. Each action can have a positive or negative outcome over one or more indicators in the system, depending on the context* $\mathbb{C}$ *in which it is applied.*

A repair action can contribute to improve the state of the specific indicator but can also be damaging for it. This effect is depicted in the model using an annotated link between actions and indicators. The annotation on the link indicates the ability of the action of increasing ("+") or decreasing ("-") the value of the indicator when applied. If this effect is positive or negative for the indicator is evaluated according to the context. Examples of repair actions are VM reconfiguration or migration.

Starting from the defined model, we propose a methodology for managing QoS and EE in data centers from an application perspective. Modern data centers continuously evolve their configuration and the way applications are deployed. Thereby, an approach aiming at dealing with such an environment needs being able to adapt to modifications, reducing human intervention. According to this, we propose a technique to automatically learn relations among the goals of the system, represented using a BN, as described in Sect. 4. We also define available actions and automatically learn the links connecting actions and goals by proposing an adaptive approach called Adaptive Action Selection (AAS), inspired to the AOS approach (Sect. 5). The system is able to respond to changes in the environment, keeping these relations updated to better fit the current situation.

The proposed framework can be an important tool for assisting the system administrator in the management of a data center oriented towards EE. Connections among goals can be used to predict the outcome of modifications in the system and can put light over unexpected connections occurring between the sometimes contrasting goals. Moreover, action-to-goal connections enable a repair mechanism suggesting the most likely successful repair action(s) to improve the system state, both in terms of EE and QoS. The AAS algorithm is designed to dynamically update its knowledge about the actions effects, fitting a system that can frequently change in time.

4. Defining and Learning the Goal Layer

In this section we analyze the goal layer, composed of a set of indicators, described in Sect. 4.1, and of the relations existing among them (Sect. 4.2).

Table 1: Green Performance Indicators and Key Performance Indicators

Indicator	Levels	Description
Energy	all levels	The amount of energy used by a component in the system. It is measured in kilowatt-hour
CPU Usage	VM, host	The ratio between the amount of CPU used and the amount of CPU allocated
Memory Usage	VM, host	The amount of memory used as a fraction of memory allocated. It is a number between 0 and 1
Response Time	application	Time elapsed between the start and the completion of an instance of an activity. It is measured in seconds
I/O throughput	application	The number of I/O operations that each activity executes per time unit
Performance per Energy	application	The number of transactions that each activity executes per energy unit expressed in kilowatt-hour
Throughput	application	The number of operations that each activity executes per time unit
Storage Usage	application, host	The amount of disk space used as a fraction of total disk space allocated

4.1. Goal Definition

An indicator can be defined as a measure of a relevant aspect of an organization or a system, and it can be used to assess the success of an entire organization or of a specific department. The assessment is lead by the definition of strategic goals or, at a lower level, of organizational goals. The selection of a relevant set of indicators is an important and non trivial activity in the life-cycle of a system, since it requires a good understanding of what is important to the organization. Usually, performance aspects are privileged, but even the economical and green aspects should be considered. Indicators allow the organization to assess the current state of the system and to identify the key activities. They also enable the identification of potential improvements. Indicators can be defined at different granularity levels, depending on the level of detail used to collect the information that participates in their computation. Possible levels are: host, VM, and application. Values at the different levels can be computed starting from the measurements in the finest level and aggregating them along different dimensions.

Several metrics and their definition have been analyzed and discussed in

[23], and a subset of indicators is shown in Tab. 1. This subset has been selected considering the most considered metrics in the literature for measuring both EE and QoS. However, this set is only used as an example and other metrics can be considered without changing the nature of the approach.

The selected indicators allow the data center administrator to investigate the state of the system in terms of QoS and EE. In order to decide the satisfaction intervals for each indicator we referred to two main sources. The first source is the client who establishes constraints about the performance of his applications through the SLA. The second source is the Green Grid Data Center Maturity Model (DCMM) [24], a document suggesting best practices for energy efficient data centers by indicating some thresholds related to resource usage percentages. The threshold definition issue has been discussed in more details in a previous work [25].

4.2. Learning Relations among Goals

In a data center, the set of metrics used to assess EE and QoS are not independent in many cases. In some contexts, metrics can have a negative effect on others, but they can be also related by positive interactions. It means that improving the state of one indicator can improve the state of another one related to it. Also, these influences can change in time if the environment where the metrics are collected is subject to modifications. For these reasons, it seems very important to investigate about the relations between indicators. Knowing these relations has multiple advantages when trying to improve the efficiency (in terms of quality and energy) of the data center. First of all it enables an indirect fixing approach: the violation of an indicator can be fixed acting directly over the indicator, or intervening over other indicators that result to be related to it. Moreover, a what-if analysis approach is also possible: the knowledge about relations allows us to predict how the variables of the system react to a modification of an indicator state, predicting possible critical outcomes.

In this paper, we propose a methodology for automatically learning relations between indicators by automatically building a Bayesian Network representing the relations between the indicators violation and/or satisfaction, starting from the data collected by the monitoring system. Indicators are represented by the nodes of the network, while relations between indicators are represented through edges. A Conditional Probability Table (CPT) corresponds to each edge expressing the relations between each of the states of the two indicators involved. The proposed approach is used to build the

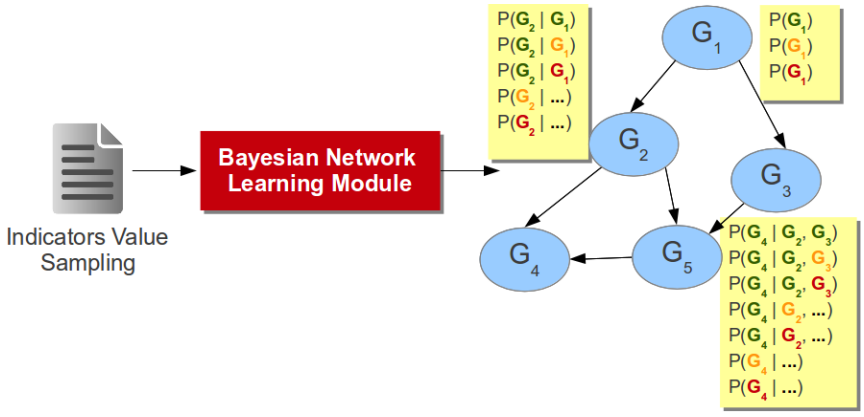


Figure 3: Bayesian Network Learning Approach

higher level of the goal oriented model represented in Fig. 1. A graphical synthesis of the approach is represented in Fig. 3.

The first step in learning the BN is to discover the relations between indicators which can be translated in learning the edges connecting variables. There are plenty of techniques in the state of the art to learn the structure of a BN from collected data [26][27][28]. However, they require discrete or Gaussian variables. The values obtained through the monitoring system are continuous variables. Transforming continuous values into discrete values applying thresholds is not a solution since it reduces the amount of information available to compute relations. The noise introduced by thresholds application in learning BN is also discussed in [9]. We propose an algorithm to learn a BN starting from continuous variables. First, we use correlations to discover relations between variables, using the Pearson product-moment correlation coefficient, by dividing the covariance for the product of the standard deviations σ as shown in Eq. 3. The covariance is a value expressing how two variables change together and is computed using the expected value function $E[X]$. Values of the correlation can be both positive and negative, and are included in the range $[-1, 1]$. A positive value indicates an increasing linear relationship between X and Y, while a negative value indicates an anti-correlation, or a decreasing linear relationship. The nearer is the correlation value to 1 (or -1), the stronger is the relation.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y} = \frac{E[(X - E[X])(Y - E[Y])]}{\sigma_X \cdot \sigma_Y} \tag{3}$$

The second step concerns learning the directions of edges. In fact, using the correlation matrix to obtain the structure of the BN is not straightforward, since correlation is a symmetric property and no information is given about the directionality of the relations. On the contrary, BNs are oriented graphs, so a direction needs to be specified for each edge in the network. In order to find the network that better describes our dataset, we apply a technique to learn directionality inspired to the approach described in [8]. Knowing the possible parent-children set for each node, obtained putting a threshold over the correlation matrix, we can apply a MMHC-like algorithm to orient the edges. The algorithm starts from an empty matrix and at each step explores all the neighbor networks, by adding, removing or reverting edges. A Directed Acyclic Graph (DAG) is considered neighbor of another Directed Acyclic Graph (DAG) if it has the same structure except for one connection that can be added, removed or reverted. All the networks obtained are scored and the one with the highest score is chosen for the next step. Scoring functions are available in literature [29] and they assign a score to a BN according to its ability to describe a training set. The MMHC algorithm is enriched using the concept of TABU list during the greedy search, following the Sparse Candidate implementation [30]. This list contains the last $\mathcal{M}$ explored structures and select the best DAG from the neighbors list that does not belong to the TABU list.

Once the structure of the BN has been learned, the model has to be completed learning CPT tables. Here, we use the Maximum a Priori Estimation, that learns parameters given a training set of data. The training set is obtained applying thresholds to the indicators effective values, since we are interested in predicting the state of indicators.

The complete network can be used to predict the state of an indicator, when it is unknown. In fact, the network is able to suggest the most likely values given a context (the state of the other indicators in our system). This property can be used in what-if analysis. Unlike other techniques introduced in Sect. 2, the model is valid for each kind of load and does not require to work with predefined stationary workloads or a homogeneous physical environment. However, the elasticity of the model depends on the conditions under which data have been collected: only collecting training data under a variable workload can generate a workload independent network.

5. Defining and Learning the Treatment Layer

The treatment layer defines a set of repair actions that can be applied to solve critical situations and to improve the state of the system. Since the environment is continuously changing, it is important to learn the impact of the actions over the indicators and to keep this information updated. After defining a set of repair actions in Sect. 5.1, we describe an algorithm for learning and updating their effect on the indicators (Sect. 5.2).

5.1. Repair Actions Definition

A repair action is a modification of the system configuration that can affect the state of one or more indicators. In this work, a repair action is described by the following set of elements: (i) *ID*, an identifier for referring to the action; (ii) *description*, a brief description of the repair action; (iii) *parameters*, the parameters needed to apply the action; (iv) *execution time*, the average time needed to apply and observe the outcome of the action; (v) *energy consumption*: the amount of energy needed to execute the action; (vi) *preconditions*, the conditions that have to be verified for applying the action.

A cost can be associated to each action. This cost is expressed through the variable c_{A_l}, which is dependent both on the execution time of the action (t_{A_l}) and on its energy cost (En_{A_l}):

$$c_{A_l} = f(t_{A_l}, En_{A_l}) \tag{4}$$

Several repair actions have been identified. The selected actions have been chosen starting from the most common strategies proposed in the state of the art for dealing with EE [2], also discussed in Sect. 2. Actions are classified into four categories: VM level, host level, application level, and general. A summary of the available actions is reported in Tab. 2. The ones that we have tested in the experiments are discussed in more details.

The actions defined at the VM level concern the management of VMs configuration, placement, and deployment.

VM Migration (A_1) This repair action moves a VM from a server to another. Parameters are the ID of the VM that has to be migrated and the ID of the server where it has to be moved. In order to be executable the selected server must have the resources requested by the VM. The amount of time required to execute migration can be approximated using models available in the state of the art, as described in [31]. Given the time required to perform migration and the computational resources involved in the process, also the

Table 2: Repair Actions

VIRTUAL MACHINE LEVEL		
ID	**Description**	**Parameters**
A_1	VM Migration	VM ID, Server ID
A_2	VM Reconfiguration - add	VM ID, resource type
A_3	VM Reconfiguration - remove	VM ID, resource type
A_4	VM Duplication	VM ID
A_5	VM Removal	VM ID
HOST LEVEL		
ID	**Description**	**Parameters**
A_6	Turn on Server	server ID
A_7	Turn off Server	server ID
A_8	Data Migration	none
A_9	CPU Frequency Scaling - increase	server ID
A_{10}	CPU Frequency Scaling - decrease	server ID
A_{11}	Storage Mode Modification - performance	disk ID
A_{12}	Storage Mode Modification - energy saving	disk ID
APPLICATION LEVEL		
ID	**Description**	**Parameters**
A_{13}	Process Work-flow Modification - skip	activity ID
A_{14}	Process Work-flow Modification - execute	activity ID
GENERAL		
A_{15}	Do Nothing	activity ID
COMPOSED ACTIONS		
ID	**Description**	**Parameters**
A_{C1}	Migrate and Turn Off	server ID
A_{C2}	Turn On and Migrate	VM ID, server ID

cost in terms of energy has to be considered. To model the energy cost we refer to the model proposed in [32].

VM Reconfiguration - add/remove (A_2, A_3) This action enables changing the amount of resources allocated to a VM according to the incoming load. Parameters are the ID of the VM that has to be modified and the kind of resource that is involved (e.g. CPU and memory). Reconfiguration is executed using fixed steps that consist in the addition/removal of a core

in case of CPU or of a fixed amount of memory otherwise. Precondition for this action is that additional resources are available on the server or that the minimal amount of resources is guaranteed for the VM (at least one core and one block of memory). In this work we consider only hot-plugging, so resources can be added without stopping the machine. This feature is available in the most popular hypervisors (e.g., VirtualBox and VMware). According to this, both execution time and energy cost of this action are limited.

At the host level, a set of actions can be defined which concern the physical level of the system by managing the servers and their components.

Turn on/off Server (A_6, A_7) This repair action turns on/off a server. It can be useful when there are not enough physical resources available for deploying a new VM or when a server is not used. The only parameter is the ID of the server. This action can require a significant, even if limited, amount of time to be executed. Also the energy consumption, due to the utilization of resources while activating/deactivating the server, has to be taken into account. In order to compute this cost, we refer again to the model proposed in [32]. For turning off a server, the precondition is that there are no VMs deployed on it. To avoid delays, this action can be replaced by putting on and off stand-by mode, mitigating performance issues while still saving a considerable amount of energy.

In some contexts, some actions are useless or even damaging if considered in isolation. As an example, the migration of a VM (action A_1 (VM migration) in Tab. 2) is expensive both in terms of energy and performance. Applying this action by itself is not convenient. However, it is useful when combined with other actions, such as A_7 (Turn off Server). For this reason, it is useful to introduce the concept of composed action defined as follows:

Definition 5. *A composed action A_{Ci} consists of a set of simple actions organized in a plan, defining the sequence in which these actions are executed:*

$$A_{Ci} = \langle A_i \rightarrow A_j \rightarrow \ldots \rightarrow A_k \rangle \tag{5}$$

where each action A_x belongs to the set of all the available actions A and the symbol $\rightarrow$ indicates the order of execution of the actions in the plan.

Examples of composed actions are described in more details. For all the actions described in this part, the execution time and the energy cost can be obtained by summing up the costs of the simple actions composing them.

Migrate and Turn Off (A_{C1}) This action consists of migrating all the VMs on a server and then turning off the empty server. It is useful when enough resources are available in other machines and turning off a server reduces significantly the energy usage. It takes as parameter the identifier of the server to turn off and can be applied only if the preconditions for the two actions composing this complex action are both satisfied.

Turn On and Migrate (A_{C2}) This action turns on a server and migrates a VM on it. It is useful when more resources are needed and a new server can increase the performance of the system. The action takes as parameter the identifiers of a VM and of a server. As before, preconditions for both the simple actions composing it have to be satisfied.

5.2. Learning Action-Goal Relations

Actions can have a different outcome on different indicators. In fact, the same action could positively affect the state of an indicator while negatively affecting another one. A complete knowledge of the influence of the considered actions over the monitored indicators is important to avoid unexpected side effects. A detailed mapping of action-indicator relations can be a difficult task even for a domain expert, especially in a dynamic environment where the effect of an action can be influenced by the current context. In order to overcome this issue, we propose the AAS approach as a methodology for automatically learning the action-indicator relations from experience and for suggesting the most likely successful repair action given a context. The approach associates to each action a probability of success in the indicator improvement, and keeps this probability up to date in time. In this way, the proposed model is able to automatically adapt to modifications in the dynamic environment. The algorithm implements the principal features of the AOS algorithm described in [10] and [11], exploring several possibilities and favoring to the ones that have a higher confidence of being effective.

A general view of the approach is shown in Fig. 4. Given a context and a set of repair actions, the aim of the AAS algorithm is to suggest the most likely successful action and observe the outcome of this choice. The evaluation consists of comparing the values of the indicators before and after the action execution. Whenever an action is enacted, a quality value is assigned for each indicator considering also past executions:

Definition 6. *The* QUALITY $\mathcal{Q}$ *of an action* A_l *over an indicator* I_n *is a value representing the effectiveness of the action in increasing the value of the indicator.*

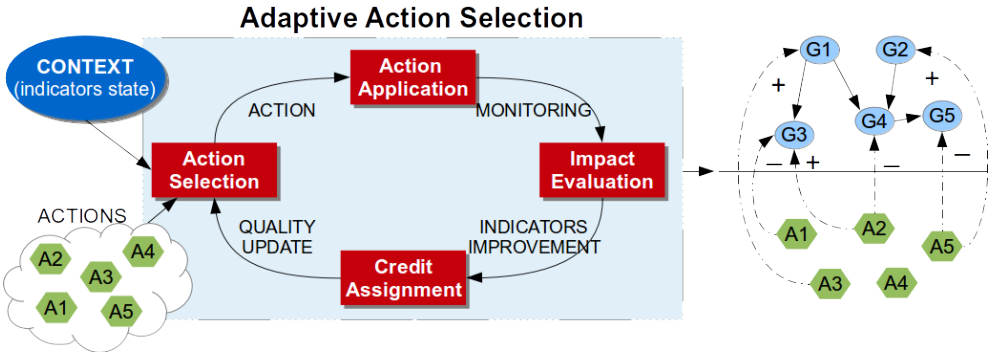


Figure 4: Adaptive Action Selection approach for adaptive applications

Quality is defined in the interval $[-1, 1]$, where positive values represents the ability of the action in increasing the value of the indicator, while negative values indicates the ability of the action in decreasing the indicator value. Increasing or decreasing the value of an indicator can be desirable in some contexts while damaging in others, and this context information will be considered later in the procedure for selecting a repair strategy.

In order to use the model we need to learn the quality of each action over all the indicators of the system and each of these values takes into account both the most recent values and also historical values collected after the application of the action. After some executions it is possible to derive information about the action-indicator relations contained in the goal-oriented model shown in Fig. 1, associated with a confidence value.

Quality computation starts from the application of an action and the observation of its effects. The impact $\mathcal{I}$ of the action can be evaluated observing the system state after the execution time required by the action. The new state is compared with the one before the application of the action. Impact can be defined as follows:

Definition 7. *The* IMPACT $\mathcal{I}$ *is the effect that the execution of an action has over the indicators defined for monitoring the system.*

An action has a different impact over different indicators, so a different value for each of them has to be used. The impact $\mathcal{I}$ of an action A_l over the indicator I_n is assigned considering the variation of the indicator after the application of the action. We model $\mathcal{I}$ in the interval $[-1, 1]$. The impact matrix is a two dimensional LxN matrix, where the value $\mathcal{I}(A_l, I_n)$ represents the ability of action A_l to increase the value of indicator I_n:

	I_1	I_2	$\cdots$	I_N
A_1	$\mathcal{I}(A_1, I_1)$	$\mathcal{I}(A_1, I_2)$	$\cdots$	$\mathcal{I}(A_1, I_N)$
A_2	$\mathcal{I}(A_2, I_1)$	$\mathcal{I}(A_2, I_2)$	$\ldots$	$\mathcal{I}(A_2, I_N)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
A_L	$\mathcal{I}(A_L, I_1)$	$\mathcal{I}(A_L, I_2)$	$\cdots$	$\mathcal{I}(A_L, I_N)$

If the value of the indicator increases (decreases) significantly, then the assigned impact is the maximum (minimum) ($\mathcal{I}(A_l, I_n) = 1(-1)$). Otherwise, if the increase (decrease) is limited, than the impact value assigned to the action is 0.5 (-0.5). In order to assess if the modification of the indicator value is significant or not, we used as a comparison the size of the warning interval. When the modification is bigger than the warning region, there is a possibility for the indicator to move from a violation to the satisfaction zone. Impact of action A_l over indicator I_n is computed as follows:

- if $(I_{n,t+1} - I_{n,t}) > size(Warning(I_n)) \Rightarrow \mathcal{I}(A_l, I_n) = 1.0$;
- if $(I_{n,t} - I_{n,t+1}) > size(Warning(I_n)) \Rightarrow \mathcal{I}(A_l, I_n) = -1.0$;
- if $size(Warning(I_n))/3 < (I_{n,t+1} - I_{n,t}) < size(Warning(I_n)) \Rightarrow \mathcal{I}(A_l, I_n) = 0.5$;
- if $size(Warning(I_n))/3 < (I_{n,t} - I_{n,t+1}) < size(Warning(I_n)) \Rightarrow \mathcal{I}(A_l, I_n) = -0.5$;

Since the system is not isolated, it is possible that its state has been influenced by external factors. That means that some noise can be introduced in the impact evaluation. For this reason, past values of the impact evaluation are kept and a window $\mathcal{W}$ of fixed size is used to compute a credit using an aggregation of the historical values inside the window. The size of the window $\mathcal{W}$ is chosen depending on the features of the system. If the system is stable, a large value for the window size is recommended. Otherwise, small values for $\mathcal{W}$ make the system more dynamically adaptable, enabling it to change strategy more frequently. We can compute the credit $\mathcal{C}$ as:

$$\mathcal{C}_t(A_l, I_n) = \frac{\sum_{s \in \mathcal{W}} \mathcal{I}_s(A_l, I_n)}{size(\mathcal{W}_t)} \tag{6}$$

As stated before, quality is the estimation of the ability of the action to increase the value of an indicator. In our system, it is represented through

a matrix $\mathcal{Q}$ of dimensions LxN where L is the number of available actions and N is the number of indicators. The quality matrix is updated at each iteration. While impact is computed instantly at each application, quality depends also from past values. In general, quality $\mathcal{Q}$ can be expressed as:

$$\mathcal{Q}_t = f(\mathcal{Q}_{t-1}, \mathcal{C}_t) \tag{7}$$

According to Eq. 7, quality depends on two factors: (i) *Credit*: this value depends exclusively on the last execution of the action; (ii) *History*: this value summarizes the outcome of old executions of the action. Different methods for estimating quality by combining these two components are used in different algorithms. Here, we decided to use the Probability Matching approach, proposed by Goldberg in 1990 [33]. In Probability Matching, the probability of selecting an operator is proportional to its quality estimation. Given the number of available operators K, for each operator the algorithm keeps two values: the probability $p_{k,t}$ and the quality estimation $q_{k,t}$. Quality estimation for an action can be performed as follows:

$$\mathcal{Q}_t(A_l, I_n) = (1 - \alpha)\mathcal{Q}_{t-1}(A_l, I_n) + \alpha \cdot \mathcal{C}_t(A_l, I_n) \tag{8}$$

where α is an adaptation rate, indicating the importance given to the memory of past values. The definition of α can be problematic and a fixed value is not the best solution. The importance of history in the quality computation should be directly dependent on the affordability of the historical value. If the action has not been applied for a long time, its historical value does not reflect the current situation, and more importance should be given to the credit computed from the current application of the action. On the contrary, if the action has been recently applied, the historical value reflects the current situation and it should have a higher importance. According to this, the importance of history should depend on the elapsed time since the last application of the action. We propose a value for α which is dependent on the freshness and affordability of the historical information about quality:

$$\alpha(A_l) = 1 - \frac{Last_{op}(A_l)}{T_{op}} \tag{9}$$

where $Last_{op}(A_l)$ is the timestamp of the last application of action A_l, while T_{op} is the current timestamp. Let us consider the behavior of α under different situations. When the action A_l is applied for the first time, history has no value and all the importance is given to the credit ($Last_{op}(A_l) = 0 \Rightarrow \alpha(A_l) =$

ALGORITHM 1: The Probability Matching algorithm for Adaptive Action Selection

Input: the values of the indicators I, the current quality matrix $\mathcal{Q}$, the action list A
Output: an updated quality matrix $\mathcal{Q}$
compute context $\mathbb{C}$ from I ;
compute w_n for each indicator in I according to its context in $\mathbb{C}$;
verify preconditions for each action A_i in A;
create the set of actions A' which satisfy the preconditions;
verify if there are actions in A' never tested and eventually remove all tested actions from A';
select an action A_l^* via a random selection scheme;
wait for $t_{A_l^*}$;
collect indicators values I' from the monitoring system;
compute impact $\mathcal{I}$: **if** $((I'_n - I_n) > size(Alarm(I_n)))$;
then
    $\mathcal{I}(A_l^*, I_n) = 1.0$;
end
else if $((I_n - I'_n) > size(Alarm(I_n)))$;
then
    $\mathcal{I}(A_l^*, I_n) = -1.0$;
end
else if $((I'_n - I_n) > size(Alarm(I_n))/3)$;
then
    $\mathcal{I}(A_l^*, I_n) = 0.5$;
end
else if $((I_n - I'_n) > size(Alarm(I_n))/3)$;
then
    $\mathcal{I}(A_k, I_n) = -0.5$;
end
compute credit: $\mathcal{C}_t(A_l, I_n) = \frac{\sum_{s \in \mathcal{W}} \mathcal{I}_s(A_l, I_n)}{size(\mathcal{W}_t)}$;
compute $\alpha(A_l^*) = 1 - \frac{Last_{op}(A_l^*)}{T_{op}}$;
update quality: $\mathcal{Q}_t(A_l^*, I_n) = (1 - \alpha)\mathcal{Q}_{t-1}(A_l^*, I_n) + \alpha \cdot \mathcal{C}_t(A_l^*, I_n)$;
$Last_{op}(A_l^*) = T_{op}$;
$T_{op} = T_{op} + 1$;
Return the updated quality matrix $\mathcal{Q}$;

$1 - \frac{0}{T_{op}} = 1$, and $1 - \alpha = 0$. Otherwise, if the action has been applied in the last execution, than $Last_{op}(A_l) = T_{op} - 1 \Rightarrow \alpha(A_l) = 1 - \frac{T_{op}-1}{T_{op}} = \frac{1}{T_{op}}$ and $1 - \alpha(A_l) = 1 - \frac{1}{T_{op}}$, giving to history a greater importance.

In order to allow the algorithm to quickly adapt in the initial phases, where the effect of the action has still to be learned, it is important to give the priority to actions never tested before, ensuring that each action is tried at least once. Since not all the actions can be applied in every moment (due to the preconditions introduced in Sect. 5.1), this condition has to be verified at each step. The algorithm for learning the action impact is shown in Alg. 1.

6. Using the Model

The proposed approach can now be used to monitor and improve the state of the data center. First of all, the state of the system is observed through the defined metrics. Whenever one or more metrics are violated or near to be violated, some repair actions are enacted to move the system towards the desired state (where no violations occur).

Two phases can be identified: in the first phase, the system suggests a repair action using the action to goal relations learned in Sect. 5. The second phase consists in exploiting the knowledge about indicators relations and using again the model by considering also indicators which are not violated, but which have an influence over violated ones. This second step can suggest a different action than the one suggested in the first phase, which can have a better impact over the violated indicators.

6.1. Phase 1: Exploiting the Action to Goal Relations

In order to use the action to goal relations, we have to assign a probability of success to each of the actions available given the current context $\mathbb{C}$ (see Def. 3). The first step for computing probability given a context consists in evaluating the quality matrix according to $\mathbb{C}$. Quality is be positive if the action increases the indicator value, negative otherwise. This information has to be contextualized, since some indicators can be under their desired values but others can be over it, as shown in Fig. 2. So, in this last case, we should prefer actions able to reduce the indicator value. The contextualized quality matrix $\mathcal{Q}'$ can be computed as follows:

$$\mathcal{Q}'_t(A_l, I_n) = \begin{cases} -\mathcal{Q}_t(A_l, I_n) & IF\ I_n > I_n^{max} \\ \mathcal{Q}_t(A_l, I_n) & otherwise \end{cases} \tag{10}$$

where I_n^{max} represents the maximum value that the indicator can take before entering in the warning zone, as depicted in Fig. 2.

Probability $p(A_l|\mathbb{C})$ depends directly from $\mathcal{Q}'_t$ and can be computed as:

$$p'(A_l|\mathbb{C}) = max\left\langle 0, \frac{1}{c_{A_l}}\left(\frac{\sum_n w_n \mathcal{Q}'_t(A_l, I_n)}{\sum_n w_n} - 0.5\frac{\sum_n w'_n |\mathcal{Q}'_t(A_l, I_n)|}{\sum_n w'_n}\right)\right\rangle \tag{11}$$

The first part of the equation expresses the ability of the action of improving the violated indicators. The term w_n is a weight expressing the indicator violation priority, used to give different importance to the indicators in the alarm and in the warning zone ($w_n = 1$ if the violated indicator is in the alarm zone, $w_n = 0.5$ if it is in the warning zone, and $w_n = 0$ if it is in the normal zone). The probability of applying the action depends on the contextualized quality of the action over the indicators that are violated or that are near to be violated. The second part represents a penalty term, which decreases the probability value if the action affects indicators that are not violated. The term w'_n is equal to 1 for each metric which is not in the violation list (for which $w_n = 0$). The penalty factor has less importance than the other one since even if the action has an effect on other metrics, this does not means that this effect will have a negative outcome. In fact, the metrics could stay inside the constraints also after the execution. Finally, the cost c_{A_l} of the action enactment is considered (see Eq. 4).

In the Probability Matching approach, a minimum probability is assigned to each action. Keeping the probability greater than 0 allows the system to select the action, even if rarely, and to learn the new behavior if the system changes. This approach is useful since the system is dynamic and the utility of an action can change over time. Probability is defined as follows:

$$p(A_l|\mathbb{C}) = p_{min} + (1 - L * p_{min}) \frac{p'(A_l|\mathbb{C})}{\sum_{m \in L} p'(A_m|\mathbb{C})} \tag{12}$$

We assign a dynamic value to p_{min} which depends on the affordability of the best action available in the considered context. It has also to be proportional number of available actions L. We propose to compute the value for p_{min} as:

$$p_{min} = \frac{1 - max(p'(A|\mathbb{C}))}{L * 10} \tag{13}$$

Once probabilities for all the actions are computed given the context, an action can be selected using a wheel-like process. In Phase 2 indirect relations are explored for refining the selection.

6.2. Phase 2: Exploiting the Goal to Goal Relations

The BN representing the relations among goals can be used to detect actions which have an indirect impact on the violated indicators by detecting which are the indicators having a strong relation with the violated ones.

The AAS algorithm is executed again with a different context information, consisting of this new set of indicators. The algorithm to exploit the goal to goal relations behaves as follows:

1. for each of the violated indicators, select the set of parents in the BN;
2. for each of these parents, use the CPT to detect which is the desired state for this indicator which can improve the children and compare it with the current state;
3. create a new context $\mathbb{C}'$ where the value of the selected parents has to be increased or decreased according to the CPT table and the context;
4. apply the AAS algorithm as described in Sect. 6.1 to $\mathbb{C}'$;
5. compute the final probability by multiplying the obtained vector $p(A_l|\mathbb{C}')$ with the value of the CPT table between the parent and the violated indicators. In this way the probability is diminished according to the strength of the relation existing between the indicators.

This second step allows the selection of a new repair action, indirectly improving the violated indicators. Given the results of both Phase 1 and Phase 2, it is possible to select the action with the highest likelihood of success. Whenever a repair action is applied, its outcome is observed and used to update the quality matrix as discussed in Sect. 5.2.

7. Evaluation of the Algorithm

In this section we show experimental results of the described approach. First, we introduce the evaluation criteria used to evaluate the approach (Sect. 7.1). Then, we apply the approach to two different applications. In Sect. 7.2, we discuss a systematic study of the behavior of the proposed algorithm applied to an e-business application in a simulated environment. This would not be possible in a real system where the variation of variables cannot be induced in a controlled way. In Sect. 7.3, we illustrate experimental results obtained for a CPU intensive application in a real data center. The learning module has been implemented in MATLAB [34]. For managing BNs, we used the MATLAB Bayes Net Toolbox (BNET) [35], providing instruments for learning and using BNs. The code of the structure learning module is available at [36]. The AAS algorithm has also been implemented in MATLAB and it is available at [37]. In Sect. 7.4 results are discussed.

7.1. Evaluation Criteria

In order to evaluate the results, first we define a distance function $\mathcal{D}$:

Definition 8. *The distance $\mathcal{D}$ of a context $\mathbb{C}$ is defined as the sum of the distances of the state of each indicator from the optimal state. More in details, an indicator in the normal zone has distance value $d(I_n) = 0$, an indicator in the warning zone has distance value $d(I_n) = 1$, and an indicator in the alarm zone has distance value $d(I_n) = 2$. The total distance is computed as:*

$$\mathcal{D}(\mathbb{C}) = \sum_n d(I_n|\mathbb{C}) \tag{14}$$

This value is used to measure the ability of the algorithm to improve the state of the system given a context, comparing the state of the system before and after the execution of a repair action.

7.2. Testing the Algorithm with Simulated Data

In this section we describe the results obtained using an e-business application composed of three activities deployed on dedicated VMs having different features about response time and average processing time. Requests of users can be directed independently to each of them. The application has been tested using the simulation system described in [38], which allows us to collect all the required information while having full control of the workload of each of the activities involved in the application. In this test we consider a small data center composed of two servers and three VMs, with VM1 and VM3 deployed on S1 and VM2 deployed on S2, and studied its behavior systematically as shown in the following.

The first step consists in learning relations among indicators. Considered indicators are CPU Usage (U(x)), Response Time (R(x)), Performance per Energy (PE(x)), and Energy (E(x)). Several workloads have been simulated while collecting monitoring data for learning the model. Using the algorithm described in Sect. 4, we have created the BN representing relations among variables. The DAGs corresponding to the best score is drawn in Fig. 5. Discovered relations reflect the structure of the data center, isolating metrics belonging to different servers while associating metrics belonging to the same servers and to the same VMs. In this way the help of an expert is not needed to model the system, but the model is learned by the algorithm. To test the ability of the network to predict the state of an indicator, we have used

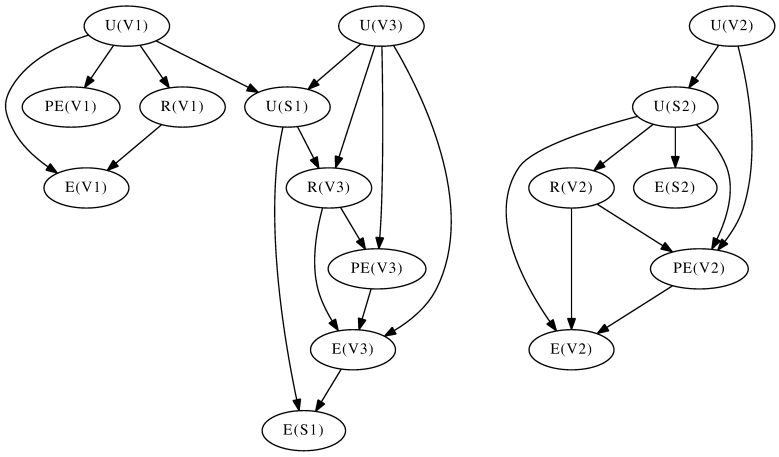


Figure 5: Highest scored directed BN obtained applying the Bayesian score

a test set different from the training set used to acquire the network. We have hidden the value of a variable in all the examples and for each example we used the network to predict this value. Comparing the result with the real value we obtained the success rate of the network prediction ability. We have run the same procedure for all the variables in our system obtaining an average success rate around 87.5%.

Once relations among variables are discovered, we can use the algorithm described in Sect. 5 to learn the effect of actions over indicators. The set of actions considered is a subset of the ones analyzed in Sect. 5.1. The resulting quality matrix is shown in Fig. 6. As it can be observed, migrating a VM in a server (A_2) is likely to increase the server usage while decreasing the usage of the previous server. The same effect can be observed on the energy consumption of the servers. Action A_2 is likely to decrease the value of CPU usage and response time for the VM, while the energy consumption has a slight probability of being increased. The same considerations are symmetrically valid for action A_3. Using action A_{C1}, which migrates all the VMs on a server and then turn this server off, the usage of the server to be turned off is reduced, while the usage of the other server is increased since new machines are migrated on it. This behavior is reflected also on the energy usage. Action A_{C1} affects also the Performance per Energy since, in our specific configuration, migrating VMs from server $S1$ increases their performance because $S2$ is more efficient than $S1$. The same is valid for the energy of the VMs. Action A_{C2} behaves similarly to migration, affecting the

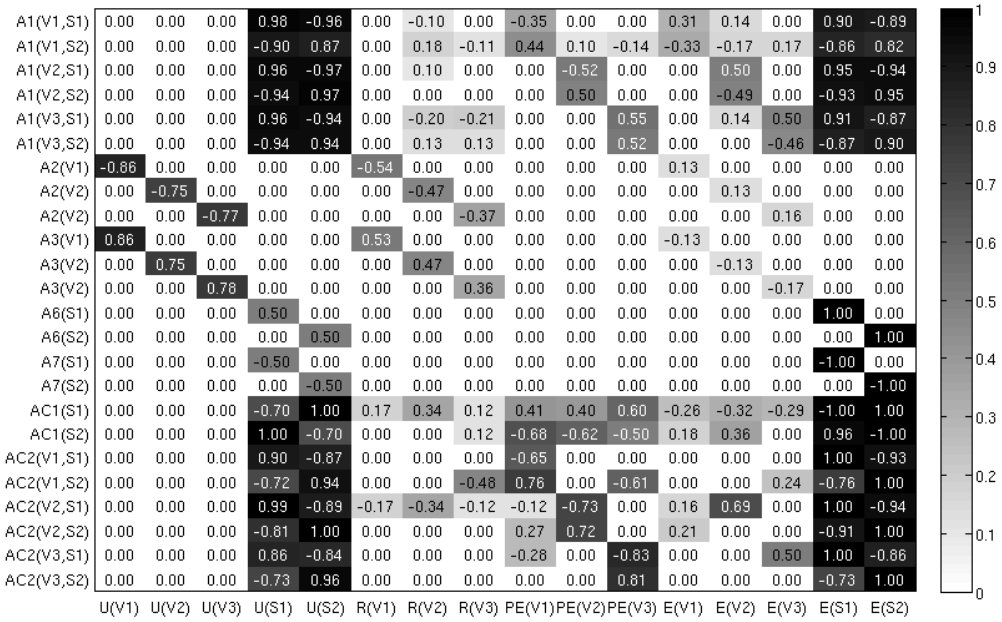
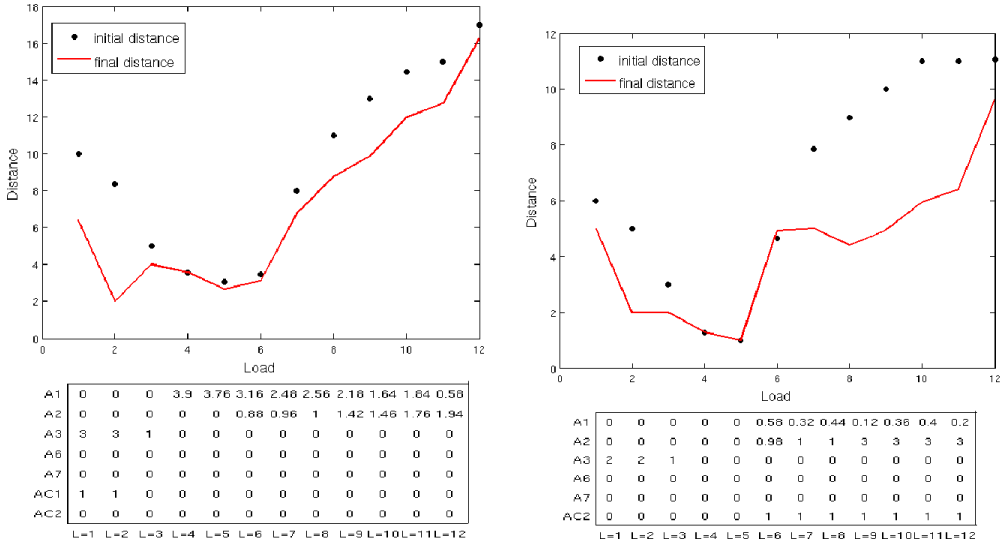


Figure 6: Quality matrix with complex actions

usage of both servers, their energy, and the efficiency and energy of the VMs.

Once the model is learned, we can evaluate its ability to recommend a good repair strategy using the distance function $\mathcal{D}$ introduced in Eq. 14. In order to evaluate the algorithm, we run it using an incremental load. For each load, the algorithm has been executed 50 times, and the average distance value for each run has been computed to have a more reliable result. We consider a test terminated when the system enters into a static condition in which the "Do Nothing" action is always selected.

Results are shown in Fig. 7(a). Here we can see that the algorithm is able to improve the system state by reducing the distance from the optimal configuration. Fig. 7(a) also shows the average number of times each action is applied. An improvement can be observed for low load rates: for load $L < 3$ action A_{C1} is selected, consolidating all the VMs on a single server, significantly improving the distance function when spare resources are available. Action A_{C2} is never applied, since there are no other servers available in the system. In another test we have used a different initial configuration, with only two VMs, both deployed in the same server, and another server available but inactive. Results are shown in Fig. 7(b). In the first steps, the



(a) Configuration with 2 servers and 3 VMs (b) Configuration with 2 servers and 2 VMs

Figure 7: Results of the AAS algorithm at different load rates

algorithm applies action A_3 to increase the CPU usage indicator at the VM level. For $L = 4$ and $L = 5$ no action is applied since the system is near to the optimum and no action is able to improve the state. For $L >= 6$ the algorithm starts selecting A_{C2} for execution. This action enables the availability of more resources, allowing for migrations and for resources reconfiguration. In fact, increasing the load, all the VMs need a higher amount of CPU for satisfying the related indicators. With $L > 12$ the load exceeds the capacity of the system so no adaptation can be effective.

7.3. Testing the Algorithm on a Real Data

In this section we apply the model to learn relations among variables collected in a real data center. We use the $ECO_2Clouds$[4] architecture which is developed as an extension of the BonFire platform[5], providing monitoring facilities for a federated data center located in different countries in Europe. In particular, we run our tests at the data center provided by INRIA[6].

[4]http://eco2clouds.eu/
[5]http://www.bonfire-project.eu/
[6]http://www.inria.fr/

ECO_2Clouds extends Zabbix to monitor its running environment from the infrastructure to the application layer, with a focus on EE and CO_2 emissions. Physical hosts are equipped with a power distribution unit (PDU), collecting information about the power consumption of a single server. Energy consumption at the virtual level is also computed using models available in the state of the art. Zabbix can be easily extended to include new metrics defined at the application level. In our experiment, monitored data were retrieved every 30 seconds.

To perform experiments on ECO_2Clouds, we istantiated three VMs running on two different physical hosts provided with 24 6-Core Intel® Xeon® Processor E5-2620 @ 2.00GHz and with 63,7 GB RAM. Each VM is provided with 1 CPU and 4096 MB RAM. The application deployed on the VMs is a CPU intensive application simulating and predicting the movement of eels in the ocean [39], in which CPU is always used at its maximum capacity.

In the experiment we run several requests, distributing the load on a single VM or on several VMs in parallel. In the tests, virtual machines V1 and V3 are deployed on the same server (indicated as S1 which correspond to the server named bonfire-1), while VM V2 is deployed on server bonfire-2, indicated as S2. We selected as indicators resources usage on each VM and on the server (CPU - U(x), memory - M(x), I/O transactions - IO(x)), energy both at the VM and at the server level (E(x)), and throughput of each activity and of the whole application (TH(x)). Relations among indicators have been discovered using the procedure described in Sect. 4.2 and are shown in Fig. 8. As in the previous example, the algorithm is able to find consistent relations among variables, properly distinguishing among variables on different VMs. Variables related to servers are not connected to any other variable in the system. This is due to the fact that the considered servers are only partially used in the experiment and their resources are shared with other applications which are not considered in the learning process. The predominance of CPU variation (which is either 100% or 0%) makes this variable the one from which any other variable depends, because it allows distinguishing between time periods when the application is running and time periods when nothing is running. Each edge in the graph expresses dependencies among the state of the involved variables. As an example we can analyze the CPT associated to the relations between U(V3) and E(V3), depicted in Fig. 8. Since in all the examples in the training set the state of U(V3) is either 1 or 5, for these two states the network is able to predict the consequent state for E(V3), while for the other states all the outcomes have the same probability. The

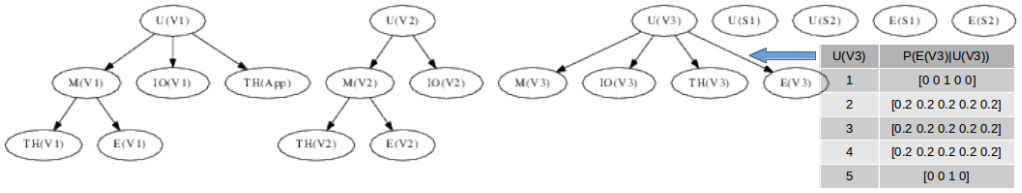


Figure 8: Relations among variable for a real application: the discovered Bayesian Network

discovered knowledge can be exploited in the adaptation phase.

The $Eco_2Clouds$ platform does not provide a way to perform adaptation actions such as live migration of VMs on the system, so the learning phase for action-goal relations has been performed using the simulation environment described in Sect. 7.2. The aim is to verify the ability of the algorithm in improving the efficiency of the system using different configurations. In the test we start from a configuration similar to the one described in Sect. 7.2, in which we have again V1 and V3 running on server S1, and V2 running on S2. From tests we observed that S1 is a more performing server, and the time of execution of the same activity is lower on S1 than on S2. So, running a request of execution for the eels case study takes 12 minutes on S1 and 20 minutes on S2 on average. Such a situation is detected by the learning algorithm and included in the quality matrix. Results are similar to what was depicted in Fig. 6 where migration of V3 from S1 to S2 (A1(V3,S2)) impacts response time increasing its value, while performing the migration from S2 to S1 (A1(V3,S1)) decreases response time. In the initial configuration, resources on both servers are underutilized and response time of V2 is violated. The total energy consumed for the execution of the application is equal to 63.76 Wh. The energy consumption of the application is computed considering the energy consumed by each VM involved in the application and a portion of the energy consumed in idle state by the server hosting the VM. The AAS algorithm suggests to migrate VM2 on S1. Migration improves the resource utilization on the servers and impact on response time of V2. In the new configuration, an execution of the application requires 25.52 Wh, 60% less than the previous configuration, and reduces the total response time of 40% since all the instances can be completed in 12 minutes instead of 20.

7.4. Analysis of the Results and Scalability

Whenever a new application is deployed on the data center, the BN related to its variables has to be created. Learning the BN used to represent

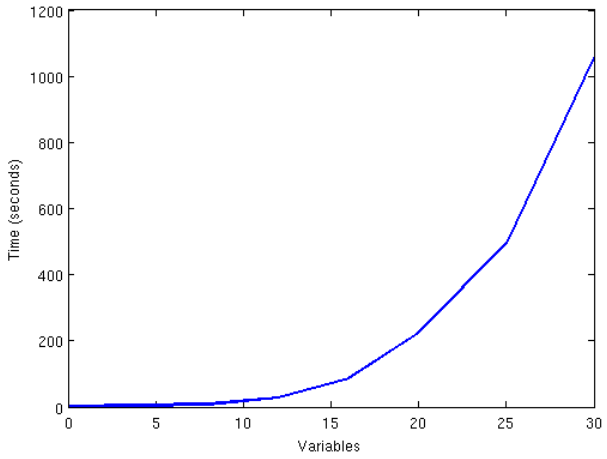


Figure 9: Computation time of the learning module with an increasing number of variables

relations among goals is computationally expensive and the time increases exponentially with the number of variables considered in the network. In order to analyze this issue, several tests have been run changing two parameters: the number of variables and the size of the dataset. From the tests, we observed that the size of the dataset does not significantly affect the computation time. On the contrary, the number of variables influences the computation time exponentially (Fig. 9). Even if in the tests the computation time stays in a manageable range, this can become an issue for larger sets of variables.

The main concern about this approach is related to scalability. However, some considerations can be done that mitigate this issue. The number of variables to be considered is limited to the variables involved in the application, while other applications running in the same data center will be explored separately, using a different set of indicators and adaptation strategies. Even with this limitation, the number of variables to be considered can be elevated. We have experimentally demonstrated that variables monitored on a VM are independent from variables monitored on other VMs, but are not independent from variables related to the server hosting the VM. Instead of computing relations among all the variables in the system, we can consider subsets of these variables for each VM. The monitoring system collects 6 variables for each VM plus 2 variables for the host. Dividing the problem into sub-problems, we have to consider networks of only 8 variables each. This

would take about 15 seconds for each network, multiplied by the number of VMs involved in the application. In case of applications involving 100 VMs, the computation time of the initial BN would be around 25 minutes. Using parallel execution can further decrease this time. In addition, the computation of the complete network needs to be performed only once, during the set up operations. Further modifications of the system (e.g., VM migrations and reconfigurations) affect only parts of the BN as indicated above and will trigger only the recalculation of the parts of the network involved.

Issues related to scalability also affect the action to goal relations learning. Defining the computation time of the quality matrix is not so trivial. The quality matrix is continuously refined during the lifetime of the application and its computation depends on the occurrence of violations. For the quality matrix to be reliable, each action needs to be applied for all possible values of the parameter at least 3 times. The computation time linearly depends on the number of available actions and the number of parameters. To improve the performance of the algorithm, relations can be learned off line, using a simulation system. This operation has to be performed only once, then the values will be kept updated incrementally during the action selection phase. An additional solution consists of performing a generalization of the relations between actions and indicators. Observing the quality matrix obtained by the application of the algorithm in the learning phase, it is possible to see that the same action behaves similarly on similar indicators, depending on the parameters used to execute the action. Adding some knowledge about the meaning of each variable and describing similarities among variables, it is possible to predict the effect of an action observed over a subset of variables for similar indicators belonging to other VMs. This information can be useful in the initial steps of the algorithm to suggest an effective adaptation action even if no direct information is available. This initial prediction can be refined by the actual enactment of the action in the VM. As an example, if we observe that action A_3 applied to VM1 increases the values of CPU usage and response time of this VM while decreasing its energy consumption, we can predict that applying the same action to VM2, it will have a similar effect over the same set of indicators. This information can be used by the algorithm to suggest this action when the CPU usage indicator of VM2 is violated and the observed outcome of the action can be used to fill in the missing information in the matrix with real data. Generalizing the action behavior can significantly reduce the number of training steps needed for obtaining a reliable initial knowledge of the effect of the actions over the

indicators and to significantly improve the performance of the algorithm in this phase. This solution will be further investigated in future work.

8. Conclusions

In this work we proposed a goal-oriented model for driving adaptation in applications having as primary goal EE and QoS constraints satisfaction. Selecting a set of metrics able to describe the state of the system and learning relations among them together with the effect of the application of repair actions, we have been able to propose a dynamic model to improve the state of the system. This model is able to automatically react to modifications in the environment. Testing the application, we have automatically built BNs able to predict future states of the indicators with a success rate of the 85%, and to prescribe adaptation actions that can improve the system state when violations occur. We have also demonstrated that the algorithm for learning the relations among goals is scalable due to the independence between variables belonging to different VMs emerged from tests. Also the scalability of the AAS algorithm can be improved with some modifications, using an ontology describing similarities among indicators for generalizing the effects of actions over different VMs.

Future development of the approach are going to analyze this ontology describing the monitored metrics and to evaluate how this can improve the behavior of the AAS algorithm in the initial steps. Also, the approach will be extended and tested in a cloud environment, in which the differences between the available sites should be considered, especially when deploying new VMs or when migrating one machine from one site to another, and additional actions can be applied. Preliminary results in the application of the approach to a cloud environment have been conducted in the framework of the Eco_2Clouds project. Here, the application of the approach together with a scheduler able to select the best host to deploy a VM have proved able to reduce energy consumption up to 50% if compared with a random deployment where no adaptation is enacted.

Acknowledgments

This work has been partially supported by the ECO_2Clouds European project (http://eco2clouds.eu/). This work expresses the opinions of the authors and not necessarily those of the European Commission. The European

Commission is not liable for any use that may be made of the information contained in this work. Authors want to thank the Progetto Rocca for the financial support of a Ph.D. visiting period at the Massachusetts Institute of Technology which allowed the development of part of this research.

References

[1] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems, Advances in Computers 82 (2011) 47–111.

[2] M. Vitali, B. Pernici, A Survey on Energy Efficiency in Information Systems, International Journal of Cooperative Information Systems (IJCIS) 23 (2014) 1–38.

[3] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali, et al., A taxonomy and survey on green data center networks, Future Generation Computer Systems 36 (2014) 189–208.

[4] G. Cook, How Clean is Your Cloud?, Technical Report, Greenpeace International, 2012.

[5] A. De Oliveira, G. Frederico, T. Ledoux, Self-Optimisation of the Energy Footprint in Service-Oriented Architectures, Proceedings of the 1st Workshop on Green Computing (GCM'10) (2010).

[6] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, F. Leymann, Adaptation of Service-Based Applications Based on Process Quality Factor Analysis, in: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, 2010, pp. 395–404.

[7] S. S. Masoumzadeh, H. Hlavacs, An Intelligent and Adaptive Threshold-Based Schema for Energy and Performance Efficient Dynamic VM Consolidation, in: Energy Efficiency in Large Scale Distributed Systems, Springer, 2013, pp. 85–97.

[8] I. Tsamardinos, L. E. Brown, C. F. Aliferis, The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm, Machine Learning 65 (2006) 31–78.

[9] S. J. Russell, P. Norvig, E. Davis, Artificial Intelligence: a Modern Approach, volume 2, Prentice Hall Englewood Cliffs, 2010.

[10] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, Hyper-Heuristics: A Survey of the State of the Art, Journal of the Operational Research Society (2010).

[11] K. Li, A. Fialho, S. Kwong, Q. Zhang, Adaptive Operator Selection With Bandits for a Multiobjective Evolutionary Algorithm Based on Decomposition, IEEE Trans. Evolutionary Computation 18 (2014) 114–130.

[12] M. Papazoglou, K. Pohl, M. Parkin, A. Metzger, eds., Service Research Challenges and Solutions for the Future Internet: S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems, volume 6500, Springer, 2010.

[13] C. Patel, K. Supekar, Y. Lee, A QoS Oriented Framework for Adaptive Management of Web Service Based Workflows, in: Database and Expert Systems Applications, 2003, pp. 826–835.

[14] H. Zhang, L. Liu, T. Li, Designing IT Systems According to Environmental Settings: A Strategic Analysis Framework, The Journal of Strategic Information Systems 20 (2011) 80–95.

[15] R. Ali, F. Dalpiaz, P. Giorgini, A Goal-Based Framework for Contextual Requirements Modeling and Analysis, Requirements Engineering 15 (2010) 439–458.

[16] J. Gao, Machine Learning Applications for Data Center Optimization, Technical Report, Google, 2014.

[17] A. Beloglazov, R. Buyya, Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints, IEEE Transactions on Parallel and Distributed Systems 24 (2013) 1366–1379.

[18] X. Zheng, Y. Cai, CMDP Based Adaptive Power Management in Server Clusters, Sustainable Computing: Informatics and Systems 3 (2013) 70–79.

[19] Y. Asnar, P. Giorgini, J. Mylopoulos, Goal-Driven Risk Assessment in Requirements Engineering, Requirements Engineering 16 (2011) 101–116.

[20] A. Mello Ferreira, B. Pernici, Managing the Complex Data Center Environment: an Integrated Energy-Aware Framework, Computing 96 (2014) 1–41.

[21] J. Radatz, A. Geraci, F. Katki, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std (1990).

[22] C. Cappiello, P. Plebani, M. Vitali, Energy-Aware Process Design Optimization, in: Proceedings of the 3rd International Conference on Cloud and Green Computing, 2013.

[23] D. Chen, E. Henis, R. I. Kat, D. Sotnikov, C. Cappiello, A. M. Ferreira, B. Pernici, M. Vitali, T. Jiang, J. Liu, A. Kipp, Usage Centric Green Performance Indicators, SIGMETRICS Performance Evaluation Review 39 (2011) 92–96. doi:doi: 10.1145/2160803.2160868.

[24] The Green Grid Consortium, Data Center Maturity Model, White Paper (2011).

[25] B. Pernici, C. Cappiello, M. G. Fugini, P. Plebani, M. Vitali, I. Salomie, T. Cioara, I. Anghel, E. Henis, R. Kat, et al., Setting Energy Efficiency Goals in Data Centers: the GAMES Approach, in: Energy Efficient Data Centers, Springer, 2012, pp. 1–12.

[26] G. F. Cooper, E. Herskovits, A Bayesian Method for the Induction of Probabilistic Networks from Data, Machine Learning 9 (1992) 309–347.

[27] R. Opgen-Rhein, K. Strimmer, From Correlation to Causation Networks: a Simple Approximate Learning Algorithm and its Application to High-Dimensional Plant Gene Expression Data, BMC Systems Biology 1 (2007) 37.

[28] R. E. Neapolitan, Learning Bayesian Networks, Pearson Prentice Hall Upper Saddle River, 2004.

[29] A. M. Carvalho, Scoring Functions for Learning Bayesian Networks, Inesc-id Tec. Rep (2009).

[30] N. Friedman, I. Nachman, D. Peér, Learning Bayesian Network Structure from Massive Datasets: the Sparse Candidate Algorithm, in: Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence, 1999, pp. 206–215.

[31] Y. Wu, M. Zhao, Performance Modeling of Virtual Machine Live Migration, in: Cloud Computing (CLOUD), 2011 IEEE International Conference on, 2011, pp. 492–499.

[32] D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, I. Brandic, Energy-Efficient and SLA-Aware Management of IaaS Clouds, in: Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet, 2012.

[33] D. E. Goldberg, Probability matching, the magnitude of reinforcement, and classifier system bidding, Machine Learning 5 (1990) 407–425.

[34] MATLAB, version 7.14.0 (R2012a), The MathWorks Inc., Natick, Massachusetts, 2012.

[35] K. Murphy, Bayes Net Toolbox for Matlab, https://code.google.com/p/bnt/, 2007.

[36] M. Vitali, StructureLearning Tool, https://github.com/monicavit164/StructureLearning, 2013.

[37] M. Vitali, ActionSelection Tool, https://github.com/monicavit164/ActionSelection, 2013.

[38] M. Vitali, U.-M. O'Reilly, K. Veeramachaneni, Modeling Service Execution on Data Centers for Energy Efficiency and Quality of Service Monitoring, in: IEEE International Conference on Systems, Man, and Cybernetics (SMC2013), 2013.

[39] P. Melià, A. J. Crivelli, C. Durif, R. Poole, D. Bevacqua, A Simplified Method to Estimate Body Growth Parameters of the European Eel Anguilla Anguilla, Journal of fish biology 85 (2014) 978–984.