Quality of Service Channelling for Latency Sensitive Edge Applications

Atakan Aral, Ivona Brandic
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
$\{$*atakan.aral, ivona.brandic*$\}$*@tuwien.ac.at*

***Abstract*—Edge data centers (EDCs) typically provide lower availability rates than Cloud counterparts since they lack expensive support systems such as air conditioning units and power generators. To avoid this limitation deteriorating response time which is critical for Edge applications, use of proactive optimization algorithms is essential. Such proactive algorithms, however, require an accurate method for estimating availability of a VM that runs on a certain EDC. Estimation is complicated by several dependent and independent probabilistic events (e.g. hardware, software or network failures, power outage, etc.) that affect VM availability in different levels. In this paper, we propose a Bayesian Network model of QoS related parameters to estimate availability level of a VM in Edge infrastructure. We compare our approach to other machine learning methods and to the common practice in reliability theory that is the use of prior failure probability. According to experimental results, proposed method can identify VMs that satisfy user defined availability objectives with up to 94% accuracy and decrease SLO violations by nearly 44%. In addition, we empirically investigate the trade-off between running time and accuracy of proposed approach.**

***Keywords*-Cloud Computing; Edge Computing; Quality of Service; Virtual Machine Availability; Bayesian Networks.**

I. Introduction

Cloud computing has revolutionized information technologies industry in the past decade by favor of efficiency and flexibility it provides. Rapid increase in the magnitude of data generated and consumed by Cloud Services, however, is beginning to cause a bottleneck in network bandwidth [1]. Moreover, highly interactive applications are demanding lower and lower network latency for accessing Cloud computing resources [2]. In response, Edge computing is an endeavour to achieve benefits of Cloud computing without causing network congestion or forfeiting low access latency. It proposes Cloud computing resources to be located in close proximity to their consumers instead of large-scale centralized data centers in order to keep network access latency to a minimum.

Extreme sensitivity to latency and huge amount of data to be processed / transferred are the typical characteristics of new generation cloud tasks (e.g. IoT, Cyber-Foraging) that Edge computing must address to accelerate its adoption [1]. Moreover, software and hardware failures are quite common at Edge data centers (EDCs) due to their distributed nature, volatile workload and the absence of advanced support systems such as air conditioning units and power generators which are present in centralized Cloud data centers. In addition to reducing availability and possibly causing service level objective (SLO) violations, failures also adversely affect response latency and bandwidth traffic since the tasks in failed EDC should be distributed to other Edge or Cloud data centers. Size of data to be transferred and arising delay after a failure restrict the use of reactive methods alone, hence estimating failures early and taking countermeasures is critical for both Edge providers and customers [3]. Proactive optimization algorithms can be applied during either application deployment or run time. For instance, a VM replication strategy can be applied to avoid service disruption and SLO violation. Another solution can be differentiating VM requests based on required availability levels and dispatching them to EDCs that are most expected to satisfy their SLO. In this way, less reliable EDCs are also utilized by noncritical tasks. One common input to any such algorithms is the accurate estimation of failure rates or VM availability levels.

However, estimation problem is nontrivial since failures in a distributed infrastructure such as Edge can originate from numerous hardware, software, and network components as well as human factors. Moreover, failures usually have different and variable rates and are not statistically independent from each other [4]. We present a method that is based on Bayesian Networks (BNs) to estimate availability of VMs in EDCs. The probabilistic model captures dynamic dependencies between different failure types and temporal failure characteristics of components. In essence, our aim is to channel all QoS related parameters through VM availability. Parameters may include, depending on the availability of data, frequency of various system failures as well as hardware and software characteristics. We propose VM as the granularity level for QoS estimation because VMs decouple hardware from software, thus they can be used for handling availability of both.

Two challenges specific to Edge computing environment that we face in estimating QoS are necessity of short response time and limited historical data for failures. Following measures are taken for these challenges respectively, in order to ensure real-world applicability of our approach: (i) A BN augmentation strategy (described in IV-C) which improves runtime performance with respect to frequent model learning, and (ii) An evidence extraction strategy (described in IV-D) fusing relevant traces to cope with data scarcity. In brief, our contributions in this study are a probabilistic model for estimating failures in Edge systems and approximations to solve possible problems in its practical application.

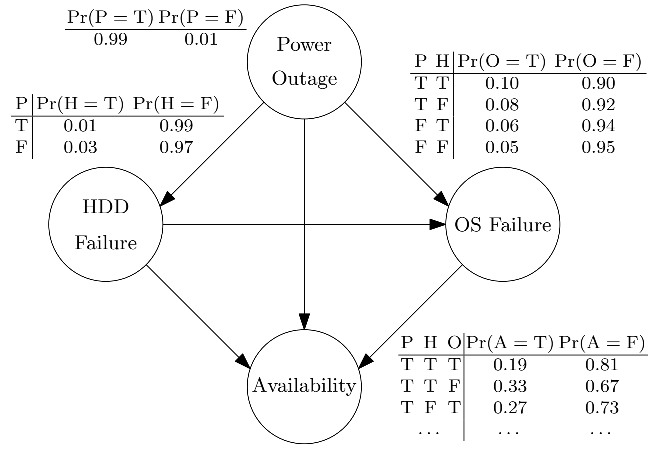
We describe BN model and introduce formalism used throughout the paper in Section II and introduce the proposed system architecture in Section III. Then we present the estimation methodology in Section IV and its evaluation in Section V. We provide related literature and conclude the paper in Sections VI and VII, respectively.

II. Background on Bayesian Networks

BN is a tool to represent and organize the general knowledge about a particular situation. Qualitative part (i.e. structure) of BN is a directed acyclic graph (DAG) with vertices representing relevant variables of the situation and edges representing the dependencies between those. Quantitative part (i.e. parameters), on the other hand, consists of the probabilities of relationships between variables and their parents (direct causes) in the form of Conditional Probability Tables (CPTs) [5].

Consider a set of random variables $R = \{r_1, r_2, \ldots, r_n\}$. A BN can be defined for R, in a formal way, as the pair $\langle G, \Theta \rangle$, where G is above explained DAG which is convenient for visualizing dependence and independence assumptions. Each variable r_i is directly dependent on its parents in G and independent of its non-descendants given these parents. Then, Θ is a set of conditional probabilities for each variable given its parents. There exists a parameter $\theta \in \Theta$ such that $\theta = \Pr(r_i \mid \text{parents}(r_i))$ for each combination of values that r_i and its parents can possibly take [6]. Computing conditional probabilities of variables that are not directly connected in G are trivial via Bayesian inference.

We choose BN as the tool for modeling availability mainly due to its capability to handle conditional dependencies and uncertainty. Another reason is compactness since a BN can specify an exponentially sized probability distribution using a polynomial number of probabilities [5]. Figure 1 demonstrates an example BN for EDC availability scenario. Three failure types that affect the availability of VM are available in this example (i.e. power outage, hard disk drive failure and operating system failure). Causal dependence of availability to these failures are represented by directed edges arriving from them. Moreover, these failures are not independent from each other neither, e.g. a power outage may cause a HDD or OS failure, and a HDD failure may cause an OS failure. Network is also annotated with CPTs that quantify direct dependencies. In these tables probability of a failure is presented for each combination of values that its direct causes can take. All variables are binary and identified by their initials in this example, hence



Pr(P = T)	Pr(P = F)
0.99	0.01

P	Pr(H = T)	Pr(H = F)
T	0.01	0.99
F	0.03	0.97

P	H	Pr(O = T)	Pr(O = F)
T	T	0.10	0.90
T	F	0.08	0.92
F	T	0.06	0.94
F	F	0.05	0.95

P	H	O	Pr(A = T)	Pr(A = F)
T	T	T	0.19	0.81
T	T	F	0.33	0.67
T	F	T	0.27	0.73
…			…	…

Figure 1. An Example Bayesian Network that Depicts Dependencies between Failures and EDC Availability.

occurrence of failures are represented by true (T) or false (F) labels. CPT belonging to availability variable indicates that the probability of achieving availability objective of the customer, $\Pr(\mathrm{A} = \mathrm{T})$ when all three failures occur, $\langle P, H, O \rangle = \langle T, T, T \rangle$, is 0.19 and it increases to 0.33 if we know that HDD failure did not occur, $\langle P, H, O \rangle = \langle T, F, T \rangle$. Similarly, according to its CPT, probability of a OS failure is 0.10 given that a power outage and HDD failure occurred. However, it drops to 0.06 if no power outage is observed.

Reasoning in BNs can be performed in both directions. One may predict causes given symptoms, e.g. determining which failure caused unavailability, or predict symptoms given causes, e.g. estimating availability when certain failures occur [7]. Former is known as diagnostic reasoning, whereas in this study, we focus on the latter which is called predictive reasoning.

III. System Architecture

A high level overview of the system is depicted in Figure 2. Distributed EDC monitors (Figure 2.1) collect relevant parameters such as failure traces or hardware / software characteristics and transfer them to a central repository (Figure 2.2). Actualization of the monitoring system is outside the scope of this study. We assume that data is being collected continuously in a fairly accurate way while BNs are quite tolerant to noise in their parameters [8]. In practice, heartbeat protocol is a well-known approach for fault detection in distributed systems [9], whereas fault tree analysis is an established tool for root cause detection of failures [10].

Parameters collected in the repository are concurrently consumed as training data by the VM availability estimation module, (Figure 2.3) which is described in detail in Section IV. Another input to the estimation module is relevant objectives, e.g. minimum availability, maximum error rate,

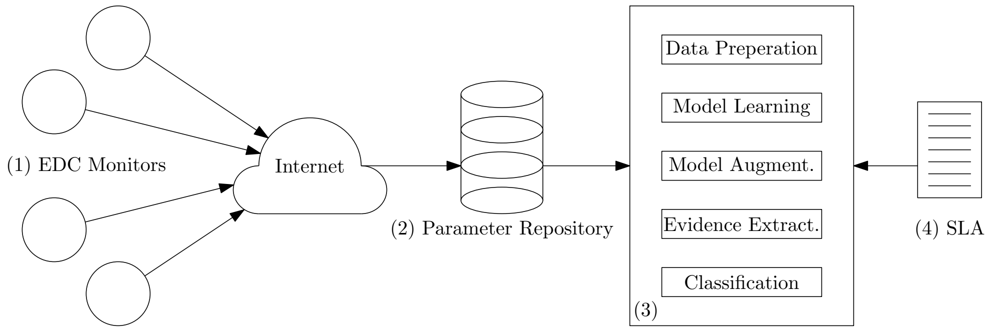


Figure 2. Runtime Architecture for the VM Availability Estimation System.

from the service level agreement (Figure 2.4) in machine-readable format. This can be as simple as an XML file or as advanced as a Smart Contract. Output of the module is a set of estimated probabilities for the satisfaction of SLOs at each VM and EDC. This information is ready to be utilized by a task scheduling algorithm.

IV. VM Availability Estimation

Figure 3 describes the information flow between the components of proposed method. Basic sequence is as follows. After data is prepared (A) from failure traces and BN model is learned (B) from that data, BN is augmented (C) to fit the availability objective at hand. BN model is then used to estimate probability (E) of fulfilling that objective at each EDC. Evidence necessary for classification is extracted (D) from data depending on the characteristics of requested VM and EDC. Following subsections describe each component in detail.

A. Data Preparation

Failure traces from Edge infrastructure are stored in a repository which is a collection of vectors in the form $\mathbf{d_i} = \langle x, y, t_s, t_e, k\rangle$. Here, x and y are unique identifiers of EDC and VM where the failure occurred, t_s and t_e are start and end timestamps of unavailability, and $k \in [1, n]$ is the error code for the cause of failure. Pre-processing of data before model learning consists of aggregation, normalization and discretization. In the first step, number of occurrences of each failure type, k, on each (x, y) pair is counted in failure traces. Then, counts are normalized by runtime duration of that pair so that they indicate frequencies of each failure. A class variable indicating historical availability of the pair is also appended to each record. Availability can be easily computed by using t_s and t_e fields of failure traces. Finally, all numeric attributes are discretized into nominal ones. Attributes are divided into b bins such that each bin of an attribute has the same number of instances (i.e. equal frequency binning). As an example, when $b = 3$, frequency bins can be considered as frequent, occasional, and rare occurrence of a failure type. Availability class, on the other hand, is discretized into binary using a pre-defined threshold. This binary version is only used for initial model learning and the numeric availability values are also saved for rediscretization during model augmentation phase where actual availability objective is used as threshold (see subsection IV-C). Each record can be represented as a vector $\mathbf{r} = \langle r_1, r_2, \ldots, r_n, c\rangle$ where r_i are failure frequency attributes and c is the availability class. In the rest of the paper, we refer to each such vector as an instance.

B. Model Learning

Second step of our approach is to learn an initial BN model of inherent dependencies among various failure causes and their influence on VM availability. Both structure (DAG) and parameters (CPTs) of the BN are automatically learned from historical failure data. The objective in learning BNs is to find a network that best describes the probability distribution over the training data. Since the problem of finding the optimal network is NP-Complete [11], various approximate algorithms are proposed. The simplest solution to the problem is to assume that all attributes are conditionally independent of each other given the class value. This approach is known as naive Bayesian classification. In this study, however, we appeal to a more realistic learning algorithm, Tree Augmented Naive Bayes (TAN) [6], which captures the dependencies among the attributes.

TAN method makes use of conditional mutual information to specify the dependent attribute pairs. In this case, conditional mutual information, $I(r_i, r_j \mid c)$, measures the amount of information that can be obtained about one attribute by observing the other and the class variable. In TAN method, a complete graph is built where vertices are attributes and edges are weighted with conditional mutual information of the incident vertices. Then, a maximum weighted spanning tree is computed on that complete graph to represent the most significant dependencies [6].

Since independence assumption of Naive Bayes is almost never valid in practice, it is significantly outperformed by TAN (see subsection V-A). However, considering inter-attribute dependence also incurs additional computational

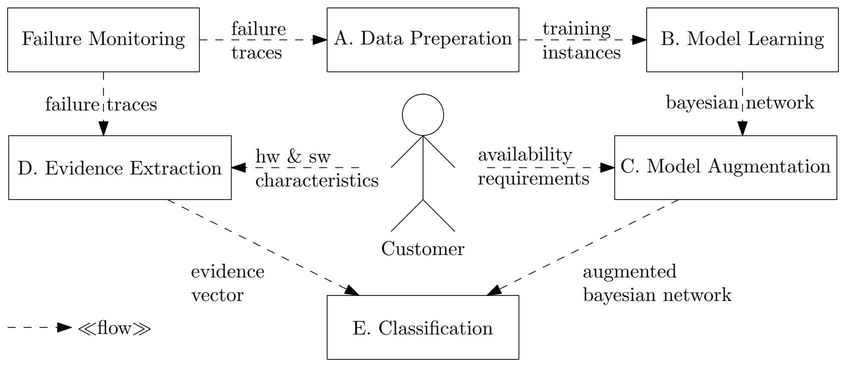


Figure 3. UML Information Flow Diagram for the Proposed Availability Estimation Method.

cost. Time complexity of learning a BN from m instances with n attributes using TAN method is $\mathcal{O}(n^2m)$ which may cause a performance bottleneck if performed at each VM request (see subsection V-C). Thus, we propose a BN augmentation strategy as described in the following subsection.

C. Model Augmentation

Since each VM request has a different availability objective, binary classes of training instances (that indicate achievement of objective) should be reset at each request and BN model should be updated, accordingly. Moreover, new failure events over the time may also change our knowledge and require updating the model. Computational cost of TAN algorithm, however, restrains us from rebuilding the entire model whenever a new evidence is observed or an instance is received to be classified. Here, it should be noted that relatively high time complexity of building model stem from structure learning. If a structure was already given, updating conditional probabilities would only be a bookkeeping problem. Time complexity of CPT update, when parent count of each node is constant (which is 2 in the case of TANs used in this study), is $\mathcal{O}(nm')$ where m' is the number of additional training instances. Thus, we resort to an approximation assuming that the structure of BN would require a change at a much lower rate than the CPTs.

Figure 4 describes the proposed algorithm for model augmentation. After the initial BN is built, the same structure is used in subsequent requests, by rediscretizing the class according to availability objective (lines 5–9) and by only updating CPTs (line 10). Complete structure and parameter rebuilding (line 13) can be triggered in fixed periods or only when the accuracy of estimation drops below a threshold. We implement former alternative in our evaluation.

```
Input: new instances $N$, availability objective $a$, initial
    Bayesian Network $B$
Output: augmented Bayesian Network $B'$
 1: $I \leftarrow I \cup N$ {$I$ is a global set of all instances}
 2: $I' \leftarrow I$ {$I'$ is a temporary variable not to override $I$}
 3: $B' \leftarrow B$
 4: for all $i \in I'$ do
 5:    if $i.numericClass >= a$ then
 6:       $i.class \leftarrow 1$
 7:    else
 8:       $i.class \leftarrow 0$
 9:    end if
10:    updateParameters($B', i$) {Only CPTs}
11: end for
12: if isObsolete($B'$) then {Fixed period or accuracy}
13:    rebuild($B'$) {DAG and CPTs}
14: end if
```

Figure 4. Pseudo Code Description of Model Learning and Augmentation.

D. Evidence Extraction

Let us consider an estimation instance where we want to know whether a given VM will satisfy its availability objective with a certain probability if it is deployed on a given EDC. Before the VM is deployed, we usually do not have any historical data for its runtime failure behaviour. Failure traces of EDC may also be quite limited due to the relative infrequency failures. To overcome these two problems, historical data (failure traces) for other EDCs and VMs that share certain characteristics with instance at hand can be aggregated to synthesize evidence. Set of common characteristics to be considered will be different for each attribute (i.e. failure type).

In practice, binary vectors $\mathbf{v} = \langle v_1, v_2, \ldots, v_\alpha \rangle$ and $\mathbf{e} = \langle e_1, e_2, \ldots, e_\beta \rangle$ are used to represent VM and EDC characteristics, respectively. For instance, if at most eight different Internet service providers (ISPs) are present, e_1, e_2, and e_3 can be used to encode the ISP that an EDC get service from. Similarly, v_1 and v_2 can identify different network connection types (e.g. bridged networking, NAT, etc.) for VMs. In addition, a masking vector $a = \langle a_1, a_2, \ldots, a_{\alpha+\beta} \rangle$ is used to describe which attributes in $\mathbf{v}$ and $\mathbf{e}$ are significant for each failure type. Continuing above example, a certain type of network error (identified by $k \in [1, n]$) may be related to ISP and connection type. This information is represented by a vector $\mathbf{a}^k$ where a^k_1, a^k_2, a^k_3, $a^k_{\alpha+1}$, and $a^k_{\alpha+2}$ are 1, whereas all other components are equal to 0. Then, set S^k in Equation 1 contains only the trace instances having same ISP in EDC and same network connection in VM. Here, $\frown$ is the vector concatenation operator and $\vec{0}$ is a vector, which has all components equal to 0. Exclusive or ($\oplus$) compares concatenated characteristic vector of each trace with the instance to be estimated, whereas masking vector a^k ensures that unrelated characteristics are ignored.

$$S^k = \left\{ s \in [1, m] \;\middle|\; \mathbf{a}^k \wedge \Big(\big(\mathbf{v}^s \frown \mathbf{e}^s\big) \oplus \big(\mathbf{v} \frown \mathbf{e}\big) \Big) = \vec{0} \right\} \quad (1)$$

Next step is to instantiate each attribute of evidence vector $\mathbf{e} = \langle e_1, e_2, \ldots, e_n \rangle$ with its most frequent value (i.e. mode) among the instances in its S set. In Equation 2, $\mathbf{r^i}$ is the attribute vector of an instance i and $[\ldots]$ indicates a multiset by an abuse of notation. Then, kth component of the evidence vector (e_k) is calculated as the mode of multiset that contains kth components of all attribute vectors which belong to an instance in S^k.

$$\begin{gathered} \forall i \in [1, m], \forall k \in [1, n] \\ \mathbf{r^i} = \langle r^i_1, r^i_2, \ldots, r^i_n, c \rangle \\ e_k = \text{mode}\Big(\big[r^j_k \mid j \in S^k \big] \Big) \end{gathered} \quad (2)$$

E. Classification

Final step of estimation is to compute posterior probabilities of both availability classes (i.e. SLO fulfilment and violation) given extracted evidence. We can use Bayesian theorem to compute posterior probabilities and predict the class with the highest probability as shown in Equation 3. $\Pr(c_i)$ and $\Pr(\mathbf{e} \mid c_i)$ can be easily calculated from data or CPTs.

$$c^* = \underset{i=1,2}{\arg\max} \Pr(c_i) \Pr(\mathbf{e} \mid c_i) \quad (3)$$

In practice, posterior probabilities of classes are provided instead of a binary output. This allows taking reliability management decisions based on how strong is the belief that an availability objective will be met or missed.

V. Evaluation

We perform three-fold evaluation of the proposed Edge availability estimation method. In the first part, we validate that taking failure dependencies into consideration, in the form a BN, is indeed beneficial to the accuracy of the estimation. Hence, we only evaluate the model learning component in this part, and compare it to more straightforward learning algorithms. Whereas in the second and third parts, we evaluate the complete methodology presented in Section IV in terms of SLO violations, running time and model accuracy. In all three parts, we used real world failure traces of high performance computing (HPC) servers collected by Los Alamos National Laboratory (LANL), USA. The data set spans 9 years between 1996–2005 and contains 19436 failures of 122 distinct types from 3577 servers that belong to 22 HPC systems [12]. In pre-processing the traces as training instances, number of bins (b) for discretization is chosen to be 5 similar to the Likert scale which is common for mapping a quantitative value to an evaluation. In addition, unknown causes that are observed in $\%5.4$ of the traces, are distributed to other failures in proportion to the frequency of each failure.

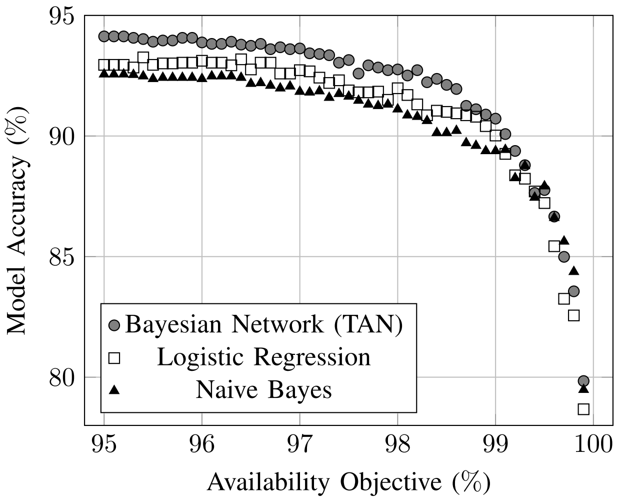


Figure 5. Accuracy Comparison of Learning Algorithms for Varying Availability Objectives.

Although the LANL data set is not collected on an Edge computing infrastructure, it possesses certain characteristics that suits our evaluation scenario well. Similar to Edge, the number of computation nodes is large, they are organized in different sites, and are highly heterogeneous in terms of processor count, memory, hardware architectures, etc., but most are not as powerful as servers in Cloud data centers.

A. Model Learning Accuracy

In order to evaluate the benefit of including dependent failures to the model, we train a Naive Bayes classifier as an alternative to Bayesian Network. Additionally, a discriminative classifier, Logistic Regression, is also trained to act as an additional baseline for the two generative classifiers. We perform 10-fold cross validation and logged accuracy, which is the ratio of number of correctly classified instances to number of all instances. The process is repeated for 100 different values of availability objectives between 90.0% and 99.9%.

In almost all cases, BN model outperforms both Naive Bayes and Logistic Regression. Results are provided in Figure 5 where we omit $[90.0\%, 95.0\%)$ in x-axis since

Table II
NUMBER OF SLO VIOLATIONS FOR EACH CONFIGURATION USING BAYESIAN NETWORK ESTIMATION (BN) OR ONLY PRIOR KNOWLEDGE (PK)

		Chi_1	Chi_2	Chi_3	Chi_4	Wei_1	Wei_2	Wei_3	Wei_4	Log_1	Log_2	Log_3	Gam_1	Gam_2	Gam_3
	μ	54.6	46.2	40.5	35.5	54.6	47.8	48.3	45.4	45.0	59.3	69.3	60.6	44.1	38.5
PK	σ	4.4	4.3	3.4	2.0	3.1	4.0	3.4	3.5	3.5	4.5	4.5	3.2	2.8	3.8
	σ/μ	8.0%	9.2%	8.5%	5.7%	5.7%	8.4%	7.0%	7.7%	7.9%	7.7%	6.5%	5.2%	6.3%	9.8%
	μ	37.9	33.2	30.2	28.4	39.3	30.4	28.3	25.5	26.7	40.4	57.9	40.8	27.8	26.7
BN	σ	3.5	3.0	2.9	2.3	4.0	2.7	3.0	2.0	1.6	4.1	3.6	2.3	2.3	2.7
	σ/μ	9.2%	9.1%	9.5%	8.1%	10.1%	8.9%	10.6%	7.9%	5.8%	10.0%	6.2%	5.7%	8.2%	10.2%
Improvement		30.6%	28.1%	25.4%	20.0%	28.0%	36.4%	41.4%	**43.8%**	40.7%	31.9%	**16.5%**	32.7%	37.0%	30.6%

Table I
PARAMETERS AND STATISTICS OF AVAILABILITY DISTRIBUTIONS

Label	Distribution	Parameters	μ	σ
Chi_1	Chi-Squared	Degrees of Freedom: 3	96.97	2.48
Chi_2	Chi-Squared	Degrees of Freedom: 4	95.97	2.83
Chi_3	Chi-Squared	Degrees of Freedom: 5	95.09	3.11
Chi_4	Chi-Squared	Degrees of Freedom: 6	94.01	3.45
Wei_1	Weibull	Shape: 1.5, Scale: 3.0	97.32	1.81
Wei_2	Weibull	Shape: 2.0, Scale: 3.0	97.35	1.40
Wei_3	Weibull	Shape: 2.5, Scale: 3.0	97.33	1.14
Wei_4	Weibull	Shape: 3.0, Scale: 3.0	97.31	0.97
Log_1	Log-Normal	Shape: 1.0, Scale: 0.5	96.91	1.66
Log_2	Log-Normal	Shape: 1.0, Scale: 1.0	95.46	6.41
Log_3	Log-Normal	Shape: 1.0, Scale: 1.5	91.70	20.95
Gam_1	Gamma	Shape: 3.0, Scale: 0.5	98.49	0.87
Gam_2	Gamma	Shape: 3.0, Scale: 1.0	97.02	1.70
Gam_3	Gamma	Shape: 3.0, Scale: 1.5	95.55	2.57
		EDC Availability:	98.62	8.01

accuracies of all three classifiers are almost constant over that interval. Over interval $[95.0\%, 99.9\%)$, however, a logarithmic decay in the accuracy is observed. This can be explained with the decreasing number of instances that satisfy strict availability objectives and thus growing imbalance in class labels. Logistic Regression, or discriminative approach in general, seems to be more sensitive to this class imbalance since it demonstrates a steeper accuracy decay than the others and is considerably outperformed by even Naive Bayes for availabilities greater than 99.0%. Results clearly indicate that exploiting inherent dependency between failures ensure more accurate availability estimation.

B. Virtual Machine Scheduler Simulation

For the second part of evaluation, we implement a VM scheduling simulator in Java. The scheduler is a prototype to compare effects of estimation methods on any proactive availability optimization algorithm. For each VM request, it identifies EDCs that are estimated to meet the availability objective, i.e. feasible nodes. Feasible nodes are chosen based on the output of VM availability estimation method described in Section IV. To act as a baseline, a simple but common scheduling method is also implemented where prior availability of a node is assumed to carry on. In both cases, VM is scheduled to the minimum cost alternative among feasible nodes where cost of a EDC is directly proportional to its observed availability. Failure times and types are taken from the LANL dataset whereas VM requests are randomly generated and received in equal time-steps. Failure and request times are normalized to the range $[0, 1]$.

140 sets of VM requests are generated with availability objectives that show 14 different configurations of the most frequently used probability distributions in reliability theory, namely; Chi-squared, Weibull, Lognormal, and Gamma [12], [13]. For each set, 10000 VMs are scheduled to EDCs, each of which has a fixed capacity of 3 VMs, so that total capacity of the system (3 VMs $\times$ 3.577 EDCs) is just enough for the demand. LANL servers are utilized as EDCs and their failure and availability data are put to use in the simulation. In evidence extraction (as described in IV-D), we use a simple characteristic vector to identify VMs and EDCs in the same site (i.e. belonging to the same cluster in LANL data set).

Table I lists the distributions and parameters that are used to generate VM availability sets. For each configuration, mean (μ) and standard deviation (σ) of 10 generated sets are also provided for reference. Note that, distributions are for failure rates while availability is calculated as $100 - failure\ rate$. Chosen configurations cover a wide range of possibilities, e.g. fixed scale and variable shape (Wei) or vice versa (Log and Gam). Parameters are chosen in order to guarantee that generated availability objectives can be met with the available EDCs, whose mean and standard deviation is provided at the bottom. Extreme cases of strict (Gam_1) or flexible (Log_3) objectives are also included.

In Table II, we provide mean, standard deviation and coefficient of variation for the number of SLO violations, i.e. assigned VM does not fulfill the availability objective at the end of simulation. Results demonstrate that proposed approach substantially decreases the number of violations in all configurations with improvement rates between 16.5% and 45.3%. Improvement gets better as the standard deviation of a distribution (in Table I) decreases. This is expected since estimating highly variable availabilities is a harder problem. However, even the improvement in extremely variable distributions such as Log_2, Log_3, and Chi_4 is significant which emphasizes the accuracy of learning.

C. Trade-off between Accuracy and Running Time

To examine runtime performance and accuracy of the proposed approach, we focus on the model augmentation component (see subsection IV-C) in the final part of evaluation. The same VM scheduler simulation described in

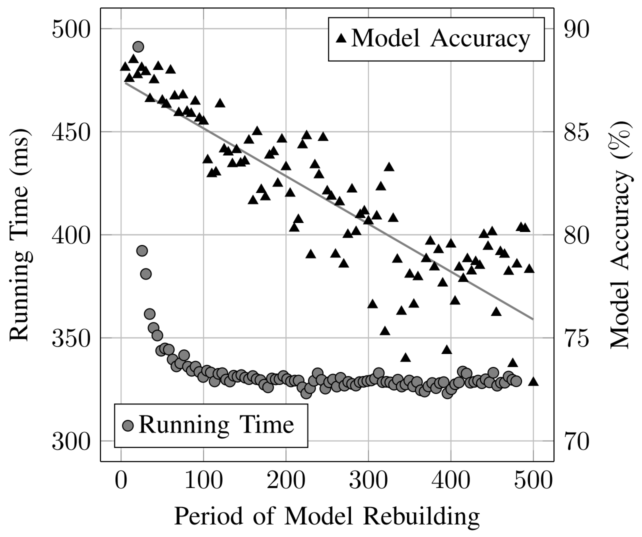


Figure 6. Running Time per Request and Average Model Accuracy for Varying Periods of Model Rebuilding.

subsection V-B is used here using Chi_1 distribution and various model rebuilding frequencies. Figure 6 combines both running time per request and average model accuracy for each period. Period stands for the number of new VM requests to be received in order to trigger complete structure and parameter rebuilding of the BN model. For reference, accuracy is 88.1% and running time $1148\,ms$ when the model is not augmented but rebuilt at each request.

As we extend the period, average running time per request drops rapidly from $1148\,ms$ (excluded in Figure to increase clarity) to around $330\,ms$ where it remains stable. Results indicate that model learning is the larger part of running time and decreasing its frequency can effectively shorten decision time. However, for lower frequencies, performance bottleneck shifts to other components. Experimental results for running time coincide with our claims in subsection IV-C. Accuracy of the BN model, calculated via 10-fold cross validation, on the other hand, demonstrate a steady decrease (indicated with a trend line) as period increases. According to experimental results, running time can be decreased by 70% with no perceivable effect on accuracy by using model augmentation with a rebuilding period of 50 requests for example. We can also conclude that increasing period after a point (roughly 100 requests) is not advisable as it deteriorates accuracy but does not improve running time. It should be noted that, mentioned values for period depend heavily on the failure characteristics of Edge infrastructure. Although we used real world failure data, optimum value can be slightly different for each application whereas the trends would remain similar.

VI. Related Work

In Edge computing research, reliability and QoS management are mostly unexplored areas. None of the well established QoS methods from other fields such as Cloud data centers or computer networks can be applied because they do not work at the scale of Edge and have completely different data flows and dependencies. One of the few studies in the area [14], propose recovery schemes in the case of overload in mobile edge computing (MEC). Main idea is to offload tasks from MEC nodes that are expected to get overloaded soon, to other nodes. Prediction of failures and proactive actions, however, are not taken into account.

Bayesian Networks have been long used in system reliability engineering. Early works [4], [15], [16] rely on expert knowledge to build networks as efficient structure learning mechanisms were not yet mature [17]. Manually defining the dependencies between failures or component of system is an extremely costly and time-consuming task for modern day complex systems. Moreover, it is particularly prone to human errors [17], [18]. Another common method for BN generation is to translate it from so called Fault Trees (FTs) [18]–[20]. FT is a tool that is used to determine the cause of a binary event (e.g. failure), via relationships between other events represented with logic gates [20]. It can be seen as special case of a BN where all probabilities in CPTs are either 0 or 1. Although BNs are automatically created in such studies, causal relations in FTs are nevertheless determined via human experience and professional judgments. Hence, they share the same drawbacks as manual BN generation.

First approach to learn BNs automatically from historical data for estimating system reliability is presented in [21]. Suggested algorithm uses data mining techniques and score-based heuristics to explore the search space of possible BN structures. Although experimental results are promising, quadratic time complexity ($\mathcal{O}(m^2)$) with respect to the number of nodes (m) restrain us to apply this algorithm in the large-scale and latency-sensitive Edge scenario. As explained in IV-B, TAN learning is linear in number of nodes. In [22], Naive Bayes learning from system parameters such as CPU, memory, and disk utilization is implemented to diagnose errors and apply self-healing. A similar approach [3] to ours applies TAN learning to use system level metrics to predict normal or abnormal behaviour in the near future (e.g. in a few minutes). The main difference between the two works lies in the usage scenario. While authors aim to predict and prevent infrequent failures in Cloud Systems in [3], we regard failures as inevitable events in Edge Systems and instead try to estimate the availability level as a consequence of them.

VII. Conclusion

Frequent failures in Edge data centers, due to their distributed nature and lack of advanced support systems, may hinder adoption of Edge computing. Thus estimating failures and taking proactive measures are vital. In this study, an availability estimation method for Edge VMs is proposed. It utilizes historical failure data to extract dependencies between failures and model their influence on VM availability via a Bayesian Network. The model is then used to identify the VMs that would meet the availability objective in SLA. To the best of our knowledge, proposed approach is the first application of a probabilistic model to Edge reliability. In addition to the analytical estimation algorithm, novel approximation techniques to ensure real world applicability, namely model augmentation and evidence extraction, are suggested. Evaluation results demonstrate the advantage of proposed method in comparison to traditional practices in reliability engineering as well as other machine learning models in terms of accuracy and number of SLO violations. An analysis of the trade-off between accuracy and running time of availability estimation is also provided. In the future, we aim to use Temporal BNs in order to capture more information from sequence and timing of failures. Further accuracy improvement can be achieved with more detailed dependency graphs, thus we are planning to test advanced learning algorithms (e.g. K2) as alternative to TAN.

Acknowledgment

The work described in this paper has been funded through the Haley project (Holistic Energy Efficient Hybrid Clouds) as part of the TU Vienna Distinguished Young Scientist Award 2011 and Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *7th ACM SIGOPS Asia-Pacific Workshop on Systems*. ACM, 2016, p. 5.

[3] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *IEEE 32nd International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 285–294.

[4] J. G. Torres-Toledano and L. E. Sucar, "Bayesian networks for reliability analysis of complex systems," in *Ibero-American Conference on Artificial Intelligence*, 1998, pp. 195–206.

[5] A. Darwiche, *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[6] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[7] K. B. Korb and A. E. Nicholson, *Bayesian artificial intelligence*. CRC press, 2010.

[8] M. J. Druzdzel and A. Onisko, "Are bayesian networks sensitive to precision of their parameters," in *Intelligent Information Systems Conference*. Warsaw, Poland: Academic Publishing House EXIT, 2008, pp. 35–44.

[9] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *IEEE Systems Journal*, vol. 7, no. 2, pp. 288–297, 2013.

[10] C. A. Ericson and C. Ll, "Fault tree analysis," in *System Safety Conference*, 1999, pp. 1–9.

[11] D. M. Chickering, "Learning bayesian networks is NP-complete," in *Learning from data*. Springer, 1996, pp. 121–130.

[12] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[13] A. Gorski, "Chi-square probabilities are poisson probabilities in disguise," *IEEE Transactions on Reliability*, vol. 34, no. 3, pp. 209–211, 1985.

[14] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer Systems*, 2016.

[15] D. C. Yu, T. C. Nguyen, and P. Haddawy, "Bayesian network model for reliability assessment of power systems," *IEEE Transactions on Power Systems*, vol. 14, no. 2, pp. 426–432, 1999.

[16] P. Weber and L. Jouffe, "Reliability modelling with dynamic bayesian networks," in *5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*. IFAC, 2003, pp. 57–62.

[17] H. Langseth and L. Portinale, "Bayesian networks in reliability," *Reliability Engineering & System Safety*, vol. 92, no. 1, pp. 92–108, 2007.

[18] H. Boudali and J. B. Dugan, "A discrete-time bayesian network reliability modeling and analysis framework," *Reliability Engineering & System Safety*, vol. 87, no. 3, pp. 337–349, 2005.

[19] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla, "Improving the analysis of dependable systems by mapping fault trees into bayesian networks," *Reliability Engineering & System Safety*, vol. 71, no. 3, pp. 249–260, 2001.

[20] N. Khakzad, F. Khan, and P. Amyotte, "Safety analysis in process facilities: Comparison of fault tree and bayesian network approaches," *Reliability Engineering & System Safety*, vol. 96, no. 8, pp. 925–932, 2011.

[21] O. Doguc and J. E. Ramirez-Marquez, "A generic method for estimating system reliability using bayesian networks," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 542–550, 2009.

[22] Y. Dai, Y. Xiang, and G. Zhang, "Self-healing and hybrid diagnosis in cloud computing," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 45–56.