Learning Spatiotemporal Failure Dependencies for Resilient Edge Computing Services

Atakan Aral, *Member, IEEE* and Ivona Brandić, *Member, IEEE*

Abstract—Edge computing services are exposed to infrastructural failures due to geographical dispersion, ad hoc deployment, and rudimentary support systems. Two unique characteristics of the edge computing paradigm necessitate a novel failure resilience approach. First, edge servers, contrary to cloud counterparts with reliable data center networks, are typically connected via ad hoc networks. Thus, link failures need more attention to ensure truly resilient services. Second, network delay is a critical factor for the deployment of edge computing services. This restricts replication decisions to geographical proximity and necessitates joint consideration of delay and resilience. In this article, we propose a novel machine learning based mechanism that evaluates the failure resilience of a service deployed redundantly on the edge infrastructure. Our approach learns the spatiotemporal dependencies between edge server failures and combines them with the topological information to incorporate link failures. Ultimately, we infer the probability that a certain set of servers fails or disconnects concurrently during service runtime. Furthermore, we introduce *Dependency- and Topology-aware Failure Resilience* (DTFR), a two-stage scheduler that minimizes either failure probability or redundancy cost, while maintaining low network delay. Extensive evaluation with various real-world failure traces and workload configurations demonstrate superior performance in terms of availability, number of failures, network delay, and cost with respect to the state-of-the-art schedulers.

Index Terms—Edge computing, failure resilience, quality of service, dependency learning, dynamic Bayesian networks

1 Introduction

Edge computing refers to the technologies for computation at the edge of the network by extending the cloud with resource-constrained and distributed servers. It renders significantly lower response times possible since the computation is carried out in proximity of where inputs are produced and/or outputs are consumed [1]. The demand for edge computing grows due to the proliferation of mobile computing and Internet of things technologies as they multiply the amount of data produced and consumed at the edge of the network. Edge resources can be utilized either by end devices to offload code or by cloud services to create proxies. In the former case, processing power of the end devices is extended and their power consumption is decreased. Whereas in the latter, response time is decreased and backhaul bandwidth is preserved since most of the traffic flows through the local area network. Failure resilience of edge computing services is still an open issue and a big obstacle to the adoption of the paradigm [2] particularly in conjunction with the cost of redundancy. Service disruptions cause significant revenue loss in a business setting. In 2017, a four-hour outage of AWS is reported affecting 54 of the top 100 online retailer services, which lost $150 million in total [3]. Worse still, contemporary services are getting less tolerant to downtime: average cost of a data center outage has increased from $505,000 in 2010 to $740,000 in 2016 [4].

We observe more frequent failures at edge servers in comparison to cloud counterparts due to geographical dispersion, which complicates management and maintenance, and absence of advanced support systems such as fully duplicated electrical lines with transfer switches, diesel backup generators, clean agent fire suppression gaseous systems, and direct liquid cooling. Moreover, the limited availability of computation and storage resources restricts redundancy at the edge. Particularly low reliability can be expected in future smart contract and blockchain-based edge computing architectures, where arbitrary resources including user equipment can be leased [5], [6]. Link failures should also be taken into account because edge servers are typically connected through less reliable networks (e.g., public wireless networks) than centralized deployments such as cloud data centers. Many failure resilience mechanisms designed for cloud computing, only focus on node failures due to very high reliability of intra data center networking.

Edge services are typically sensitive to computation and communication delays and require near real-time interaction, which adds another dimension to failure resilience. Deploying edge services on servers selected solely to avoid failures might result in the violation of quality of service (QoS) requirements and particularly delay constraints. Thus, failure resilience and network delay have to be optimized jointly. This is not the case for general distributed systems, such as grid computing, where the tasks can be executed on any node regardless geographical location or delay. Moreover, resilience techniques such as re-execution

• *The authors are with the Institute of Information Systems Engineering, Vienna University of Technology, 1040 Vienna, Austria. E-mail: {atakan.aral, ivona.brandic}@tuwien.ac.at.*

Manuscript received 1 July 2020; revised 20 Oct. 2020; accepted 30 Nov. 2020. Date of publication 22 Dec. 2020; date of current version 11 Feb. 2021. (Corresponding author: Atakan Aral.) Recommended for acceptance by P. Balaji, J. Zhai, and M. Si. Digital Object Identifier no. 10.1109/TPDS.2020.3046188

or checkpointing that are widely used in other distributed systems may not be efficient or comparably effective in an edge scenario due to high computational overhead and delay. Thus, there is a need for resilience techniques that take unique features and limitations of edge computing into consideration so that cloud grade availability is possible [7].

In this work, we exploit spatiotemporal failure dependencies among edge servers to improve the failure resilience of services with minimum possible redundancy. To this end, we focus on the probabilistic analysis of concurrent failures. Previous work in the field of system reliability already showed a correlation between failures in various distributed computing systems [8], [9] including edge computing [10]. Therefore, having replicas deployed at edge servers that probabilistically fail in overlapping periods will deteriorate or even nullify availability benefits of replication. To avoid this, we propose a machine learning approach to compute the joint failure probability (JFP) of edge servers. We model failure dependencies as a dynamic Bayesian network (DBN) trained from past traces. Then, we employ an efficient inference algorithm to compute the JFP of a given service deployment. Finally, we combine JFP with the link failure probability (LFP) that is based on the edge network topology to obtain overall service failure probability (SFP).

Furthermore, we propose *Dependency- and Topology-aware Failure Resilience* (DTFR) algorithms, which optimize the deployment of services on edge servers in terms of failure probability, response time, and number of replicas. The main idea is to deploy active replicas at the most proximate servers in the first stage, and then to create as many standby replicas as needed based on SFP to fulfill availability requirements. Consequently, the minimum possible response time would be achieved during the failure free period, which constitutes most of the service run time. We evaluate proposed algorithms using multiple real-world failure traces with various availability characteristics. The results demonstrate the effectiveness of our approach with respect to state-of-the-art baselines in terms of service downtime, number of failures, network delay, and redundancy cost. Briefly, the main contributions in this work are as follows.

- A machine learning mechanism to forecast outages in replicated, near real-time edge services,
- The replication and scheduling algorithms for such services that ensure failure resilience and QoS.

Our hypothesis is that there exist spatiotemporal dependencies among edge computing failures to such an extent that dependency-aware failure prediction and task replication results in substantially higher resilience. We believe, this work is the first attempt to minimize the failure probability of edge computing services under QoS or cost constraints. Besides, it is the first analysis of dependent failures in large-scale edge computing systems, to the best of our knowledge. This work builds on our previous study of edge computing failures [10], which introduces correlated node failures in edge computing. In this work, we additionally incorporate link failures as well as delay considerations. In the rest of the paper, we first give an overview of our approach in Section 2. Then in Section 3, we formally define the addressed problem. We propose a failure model in Section 4 and present DTFR algorithms in Section 5. Experiments and numerical results are discussed respectively in Sections 6 and 7. Finally, we review the related work in Section 8 and conclude the paper in Section 9.

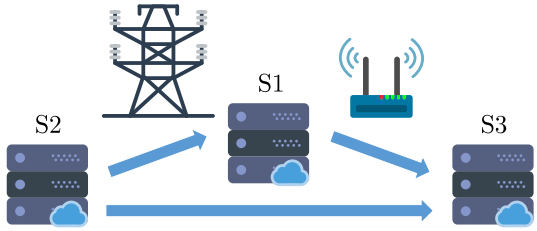


Fig. 1. The running example scenario to illustrate failure dependencies.

2 Approach

2.1 Use Case Scenario (Running Example)

In this work, we consider *InTraSafEd 5G – Increasing Traffic Safety with Edge and 5G* project[1] as our use case scenario. The project aims to improve traffic safety through real-time video analytics at geographically distributed smart traffic lights in Vienna. A very high level of availability (99.99 percent) is required particularly for the detection of humans and animals on crosswalks. Although 1369 such smart traffic lights could be available in Vienna in the future, the service provider has a limited budget for edge computing resources. As a trivial example, consider three edge servers illustrated in Fig. 1, none of which satisfying the availability requirement alone. The limited budget allows at most two replicas of the video analytics to be deployed at these servers; however, the dependencies between the servers might nullify the availability benefits of replication if the replicas are placed indifferently. Specifically, S1 and S2 are powered by the same electricity grid causing joint failures. Similarly, S1 and S3 share the same network connection. In addition, edge servers are configured to dispatch tasks to others (indicated with arrows) when overloaded. This results in cascading failures and further temporal dependencies. Note that, dependency causes are not usually known in real systems; therefore, we propose automatic extraction from past traces.

2.2 Edge Computing Failures

Two broad categories of spatiotemporal correlations between node failures are considered in literature [8]. In multiplication, failures occur simultaneously in multiple servers due to a common cause, whereas in propagation, the failure in a server eventually triggers further failures in others. In the edge computing context, examples of dependencies that belong to the former category include: a network failure affecting multiple servers in the same physical/virtual network; a power outage affecting multiple servers in the same grid; and multiple servers deployed in hostile locations failing due to environmental interference. Yet, cascading failures, which occur after a single failure and later spread due to workload redistribution, belong to the failure propagation category. Reattempting failed tasks often amplifies such failures. These factors are not

1. https://newsroom.magenta.at/2020/01/16/5g-anwendungen/

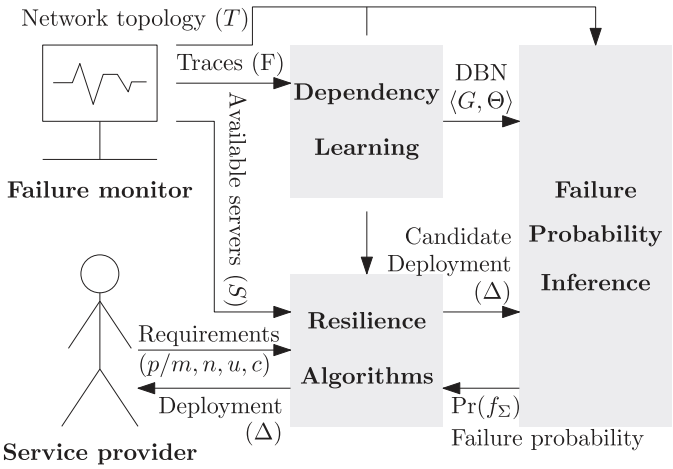


Fig. 2. Main components and data flow in the proposed technique.

transparent to the user and it is exhaustive to take measures for each.

Post failure recovery mechanisms such as re-execution or checkpointing alone are not sufficient in the edge scenario due to their high overhead and the limited computational capacity of edge servers. Additionally, the unstructured, dynamic, and heterogeneous edge computing architecture hinders approaches based on shared risk groups or availability zones. Due to strict locality requirements, all admissible candidates for replication may belong to a few such groups or zones , which adds to inherent dependency. Another major reason for edge service unavailability is link failures. Communication at the edge is typically enabled via ad hoc networks, which in combination with mobility, might result in intermittent connectivity. Previous work on consistency in edge computing [11] shows that substantial communication is needed to keep the state consistent between multiple copies. To this end, edge servers can communicate either directly or through a central server. For applications with strong consistency requirements, a link failure on the path between two replicas would result in service unavailability.

2.3 Methodology Overview

In Fig. 2, we present a high-level overview of the main components of our approach and data flows. In this architecture, dependency learning (Section 4.2) is a one-time process that occurs in a resource-rich environment such as the cloud. It receives past failure traces (F) and trains a DBN $\langle G, \Theta \rangle$ based on these. Failure probability inference (Sections 4.3 and 4.4), in turn, utilizes the DBN as well as the network topology (T) at runtime to compute the failure probability of a candidate deployment ($\Pr(f_\Sigma)$). This component can either be deployed on the cloud and provided as a programming interface or run at the edge as part of the service provider software (e.g., scheduler). The candidate deployments (Δ) to be evaluated by this module are continuously generated by the resilience algorithms depending on currently available servers (S) and network conditions (T). After several trials, these algorithms return the optimized deployment in terms of requirements of the service provider such as maximum acceptable failure probability (p) or the number of replicas (m). Resilience algorithms should run on an edge server due to their interactive nature. Therefore, our design goal for these algorithms is to achieve low computational complexity. This is demonstrated both theoretically (Section 5) and empirically (Section 7.2) in the rest of the paper. Further notations used in this paper are defined in Table 1.

TABLE 1
Common Notation and Symbols Used Throughout the Paper

Symbol	Definition
δ	A deployment $\langle c, s \rangle$ of a service copy and an edge server
Δ	Set of all deployments of a service, $\delta \in \Delta$
λ	A link on the path between two deployments
Λ	Set of all links on the path between two deployments
S	Set of currently available edge servers
S_Δ	Set of edge servers that host a service, $S_\Delta \subseteq S$
T	Network topology among the edge servers, $T = \langle S, L \rangle$
c	A copy or replica of the service
p	Maximum acceptable failure probability of a service
p_{min}	Failure probability of an optimum deployment
m, n	Number of copies and active copies for a service
u	Edge server that is closest to the service users, $u \in S$
f_s^t	Binary random event representing a server failure at time t
f_δ	Binary random event representing a deployment failure
f_Δ	Binary random event representing joint failure of all deployments
f_λ	Binary random event representing a link failure
f_Λ	Binary random event representing a path failure
f_Σ	Binary random event representing an overall service failure
F	Set of all f_s^t for all $s \in S$ and for all t
G, Θ	Structure graph and parameters of a DBN

2.4 Practical Implementation

We make no assumption on the virtualization technology and believe that the proposed scheduling algorithms will be applicable to virtual machines as well as more light-weight implementations such as containers or pods. Hence, we use the generic terms of *task*, *copy*, or *replica*. State-of-the-art edge orchestration systems such as K3S and KubeEdge are based on Kubernetes [12]. Failure probability inference and resilience algorithm modules of DTFR in Fig. 2 are to run on the Kubernetes master and to extend the scheduler and replication controller modules. Dependency learning, on the other hand, can be executed on a more resource-rich server and the pre-trained model can be stored as a volume at the master. To the best of our knowledge, the aforementioned systems do not extend the replication capabilities of Kubernetes. DTFR requires support for dynamic number of replicas and active-standby replication. In Kubernetes, the former is possible via horizontal pod autoscaler and latter via a readiness probe. Failure forecasts is a valuable input to the scheduler and replication controller of any edge computing service to make better-informed decisions about resilience. While we utilize forecasting to evaluate replication plans at deployment time, it can also be directly employed at execution time to trigger proactive failure mitigation mechanisms. To this end, the inference module can be recalled either periodically or when a failure is detected.

It should also be noted that, the proposed techniques for edge service resilience are probabilistic in nature, thus they are not to guarantee absolute availability for safety-critical applications. They do, however, promise satisfactory QoS under a limited budget for most prospective real-time edge computing applications. As a reference, Open Data Center Alliance defines in *Standard Units of Measure for IaaS* report that highest category of cloud data centers (i.e., Platinum) must offer 99.99 percent availability. The same monthly availability level is also promised in Amazon EC2 SLA. According to our evaluation, DTFR algorithms can achieve comparable QoS levels despite highly unreliable edge resources.

3 Problem Definition

3.1 Node Failures

Services deployed at the edge have various resilience requirements. These are often communicated through minimum service level, number of nines, or maximum acceptable downtime. On the other hand, failure characteristics of edge servers can be represented with mean time between failures (MTBF) hazard rate, availability, etc. In an attempt to standardize and simplify the terminology, we introduce the notion of a deployment pair, $\delta = \langle c, s \rangle$, and its failure, f_δ, as a binary random event. A deployment pair (*deployment* for short in the remainder of the paper) consists of a copy of the service (c) and an edge server (s) which hosts that copy.

We further define an edge service as a set of deployments, Δ. Each $\delta \in \Delta$ runs a copy of the service. Joint failure probability, $\Pr(f_\Delta)$, can be stated in different ways based on the availability definition of the client. For instance, in active-standby replication given in the left part of (1), the service is assumed available unless all m deployments fail, since a standby deployment takes over when the active one fails. In load sharing replication, however, all deployments are active and share the workload. As given in the right part of (1), the service is available as long as at least n deployments out of m are active. In other words, up to $k = m - n$ failed deployments are tolerated.

$$\Pr(f_\Delta) = \Pr\left(\bigcap_{\delta\in\Delta} f_\delta\right), \quad \Pr(f_\Delta) = \Pr\left(\bigcup_{\substack{D\subseteq\Delta \\ |D|=k+1}} \bigcap_{\delta\in D} f_\delta\right) \tag{1}$$

In this work, we assume single-component edge services where all copies execute the same task, however, definitions in (1) can be easily generalized to a multi-component case as shown in (2). Here, K is the set of service components and Δ^κ is the set of all deployments that run component κ. It is also possible to define a custom JFP function where each component has a different availability definition or some components are noncritical and have no availability impact.

$$\Pr(f_\Delta) = \Pr\Big(\bigcup\nolimits_{\kappa\in K} f_{\Delta^\kappa}\Big), \;\; \Delta^\kappa = \{\langle c, s\rangle \in \Delta \,|\, c \leftarrow \kappa\} \tag{2}$$

3.2 Link Failures

The failure probability, $\Pr(f_\lambda)$, of a link, λ, is defined as its unavailability, that is, downtime divided by total time. The end points of the Internet such as edge servers typically communicate through a network path that can be altered unpredictably due to failures or load balancing decisions. Although the changes can be frequent especially in the case of programmable networks, we assume that new paths would have comparable length and hence failure probability. Thus, we compute the failure probability of the initial path between each pair of replicas, Λ, as the probability that at least one link fails as given in (3). Then, we define the SFP as the union of all node and link failures as shown in (4).

$$\Pr(f_\Lambda) = \Pr\Big(\bigcup\nolimits_{\lambda\in\Lambda} f_\lambda\Big) = 1 - \Pr\left(\bigcap_{\lambda\in\Lambda} \neg f_\lambda\right) \tag{3}$$

$$\Pr(f_\Sigma) = \Pr\Big(f_\Delta \cup \bigcup f_\Lambda\Big). \tag{4}$$

3.3 Optimized Deployment

We believe, failure forecasting comes in useful for management of edge resources in various stages. It can be used (i) at design time to evaluate software models in terms of resilience; (ii) at deployment time to compare replication and deployment alternatives; or (iii) at execution time to take measures (e.g., migrate, replicate etc.) before failures. Among these possible use cases, we focus on optimizing the failure resilience at service deployment as the second part of the problem. Given a set of available edge servers, S, we aim to minimize either SFP or number of deployments.

Optimization Problem 1 (OP1). Number of copies to be deployed is predefined and objective function minimizes the SFP of deployment set. First constraint in (5) states that each deployment is between a service copy and a server, whereas the second one ensures that the total number of copies is m.

$$\begin{aligned} \underset{\Delta}{\text{minimize}} \quad & \Pr(f_\Sigma) \\ \text{subject to} \quad & \forall\delta \in \Delta(\delta = \langle c, s\rangle \wedge s \in S), \; |\Delta| = m. \end{aligned} \tag{5}$$

Optimization Problem 2 (OP2). Maximum acceptable failure probability, p, is predefined and objective function minimizes deployment set cardinality (i.e., copy count). Second constraint in (6) satisfies the resilience requirement.

$$\begin{aligned} \underset{\Delta}{\text{minimize}} \quad & |\Delta| \\ \text{subject to} \quad & \forall\delta \in \Delta(\delta = \langle c, s\rangle \wedge s \in S), \; \Pr(f_\Sigma) \leq p. \end{aligned} \tag{6}$$

4 Failure Model

4.1 Joint Failure Probability

JFP is the probability that all copies of an edge service are unavailable due to concurrent failures at the servers in which they are hosted. Note that, this definition corresponds to the service interruption definition of active-standby replication in (1). Techniques described in this section can also be applied to load sharing, but we omit this scenario for brevity. There exist efficient and accurate algorithms in distributed systems literature to forecast availability or marginal failure probability (MFP) of a single

deployment, $\Pr(f_\delta)$. Some example approaches include use of recent availability ratio [13], support vector machines [14], and probabilistic graphical models [15]. We use the retrospective unavailability of an edge server to estimate its MFP. However, JFP is substantially harder to forecast unless independence is assumed. A naive solution to the JFP computation problem is to assume that the failure of a deployment is independent of failures at other deployments. In this case, it is sufficient to compute and store MFP of each deployment resulting in $\mathcal{O}(|\Delta|)$ probabilities as shown in (7). In contrast, one may assume that each deployment is dependent to all others and compute JFP via chain rule as in (8). In this case, $\mathcal{O}(2^{|\Delta|})$ probabilities must be computed.

$$\Pr\left(\bigcap_{\delta\in\Delta} f_\delta\right) = \prod_{\delta\in\Delta} \Pr(f_\delta) \tag{7}$$

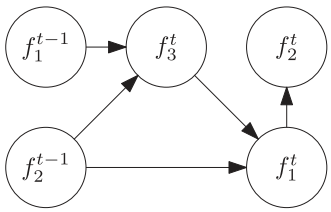
$$\Pr\left(\bigcap_{\delta\in\Delta} f_\delta\right) = \prod_{i=1}^{|\Delta|} \Pr\left(f_{\delta_i} \,\middle|\, \bigcap_{j=1}^{i-1} f_{\delta_j}\right). \tag{8}$$

Both of these extreme solutions, however, have substantial shortcomings. The former, while being computationally efficient, ignores valuable information about concurrent failures. The latter, on the other hand, has high time and space complexity as well as sensitivity to noise from coincidental correlations. Consequently, we make use of probabilistic graphical models in order to model the most significant conditional dependencies along with uncertainty in a compact and efficient way. More specifically, we model spatial and temporal failure dependencies via a dynamic Bayesian network (DBN) and make inferences with DBN via an algorithm based on variable elimination technique.

4.2 Dynamic Bayesian Networks

Among other probabilistic graphical models, we choose DBN for our purpose mainly because of its capability to represent temporal dependencies between events, unlike regular Bayesian networks for example. This is crucial to capture cascading failures, which are dependent but occur at different times. DBN infers not only dependencies themselves but also the direction of causality because it incorporates temporal information [16]. Thus, it can distinguish between causes and effects. Failure events are nonlinear, so their dependency can be captured by DBN but not by linear estimators such as Kalman filters [8]. Moreover, DBN hold performance improvements with respect to hidden Markov models in which the number of states grows exponentially.

A DBN is defined as the pair $\langle G, \Theta\rangle$ for a set of random variables $R = \{r_1^t, r_2^t, \ldots, r_n^t\}$, where $t \in \mathbb{N}$ is the time step. Here, G is a directed acyclic graph (DAG) with vertices representing variables at different time steps and links representing their dependencies. According to the independence assumption in Bayesian networks, each variable r_i^t is directly dependent on its parents in G and independent of its non-descendants given these parents. The second element of the pair, Θ, is a set of probabilities for each variable conditional to its parents. There exists a parameter $\theta \in \Theta$ for each possible combination of values that r_i^t and its parents can take, such that $\theta = \Pr(r_i^t \mid \text{parents}(r_i^t))$.



(a) Structure

f_2^{t-1}	f_3^t	$\Pr(f_1^t)$	$\Pr(\neg f_1^t)$
T	T	0.10	0.90
T	F	0.08	0.92
F	T	0.06	0.94
F	F	0.05	0.95

(b) Parameters

Fig. 3. Dynamic Bayesian network for the running example scenario.

In our case, the variables for which DBN is defined, are binary failure events of servers (i.e., $R = \{f_1^t, f_2^t, \ldots, f_n^t\}$). There may be multiple variables in DBN that correspond to the same server but at different times. This way, DBN can effectively represent spatial (btw. edge servers) and temporal (btw. time steps) dependencies of edge computing failures. Moreover, it can be queried to estimate the future joint failure probability of certain servers. Let us illustrate how failure dependencies in our video analytics scenario (Fig. 1) are modeled. Fig. 3a is the simplified structure of a corresponding DBN. Joint failures due to the shared electricity or network result in dependent failure events $f_1 - f_2$ and $f_1 - f_3$ at the same time step t. Direction is trivial in concurrent dependencies. In addition, cascading failures are represented with dependencies in consecutive time steps ($t-1$ and t). Furthermore, in Fig. 3b, we provide the conditional probability table (CPT) for S1. As an example interpretation from the CPT, failure probability of S1 at time t given that S2 failed in the previous at $t-1$ and that S3 did not fail at t is $\Pr(f_1^t \mid f_2^{t-1} \neg f_3^t) = 0.08$. In the proposed technique, both DBN structure and CPTs are automatically trained from past failure traces. Traces are ordered chronologically and clustered into fixed-length time steps, such that overlapping failures are regarded as concurrent. Failures in consecutive time steps, on the other hand, are used to infer spatiotemporal dependencies. We employ simulated annealing heuristic to search for DAGs that represent dependencies accurately. Practical implementation details about DBN learning are given in Section 6.

4.3 Joint Probability Inference

Given a DBN model, $\langle G, \Theta\rangle$, we are interested in inferring the joint probability of certain events. More specifically, we aim to compute the failure probability of the edge servers that are to be allocated by a given service deployment Δ. The computed value (i.e., JFP) in (9) is treated as the forecast.

$$\Pr(f_\Delta) = \Pr\left(\bigcap_{s\in S_\Delta} f_s\right), \quad S_\Delta = \{s \in S \mid \langle c, s\rangle \in \Delta\}. \tag{9}$$

Independence assumption of Bayesian networks states that a variable is conditionally independent of its non-descendants, given its parents. This allows us to factorize the joint distribution of all variables by conditioning each variable only on its parents in the DBN. This is given in (10) where P_s is the parent set of variable f_s. Note that, significantly fewer conditional variables are needed with respect to (8), decreasing from $\mathcal{O}(|S|)$ to $\mathcal{O}(|P_s|)$.

$$\Pr\left(\bigcap_{s\in S} f_s\right) = \prod_{s\in S} \Pr\left(f_s \,\middle|\, \bigcap_{\gamma\in P_s} f_\gamma\right). \tag{10}$$

All these probabilities are already available in Θ. Hence, one way of computing the JFP is to leave interested variables (S_Δ) and marginalize out all others ($S \setminus S_\Delta$) by summing up the probabilities for all possible combinations of them.

$$\Pr(f_\Delta) = \sum_{S \setminus S_\Delta} \prod_{s \in S} \Pr\left(f_s \,\middle|\, \bigcap_{\gamma \in P_s} f_\gamma \right). \tag{11}$$

In (11), we sum the probability over all $2^{|S \setminus S_\Delta|}$ possible instantiations of uninterested variables. The summation can be computationally optimized in several ways. Below introduced steps significantly reduce the time-complexity of exact inference. Resulting performance suffices for the size and complexity of DBNs that are learned from data in our evaluation, as described in Section 6. However, for models with greater number of variables (> 1000) and allowed parents per variable (> 5), approximate inference algorithms can be useful [17], including sampling techniques such as *Monte-Carlo*, or variational inference algorithms such as *mean field*.

Algorithm 1. ANCESTRAL − GRAPH

Input DBN structure: $G = \langle V, E \rangle$; Deployment servers: S_Δ
Output Ancestral graph for S_Δ: A = $\langle V', E' \rangle$
1: $V' \leftarrow \emptyset$, $E' \leftarrow \emptyset$ {A is initially a null graph}
2: $Q \leftarrow S_\Delta$ {Initialize the queue}
3: **for all** $q \in Q$ **do** {While the queue is not empty}
4: $P \leftarrow \{p \in V \mid \langle p, q \rangle \in E\}$ {Parents of q in G}
5: $Q \leftarrow \{Q \cup P\} \setminus \{q\}$ {Queue P and dequeue q}
6: $V' \leftarrow V' \cup \{q\}$ {q belongs A}
7: $E' \leftarrow E' \cup \{\langle x, y \rangle \in E \mid y = q\}$ {Links to q belong A}
8: **end for**

First, some of the variables in $S \setminus S_\Delta$ may be independent of, thus have no contribution to the joint probability of the variables in S_Δ. More specifically, we only need the variables that are ancestors of at least one variable in S_Δ according to the *d-separation* algorithm [18]. Hence, we build a subgraph of the original DBN which consists of only concerned variables (S_Δ) and their ancestors. Extraction of this so-called ancestral graph is described in Algorithm 1. Once we obtain the set of ancestor nodes V', it can be safely used instead of S in (11). Consider the DBN in Fig. 3a and assume that we need to deploy a service with two copies, c_1 and c_2. Among other alternatives, let us evaluate the failure resilience of deployment set $\Delta = \{\langle c_1, s_1 \rangle, \langle c_2, s_3 \rangle\}$, so we are interested in the servers $S_\Delta = \{s_1, s_3\}$ and their JFP, $\Pr(f_{s_1} f_{s_3})$. From Algorithm 1, the ancestral graph of S_Δ contains the variables f_1^{t-1}, f_2^{t-1}, f_1^t, and f_3^t. Hence, we can factorize and marginalize the joint probability as follows.

$$\begin{aligned} \Pr(f_{s_1} f_{s_3}) &= \sum_{f_1^{t-1}} \sum_{f_2^{t-1}} \Pr(f_1^{t-1} f_2^{t-1} f_1^t f_3^t) \\ &= \sum_{f_1^{t-1}} \sum_{f_2^{t-1}} \Pr(f_1^{t-1}) \Pr(f_2^{t-1}) \Pr(f_1^t \mid f_2^{t-1}) \Pr(f_3^t \mid f_1^{t-1} f_2^{t-1}). \end{aligned} \tag{12}$$

As a second performance optimization, we implement a *variable elimination* algorithm, which reduces the number of summation steps via dynamic programming. Details and time-complexity are discussed in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3046188.

4.4 Link Failure Probability

Contrary to node failures, we model link failures as independent random events. Independence assumption is reasonable in this case since link failures are not as multi-dimensional as node failures in terms of their variety, multiplication, or propagation. To support this claim, we analyzed the cross-correlation of link failures from two real-world systems. The first data set is collected from the computing system MPP2 located at the Pacific Northwest National Laboratory [19]. It contains 121 link and 5591 node failures between 2003 and 2008. Our analysis of link failures resulted in an unnormalized cross-correlation function values in the range [0,0.15] with a median of 0.03, whereas node failures are more significantly correlated up to 0.83 with a median of 0.26. We repeated our analysis on another data set by Telecom ParisTech and Cisco, collected from a system with a typical topology of a content service provider [20]. Similar to the first analysis, we detected no correlation between the 131 link failures that occurred in January 2018.

For simpler notation, we consider an edge network with a tree topology and the failure of a single link on the path results in disconnection. However, formulations in this section can be easily generalized to any network topology. According to the MEC architecture defined by ETSI, the edge orchestrator, which is responsible for the service deployment, is aware of the network topology between the edge servers. Thus, we are able to compute the LFP of an edge service, given the network paths between the deployed nodes and failure probability of each link on these paths. We compute the failure probability of a path, λ in (13). Finally, overall service failure probability (SFP) is defined in (14) as the probability that the nodes jointly fail (JFP) or at least one path fails (LFP). LFP can be computed in constant time as it is independent of the number of candidate edge servers.

$$\Pr(f_\Lambda) = 1 - \prod_{\lambda \in \Lambda} \Pr(\neg f_\lambda) \tag{13}$$

$$\Pr(f_\Sigma) = 1 - \Pr(\neg f_\Delta) \prod_{\Lambda} \Pr(\neg f_\Lambda). \tag{14}$$

5 Resilient Service Deployment

5.1 Stage 1: Topology Awareness

First stage of the resilient service deployment given in Algorithm 2 deploys the active copies at edge servers with minimum network delay. This guarantees that during the failure-free execution, lowest possible response time is achieved. We do not deploy standby replicas in this stage due to the following reasons. First, this may not be the optimal in terms of joint failures due to the high possibility of dependency between edge servers in proximity. This would also quickly consume the limited resources at the areas with high number of users. Finally, since failures are relatively

rare, higher response times can be tolerated in short failure periods of active replicas. $\mathrm{OPT}-\mathrm{RT}$ starts with an empty two-dimensional array (line 1). Then, it iterates over all servers and computes their distance to the user (line 4). The distance and server is stored in an array (line 5), which is then sorted by distance (line 7). A new copy of is deployed at the top n servers in terms of network distance (lines 9-12). The time-complexity of $\mathrm{OPT}-\mathrm{RT}$ is $\mathcal{O}(N \log N)$ due to sorting, where $N = |S|$ is the number of available servers.

Algorithm 2. $\mathrm{OPT}-\mathrm{RT}$

Input Available servers: S; Initial copy: c; Number of active copies to deploy: $n \leq m$; Network topology graph: $T\langle S, L\rangle$; Most proximate server to the user: $u \in S$
Output Partial deployment set: Δ
1: $O \leftarrow [][]$ {Initially array O is empty}
2: $i \leftarrow 0$ {Next index of O}
3: **for all** $s \in S$ **do** {For each candidate server}
4: $d \leftarrow \mathrm{DISTANCE}(T, u, s)$ {Shortest path length to u}
5: $O[i][0] \leftarrow s, O[i][1] \leftarrow d, i \leftarrow i + 1$
6: **end for**
7: $O \leftarrow \mathrm{SORT}(O, 1)$ {Sort O by $O[][1]$, i.e., distance}
8: $\Delta \leftarrow \emptyset$ {Initially deployment set is empty}
9: **for** $i = 0$ to $n - 1$ **do** {First n items in O}
10: $c' \leftarrow \mathrm{CLONE}(c)$ {Create another copy of c}
11: $\Delta \leftarrow \Delta \cup \{\langle c', O[i][0]\rangle\}$ {Add a new deployment}
12: **end for**

Algorithm 3. $\mathrm{OPT}-\mathrm{FP}$

Input Available servers: S; Initial copy: c; Number of copies to deploy: m; Partial deployment set: Δ
Output Optimum deployment set: Δ; JFP of that set: p_{min}
1: $i \leftarrow m - |\Delta|$ {Number of copies yet to be deployed}
2: $\mathcal{C} \leftarrow \{C \subseteq S \setminus S_\Delta \,|\, |C| = i\}$ {i-subsets of candidates}
3: $p_{min} \leftarrow +\infty$
4: **for all** $C \in \mathcal{C}$ **do** {For each candidate set}
5: $p \leftarrow \Pr(f_{C \cup S_\Delta})$ {Compute as in (14)}
6: **if** $p < p_{min}$ **then** {If better than current best}
7: $p_{min} \leftarrow p$ {Update minimum p}
8: $C' \leftarrow C$ {Update best candidates set}
9: **end if**
10: **end for**
11: **for all** $s \in C'$ **do** {For each server in C'}
12: $c' \leftarrow \mathrm{CLONE}(c)$ {Create another copy of c}
13: $\Delta \leftarrow \Delta \cup \{\langle c', s\rangle\}$ {Add a new deployment}
14: **end for**

5.2 Stage 2: Dependency Awareness

For the second stage, we introduce two variants that correspond to OP1 and OP2 from Section 3.3. Both algorithms start from the partial deployment set computed by $\mathrm{OPT}-\mathrm{RT}$ and deploy remaining copies based on SFP. The first one, described in Algorithm 3, finds the deployment set of given cardinality m with the lowest SFP. This is useful when the service provider has a fixed budget and expect the highest possible resilience under current state of the edge servers. $\mathrm{OPT}-\mathrm{FP}$ iterates over all i-subsets of available servers in order to identify the combination that yields the lowest JFP (lines 3–9). Then, a new copy is deployed on every server in this identified combination (lines 11–14). Considering $|S| \geq m$, there exists $\mathcal{O}(2^N)$ subsets and hence calls to $\Pr(f_\Sigma)$ calculation, which itself has the time-complexity of $\mathcal{O}(N\,2^M)$. Thus, the time-complexity of the algorithm is $\mathcal{O}(N\,2^{N+M})$, where M is the length of the longest factor (in terms of number of variables) in the JFP inference step.

Algorithm 4. $\mathrm{OPT}-\mathrm{SIZE}$

Input Available servers: $S \subseteq S$; Initial copy: c; Acceptable failure probability: p; Partial deployment set: Δ
Output Optimum deployment set: Δ
1: **for** $m = 1$ to $|S|$ **do** {Test increasing Δ sizes}
2: $\{\Delta', p_{min}\} \leftarrow \mathrm{OPT}--\mathrm{FP}(S, c, m, \Delta)$ {Find the optimum deployment of size m}
3: **if** $p_{min} \leq p$ **then** {If JFP is acceptable}
4: $\Delta \leftarrow \Delta'$
5: **break** {Stop the search}
6: **end if**
7: **end for**

The second algorithm, $\mathrm{OPT}-\mathrm{SIZE}$ in Algorithm 4 takes maximum acceptable JFP (p) as input instead of deployment size. Starting from $m = 1$ and incrementing m at each iteration (line 1), it calls $\mathrm{OPT}-\mathrm{FP}$, which in turn returns the optimum deployment set and corresponding JFP for the given m (line 2). When a deployment set that satisfies p is found, search is stopped and the algorithm outputs the set (lines 3–5). Consequently, the outputted deployment set is of not only minimum size but also minimum JFP given its size. If it is not possible to find a deployment that satisfy the requirement, the one with the highest JFP and size is returned. In the worst case, N calls to $\mathrm{OPT}-\mathrm{FP}$ are made, resulting in an overall time-complexity of $\mathcal{O}(N^2\,2^{N+M})$.

6 Experimental Setup

We evaluate SFP forecaster and DTFR algorithms through a larger-scale and more realistic version of our video analytics scenario. Here, service providers with certain resilience requirements aim to minimize over-provisioning of replicas to reduce costs. To that end, we implement the architecture shown in Fig. 2. DTFR and baseline algorithms are implemented in Java (JDK 1.8). The main program executes them sequentially by generating a service request at each iteration. We generate 10,000 service requests at uniform time intervals. The availability definition of services belongs to categories described in Section 3.1, namely load sharing and active-standby, whereas the number of copies is chosen uniformly at random from the range [1,5]. Then, generated tasks are deployed on a subset of currently available servers via proposed and baseline algorithms. Deployments outputted by each algorithm are evaluated with the failure traces that correspond to the task running time. Evaluation is repeated for 10 disjoints sets of 100 randomly selected servers from each data set. For each set, failures in the first half of the total time span are reserved for DBN learning and the rest for task scheduling. Data communication between the modules is implemented through shared variables.

6.1 Network Topology

An undirected network topology graph is generated with Barabási–Albert scale-free network generation model [21]. This model is widely used to represent human-made networks such as the Internet including edge topologies [22], [23], mainly due to two characteristics borrowed from real networks: incremental growth and preferential node connectivity. When new nodes are being added, a probability function for edge generation ensures that new nodes tend to link to the more connected nodes, i.e., hubs. Generated topology contains 1000 nodes placed on a 1000×1000 coordinate plane, 2994 edges, and a heavy-tailed distribution (Pareto with shape 1.2) of network latency. We consider fixed locations for edge resources such as base stations. Network delay is measured from the edge node that is closest to the user at the time of deployment.

We collected real link failure data on an edge computing testbed for the aforementioned InTraSafEd 5G project. The data set includes round-trip time (RTT) measurements of 25000 messages sent periodically from a Galaxy S10 5G smartphone to a nearby Kubernetes cluster consisting of 12 Raspberry Pi 3B+ single-board computers (see supplementary file, available online) over a period of 4 days. The smartphone was connected to either 5G or 4G networks during the data collection. The messages were transmitted using the MQTT protocol, which is the industry standard for IoT messaging, using *exactly once* (highest) QoS level. RTT values greater than $\mu + 2\sigma$ ($\approx 481ms$) were assumed link failures ($\approx 2.3\%$).

6.2 Failure Traces

To the best of our knowledge, there does not exist an edge computing reliability data set that is available to the research community at present. This is because of not only the novelty of the technology, but also the obstacles to making workload traces of commercial systems publicly available, such as competitive concerns, privacy obligations, and hardness of data anonymization [24]. Consequently and as with the previous work in system reliability literature [25], [26], [27], we take failure traces and infrastructure information from real-world distributed systems and synthetically generate the workload. To generalize our results, failure traces are collected from three distributed hardware systems that represent different deployment strategies that are proposed in the edge computing literature as depicted in Fig. 4. Here, the mean availability of servers reflects the values in corresponding data sets, whereas the ordinal values of computing power (shown as server icons) and average RTT are evaluated based on literature [1], [28], [29].

The most widely distributed case is when edge computing tasks are executed directly on client devices (e.g., desktop PCs, tablets, smartphones, etc.) [30], [31]. It is characterized by the lowest possible latency but also significantly high churn, low reliability, and limited computing power. The DEVICE data set [32] that contains failure traces from 226,208 personal computers between April 1, 2007 and January 1, 2009 is used to represent this deployment. The second alternative is the cloudlets located on-site on business premises [33]. This form of deployment exhibits relatively higher reliability but still lacks cloud-level extensive

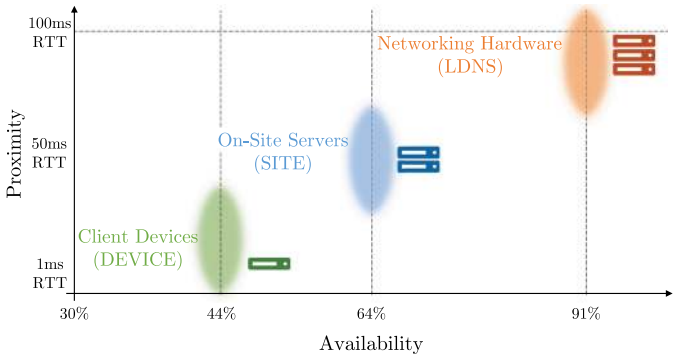


Fig. 4. Edge computing implementations and corresponding data sets.

support systems. We utilize the SITE data set [34], which contains 2,081 supernodes pinged in 30-minute intervals between September 18 to October 4, 2005. Supernodes are identified based on reachability and spare bandwidth and they represent the reliability middle ground between implementing edge computing on regular client devices and dedicated servers. Finally, edge virtualization infrastructure can be deployed on the networking hardware such as routers, switches, or proxy servers, similar to fog computing [35]. This would result in the highest level of reliability and computation power at the cost of increased RTT due to distance. Local Domain Name Servers (LDNS) data set [36] contains ping probes initiated to servers at exponential intervals with a mean of 1 hour, between March 17 to 24, 2004. In this data set, 62,201 LDNS servers substitute for edge servers deployed on networking hardware.

These data sets also include the workload traces; however, we exclude this part of the data in our experiments because the tasks are not typical edge computing services. We rather focus on the hardware characteristics as they run on a infrastructure similar to the prospective edge computing deployments described above. Availability distribution of data sets are given in the supplementary file, available online.

6.3 DBN Learning

For learning the DBN structure, we utilize Banjo framework[2] by Duke University. Finding the optimum structure that best describes the data is an NP-complete problem [37]. Hence, structure learners almost always include heuristic and approximation steps. Banjo searches for candidate graphs via simulated annealing, a Monte Carlo metaheuristic. We configured Banjo to allow Markov lags of 0 and 1, which means only the dependencies between failures in the same or consecutive time steps are captured.

Banjo does not support parameter learning (i.e., CPTs), so we implement maximum likelihood estimation (MLE) to obtain each conditional probability. MLE is a standard technique for parameter learning and it assumes that the probability is equal to the number of historical occurrence of all events (interested and given) divided by that of only given events. Continuing from the previous example in Fig. 3a, second row in Fig. 3b is estimated via MLE as shown in (15). Here, σ is a function that maps logical true to integer 1

2. https://users.cs.duke.edu/~amink/software/banjo/

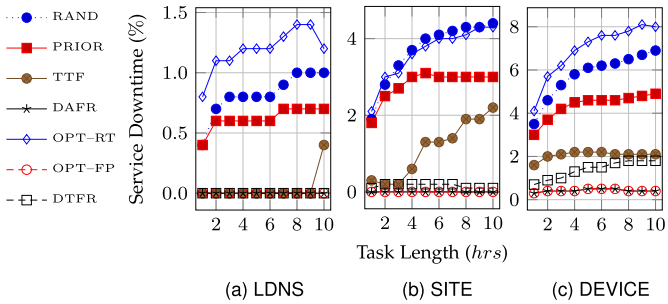


Fig. 5. Mean Service downtime results for all data sets.

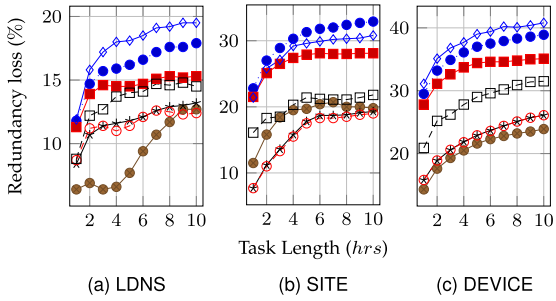


Fig. 6. Mean Redundancy loss results for all data sets.

and false to 0, whereas f_1^t is the interested event.

$$\Pr(f_1^t \mid f_2^{t-1}, \neg f_3^t) = \frac{\sum_{t=2}^{T} \sigma(f_1^t \wedge f_2^{t-1} \wedge \neg f_3^t)}{\sum_{t=2}^{T} \sigma(f_2^{t-1} \wedge \neg f_3^t)}. \quad (15)$$

6.4 Baseline Algorithms

Random (RAND). Each copy is placed on a server chosen uniformly at random. System time is the seed.

Prior-Based (PRIOR). Availability of a server is assumed to remain the same as its recent past and tasks are scheduled to the servers with highest past availability. This technique is applied in [13] to improve fault tolerance of Apache Storm applications. In our experiments, availability in the last five hours yielded the best accuracy for this baseline algorithm.

TTF-Based (TTF). Support Vector Machine (SVM) regression is applied in [14] to forecast future time to failure (TTF) values. We implement sequential minimal optimization [38] algorithm for SVM regression. Tasks are scheduled to the servers with the longest remaining TTF. Sample data size for SVM is 50 in our experiments.

Dependency-Aware (DAFR). The algorithm, introduced in [10], utilizes the same JFP calculation as DTFR but it is unaware of the topology, response time, and link failures.

Proximity-Based (OPT – RT). The first stage of DTFR is executed for both active and standby replicas, thus copies are placed on the edge servers closest to the users.

Availability-Based (OPT – FP). This is the DTFR algorithm without the first stage, so that replicas are placed solely based on SFP without considering their proximity.

7 Numerical Results and Discussion

7.1 Availability

Fig. 5 shows mean downtime percentages of 10,000 services that are deployed by each algorithm using different traces. A general trend is that availability decreases as the task length increases. This is expected since none of the algorithms re-evaluate the deployments or propose migrations after the initial decision. Additionally, services have higher downtime as the availability of the resources decreases (i.e., LDNS > SITE > DEVICE). DTFR avoids any downtime with LDNS regardless the task length and achieves a very high availability of 99.8 percent in the worst case with SITE. DAFR and OPT – FP achieve 100 percent availability for both data sets as they exploit the same JFP values as DTFR but their sole objective is to avoid unavailability. With highly unreliable DEVICE resources, none of the algorithms achieve zero downtime and the availability of DTFR ranges between 98.2 percent and 99.3 percent Forecasting-based approach TTF performs comparably well for short-term tasks and reliable resources; however, it gets increasingly inaccurate with longer ones. We conclude that dependency and link failures are critical in node selection especially for long-term tasks.

Fig. 6, on the other hand, presents the redundancy loss rates. We define redundancy loss as the case that at least one deployment fails but the service is still available according to its availability definition (e.g., maximum tolerable failures), which is described in detail in Section 3.1. The results demonstrate that considerable amount of failures do occur but service resilience is preserved by DTFR, DAFR and OPT – FP mechanisms. Interestingly, TTF incurs significantly less failures than DTFR, which shows its effectiveness in detecting individually most reliable servers; however, it suffers the same or higher downtime as shown by Fig. 5. Proposed DTFR mechanism, instead, achieves failure resilience by exploiting the co-occurrence of failures, which justifies our claim that consideration of failure dependency makes substantial contribution to service availability.

7.2 Network Delay and Overhead

Different from other distributed systems in general, scheduling for edge computing infrastructure is highly dependent to the proximity of chosen servers to the user. Thus, failure resilience has to be co-optimized with proximity in order to achieve acceptable response times. In Fig. 7 (left), we present the average end-to-end delay between the user and the closest failure-free deployment. Downtime periods are excluded to provide a fair comparison of all baselines in terms of network delay. Network delay of all task lengths are aggregated because the results are unaffected by length.

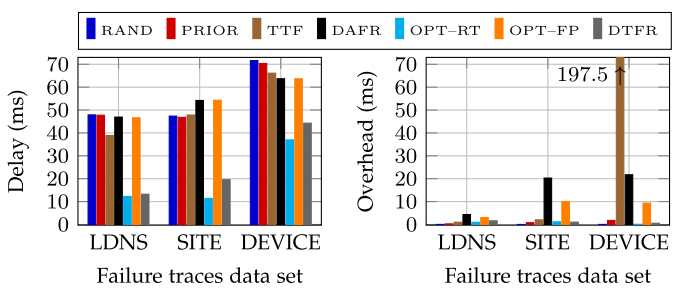


Fig. 7. Mean network delay and computational overhead results.

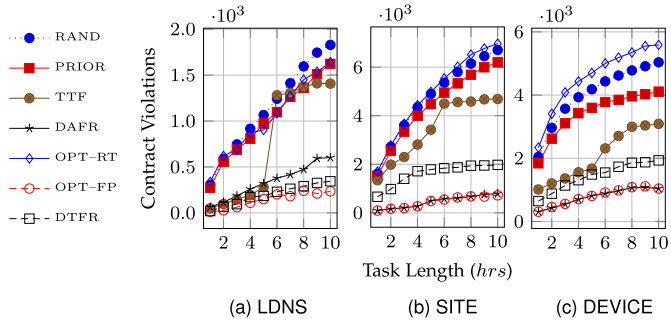


Fig. 8. Mean Contract violation results for all data sets.

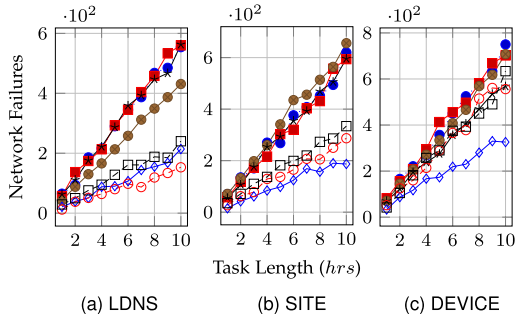


Fig. 9. Mean Network failure results for all data sets.

Higher average delays with DEVICE are due to the higher number of nodes and hence larger network topology.

OPT – RT, which chooses the most proximate servers, achieves the minimum delay with all data sets. However, as our previous results showed, it also suffers the highest downtime, which makes it infeasible for edge services. A close competitor is DTFR with only 1.0 ms of additional delay with LDNS, 8.2 ms with SITE, and 7.3 ms with DEVICE, despite maintaining very high failure resilience at the same time. The baselines with the highest availability performance, namely DAFR and OPT – FP, fail to achieve proximate deployments and incur between 26.7 and 34.7 ms additional delay with respect to OPT – RT. Our results show that only DTFR is able to fulfill both availability and network delay requirements of edge services simultaneously.

Responsiveness of edge services is also sensitive to their scheduling time, especially when they are short-lived. In Fig. 7 (right), we present a comparison of the overheads, which corresponds to the inference time in learning-based algorithms (i.e., DTFR, DAFR, OPT-FP, and TTF). Experimental results regarding the effect of training time to the DBN accuracy are provided in the supplementary file, available online. DTFR overhead remains in the range [0.5,1.6] ms, which is negligible in comparison to total network delay. This is also the case for the simpler baselines RAND and PRIOR as well as OPT-RT which is shown to have linear time-complexity in Section 5.1. OPT-FP, which corresponds to stage 2 of DTFR, has 2x to 18x higher overhead than the full two-stage version. The reason is that stage 1, which has linear time complexity, greatly reduces the search space for stage 2. Since it already deploys active replicas, N in stage 2 gets considerably small (≤ 10). The same applies to DAFR too. TTF, however, does not scale well with the number of nodes as it has low overhead (1 to 2 ms) with small-scale data sets but extremely high (197.5 ms) with DEVICE.

7.3 Failures

Although the mean availability percentages show an overall picture, a practical concern for service and infrastructure providers alike is the frequency of contract violations caused by failures. Thus, we present the number of contract violations incurred by each algorithm in Fig. 8. We enforce a strict availability requirement of 99.9 percent in these experiments. Our first observation is about TTF, which achieves relatively low downtime but incurs high number of individual violations as all three figures show. For the tasks longer than five hours, contract violations explode, which is particularly evident in Fig. 8a. In the same figure, we also observe that the proposed DTFR algorithm and its variant OPT – FP outperform DAFR. We assume the difference is due to the network failures, which are typically shorter-lived than node failures, hence do not affect the average downtime significantly, but cause violations nevertheless.

Above assumption is confirmed by our second set of results, which show the number of contact violations caused by the network failures. It is clear in Fig. 9a that link failure-aware algorithms OPT – FP and DTFR incur fewer network failures. However, network failure numbers are closer to other baselines in Fig. 9b and nearly the same in Fig. 9c. This is due to fewer available servers, hence fewer options for resilient deployment. Consequently, nodes with higher JFP are preferred despite relatively higher LFP. Overall, OPT – RT incurs the fewest network failures. Although, it does not consider link failures, it chooses the servers that are closest to the user, which are consequently close to each other as well, reducing the hop count and LFP. Since link failures are less frequent than node failures, this does not translate to lower downtime or fewer contact violations for OPT – RT. In contrast, high number of network failures in the case of DAFR are compensated by node failure avoidance, unless the nodes are extremely reliable (e.g., LDNS).

7.4 Cost

Finally, we conduct several experiments with variable number of copies using the dynamic variant of DTFR with the OPT – SIZE algorithm. In Figs. 10a and 10b, the number of deployed copies and percentage downtime are reported with various values of p for SITE and DEVICE traces. To illustrate the benefits, proposed algorithm is compared to the best performing static baseline TTF, configured to deploy two copies of each service, which corresponds to 20,000 deployments. Task length is chosen as two hours in these experiments. As maximum acceptable failure probability (p) increases, proposed algorithm manages to satisfy the requirement with fewer and fewer copies but services experience an adverse impact on downtime. TTF suffers around 1 percent downtime irrespective of p in both cases. In 300 different values of p evaluated for three data sets (not all are reported for brevity), there does not exist a single case that DTFR suffers higher downtime with the same number of copies as the baselines or that baselines achieve

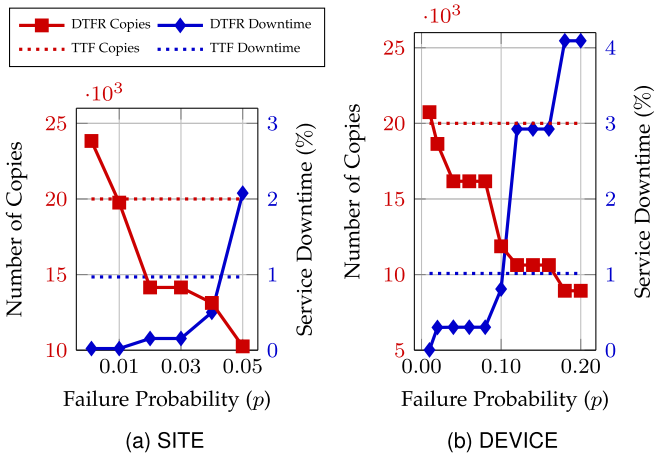


Fig. 10. The trade-off between downtime and number of copies.

the same downtime with less copies. Hence, it is a nondominated solution as far as our results are concerned. Furthermore, in certain ranges of p (e.g., roughly [0.01,0.04] in Fig. 10a and [0.02,0.11] in Fig. 10b), it is Pareto dominant, that is, proposed algorithm achieves higher availability with fewer copies. We omit the results for the LDNS data set with more reliable servers, because DTFR achieves 0 percent downtime with one or two copies regardless the value of p. Thus, the trade-off between cost and availability does not exist with LDNS.

8 Related Work

Failure resilience is a well studied topic within the context of cloud computing. Widely used strategies can be grouped under checkpointing [39], [40], [41], re-execution [40], [42], and replication [41], [42], [43]. High overhead and long re-activation time makes first and second groups of strategies infeasible when real-time computing is required [44]. Replication-based approaches, on the other hand, do not make a distinction between availability levels or failure probabilities of different nodes, and rightly so because the availability levels of servers within a cloud data center are not noticeably different. One of the few exceptions is [44], where so-called deteriorating physical machines are identified and proactive measures (e.g., migration, rescheduling) are taken. However, failure prediction is limited to CPU temperature forecasting against overheating. Similarly, in [45], an analytical model is proposed to estimate the reliability of subscribers in publish–subscribe systems. Interested reader may refer to the recent survey by Welsh and Benkhelifa [46] for details about resilience in the cloud context.

Although essential for its success, resilience in edge computing is an open issue [7]. An early discussion of reliability challenges in fog computing is presented in [47]; however, few attempts are made to address these challenges. Aral and Brandić introduce a technique that exploits causal relationships between different types of failures and channel all QoS related parameters through virtual machine availability [15]. Nebula [48], an edge-based computation and storage architecture, handles fault tolerance of compute nodes via re-execution. Although data is replicated, availability is not a factor in site selection. Cloud visitation platform [49], which copes with the hardware heterogeneity problem in a federated cloud and fog setting via hardware awareness, solves failure resilience only at a local level. When a server fails, deployed applications are migrated to another one, possibly in a different node. Cardellini *et al*. [13] extends the well known distributed stream processor, Apache Storm, by adding QoS awareness capability. Here, recent availability of nodes is used instead of predicting future values. FogStore [50], a distributed data store, handles replica and consistency management. As only data blocks are replicated, the focus of this work is on read and write latency. A recovery scheme for edge computing failures is proposed in [51]; however, only the failures that are caused by overloaded resources are considered. Traffic data is monitored to detect overloaded nodes and their load is shared with others. Odin [52], is a practical application of fault tolerance for distributed servers in CDNs via backups. A checkpointing mechanism for stateful fog computing that saves message and function call records is proposed in [53]. This work focuses on failure recovery rather than avoidance.

Almost all studies above either ignore network failure resilience or reduce it to the individual connectivity so that it can be embedded in node availability. Although this is acceptable for cloud resilience, where data center networks are reliable; specific consideration of link failures is imperative for edge due to the utilization of ad hoc public networks. Network reliability and resilience are well studied within the telecommunication field [54]. The most widely employed mechanisms including automatic protection switching [55], pre-configured cycle protection [56] and path restoration [57] are not applicable to our scenario since the service provider has no control over the network infrastructure, and can only optimize edge server selection. Ride [58] is an SDN middleware for resilient edge networks, which suffers the same problem as it relies on rerouting.

9 Conclusion

In this work, we propose a failure resilience mechanism for edge computing services that is dependency- and network-aware. Dependency awareness ensures that deployed copies are unlikely to fail concurrently. This not only increases overall service availability but also decreases the number of replicas or utilizes less reliable servers. Network-awareness, on the other hand, decreases both end-to-end network delay and probability of link failures. Extensive evaluation with real-world failure traces demonstrate the superiority of the algorithms against the state-of-the-art in terms of availability, number of failures, network delay, and cost. This work is a step towards realizing promised benefits of edge computing paradigm by offering a practical solution to one of the major obstacles to its adoption: failure resilience. Many applications, which cannot be included to the cloud ecosystem due to their network delay constraints, would be viable for an edge–cloud or pure edge solution provided that sufficient level of failure resilience is achievable. We demonstrate that DTFR techniques proposed in this paper can achieve similar availability levels to cloud, in the presence of low-latency yet failure-prone edge servers.

ACKNOWLEDGMENTS

This work was supported in part by the Rucon project (Runtime Control in Multi Clouds), Austrian Science Fund (FWF): Y904-N31 START-Programm 2015 and 5G Use Case Challenge InTraSafEd 5G (Increasing Traffic Safety with Edge and 5G) funded by the City of Vienna. Icons in Fig. 1 are designed by Flaticon.com.

REFERENCES

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

[2] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, 2019.

[3] J. Bort, "The massive AWS outage [...]," Accessed: Apr. 28, 2020. [Online]. Available: http://www.businessinsider.com/aws-outage-hurt-internet-retailers-except-amazon-2017–3

[4] L. Ponemon, "Cost of data center outages," Ponemon Institute, Traverse City, MI, Tech. Rep. 5, 2016.

[5] V. De Maio, R. B. Uriarte, and I. Brandic, "Energy and profit-aware proof-of-stake offloading in blockchain-based VANETs," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2019, pp. 177–186.

[6] A. Aral, R. B. Uriarte, A. Simonet-Boulogne, and I. Brandic, "Reliability management for blockchain-based decentralized multi-cloud," in *Proc. IEEE/ACM Int. Symp. Cluster Cloud Internet Comput.*, 2020, pp. 21–30.

[7] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog *et al.*: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, 2018.

[8] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proc. ACM/IEEE Conf. Supercomputing*, 2007, Art. no. 41.

[9] W. Zheng, Z. Wang, H. Huang, L. Meng, and X. Qiu, "SPSRG: A prediction approach for correlated failures in distributed computing systems," *Cluster Comput.*, vol. 19, no. 4, pp. 1703–1721, 2016.

[10] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 228–242.

[11] A. Aral, M. Erol-Kantarci, and I. Brandic, "Staleness control for edge data analytics," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, 2020, Art. no. 38.

[12] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016.

[13] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "On QoS-aware scheduling of data stream applications over fog computing infrastructures," in *Proc. IEEE Symp. Comput. Commun.*, 2015, pp. 271–276.

[14] M. das Chagas Moura, E. Zio, I. D. Lins, and E. Droguett, "Failure and reliability prediction by SVM regression of time series data," *Rel. Eng. Syst. Saf.*, vol. 96, no. 11, pp. 1527–1534, 2011.

[15] A. Aral and I. Brandic, "QoS Channelling for latency sensitive edge applications," in *Proc. IEEE Int. Conf. Edge Comput.*, 2017, pp. 166–173.

[16] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proc. Conf. Uncertainty Artif. Intell.*, 1998, pp. 139–147.

[17] X.-G. Gao, J.-F. Mei, H.-Y. Chen, and D.-Q. Chen, "Approximate inference for dynamic bayesian networks: sliding window approach," *Appl. Intell.*, vol. 40, no. 4, pp. 575–591, 2014.

[18] D. Geiger, T. Verma, and J. Pearl, "Identifying independence in Bayesian networks," *Networks*, vol. 20, no. 5, pp. 507–534, 1990.

[19] D. Brown and G. Smith, "Mpp2 syslog data (2006–2008)," Pacific Northwest National Laboratory, Richland, WA, Tech. Rep. PNNL-SA-61371, 2008.

[20] A. Putina *et al.*, "Telemetry-based stream-learning of BGP anomalies," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw.*, 2018, pp. 15–20.

[21] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[22] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 516–529, Jun. 2018.

[23] M. Huang, W. Liang, M. Huang, and X. Jia, "Reliability-aware virtualized network function services provisioning in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2699–2713, Nov. 2020.

[24] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Obfuscatory obscanturism: Making workload traces of commercially-sensitive systems safe to release," in *Proc. IEEE Netw. Operations Mgnage. Symp.*, 2012, pp. 1279–1286.

[25] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, Art. no. 33.

[26] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid Cloud infrastructure," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1318–1331, 2012.

[27] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2899–2917, 2014.

[28] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Berlin, Germany: Springer, 2014, pp. 169–186.

[29] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[30] P. G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[31] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for Internet of Things computations," in *Proc. Cloudification Internet Things*, 2016, pp. 1–6.

[32] B. Javadi, D. Kondo, J.-M. Vincent, and D. P. Anderson, "Mining for statistical availability models in large-scale distributed systems: An empirical study of SETI@home," in *Proc. IEEE/ACM Int. Symp. Modelling Anal. Simul. Comput. Telecommun. Sys.*, 2009, pp. 1–10.

[33] M. Satyanarayanan, P. Bahl, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Fourth Quarter 2009.

[34] S. Guha and N. Daswani, "An experimental study of the skype peer-to-peer VoIP system," Cornell Univ., Ithaca, NY, Tech. Rep. TR2005-2011, 2005.

[35] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, Oct. 2014.

[36] J. Pang, J. Hendricks, A. Akella, R. De Prisco, B. Maggs, and S. Seshan, "Availability, usage, and deployment characteristics of the domain name system," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 1–14.

[37] D. M. Chickering, "Learning Bayesian networks is NP-complete," in *Learning From Data*. Berlin, Germany: Springer, 1996, pp. 121–130.

[38] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to the SMO algorithm for SVM regression," *IEEE Trans. Neural Netw.*, vol. 11, no. 5, pp. 1188–1193, Sep. 2000.

[39] Í. Goiri, F. Julia, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, 2010, pp. 455–462.

[40] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "Correlation modeling and resource optimization for cloud service with fault recovery," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 693–704, Third Quarter 2019.

[41] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud comp. environments," *J. Supercomput.*, vol. 66, no. 1, pp. 193–228, 2013.

[42] Q. Zheng, "Improving MapReduce fault tolerance in the cloud," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2010, pp. 1–6.

[43] W. Zhao, P. Melliar-Smith, and L. E. Moser, "Fault tolerance middleware for cloud computing," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2010, pp. 67–74.

[44] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1191–1202, Fourth Quarter 2016.

[45] T. Pongthawornkamol, K. Nahrstedt, and G. Wang, "Reliability and timeliness analysis of fault-tolerant distributed publish/subscribe systems," in *Proc. Int. Conf. Auton. Comput.*, 2013, pp. 247–257.

[46] T. Welsh and E. Benkhelifa, "On resilience in cloud computing: A survey of techniques across the cloud domain," *ACM Comput. Surv.*, vol. 53, no. 3, 2020, Art. no. 36.

[47] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *Proc. Int. Conf. Syst. Signals Image Process.*, 2013, pp. 43–46.
[48] M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing" in *Proc. IEEE Int. Conf. Cloud Eng.*, 2014, pp. 57–66.
[49] M. Zhanikeev, "A cloud visitation platform to facilitate cloud federation and fog computer" *Computer*, vol. 48, no. 5, pp. 80–83, May 2015.
[50] R. Mayer, H. Gupta, E. Saurez, and U. Ramachandran, "FogStore: Toward a distributed data store for fog computing," in *Proc. Fog World Congress*, 2017, pp. 1–6.
[51] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Gener. Comput. Syst.*, vol. 70, pp. 138–147, 2017.
[52] M. Calder *et al.*, "Odin: Microsoft's scalable fault-tolerant CDN measurement system," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 501–517.
[53] U. Ozeer *et al.*, "Resilience of stateful IoT applications in a dynamic fog environment," in *Proc. Int. Conf. Mobile Ubiquitous Syst.*, 2018, pp. 332–341.
[54] J. P. Sterbenz *et al.*, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, 2010.
[55] G. Ellinas and T. E. Stern, "Automatic protection switching for link failures in optical networks with bi-directional links," in *Proc. IEEE Global Telecommun. Conf.*, 1996, pp. 152–156.
[56] M. S. Kiaei, C. Assi, and B. Jaumard, "A survey on the p-cycle protection method," *IEEE Commun. Surv. Tuts.*, vol. 11, no. 3, pp. 53–70, Third Quarter 2009.
[57] B. Bassiri and S. S. Heydari, "Network survivability in large-scale regional failure scenarios," in *Proc. Canadian Conf. Comput. Sci. Softw. Eng.*, 2009, pp. 83–87.
[58] K. E. Benson, G. Wang, N. Venkatasubramanian, and Y.-J. Kim, "Ride: A resilient IoT data exchange middleware leveraging SDN and edge cloud resources," in *Proc. IEEE/ACM Int. Conf. IoT Des. Implementation*, 2018, pp. 72–83.

Atakan Aral (Member, IEEE) received the first MSc degree in computer science and engineering from Politecnico di Milano, in 2011, and second MSc degree in computer science and engineering from Istanbul Technical University (ITU), in 2012, and the PhD degree in computer engineering from Istanbul Technical University, in 2016. He is a postdoctoral research fellow with the Vienna University of Technology. His research interests include resource management for geo-distributed and virtualized computing systems such as inter-cloud and edge computing, as well as the optimization of edge computing architecture for AI services.

Ivona Brandić (Member, IEEE) received the PhD degree from the Vienna University of Technology, in 2007. She is a professor with the Vienna University of Technology. In 2015, she was awarded FWF START prize, the highest Austrian Award for early career researchers. In 2011, she received the Distinguished Young Scientist Award from the Vienna University of Technology for her project on the Holistic Energy Efficient Hybrid Clouds. Her main research interests include cloud computing, large scale distributed systems, energy efficiency, quality of service, and autonomic computing.