Dependency Mining for Service Resilience at the Edge

Atakan Aral, Ivona Brandic
Institute of Information Systems Engineering
Vienna University of Technology
Vienna, Austria
{*atakan.aral, ivona.brandic*}@*tuwien.ac.at*

***Abstract*—Edge computing paradigm is prone to failures as it trades reliability against other quality of service properties such as low latency and geographical prevalence. Therefore, software services that run on edge infrastructure must rely on failure resilience techniques for uninterrupted delivery. Unique combination of hardware, software, and network characteristics of edge services is not addressed by existing techniques that are designed or tailored for cloud services. In this work, we propose a novel method for evaluating the resilience of replicated edge services, which exploits failure dependencies between edge servers to forecast probability of service interruption. This is done by analyzing historical failure logs of individual servers, modeling temporal dependencies as a dynamic Bayesian network, and inferring the probability that certain number of servers fail concurrently. Furthermore, we propose two replica scheduling algorithms that optimize different criteria in resilient service deployment, namely failure probability and cost of redundancy.**

***Keywords*-Edge Computing; Failure Resilience; Fault Tolerance; Availability; Quality of Service; Dynamic Bayesian Networks.**

I. Introduction

One of the major determinants behind the success of cloud computing is the economical gain that consolidation of compute resources brings. This leads to proliferation of large centralized computing facilities such as massive cloud data centers, which benefit from economies of scale. However, latency-sensitive and data-intensive services, that are coming into more prominence by means of Internet of Things and mobile computing technologies, are adversely affected by centralization due to the increasing distances between where data is produced, processed, and consumed [1]. Emerging edge and fog computing paradigm aims to remedy this defect of cloud computing by bringing part of the computation to the close proximity of data producers and consumers. It is meant to seamlessly interoperate with cloud computing rather than to replace it.

In addition to evident latency and bandwidth advantages, edge computing can also help to mask cloud outages temporarily and to enforce privacy policies by determining which data are sent to the cloud [2]. However, as any emerging technology, edge computing also is not without its problems to be addressed. Particularly, edge servers are more prone to failures and outages than cloud counterparts due to (i) geographical dispersion which complicate management and maintenance, (ii) small scale in terms of computation and storage resources which preclude hardware redundancy, (iii) limited horizontal and vertical scaling opportunity in case of volatile workload, and (iv) 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. Therefore, an effective resilience technique, such as replication, is needed for edge services in order to avoid service interruption or service-level agreement (SLA) violation due to fail over to cloud or recovery from backup [3]. Such disruptions cause significant revenue loss in a business environment. Recently, a four-hour outage of Amazon Web Services is reported to affect 54 of the top 100 online retailer services, which lost $150 million in total [4]. Whereas, Amazon retail website is estimated to lose 1% of sales for every 100 ms of delay [5]. Worse still, modern Internet services are getting less tolerant to downtime: average cost of a data center outage has increased from $505,000 to $740,000 between 2010 and 2016 [6].

Previous work in the field of system reliability already showed that there exists correlation between failures in distributed computing systems [7]–[9]. Agreeing with the literature, we assume that edge server failures are intercorrelated, hence having copies deployed at servers that probabilistically fail in overlapping periods will deteriorate or even nullify availability benefits of replication. Our experimental results confirm the validity of this assumption. In this work, we propose a probabilistic technique for forecasting interruptions in replicated, near real-time edge services due to failures in edge servers. Our approach exploits spatial and temporal failure dependencies among edge servers to compute joint failure probability (JFP). We model failure dependencies as a dynamic Bayesian network (DBN) and automatically learn them from historical traces. Then, we employ an efficient inference algorithm based on variable elimination to compute the JFP of a given service deployment. Proposed technique is unique in the way that historical trends are combined with failure dependencies, and that availability of multiple edge servers are evaluated in the aggregate. Our hypothesis is that, JFP is a more precise criteria for resilience than failure probabilities of individual servers when evaluating deployment and replication plans

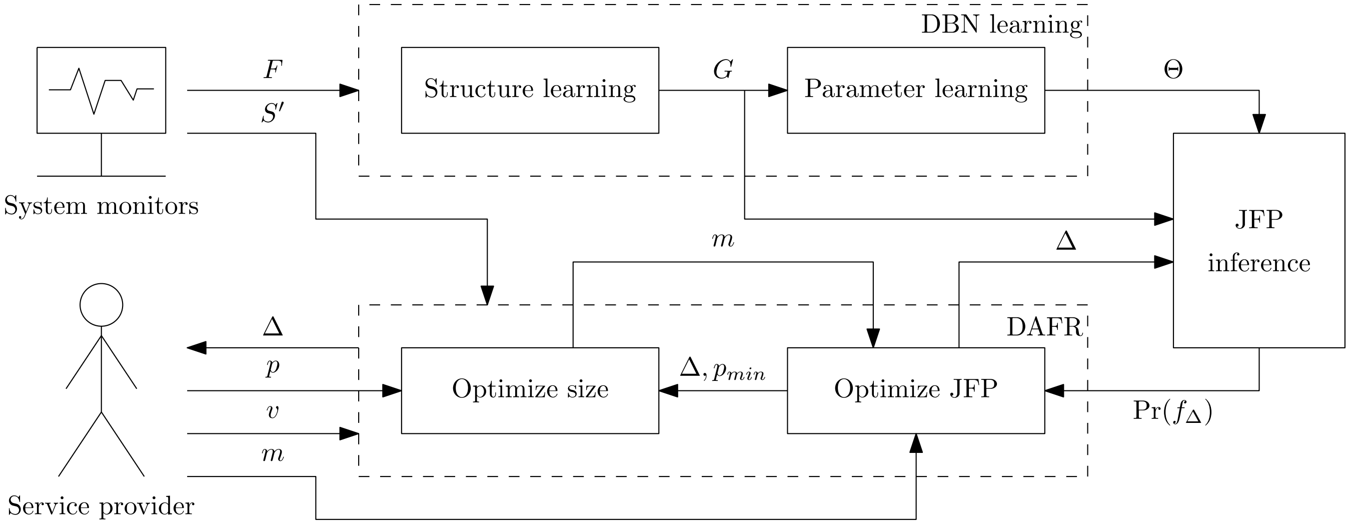

Figure 1. Main components of the proposed technique and data flow between them.

of edge services. Our second contribution is to support this hypothesis by utilizing JFP for task scheduling at the edge. We propose two dependency-aware failure resilience (DAFR) algorithms that optimize the deployment of edge services. Finally, we simulate proposed algorithms using multiple real-world failure traces with various availability characteristics and we demonstrate the effectiveness of our approach with respect to dependency-oblivious baselines in terms of both increasing failure resilience and decreasing the cost of redundancy.

Failure forecasting is a valuable input for the scheduler and replica manager of any edge computing service to make better informed decisions about resilience. In addition to our utilization in evaluating replication plans at deployment time, it can also be employed at design time to compare best, average, worst case resilience of models; or at execution time to trigger proactive failure mitigation mechanisms. Once latency benefits of edge computing are complemented with failure resilience, many near real-time services such as live video analytics [10] and streaming [11], vehicular applications [12], [13], and smart city information systems [14] could be included to the cloud and edge computing ecosystem. Note that, proposed techniques for edge service resilience in this work are probabilistic in nature, thus they are not to guarantee absolute availability under failures for safety-critical and hard real-time applications. They do, however, promise satisfactory quality of service (QoS) under a limited budget for most prospective edge computing applications as exemplified above. As a reference, Open Data Center Alliance defines in *Standard Units of Measure for IaaS* report[1] that highest category of cloud data centers (i.e. Platinum) must offer 99.99% availability. The same monthly availability level is also promised in Amazon EC2 SLA[2]. According to our evaluation, DAFR algorithms achieve comparable QoS levels despite highly unreliable edge resources. Another important remark at this point is that, our methods make no assumption on the virtualization technology or data processing architecture. We believe proposed scheduling algorithms will be applicable to virtual machines (VM) as well as more light-weight implementations such as containers and microservices. Hence, we use the general terms of *task* or *copy* throughout the paper. While applicable to both lambda and kappa architectures, our approach can particularly advance the latter; in which absence of batch layer might result in errors and thus, high availability is a strong requirement.

Fig. 1 gives an overview of the proposed architecture with the flow of data between components. Whereas the details of each component as well as inputs and outputs (with notation in Table I) are described thoroughly in the rest of the paper, the main process can be summarized as follows. As the one-off preliminary step, failure traces of edge services are collected from system monitors and a DBN model is trained with data. This model represents the existence and strength of dependencies between servers. Then, service providers define characteristics and resilience requirements of their edge services. Often, these are the outputs of a service deployment planning process such as *UML Deployment Diagram* or *OASIS TOSCA*[3]. DAFR algorithms in return, evaluate various deployment and replication plans (i.e. number and location of copies) via JFP inference in the DBN. Then, they inform the service provider about the optimum deployment in terms of JFP and/or replication cost. Inference and evaluation steps are repeated for each

[1] https://joinup.ec.europa.eu/solution/standard-units-measure-iaas-rev-11-standard-units-measure-iaas/

[2] https://aws.amazon.com/ec2/sla/

[3] https://www.oasis-open.org/committees/tosca/

new service request, whereas DBN model needs to be updated only when underlying edge computing infrastructure changes substantially.

After providing brief background information on edge computing failures in the following section, we formally define the concerned problem and our model in Section III. Whereas, in Sections IV and V, we elaborate on our main contributions, namely computing JFP and optimizing service deployment. Then, in Section VI, we provide the evaluation of our approach via extensive simulation and real-world failure traces. Finally, we review relevant literature in Section VII and make concluding remarks in Section VIII.

II. Background on Edge Failures

Although our approach does not differentiate between the types of dependency (e.g. association, correlation, causation), it is worthwhile to identify possible reasons behind this phenomenon. Two broad categories of spatial correlation between failures are considered in the literature [7].

Multiplication: Failures occur simultaneously in multiple servers due to a common cause.

Propagation: Failure in one server eventually causes further failures in other servers.

In the context of edge computing, some examples of dependency 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 power grid; multiple servers deployed in hostile locations failing due to environmental/weather interference; an extraordinary public event (e.g. election, sport competition, etc.) overloading servers in a geographical area; or failure/cyber-attack in a centralized controller that manages multiple servers. Whereas, cascading failures, which occur after a single server failure and spread due to workload redistribution (network or computation), belong to the failure propagation category. Reattempts of failed actions by users often amplify the outcomes of cascading failures. Another example can be a latent software defect or malicious software transmitting to other components, which are deployed at distributed servers.

Above listed factors are not usually transparent to the user and it is too exhaustive to take measures for each factor independently. Moreover, failures in edge servers are substantially more critical than those in massive cloud data centers since almost all use cases are time sensitive. Widely used cloud resilience techniques such as very high redundancy, re-execution, or checkpointing, may not be efficient or comparably effective in edge scenario due to their high computational overhead and limited capacity of edge servers. Additionally, unstructured and ad hoc implementation of edge computing topology, as well as its dynamicity hinder approaches based on shared risk groups or availability zones. Due to strict latency requirements, all admissible candidates for replication may belong to few such groups or zones (e.g. servers in the same region, site, rack, etc.), which adds to inherent dependency. Although edge providers may understandably adapt methods from cloud computing, there is still a need for resilience techniques that take unique features and limitations of edge computing into consideration so that comparable availability levels are possible [15].

Table I
Notation and Symbols Used in 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, $\delta \in \Delta$
K	Set of all components of a service
S	Set of all defined edge servers
S'	Set of admissible edge servers for a service, $S' \subseteq S$
S_Δ	Set of edge servers that host a service, $S_\Delta \subseteq S'$
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	Number of copies for a service
f_δ	Random binary event representing failure of a deployment
f_Δ	Random binary event representing failure of a service
f_s^t	Random binary event representing failure of server s at time t
F	Set of all f_s^t for all $s \in S$ and for all t
G	Structure graph of a dynamic Bayesian network, $G = \langle V, L\rangle$
V	Vertices of graph G
L	Links (edges) of graph G
A	Ancestral graph for S_Δ in G, $A = \langle V', L'\rangle$, $A \subseteq G$
P	Set of parent vertices of event f_s in G, $P \subseteq V$
Θ	Parameters (CPTs) of a dynamic Bayesian network

III. Problem Modeling and Definition

We provide the notation and symbols used throughout the paper in Table I, whereas few less frequent symbols are defined where they are used. In the following, we first introduce the failure probability model and then the optimization problems.

A. Failure Probabilitization

Services deployed at the edge have various resilience requirements. These are often communicated through reliability SLAs such as minimum service level, availability level (e.g. number of nines), or maximum acceptable downtime, which are subject to a resource cost budget (e.g. maximum number of replicas). 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. This also allows us to incorporate failures that occur in the client software, virtual platform, host operating system, hardware, or infrastructure as a single consistent event. In (1), we define failure probability of a deployment as its unavailability, which is the ratio of the duration that it is not accessible by users to its lifetime.

$$\Pr(f_\delta) = \frac{\{\text{Downtime}\}}{\{\text{Uptime}\} + \{\text{Downtime}\}} \tag{1}$$

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 (2), the service is assumed available unless all 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 (3), the service is available as long as at least n deployments out of $n+k$ are available (i.e. minimum service level). Or in other words, up to k failed deployments are tolerated.

$$\Pr(f_\Delta) = \Pr\left(\bigcap_{\delta \in \Delta} f_\delta\right) \tag{2}$$

$$\Pr(f_\Delta) = \Pr\left(\bigcup_{\substack{D \subseteq \Delta \\ |D| = k+1}} \bigcap_{\delta \in D} f_\delta\right) \tag{3}$$

In this work, we assume single-component edge services where all copies carry out the same task, however, definitions in (2) and (3) can be easily generalized to multi-component case as shown in (4). Here, K is the set of service components and $[\Delta]^\kappa$ is the set of all deployments that run component κ as defined in (5). It is also possible to define a custom JFP function where each component has a different availability definition or some components are noncritical and do not have any impact on service availability.

$$\Pr(f_\Delta) = \Pr\left(\bigcup_{\kappa \in K} f_{[\Delta]^\kappa}\right) \tag{4}$$

$$[\Delta]^\kappa = \{\langle c, s\rangle \in \Delta \mid c \leftarrow \kappa\} \tag{5}$$

First part of the problem that we are addressing in this study is to accurately forecast the failure ratio during the interval that an edge service is running. The forecast is based on the JFP of the deployment set. We believe, failure forecasting comes in very useful for management of edge resources in different 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.

B. Deployment Optimization

Among the possible use cases pointed out in Section III-A, we focus on optimizing the failure resilience at service deployment as the second part of the problem. Given a set of admissible edge servers, S', we aim to minimize either JFP or number of deployments. Here, the set S' contains only the servers that satisfy QoS requirements among all defined servers, S. More formal definitions of the two complementary optimization problems are as follows.

Optimization Problem 1 (OP1): Number of copies to be deployed is predefined and objective function minimizes the JFP of deployment set. First constraint in (6) states that each deployment is between a service copy and an admissible server, whereas the second one guarantees that the total number of copies is fixed to given m.

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

Optimization Problem 2 (OP2): Maximum acceptable failure probability, p, is predefined and objective function minimizes deployment set cardinality (i.e. the number of copies). Second constraint in (7) ensures satisfaction of resilience requirement.

$$\begin{aligned} \underset{\Delta}{\text{minimize}} \quad & |\Delta| \\ \text{subject to} \quad & \forall \delta \in \Delta(\delta = \langle c, s\rangle \wedge s \in S'), \\ & \Pr(f_\Delta) \leq p. \end{aligned} \tag{7}$$

IV. Joint Failure Probability (JFP)

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 (2). Techniques described in this section can also be applied to (3), 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 [16], support vector machines [17], and probabilistic graphical models [18]. As defined in (1), we use unavailability ratio of the edge server to estimate 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 from failures of other deployments. In this case, it is sufficient to compute and store MFP of each deployment resulting in $\mathcal{O}(|\Delta|)$ probabilities. JFP with independence assumption is calculated as follows.

$$\Pr\left(\bigcap_{\delta \in \Delta} f_\delta\right) = \prod_{\delta \in \Delta} \Pr(f_\delta) \tag{8}$$

In contrast, one may assume that each deployment is dependent to all others to some extent, where JFP can be computed via chain rule as below. In this case, computation and storage of $\mathcal{O}(2^{|\Delta|})$ probabilities are needed.

$$\Pr\left(\bigcap_{\delta\in\Delta} f_\delta\right) = \prod_{i=1}^{|\Delta|} \Pr\left(f_{\delta_i} \,\middle|\, \bigcap_{j=1}^{i-1} f_{\delta_j}\right) \tag{9}$$

Both of these extreme solutions, however, have substantial shortcomings. The former, while being computationally efficient, ignores valuable information about concurrent failures. Hence, the forecast can be inaccurate. The latter, on the other hand, has high time and space complexity as well as possibly low accuracy due to noise from coincidental dependencies. 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.

A. 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 [19], for example. This is crucial to capture cascading failures, which are dependent but occur at different times. Other relevant strengths of DBN in the context of this work are listed below.

- DBN infers not only dependencies themselves but also the direction of causality because it incorporates temporal information [20]. This way we can distinguish between failures that are cause and effect.
- Failure events are nonlinear, hence their dependency can be captured by DBN but not by linear estimators such as Kalman filters [7], [20].
- DBN holds significant performance improvements with respect to hidden Markov models (HMM) because number of states in HMM grows exponentially [20].
- DBN is the most general graphical model which includes HMM and Kalman filters as special cases [21].

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 is the time step and $t = 1, 2, \ldots, T$. Here, G is a directed acyclic graph (DAG) with vertices representing variables at different time steps and links representing the dependency assumptions between them. According to 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. 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 $\theta = \Pr(r_i^t \mid \text{parents}(r_i^t))$.

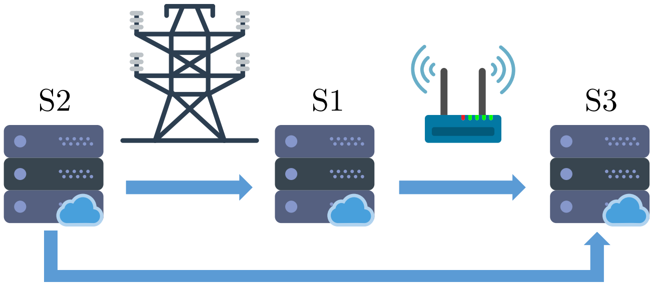

Figure 2. An example scenario to illustrate failure dependencies.

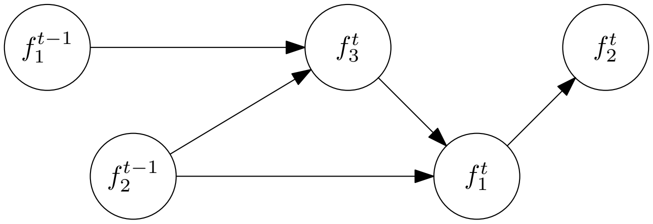

Figure 3. Simplified structure of an example dynamic Bayesian network.

Table II
An Example CPT for f_1^t from Fig. 3

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

In our case, the variables, for which DBN is defined, are binary failure events of edge 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 time steps. Let us illustrate how failure dependencies are modeled with the small scale example in Fig. 2, which consists of three edge servers: S1, S2, and S3. Fig. 3 is the simplified structure of a corresponding DBN. In this example, S1 and S2 are powered by the same electricity grid causing joint failures in the past. Hence in Fig. 3, failure events f_1 and f_2 are dependent at the same time step t. Similarly, f_1 and f_3 are concurrently dependent because S1 and S3 share the same network connection. Direction of the dependency is trivial in concurrent dependencies. In addition, edge servers are configured to dispatch their tasks to others in the case of overload as indicated with blue arrows in Fig. 2. This is a typical cause of cascading failures, which are represented with dependencies in consecutive time steps ($t-1$ and t) in Fig. 3. Circular dependencies are not captured by the definition of DBN.

Furthermore, in Table II, 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 time step ($t-1$) and that S3 does 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 based on historical failure traces as shown in Fig. 1. Traces are ordered chronologically and clustered into fixed-length time steps, such that overlapping failures are regarded as concurrent. Failures in different but consecutive time steps, on the other hand, are used to infer temporal dependencies as in the case of cascading failures. Further details of the DBN training are described in Section VI.

B. 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. This process is illustrated in Fig. 1, whereas specific set of servers is defined in (10). Then, computed probability value (i.e. JFP) in (11) is treated as the forecast.

$$S_\Delta = \{s \in S \mid \langle c, s\rangle \in \Delta\} \tag{10}$$

$$\Pr(f_\Delta) = \Pr\left(\bigcap_{s \in S_\Delta} f_s\right) \tag{11}$$

Below introduced optimization 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 VI. However, for models with greater number of variables (> 1000) and allowed parents per variable (> 5), approximate inference algorithms can be useful. Some examples include sampling techniques such as *Monte-Carlo*, or variational inference algorithms such as *mean field*.

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 (12) where P_s is the parent set of variable f_s. Note that, significantly fewer conditional variables are needed with respect to (9), decreasing from $\mathcal{O}(|S|)$ to $\mathcal{O}(|P_s|)$.

$$\Pr\left(\bigcap_{s \in S} f_s\right) = \prod_{s \in S} \Pr\left(f_s \,\middle|\, \bigcap_{\gamma \in P_s} f_\gamma\right) \tag{12}$$

Moreover, all these conditional probabilities are already available in Θ. Hence, one way of computing the JFP, as in (13), is to leave interested variables (S_Δ) and marginalize out all others ($S \setminus S_\Delta$) by summing up the probability values for all possible combinations of them.

$$\Pr(f_\Delta) = \sum_{S \setminus S_\Delta} \prod_{s \in S} \Pr\left(f_s \,\middle|\, \bigcap_{\gamma \in P_s} f_\gamma\right) \tag{13}$$

```
Algorithm ANCESTRAL–GRAPH
Input DBN structure graph: $G = \langle V, L\rangle$
    Deployment servers: $S_\Delta$
Output Ancestral graph for $S_\Delta$: A = $\langle V', L'\rangle$
$V' \leftarrow \emptyset$, $L' \leftarrow \emptyset$        # $A$ is initially a null graph
$Q \leftarrow S_\Delta$                          # Initialize the queue
for all $q \in Q$ do                # While the queue is not empty
    $P \leftarrow \{p \in V \mid \langle p, q\rangle \in L\}$     # Parents of $q$ in $G$
    $Q \leftarrow \{Q \cup P\} \setminus \{q\}$          # Queue $P$ and dequeue $q$
    $V' \leftarrow V' \cup \{q\}$                   # $q$ belongs to $A$
    $L' \leftarrow L' \cup \{\langle x, y\rangle \in L \mid y = q\}$        # Links to $q$
end for                                    # belong to $A$
```

Figure 4. Pseudo code description of ancestral graph extraction.

In (13), we sum the probability over all $2^{|S \setminus S_\Delta|}$ possible instantiations of uninterested variables. Although the result would be correct, the summation can be optimized computationally in several ways. 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 *d-separation* algorithm for independence in Bayesian networks [22]. Hence, we build a subgraph of the original DBN which consists of only concerned variables (S_Δ) and all their ancestors. Extraction of this so-called ancestral graph is described via pseudo code in Fig. 4. Once we obtain the set of ancestor nodes V', it can be safely used instead of S in (13).

Consider the DBN in Fig. 3 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 with the servers $S_\Delta = \{s_1, s_3\}$ and their JFP, $\Pr(f_{s_1} f_{s_2})$. From Fig. 4, ancestral graph of S_Δ contains 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.

$$\begin{aligned}\Pr(f_{s_1} f_{s_3}) &= \sum_{f_1^{t-1}} \sum_{f_2^{t-1}} \Pr(f_1^{t-1} f_2^{t-1} f_1^t f_3^t) \\ &= \sum_{f_1^{t-1}} \sum_{f_2^{t-1}} \Pr(f_1^{t-1}) \Pr(f_2^{t-1}) \\ &\qquad \Pr(f_1^t \mid f_2^{t-1}) \Pr(f_3^t \mid f_1^{t-1} f_2^{t-1})\end{aligned} \tag{14}$$

As a second performance optimization, we implement well-known *variable elimination* algorithm, which reduces the number of summation steps via dynamic programming. While the formal definition of the algorithm is given elsewhere [23], the main idea behind is to move probabilities out of certain summations. This can be safely done when the index variable of the summation does not appear in the conditional probability. Continuing the example from (14), we can move the probability $\Pr(f_1^{t-1})$ out of the summation $\sum_{f_2^{t-1}}$ because variable f_2^{t-1} does not appear in it.

$$\begin{aligned}\Pr(f_{s_1} f_{s_3}) = \sum_{f_1^{t-1}} \Pr(f_1^{t-1}) \sum_{f_2^{t-1}} \Pr(f_2^{t-1}) \\ \Pr(f_1^t \mid f_2^{t-1}) \Pr(f_3^t \mid f_1^{t-1} f_2^{t-1})\end{aligned} \tag{15}$$

However, different orders of parameter elimination are possible and significantly affect the number of computation steps (but not the outcome). Since finding the optimum ordering is an NP-complete problem, various heuristics are used in practice [23]. Due to its simplicity and being one of the four heuristics that perform well in practice [24], we resort to *Min-neighbors* heuristic which eliminates variables in ascending order of dependents. For example, eliminating f_1^{t-1} (one dependent) before f_2^{t-1} (two dependents) would result in following equation with fewer summations.

$$\begin{aligned}\Pr(f_{s_1} f_{s_3}) = \sum_{f_2^{t-1}} \Pr(f_2^{t-1}) \Pr(f_1^t \mid f_2^{t-1}) \\ \sum_{f_1^{t-1}} \Pr(f_1^{t-1}) \Pr(f_3^t \mid f_1^{t-1} f_2^{t-1})\end{aligned} \tag{16}$$

Finally, time-complexity of exact inference is reduced from $\mathcal{O}(2^{|S|})$ to $\mathcal{O}(|V'|\, 2^M)$ where M is the maximum number of variables in a summation term.

V. Dependency-Aware Failure Resilience (DAFR)

Having an effective method for forecasting failure probability of prospective deployment sets, it is possible to optimize task scheduling at the edge to develop failure resilience. DAFR algorithms that we describe in this section, propose a novel way of incorporating failure dependency information into the decision making process at service deployment time. Hereby, we aim to maximize the efficiency of replication in service resilience. In this context, efficiency means achieving either higher availability with the same number of copies or same availability with fewer copies.

We introduce two algorithms that correspond to OP1 and OP2 from Section III-B. Input and output relationships between the algorithms and JFP inference are given in Fig. 1. The first one, described via pseudo code in Fig. 5, finds the deployment set of given cardinality with the lowest JFP. This is useful when the service provider has a fixed budget and expect the highest possible resilience under current state of the edge servers. The set of admissible servers, S', which is one of the inputs to the algorithm, deserves specific consideration. In most cases, resilience is not the only QoS requirement of a service and other factors such as network latency, available bandwidth, computational capacity are also considered. Since optimization of all these factors is beyond the scope of this paper and the purpose is to demonstrate the significance of JFP, we assume that S' is prefiltered to contain only the servers that satisfy aforementioned requirements. In a real world setting, JFP can be used as one of the determinants in a multi-objective scheduler, along with aforementioned QoS requirements.

```
Algorithm OPTIMIZE–JFP
Input Available servers: $S' \subseteq S$
    Initial copy of the service: $c$
    Number of copies to deploy: $m$
Output Optimum deployment set: $\Delta$
    JFP of the optimum deployment set: $p_{min}$
 1: $\mathcal{C} \leftarrow \{C \subseteq S' \mid |C| = m\}$ # m-subsets of candidates
 2: $p_{min} \leftarrow +\infty$
 3: for all $C \in \mathcal{C}$ do                # For each candidate set
 4:     $p \leftarrow \mathrm{JFP}(C)$             # Compute JFP as in (13)
 5:     if $p < p_{min}$ then       # If better than current best
 6:         $p_{min} \leftarrow p$                 # Update best JFP
 7:         $S_\Delta \leftarrow C$          # Update best candidates set
 8:     end if
 9: end for
10: $\Delta \leftarrow \emptyset$           # Initially deployment set is empty
11: for all $s \in S_\Delta$ do            # For each server in $S_\Delta$
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
```

Figure 5. Pseudo code description of failure probability optimization given the size of deployment set.

```
Algorithm OPTIMIZE–SIZE
Input Available servers: $S' \subseteq S$
    Initial copy of the service: $c$
    Maximum acceptable failure probability: $p$
Output Optimum deployment set: $\Delta$
for $m = 1$ to $|S'|$ do        # Test increasing $\Delta$ sizes
    $\{\Delta, p_{min}\} \leftarrow$ OPTIMIZE–JFP$(S', c, m)$
                  # Find the optimum deployment of size $m$
    if $p_{min} \le p$ then              # If JFP is acceptable
        break                            # Stop the search
    end if
end for
```

Figure 6. Pseudo code description of deployment set size optimization given maximum acceptable JFP.

OPTIMIZE–JFP() iterates over all m-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 $|S'| \ge m$, there exists $\mathcal{O}(2^{|S'|})$ subsets and hence calls to JFP(), which itself has the time-complexity of $\mathcal{O}(|S'|\, 2^M)$. Thus, the time-complexity of the algorithm is $\mathcal{O}(N\, 2^{N+M})$, where N is the number of admissible servers (i.e. $|S'|$) and M is the length of the longest factor (in terms of number of variables) in the JFP inference step.

The second algorithm, OPTIMIZE–SIZE() in Fig. 6, on the other hand, does not have an input parameter for deployment size, but for maximum acceptable JFP value. Starting from $m = 1$ and incrementing m at each iteration (line 1), it calls OPTIMIZE–JFP(), which in turn returns the optimum deployment set and corresponding JFP for the given value of m (line 2). When a deployment set that satisfies the maximum JFP constraint is found, search is stopped and the algorithm outputs the set (lines 3–5). Consequently, 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 OPTIMIZE–JFP() are made, resulting in an overall time-complexity of $\mathcal{O}(N^2\,2^{N+M})$.

An important point is that, although DAFR algorithms find optimum solutions, neither of them gives any availability guarantee since the optimized parameter, JFP, is stochastic. In other words, they provide the best effort given the function for forecasting future failures. As emphasized in the introduction of this paper, this suffices for typical latency-sensitive edge services except safety-critical and hard real-time systems such as self-driving cars or telemedicine.

VI. Evaluation

A. Experimental Setup

We evaluate JFP forecasting technique and DAFR algorithms via simulation of a practical use case. An edge computing vendor receives resource requests from its customers (i.e. service providers) who have certain resilience objectives. The vendor must, not only satisfy customer requirements and avoid SLA violations, but also minimize over-provisioning of service replicas to more effectively monetize limited resources.

To that end, we implement a task scheduling simulator for edge computing in Java. The simulator generates 10,000 service requests at equal time intervals. Availability definition of the service can randomly belong to either category described in Section III-A, namely load sharing and active-standby, whereas number of copies are chosen uniformly at random from the range $[1, 5]$. Then, generated tasks are deployed by the vendor on a subset of currently available servers via proposed and baseline algorithms. Availability of servers at a time is obtained from real failure traces. Unless stated otherwise, we repeat our simulations for 10 disjoints sets of 100 randomly selected servers from each data set and report average results. For each set of servers, failures in the first half of the total time span are reserved for DBN learning and the rest for task scheduling simulation.

OPTIMIZE–JFP, OPTIMIZE–SIZE, and baseline algorithms are implemented in Java (JDK 1.8). The main program executes them sequentially by generating a new service request at each iteration. Available edge servers based on real failure traces at each interval are taken as the set of admissible candidates. Deployments outputted by each algorithm are compared to future failure traces in running time of the task in order to log statistics such as percentage downtime or number of SLA violations. Data communication between the modules shown in Fig. 1 is implemented through shared variables for DBN graph, CPTs, user requirements (p and m), etc. Simulations are executed on a machine with quad-core i7-6700HQ CPU and 8GB of memory.

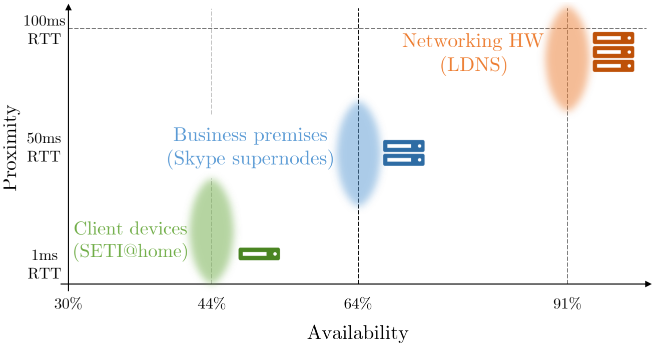

Figure 7. Comparison of edge computing implementations and corresponding data sets in terms of proximity, availability, and computing power.

B. 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 the present time. This is because of not only the novelty of 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 [25]. Consequently and as with the previous work in system reliability literature [26]–[30], we take failure traces and infrastructure information from real world systems, and simulate a distributed system with synthetically generated workload. Moreover, implementation of edge computing paradigm is not yet standardized which results in a diverse set of reliability characteristics. In the case of fog computing [31], for instance, virtualization infrastructure is the networking hardware such as routers, switches, or proxy servers, which are expected to have a certain level of reliability. In other approaches, however, virtual computing environment is formed by, for instance, cloudlets located in business premises without extensive support systems [32] or even client devices (e.g. desktop PCs, tablets, smart phones, etc.) [33], [34] which results in significantly lower availability. Due to their distinct strengths, these different infrastructures will most probably coexist. To generalize the outcomes, we run separate simulations with real failure traces from three large-scale distributed systems with distinctive characteristics as depicted in Fig. 7. Here, mean availability of servers are calculated from the values in corresponding data sets, whereas the ordinal

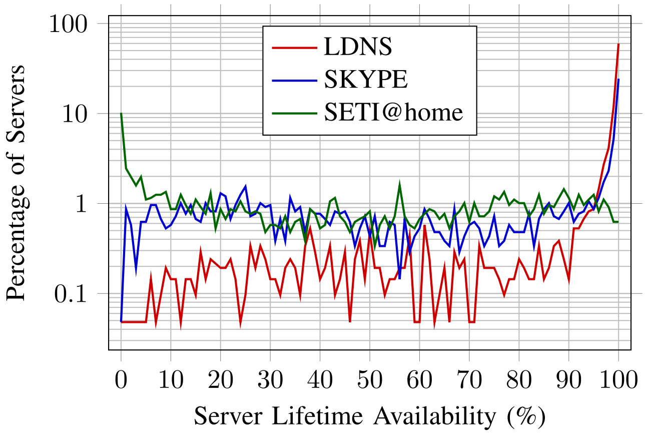

Figure 8. Distribution of servers to availability levels.

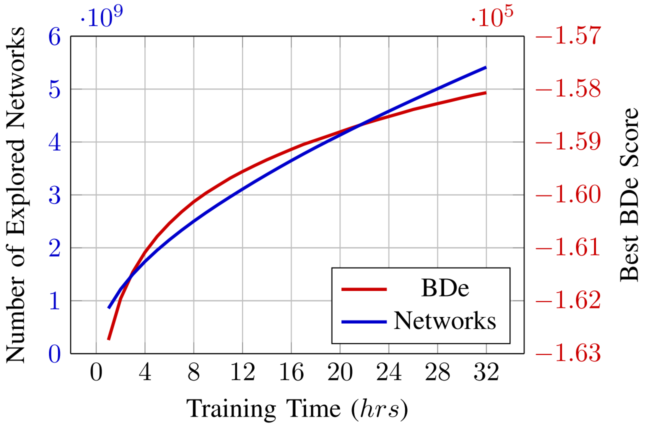

Figure 9. DBN learning performance over time.

values of computing power (represented with server icons) and average round-trip time (RTT) are evaluated based on literature [1], [2], [35].

In addition, availability distribution of servers in each data set is given in Fig. 8. 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 March 24, 2004. 62,201 LDNS servers in this data set substitute for edge servers deployed on networking hardware, hence exhibit high reliability. As shown in Fig. 8, around 80% of the servers achieve higher than 95% availability. Second data set [37] belongs to the hybrid peer-to-peer system of Skype voice over IP service. Supernodes, which are selected by Skype protocol based on reachability and spare bandwidth, are pinged in 30-minute intervals between September 18 to October 4, 2005. We use a distilled version of the data sets which contains 2,081 supernodes. SKYPE data set represents a reliability middle ground between implementing edge computing on dedicated servers and on client devices. Around 35% of nodes are available more than 95% of the time and the rest are distributed to remaining availability intervals in a roughly uniform way. Finally, failure traces of UC Berkeley SETI@home volunteer computing project is collected [38] from 226,208 personal computers between April 1, 2007 and January 1, 2009. As might be expected, this data set models an edge infrastructure that is formed by virtualized client devices with high churn and low reliability. Nearly one fifth of the nodes have 5% or lower availability whereas only 4% achieve 95% or higher availability.

C. *DBN Learning*

For learning the DBN structure, G, we utilize Banjo framework[4] by Duke University. Use of an open-source and well tested code for machine learning tasks instead of our own implementation provides us with superior reliability and error-free results. Other machine learning toolboxes such as *Weka*[5] and *bnlearn*[6] do not have native support for training DBNs. Finding the optimum structure that best describes the data is an NP-complete problem [39]. 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 our custom program in Java that obtains each conditional probability via maximum likelihood estimation (MLE). 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. 3 to illustrate CPT training, second row in Table II is estimated via MLE as shown in (17). Here, σ is a function that maps logical true to integer 1 and false to 0, whereas f_1^t is the interested event.

$$\Pr(f_1^t \mid f_2^{t-1}, \neg f_3^t) = \frac{\sum_{t=2}^{T} \sigma(f_1^t \wedge f_2^{t-1} \wedge \neg f_3^t)}{\sum_{t=2}^{T} \sigma(f_2^{t-1} \wedge \neg f_3^t)} \quad (17)$$

We run several experiments to investigate the effect of training time to the network accuracy. Network accuracy is defined as the extent that a network represents data, and measured by likelihood-equivalence Bayesian Dirichlet (BDe) metric [40]. As Fig. 9 demonstrates, number of explored networks grow at a near-linear rate, whereas BDe score improves quite slowly and with a decreasing rate. 6.4-fold increase in network size results in only 2.9% higher

[4] https://users.cs.duke.edu/ amink/software/banjo/

[5] https://www.cs.waikato.ac.nz/ml/weka/

[6] http://www.bnlearn.com/

score. The results here are from a DBN training on the 750 randomly chosen servers of the SETI@home data set, however experiments with others, which are omitted for brevity, yield almost identical results. Based on this analysis, we resolve that training time in the order of minutes is sufficient for relatively small DBNs ($\leq$ 1000 variables). In the rest of evaluation, we used DBNs that are trained for 10 minutes which corresponds to 300 Million explored networks. Parameter learning time, on the other hand, is in the order of seconds and negligible compared to structure learning time.

D. Baseline Algorithms

Both DAFR algorithms proposed, OPTIMIZE–JFP and OPTIMIZE–SIZE, are compared against two other techniques from the literature as well as random (i.e. failure-oblivious) scheduling to evaluate the significance of improvement.

*Random Scheduling (*RANDOM*):* Each copy is placed to a server chosen uniformly at random. Simulation time is used as the random seed.

*Prior-based Scheduling (*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 proposed in [16] 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 Scheduling (*TTF*):* Support Vector Machine (SVM) regression is successfully utilized in [17] for forecasting future time to failure (TTF) values. To compare this technique to ours, we implement the sequential minimal optimization [41], which is arguably fastest known algorithm for SVM regression. In this baseline, tasks are scheduled to the servers with the longest remaining time to failure, which can be calculated as RTTF = PTTF − TSLF, i.e. the difference between predicted time to failure and time since last failure. Sample data size for historical TTF values is 50 in our experiments.

E. Results and Discussion

1) OPTIMIZE–JFP: Fig. 10 and 11 show mean downtime percentages of deployed services using LDNS and SKYPE data sets respectively. In both experiments, OPTIMIZE–JFP achieves very high availability: 99.9998% in the worst case for LDNS and 99.9252% for SKYPE, outperforming all baselines. Among them, TTF performs the best, whereas failure-oblivious scheduling (represented with RANDOM) leads to up to 1.0% downtime in LDNS and 4.4% in SKYPE. Downtime percentages of all algorithms are significantly lower in LDNS reflecting higher reliability of the DNS servers. In Fig. 13 and 14, on the other hand, we present the mean percentage of the time that at least one deployment fails but the minimum service level is maintained. This does not necessarily lead to downtime, but temporary loss of redundancy unless more failures follow. The minimum service level is based on the availability definition of each service described in Section III-A and corresponds to a maximum tolerable number of failed deployments. Interestingly, in LDNS case, the loss of redundancy with OPTIMIZE–JFP is slightly more common than that of TTF, and total unavailability (sum of downtime and loss of redundancy) in the case of these two algorithms are almost the same. This demonstrates that, availability enhancement by OPTIMIZE–JFP arises from not only detecting individually most reliable servers but also accounting for their joint failure. Hence, the significance of dependencies in failure forecasting and failure-aware scheduling is confirmed.

Since proposed and baseline scheduling algorithms do not feature re-evaluation of failures and migration of tasks, it is expected that the service availability decreases as the task length increases. It is not likely that certain edge servers will run failure free when the task last several days. This trend is particularly evident in experiments with SETI data set, in which we are able to evaluate long term services owing to the vast amount of available data. SETI experiments are conducted with random sets of 750 servers, instead of 100 as with others. As seen in Fig. 12, TTF outperforms OPTIMIZE–JFP for tasks longer than 60 hours. We believe the reason is that the dependency information, which is better captured by OPTIMIZE–JFP, loses its currency earlier than the reliability trends of individual servers, which are better captured by TTF. Indeed, the loss of redundancy (Fig. 15) in all methods seem to converge to a value representing the typical availability of a server in the data set (44%). This shows that the initial selection of servers does not maintain high availability indefinitely and migrations are necessary. Considering a reasonable task length up to 10 hours, worst case availability using the proposed algorithm is 99.7478%.

Characteristics of downtime and loss of redundancy trends with SETI data set are similar to those with LDNS (e.g. OPTIMIZE–JFP has higher loss of redundancy but lower downtime than TTF for tasks up to 60 hours). Consequently, we identify two classes of edge computing infrastructures for service resilience based on their availability distribution. In the first case represented by SKYPE data set, edge servers are highly heterogeneous in terms of their availability levels. Thus, it is of capital importance to detect the highly available ones. In the second case however, availability of vast majority of the servers are quite similar: either very high as with LDNS or very low as with SETI. This poses an additional challenge of joint failure prediction. In other words, co-occurrence of failures gains more importance and demands more attention, when all alternative servers have roughly equal availability. According to our results, proposed algorithm accomplishes both these tasks effectively.

Although mean availability percentages show an overall

◆ RANDOM ■ PRIOR ● TTF ✶ OPTIMIZE–JFP

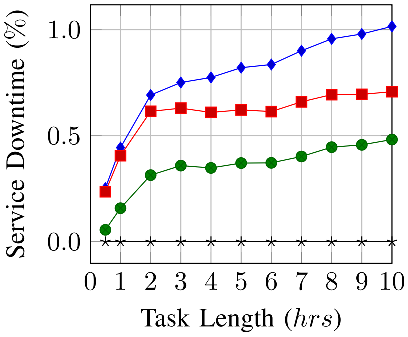

Figure 10. Service downtime results (LDNS).

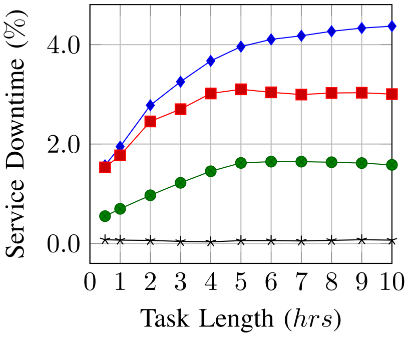

Figure 11. Service downtime results (SKYPE).

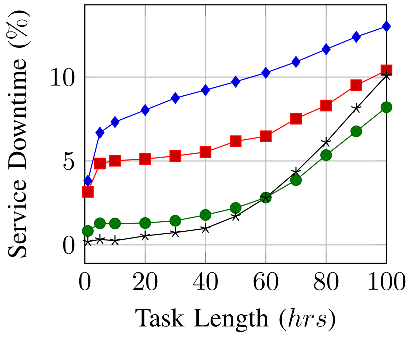

Figure 12. Service downtime results (SETI).

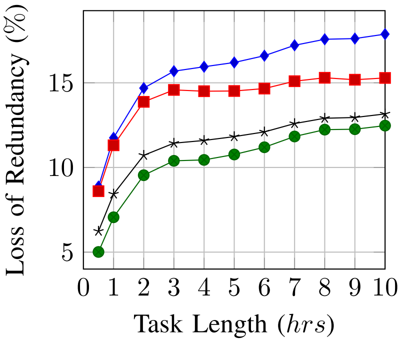

Figure 13. Loss of redundancy results (LDNS).

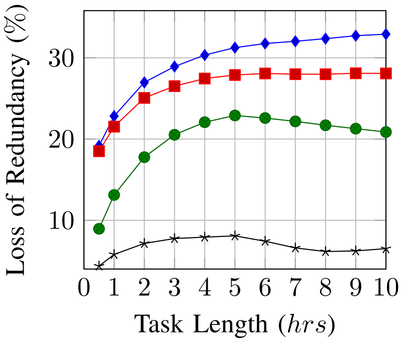

Figure 14. Loss of redundancy results (SKYPE).

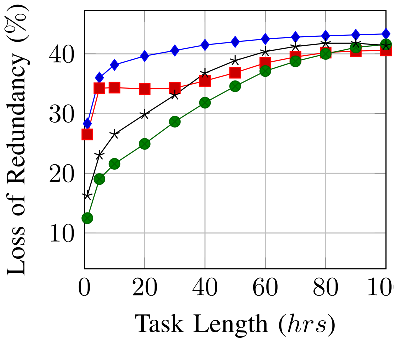

Figure 15. Loss of redundancy results (SETI).

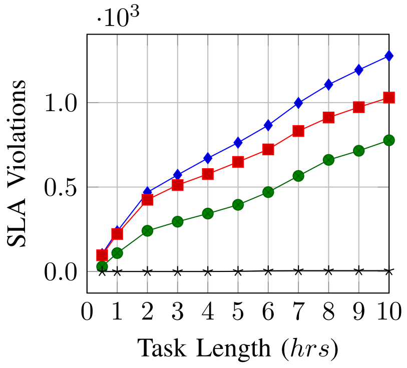

Figure 16. SLA violation results (LDNS).

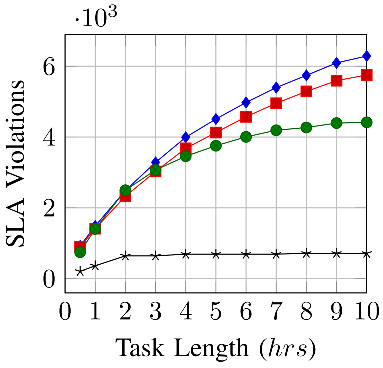

Figure 17. SLA violation results (SKYPE).

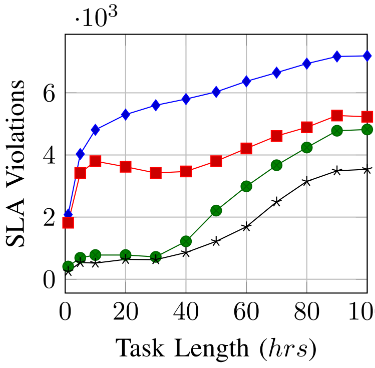

Figure 18. SLA violation results (SETI).

picture, a practical concern for service and infrastructure providers alike is the frequency of SLA violations. Details of particular violations may be overwhelmed by thousands of deployments when only the averages are reported. Thus, in Fig. 16 to 18, we present the number of SLA violations caused by each algorithm. SLA requires a strict 99.9% availability in these experiments. Again, OPTIMIZE–JFP outperforms all baselines significantly. The maximum number of violations is 5 in 10,000 requests with LDNS data set, 717 with SKYPE, and 3,540 with SETI.

2) OPTIMIZE–SIZE: We also conduct several experiments with dynamic number of copies using the proposed OPTIMIZE–SIZE scheduling algorithm. In Fig. 19 and 20, number of deployed copies and percentage downtime are reported with various values of p for SKYPE and SETI, which are the representatives for the two classes of data sets as described above. To illustrate the benefits, proposed algorithm is compared to the static TTF algorithm, the best performing baseline, configured to deploy two copies of each service. This corresponds to 20,000 deployments that are

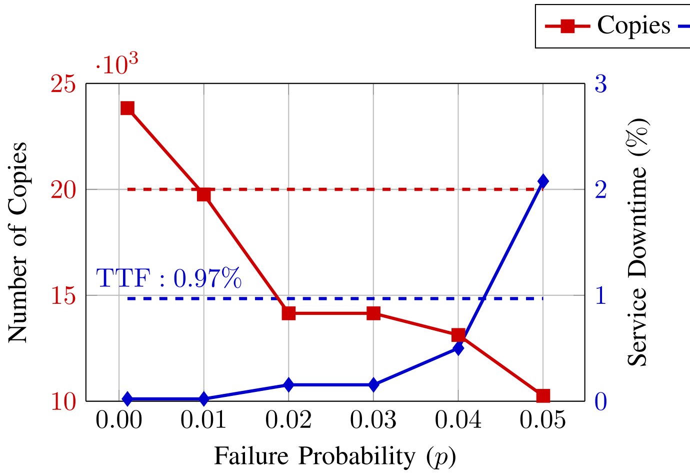

Figure 19. Downtime and copies for OPTIMIZE–SIZE (SKYPE).

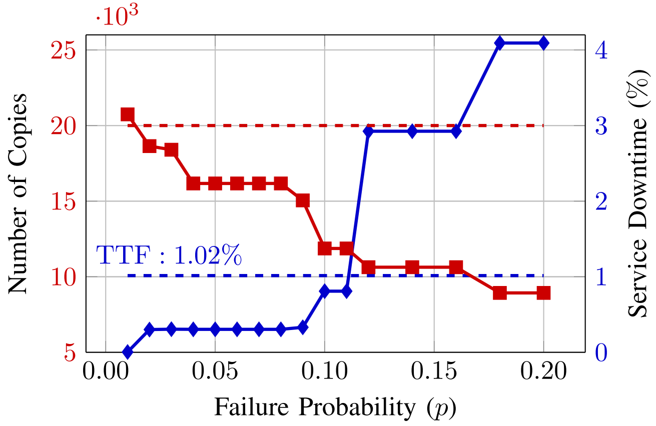

Figure 20. Downtime and copies for OPTIMIZE–SIZE (SETI).

indicated with red dashed lines. 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% downtime (indicated with blue dashed lines) 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 OPTIMIZE–SIZE suffers higher downtime with the same number of copies as the baselines or that baselines achieve 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. 19 and $[0.02, 0.11]$ in Fig. 20), it is Pareto dominant, that is, proposed algorithm achieves higher availability with fewer copies.

VII. Related Work

Fault tolerance or resilience is a well studied topic within the context of cloud computing. Widely used strategies can be grouped under checkpointing [42]–[44], re-execution [43], [45], and replication [44]–[46]. As we also discussed in Section I, high overhead and long re-activation time makes first and second groups of strategies infeasible when real-time computing is required [47]. 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 availability levels of servers within a cloud data center are not noticeably different. One of the few exceptions is [47] 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 [48], an analytical model is proposed to estimate the reliability of subscribers in publish / subscribe systems. Interested reader may also refer to the comprehensive survey [49] for further details about other fault tolerance approaches within the cloud computing context.

Although essential for its success, resilience in edge computing is still an open research area [15]. An early discussion of reliability challenges in fog computing with respect to cloud is presented in [50]. However, few attempts are made to address these challenges. In our previous work [18], we introduced a technique that exploits causal relationships between different types of failure and channel all QoS related parameters through VM availability. Nebula [51], 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 replica site selection. Cloud visitation platform [52], which aims to cope with hardware heterogeneity problem in federated clouds and fog via hardware awareness, solves failure resilience only at local level. When a server fails, applications deployed on it are migrated to another server, possibly in a different cloud or fog node. Cardellini et al. [16] extends well known distributed stream processor, Apache Storm, by adding QoS awareness capability. Proposed scheduler chooses resources based on network latency as well as utilization and availability. Here, recent availability of nodes is used instead of predicting future availability. FogStore [53], is a distributed data store, which handles replica placement and consistency management. As the replicated objects are data blocks, the focus of this work is on read and write latency. Recently, a recovery scheme for edge computing failures is proposed in [54]. However, only the failures that are caused by overloaded resources are taken into account. Traffic data is monitored to detect overloaded nodes and their load is shared with others. Odin platform [55], is a practical application of fault tolerance for distributed servers. It detects failures and create backup scenarios in CDNs.

Failure dependency is considered in the context of other distributed systems. In [7], a failure prediction framework that exploits temporal and spatial correlation between failure events is suggested. High accuracy of several prediction algorithms including neural networks leads the authors to the conclusion that failures are indeed correlated in distributed systems and this information is useful for failure prediction. Similarly, CFPA [9] is a failure prediction algorithm that learns shared risk groups from data and assigns failure probabilities to nodes. Both these works benefit from failure dependencies in predicting failures and availability of individual nodes. However, they do not consider overall availability of services with components and replicas in different nodes. Another study that focuses on spatially correlated failures is [8]. Here, authors analyze traces from 15 distributed systems to understand and model correlation characteristics of failures, specifically group arrival process, group size, and downtime. Their results demonstrate that correlated failures are present in all analyzed systems and they are the reason for the majority of downtime in 7 of these systems.

VIII. Conclusion

Edge computing is a promising advancement in order to incorporate future Internet services such as Internet of Things, mobile and body computing, Industry 4.0, and autonomous vehicles into the cloud ecosystem by bringing computation to the edge of the network, where data is produced and consumed. This could be otherwise impossible due to two crucial characteristics that are common to these services, namely data-intensity and time-sensitivity. However, renouncing protective environment of massive data centers in favour of geographical prevalence and locality, means less reliable compute resources in the face of frequent hardware and software failures. Existing resilience techniques for cloud systems are not applicable due to scale, distribution, and heterogeneity of edge computing resources.

In distributed systems, a failure in one server may be correlated to failures in others due to either common causes or propagation. Being informed about such dependencies between edge servers allows not only forecasting future availability of servers, but also identifying servers that are less likely to fail concurrently. In this work, we propose DBN based modeling and learning of failure dependencies in edge computing systems. DBN facilitates efficient and effective JFP inference for any set of servers. Thus, we make use of JFP as a future availability forecast to evaluate candidate deployments of an edge service that is replicated in multiple distributed sites. Simulation results show that proposed technique outperforms widely-used algorithms in the literature in terms of both availability and cost. The outcomes are generalized to different replication schemes as well as failure characteristics from multiple large-scale distributed systems.

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 latency constraints, would be viable for an edge–cloud or pure edge solution provided that sufficient level of failure resilience is provided. We demonstrate that DAFR techniques proposed in this paper can achieve similar availability levels to cloud data centers, in the presence of low-latency yet failure-prone edge servers. This would be impossible or cost-inefficient via existing resilience techniques that are intended for centralized systems. On the other hand, network failures, which are currently handled the same way as hardware and software failures, have particular impact on the availability of edge services since they are deployed in distributed sites with possibly intermittent network connection. Our future objectives include explicitly factoring communication reliability in our resilience model to cover all relevant aspects of the edge computing landscape in sufficient depth. Another direction is the consideration of safety-critical applications with hard real-time requirements. To this end, failover techniques can be supported with a proactive failure mitigation mechanism, which reevaluates JFP at runtime and dynamically adjusts the number and locations of copies.

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. Icons in Fig. 2 are designed by Flaticon.

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

[3] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[4] J. Bort, "The massive AWS outage hurt 54 of the top 100 internet retailers – but not Amazon," Business Insider, http://www.businessinsider.de/aws-outage-hurt-internet-retailers-except-amazon-2017-3, accessed: 2018-03-21.

[5] G. Linden, "Make Data Useful," http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt, accessed: 2018-03-21.

[6] L. Ponemon, "Cost of Data Center Outages," Ponemon Institute, Tech. Rep., 2016.

[7] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 41.

[8] M. Gallet, N. Yigitbasi, B. Javadi, D. Kondo, A. Iosup, and D. Epema, "A model for space-correlated failures in large-scale distributed systems," in *European Conference on Parallel Processing*. Springer, 2010, pp. 88–100.

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

[10] G. Ananthanarayanan, P. Bahl, P. Bodk, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[11] J. Yoon, P. Liu, and S. Banerjee, "Low-cost video transcoding at the wireless edge," in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 129–141.

[12] K. Lee, J. Flinn, and B. D. Noble, "Gremlin: scheduling interactions in vehicular computing," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 4.

[13] G. Kar, S. Jain, M. Gruteser, F. Bai, and R. Govindan, "Real-time traffic estimation at vehicular edge nodes," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 3.

[14] A. Chowdhery, M. Levorato, I. Burago, and S. Baidya, "Urban iot edge analytics," in *Fog Computing in the Internet of Things*. Springer, 2018, pp. 101–120.

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

[16] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "On QoS-aware scheduling of data stream applications over fog computing infrastructures," in *IEEE Symposium on Computers and Communication*. IEEE, 2015, pp. 271–276.

[17] M. das Chagas Moura, E. Zio, I. D. Lins, and E. Droguett, "Failure and reliability prediction by support vector machines regression of time series data," *Reliability Engineering & System Safety*, vol. 96, no. 11, pp. 1527–1534, 2011.

[18] A. Aral and I. Brandic, "Quality of Service Channelling for Latency Sensitive Edge Applications," in *IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2017, pp. 166–173.

[19] Z. Ghahramani, "Learning dynamic Bayesian networks," in *Adaptive processing of sequences and data structures*. Springer, 1998, pp. 168–197.

[20] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 139–147.

[21] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 1995.

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

[23] N. L. Zhang and D. Poole, "Exploiting causal independence in bayesian network inference," *Journal of Artificial Intelligence Research*, vol. 5, pp. 301–328, 1996.

[24] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[25] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release," in *2012 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2012, pp. 1279–1286.

[26] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. ACM, 2011, p. 33.

[27] T. Estrada and M. Taufer, "On the effectiveness of application-aware self-management for scientific discovery in volunteer computing systems," in *Proceedings of 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[28] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of parallel and distributed computing*, vol. 72, no. 10, pp. 1318–1331, 2012.

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

[30] E. M. Heien, D. P. Anderson, and K. Hagihara, "Computing low latency batches with unreliable workers in volunteer computing environments," *Journal of Grid Computing*, vol. 7, no. 4, p. 501, 2009.

[31] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[32] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.

[33] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[34] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[35] 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*. Springer, 2014, pp. 169–186.

[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 *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 1–14.

[37] S. Guha and N. Daswani, "An experimental study of the skype peer-to-peer voip system," Cornell University, Tech. Rep., 2005.

[38] B. Javadi, D. Kondo, J. Vincent, and D. Anderson, "Mining for Statistical Availability Models in Large-Scale Distributed Systems: An Empirical Study of SETI@home," in *17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2009.

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

[40] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

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

[42] Í. Goiri, F. Julia, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in *IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2010, pp. 455–462.

[43] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "Correlation Modeling and Resource Optimization for Cloud Service with Fault Recovery," *IEEE Transactions on Cloud Computing*, 2017.

[44] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 193–228, 2013.

[45] Q. Zheng, "Improving MapReduce fault tolerance in the cloud," in *IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2010, pp. 1–6.

[46] W. Zhao, P. Melliar-Smith, and L. E. Moser, "Fault tolerance middleware for cloud computing," in *IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010, pp. 67–74.

[47] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, 2016.

[48] T. Pongthawornkamol, K. Nahrstedt, and G. Wang, "Reliability and timeliness analysis of fault-tolerant distributed publish/subscribe systems." in *ICAC*, 2013, pp. 247–257.

[49] K. Bilal, O. Khalid, S. U. R. Malik, M. U. S. Khan, S. U. Khan, and A. Y. Zomaya, "Fault Tolerance in the Cloud," in *Encyclopedia of Cloud Computing*. John Wiley & Sons, Ltd, 2016, pp. 291–300.

[50] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *20th International Conference on Systems, Signals and Image Processing*. IEEE, 2013, pp. 43–46.

[51] M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing," in *IEEE International Conference on Cloud Engineering*. IEEE, 2014, pp. 57–66.

[52] M. Zhanikeev, "A cloud visitation platform to facilitate cloud federation and fog computing," *Computer*, vol. 48, no. 5, pp. 80–83, 2015.

[53] R. Mayer, H. Gupta, E. Saurez, and U. Ramachandran, "Fogstore: Toward a distributed data store for fog computing," in *Fog World Congress*, 2017.

[54] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer Systems*, vol. 70, pp. 138–147, 2017.

[55] M. Calder, M. Schröder, R. Gao, R. Stewart, J. Padhye *et al.*, "Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System," in *15th USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2018.