



Deposited via The University of Leeds.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/179226/>

Version: Accepted Version

Article:

Nikolić, J, Jubatyrov, N and Pournaras, E (2021) Self-Healing Dilemmas in Distributed Systems: Fault Correction vs. Fault Tolerance. IEEE Transactions on Network and Service Management, 18 (3). pp. 2728-2741. ISSN: 1932-4537

<https://doi.org/10.1109/TNSM.2021.3092939>

This item is protected by copyright. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Self-healing Dilemmas in Distributed Systems: Fault Correction vs. Fault Tolerance

Jovan Nikolić, Nursultan Jubatyrrov and Evangelos Pournaras

Abstract—Large-scale decentralized systems of autonomous agents interacting via asynchronous communication often experience the following self-healing dilemma: fault detection inherits network uncertainties making a remote faulty process indistinguishable from a slow process. In the case of a slow process without fault, fault correction is undesirable as it can trigger new faults that could be prevented with fault tolerance that is a more proactive system maintenance. But in the case of an actual faulty process, fault tolerance alone without eventually correcting persistent faults can make systems underperforming. Measuring, understanding and resolving such self-healing dilemmas is a timely challenge and critical requirement given the rise of distributed ledgers, edge computing, the Internet of Things in several energy, transport and health applications. This paper contributes a novel and general-purpose modeling of fault scenarios during system runtime. They are used to accurately measure and predict inconsistencies generated by the undesirable outcomes of fault correction and fault tolerance as the means to improve self-healing of large-scale decentralized systems at the design phase. A rigorous experimental methodology is designed that evaluates 696 experimental settings of different fault scales, fault profiles and fault detection thresholds in a prototyped decentralized network of 3000 nodes. Almost 9 million measurements of inconsistencies were collected in a network, where each node monitors the health status of another node, while both can defect. The prediction performance of the modeled fault scenarios is validated in a challenging application scenario of decentralized and dynamic in-network data aggregation using real-world data from a Smart Grid pilot project. Findings confirm the origin of inconsistencies at design phase and provide new insights how to tune self-healing at an early stage. Strikingly, the aggregation accuracy is well predicted as shown by high correlations and low root mean square errors.

Keywords—self-healing; fault correction; fault tolerance; fault detection; distributed system; agent; gossip; aggregation

I. INTRODUCTION

SEVERAL complex systems in nature and society often exhibit striking reliability, a result of timely choosing, applying and orchestrating multiple self-healing and adaptation strategies. For instance, effectively mitigating blackouts in power grids requires several tailored fault-correction and fault-tolerance mechanisms, whose coordination is way more sophisticated than simply repairing the originating fault of a power line. These include resilient topological design, load-shedding, operating reserves, islanding and active devices among others [1]. While a level of sophisticated self-healing in

natural systems is usually a result of self-adaptation and evolution, in artificial socio-technical systems with central control such as power grids, reliability remains to a high extent a result of planning based on past experience, adaptations based on precomputed simulations and manual human interventions by system operators.

Decentralized autonomous systems recently witness a phenomenal rise with the applicability of distributed ledgers, edge computing, multi-agent systems and the Internet of Things in several sectors of society, e.g. energy, transport, health, agriculture, etc [2]. Large-scale asynchronous distributed environments experience unprecedented network/system uncertainties that challenge the orchestration of self-healing strategies: Fault detection inherits these uncertainties that can make a faulty remote process indistinguishable from a slow process [3], [4]. As a result, a reactive system recovery may turn into an undesirable fault correction of a non-faulty system, which in turn may cause an actual fault that could be prevented instead with fault tolerance, i.e. a more proactive and preventive system maintenance. Figure 1 illustrates the self-healing dilemma problem studied in this paper.

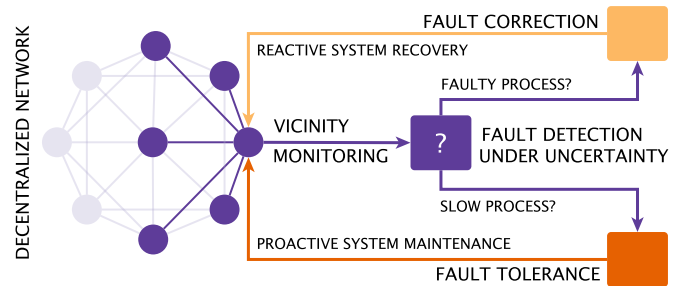


Figure 1. The self-healing dilemma of agents comprising a large-scale decentralized networked systems: Fault detection within the vicinity of an agent comes with network uncertainties: Is a remote process faulty or slow? Which self-healing strategy should be applied? Fault correction as a reactive system recovery with the risk of introducing new faults? Or, fault tolerance as a proactive system maintenance with the risk of letting a fault affecting system performance?

The effective resolution of such fault-correction vs. fault-tolerance dilemmas promises more effective self-healing mechanisms for large-scale decentralized networked systems with uncertainties. A more informed and timely application of fault correction and fault tolerance has the potential to decrease the likelihood of new faults by self-healing itself, against which decentralized systems remain more resilient with a lower communication and processing cost. This paper models and classifies the possible outcomes of self-healing dilemmas between pairs of agents, where one monitors the

J. Nikolić is with Google, Zurich, Switzerland, e-mail: jovan-nikolic@google.com.

N. Jubatyrrov is with Facebook, London, UK, e-mail: nurs@fb.com.

E. Pournaras is with University of Leeds, Leeds, UK, e-mail: e.pournaras@leeds.ac.uk.

Manuscript received November 20, 2020; revised March, 2021.

health status of the other, while both can arbitrary defect. These outcomes are possible desirable and undesirable states (inconsistencies) into which self-healing can fall. The cost of these inconsistencies by undesirable outcomes is further formalized and distinguished within fault scenarios during system runtime. These fault scenarios have the novelty of predicting the performance of self-healing mechanisms without knowledge of the computational/application scenario, overlying algorithms or application data. The modeled fault scenarios are applied and studied to the computational case study of decentralized dynamic in-network aggregation [5] by introducing a new prototyped fault-detection mechanism based on gossip-based communication [6] and agent migrations [7]. Fault correction and fault tolerance are employed to improve the estimates of aggregation functions made by each node in the network. These estimations approximate, for instance, the total power demand based on which decentralized demand-response programs and power markets operate [8]. To preserve accuracy, fault correction performs risky recomputations of the total power demand reactively when faults are detected with uncertainty, while fault tolerance relies entirely on occasional proactive recomputations to capture changes.

A rigorous experimental methodology is introduced to tackle three objectives: (i) Profiling of the inconsistency cost generated by the modeled fault scenarios across 696 experimental settings with varying fault scales, fault profiles and fault-detection thresholds. (ii) Validation of whether the inconsistency cost measured by the modeled fault scenarios is a good general predictor of the accuracy observed in the application scenario of decentralized aggregation of real-world power consumption data. (iii) Comparison of different model calibrations for the prediction of aggregation accuracy by relying on application-independent features.

The findings of the experimental evaluation have significant implications and impact for system designers and operators: By (re)using the general-purpose fault scenarios for vulnerability analysis, the resilience of different system designs can be assessed at an early stage with low cost and under different fault characteristics, while fault-detection mechanisms can be parameterized more effectively. Application developers can improve the self-healing capabilities of applications at the design phase by predicting the impact of faults and tuning appropriately the application before deployment to lower its cost. They can also plan computational resources for self-healing more effectively.

The contributions of this paper are summarized as follows: (i) The modeling of possible outcomes in agents' self-healing dilemmas. (ii) The modeling of application-independent fault scenarios during system runtime that sufficiently formalize the overall health status of decentralized systems and their impact on self-healing performance. (iii) A general-purpose novel fault-detection mechanism based on gossip-based communication and migrating agents. (iv) The applicability of the fault-detection mechanism to the Dynamic Intelligent Aggregation Service (DIAS) [5] for the improvement of its aggregation accuracy. Self-corrective operations of DIAS are expanded when nodes massively fail [9]. (v) The profiling of the predicted inconsistencies that different fault scenarios

cause under different fault scales, profiles and fault-detection thresholds. (vi) Three model calibration methods to improve the accuracy of the predicted inconsistencies that rely entirely on application-independent features.

This paper is organized as follows: Section II positions and compares this study with related work. Section III models the uncertainties in fault detection for large-scale asynchronous decentralized systems and introduces the possible outcomes in agents' self-healing dilemmas. Section IV formalizes fault scenarios that predict the cost of inconsistencies caused by faults. Section V illustrates the applicability of the proposed model in a case study of decentralized aggregation. The mechanisms for fault detection, fault correction and fault tolerance to improve aggregation accuracy are outlined. Section VI introduces the experimental methodology that addresses the objectives of this study and Section VII illustrates the findings of the experimental evaluation. Finally, Section VIII concludes this paper and outlines future work.

II. POSITIONING AND COMPARISON TO RELATED WORK

Self-healing mechanisms usually address different types of faults classified according to recent taxonomies [10], [11]. Assuming reliable communication channels, faults are differentiated as follows [10]: (i) *Crash* - agents stop responding and terminate. (ii) *Omission* - agents sporadically skip sending/receiving messages. (iii) *Timing* - agents do not complete a task in a certain time frame. (iv) *Arbitrary* (Byzantine)- agents deviate from the expected behavior and operate unpredictably. Another classification distinguishes between (i) *transient*, (ii) *intermittent*, (iii) *permanent* and (iv) *Byzantine* faults [11]. While transient faults draw parallels with omission ones and are a result of a temporary affecting condition, e.g. network connectivity, intermittent faults are random, temporary and usually result of hardware failure. In contrast, crash hardware faults require repair of the root cause and are a subset of permanent faults. Byzantine faults result in corrupted/malicious agents sharing manipulated, forged or incorrect data. In large-scale decentralized systems, designing self-healing mechanisms exclusively for certain fault types is ineffective. Several such faults can co-occur, cascade or even have a cause-effect relationship resulting in vicious adaptation cycles, e.g. a faulty fault detection causing faulty fault correction and vice versa. Modeling the interplay of faults and formalizing complex fault scenarios as well as their impact on self-healing performance is fundamental and missing.

In such fault scenarios, the reliability of fault detection plays a key role [10]. Two main fault-detection approaches of periodic *heartbeat messages* and *agent interactions* are identified. The latter further distinguishes between *timeout* and *missing callback* detection. Replication [12], [10] is a common approach that supports both fault tolerance and fault correction in terms of guarantying the availability of (backup) resources and repair modules for self-healing [13]. Replication can be *active vs. passive* based on whether replicas are used only when faults occur [14], *adaptive* based on criteria for replication [15], [16], *dynamic* by switching on-the-fly replication schemes [14], or *homogeneous vs. heterogeneous*

based on whether replicas are identical copies or equivalent processes [13]. Replication is applied to check-point schemes based on rollback protocols [17], [18], consensus protocols and hybrid approaches [19]. Replication methods are particularly applicable in multi-agent systems, for instance, group replication via proxy servers [13], replication of agents based on the criticality of their planned actions [16], adjustable group replication with a leader agents [14] or introducing a special class of agents for redundancy maintenance [20]. New replication strategies designed for the Internet of Things and cyber-physical systems are subject of recent work [21], [22]. Self-healing methods [23] can be *preventive (proactive)* [12] or *reactive (resilient)* [11]. The former methods require prediction based on probabilistic modeling and monitoring [24]. The latter ones require learning capabilities from historic data and observations [25], [26].

Despite the large body of work on fault correction and fault tolerance, a recent comprehensive review of such approaches for multi-agent systems identifies as imperative the need for generalized and standardized evaluation of fault-tolerance approaches [10]. Another recent systematic evaluation of 36 state-of-the-art self-healing systems from the research communities of autonomic computing, self-adaptive, self-organizing and self-managing systems (ICAC, SASO, TAAS, SEAMS) is illustrated [27]. Empirical assessments conclude that multiple input traces covering a vast spectrum of failure characteristics are required to predict the performance of a self-healing system. Therefore, generalized models that predict the impact of faults and their correction on large-scale decentralized systems are missing so far [12]. Predicting without knowledge about the computational/application scenario, executed application algorithms and application data is challenging [12]. Such models have the potential to fundamentally influence the understanding of how to design and deploy more cost-effective decentralized self-healing systems. What makes particularly challenging the inception of such general models is the absence of central control units, the agents' autonomy, the network uncertainties and the non-determinism of system operations [13]. In particular, failed processes are often indistinguishable from slow processes in asynchronous decentralized environments that inherit the impossibility of distributed consensus [28], [29]. As a consequence, fault detection inherits such uncertainties [3], [30] (is the process faulty or slow?), which in turn results in dilemmas on what self-healing adaptations to apply, i.e. fault correction vs. fault tolerance. This paper addresses these self-healing dilemmas.

III. SELF-HEALING DILEMMAS

This paper studies self-healing of large-scale decentralized networked systems with faulty nodes. Decentralization means that no single node has full information about all other nodes in the network at a time and each node is connected with a limited number of other nodes. Faults can be a result of system failure, software failure, security attack or any other type of error that makes a faulty node inaccessible to other healthy nodes [13]. Nodes usually depend on each other to perform distributed operations by communicating with each other in a

peer-to-peer fashion. Even if communication is asynchronous, a fault introduces a cost that hinders (i) performance and/or (ii) consistency of a distributed operation. The latter is referred to as *inconsistency cost* and is the main focus of this paper.

Two approaches are distinguished to eliminate these costs: (i) *fault correction* vs. (ii) *fault tolerance*. Fault correction eliminates the performance and inconsistency cost of faults via an effective and timely fault detection and its correction. For instance, consider a master-slave heartbeat mechanism with which a master node monitors the health status of a slave node by receiving periodically heartbeat messages. A fault detection by the master node is the passage of time period without receipt of a heartbeat message. This period is usually selected empirically and universally [31]. In contrast, fault-tolerance mechanisms are designed to decrease the performance and inconsistency cost of faults by preventing a total system break down and allowing a system to continue its operation with an operating quality proportional to the severity of the fault.

The following assumptions are made: (i) Both fault correction and fault tolerance have both, a performance and inconsistency cost. They have performance cost because their operations usually introduce communication and processing overhead. They have inconsistency cost because of uncertainties in fault detection. A fault may be erroneously detected because of high network latency, low convergence speed of the underlying communication model, misconfiguration or poor design in fault detection. For instance, a heartbeat message may not be received because of network fault rather than because of a node fault. The unnecessary correction process consumes resources and introduces potential inconsistencies as system operations are usually not designed to tolerate unnecessary fault corrections. (ii) The performance and inconsistency cost of fault correction and fault tolerance is significantly lower than the respective costs of a system left to be faulty, i.e. without any self-healing. In other words, it makes sense to take care of faults either via fault tolerance or fault correction (or both). (iii) The performance and inconsistency cost of fault correction vs. the ones of fault tolerance depend on the operational state of the system during system runtime and therefore, it is unclear which self-healing approach should be adopted. Based on these three assumptions as well as the focus of this paper on inconsistency cost to eliminate the number of studied dimensions, a study on how to minimize the inconsistency cost in fault correction vs. fault-tolerance dilemmas is illustrated.

Figure 2a models self-healing dilemmas in a decentralized networked system that consists of Node A, B and C. Each node runs a self-healing agent that is responsible to perform fault correction or fault tolerance. Given the focus of this paper on faulty nodes and without loss of generality, the self-healing agents need to operate remotely so that they are not affected by the faults of the parent nodes, i.e. the original host nodes that created them. As a result, they migrate to neighboring nodes as shown in Figure 2b. In practice, the scope of this model covers several systems that have backup components for redundancy. For the sake of this illustration, a heartbeat mechanism is assumed with which self-healing agents monitor the health status of parent nodes as shown

in Figure 2c. Heartbeat messages may not arrive at Node *B* because of (i) a fault in the parent Node *A* and/or (ii) a large latency or network error in the link between Node *A* and *B* [4]. See Figure 2d. Therefore, when heartbeat messages are not anymore received in Node *A*, the dilemma of the self-healing agent is whether to perform fault correction, i.e. establish the new link between *A* and *C* because the parent Node *B* is truly faulty or perform fault tolerance, i.e. not establish a new link because missing heartbeat messages are probably a result of high latency or fault in the link connecting Node *A* and *B*. It is assumed that the applied fault correction is effective if and only if nodes experience faults, otherwise correction introduces an inconsistency cost.

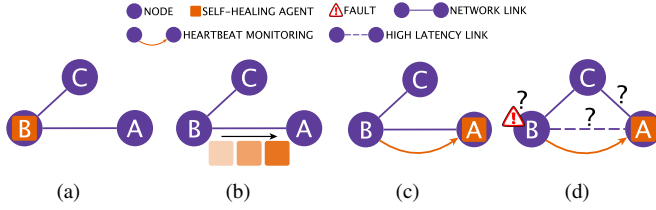


Figure 2. Modeling self-healing dilemmas in a decentralized networked system. (a) Nodes (circles) are connected in a decentralized network and initiate self-healing agents (squares), which are responsible to detect, correct or tolerate faults of their parent node. (b) Self-healing agents find a host node to migrate for redundancy. In this way, a fault on their parent node does not influence their self-healing operations. (c) Self-healing agents monitor the health status of their parent nodes via, for instance, heartbeat signals. (d) Fault detection, determined by a time period during which heartbeat messages are not received from the parent node, comes along with uncertainties. For instance, the heartbeat messages may not be received because of high latency or network error on the *A-B* link, rather than because Node *B* is faulty [4]. In this example, self-healing agents perform fault correction by initiating a new connection with another node if they detect a fault in the parent node. Fault correction eliminates inconsistency cost if a node is actually faulty, but introduces inconsistency cost if the parent node is actually not faulty. Therefore, the self-healing agent has the following dilemma: Should it establish the *A-C* link (fault correction) or wait longer for heartbeat messages to arrive (fault tolerance)? There are four possible outcomes in this decision-making illustrated in Figure 3.

The fault-correction vs. fault-tolerance dilemmas come with four possible outcomes as shown in Figure 3. Note that in Figure 3a and 3b there are two outcomes that do not have inconsistency cost (desirable outcomes). These are the *true negative* outcome that is a result of effective fault tolerance and the *true positive* outcome as a result of effective fault correction. Figure 3c and 3d show the two outcomes with an inconsistency cost (undesirable outcomes). These are the *false negative* outcome¹ by erroneous fault tolerance and the *false positive* outcome by erroneous fault correction.

The next section formalizes the fault scenarios of false negative and false positive outcomes during system runtime, which are the ones that come with an inconsistency cost. The modeled fault scenarios serve the following: (i) Predict the inconsistency cost of decentralized self-healing systems without application information. (ii) Design self-healing agents with a fault-detection capability that minimizes the inconsistency cost during system runtime.

¹False negatives also originate from faults in the node hosting the self-healing agent.

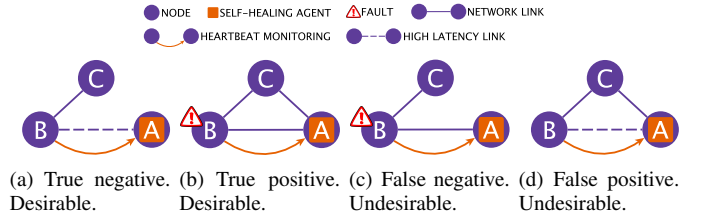


Figure 3. Self-healing dilemmas in fault detection under uncertainty. In this illustrative example, fault correction is the reactive establishment of the link between Node *A* and *C* when no heartbeat messages are received in Node *A* from Node *B*. In contrast, fault tolerance waits further for the heartbeat messages to arrive assuming a delay over the link between Node *A* and *B*. Possible outcomes: (a) *True negative*: Parent Node *B* is healthy and the self-healing agent does not perform fault correction. This outcome has no inconsistency cost. (b) *True positive*: Parent Node *B* is faulty and the self-healing agent performs fault correction. This outcome has no inconsistency cost. (c) *False negative*: Parent Node *B* is faulty but the self-healing agent does not perform fault correction. This outcome has inconsistency cost. (d) *False positive*: Parent Node *B* is healthy but the self-healing agent performs fault correction. This outcome has inconsistency cost.

IV. MODELING FAULT SCENARIOS AT SYSTEM RUNTIME

Table I summarizes the mathematical symbols used in the rest of this paper. Assume once more here the pair of Nodes *A* and *B*, where Node *A* remotely monitors the health status of Node *B*. Tracking the inconsistency cost generated by this pair of nodes during system runtime is complex and challenging due to the uncertainty over the different *fault scenarios* in the following: (i) Faults in either of the two (monitoring and monitored) nodes. (ii) Faulty detection in Node *A*, performed either too early or too late. Table II illustrates the modeled fault scenarios that can occur during system runtime.

Table I
MATHEMATICAL NOTATION USED IN THIS PAPER.

Symbol	Description
T	System runtime
$\tau \in \{1, \dots, T\}$	Time unit
$d \leq T$	Detection time
$t \leq T$	Threshold for fault detection
$F_A, F_B \leq T$	Fault time of Node <i>A</i> and <i>B</i>
l	Total number of fault scenarios
$s \in \{1, \dots, l\}$	Fault scenario
$\epsilon_s^-, \epsilon_s^+ \in \mathbb{R}$	Maximum inconsistency cost of a fault scenario <i>s</i> generated by a false negative and false positive state, during system runtime <i>T</i>
$\rho_s^-, \rho_s^+ \in [0, 1]$	Relative inconsistency cost of a fault scenario <i>s</i> generated by a false negative and false positive state, during system runtime <i>T</i>
$C, C^-, C^+ \in \mathbb{R}$	Total inconsistency cost as well as inconsistency cost generated by a false negative and false positive state during system runtime <i>T</i>
$c \leq T$	Time to restore aggregation accuracy by a corrective operation
$R_B \leq T$	Recovery time of Node <i>B</i>
p	Time to propagate a node descriptor to another node
$n \in \mathbb{N}$	Total number of nodes
$m \in \mathbb{N}$	Number of batches of faulty nodes
$k \in \mathbb{N}, mk \leq n$	Number of faulty nodes at each batch
$\lambda \in [0, 1]$	Calibration factor
$RMSE$	Root mean square error
C_R, C_{GR}	Total inconsistency cost of regression and generalized regression calibration methods
C_D	Predicted inconsistency cost of DIAS

A *system runtime* *T* is studied over which the inconsistency cost is traced as well as the *detection time* $d \leq T$ of Node *A* identifying Node *B* as faulty. Without loss of generality, Node *A* triggers at time *d* fault correction after a *threshold* time period $t \leq T$ during which $(d-t, d-t+1, \dots, d-1, d)$ the fault-detection criterion is satisfied, e.g. a heartbeat message is not received. Otherwise, Node *A* performs fault tolerance, awaiting

Table II

MODELING FAULT SCENARIOS BETWEEN THE PAIR OF NODES *A* AND *B* DURING SYSTEM RUNTIME *T* AND THE RELATIVE INCONSISTENCY COST ρ_s BY THE FALSE POSITIVE AND FALSE NEGATIVE OUTCOMES. THESE SCENARIOS ARE FORMALIZED WITH NODE *A* MONITORING THE HEALTH STATUS OF NODE *B*. THE SCENARIOS ARE MODELED BASED ON THE FOLLOWING INFORMATION: (I) THE HEALTH STATUS OF THE NODES, I.E. WHICH OF THE TWO NODES ARE ON/OFF (HEALTHY/FAULTY). (II) THE TIMING OF THE FAULTS F_A, F_B , I.E. WHICH NODE IS FAULTY FIRST. COST CALCULATIONS RELY ON THE DETECTION TIME *d* OF NODE *A* AND THE DETECTION THRESHOLD *t*.

Depiction	Fault Scenario (<i>s</i>)		Relative Inconsistency Cost (ρ_s)	
	Health Status	Faults Timing	False Positive	False Negative
(1)	A: ON B: ON	-	$\frac{T-d}{T-t}$	-
(2)	A: ON B: OFF	F_B	$\frac{F_B-d}{F_B-t}$	$\frac{d-F_B}{T-F_B}$
(3)	A: OFF B: ON	F_A	$\frac{F_A-d}{F_A-t}$	-
(4)	A: OFF B: OFF	$F_A < F_B$	$\frac{F_A-d}{F_A-t}$	$\frac{T-F_B}{T-F_A} = 1$
(5.1)	A: OFF B: OFF	$F_A > F_B$	$\frac{F_B-d}{F_B-t}$	$\frac{T-F_A}{T-F_B} = 1$
(5.2)	A: OFF B: OFF	$F_A > F_B$	-	$\frac{d-F_B}{F_A-F_B}$
(6)	A: OFF B: OFF	$F_A = F_B$	$\frac{F_B-d}{F_B-t}$	$\frac{T-F_A}{T-F_A} = 1$

further for the heartbeat messages to arrive and relying on underlying preventive maintenance mechanisms for recovery. Both Node *A* and *B* can become faulty at any time $F_A, F_B \leq T$ respectively. At each time $\tau \in \{1, \dots, T\}$ the nodes can be in one of the states of Figure 3. False negative and false positive states generate at each time τ an inconsistency cost value for a given *fault scenario* $s \in \{1, \dots, l\}$ out of *l* possible fault scenarios shown in Table II. Moreover, for a fault scenario *s*, the time during which the pair of Nodes *A* and *B* are in a false negative or false positive state out of the total time period in which they can be in such a state during system runtime is measured by ρ_s^- and ρ_s^+ respectively. Therefore, the total inconsistency cost *C* generated by a pair of agents during the system runtime *T* can be measured as follows:

$$C = C^- + C^+ = \sum_{s=1}^l (\epsilon_s^- \cdot \rho_s^- + \epsilon_s^+ \cdot \rho_s^+), \quad (1)$$

where ϵ_s^- and ϵ_s^+ are the total (maximum) inconsistency cost

that can be generated during system runtime by a fault scenario *s*. These can be a result of a constant unit of inconsistency cost value generated at each time point τ or they can be the output of functions $\epsilon_s^- = \sum_{\tau=1}^T f^-(\tau)$, $\epsilon_s^+ = \sum_{\tau=1}^T f^+(\tau)$ representing an analytical or empirical model [32], [33], [34]. The fault scenarios of Table II are illustrated as follows:

1. A: ON, B: ON – Both nodes do not defect.

The inconsistency cost by a false positive outcome is generated during the time period $T - d$ in which Node *A* erroneously detects Node *B* as faulty, while it is not. The maximum time during which nodes can be in this state is $T - t$.

2. A: ON, B: OFF – Node *B* becomes faulty at time F_B , while Node *A* does not defect.

If fault detection occurs before F_B , the inconsistency cost by a false positive outcome is generated during the time period $F_B - d$, with a maximum duration of $F_B - t$. If fault detection occurs after F_B , the inconsistency cost by a false negative outcome is generated during the time period $d - F_B$, with a maximum duration of $T - F_B$.

3. A: OFF, B: ON – Node *A* becomes faulty at time F_A , while Node *B* does not defect.

If fault detection occurs before F_A , the inconsistency cost by a false positive outcome is generated during the time period $F_A - d$, with a maximum duration of $F_A - t$.

4. A: OFF, B: OFF – Node *A* becomes faulty at time F_A and Node *B* at $F_B > F_A$.

If fault detection occurs before F_A , the inconsistency cost by a false positive outcome is generated during the time period $F_A - d$, with a maximum duration of $F_A - t$. The inconsistency cost by a false negative outcome is generated during the time period $T - F_B$ that is also the maximum duration. This is because Node *A* is faulty to perform fault detection of Node *B*.

- 5.1 B: OFF, A: OFF, $d < F_B$ – Node *A* becomes faulty at time F_A and Node *B* at $F_B < F_A$.

If fault detection occurs before F_B , the inconsistency cost by a false positive outcome is generated during the time period $F_B - d$, with a maximum duration of $F_B - t$. The inconsistency cost by a false negative outcome is generated during the time period $T - F_A$ that is also the maximum duration. This is because Node *A* becomes faulty and is unable to perform fault correction of Node *B*.

- 5.2 B: OFF, A: OFF, $d > F_B$ – Node *A* becomes faulty at time F_A and Node *B* at $F_B < F_A$.

The inconsistency cost by a false negative outcome is generated during the time period $d - F_B$, with a maximum duration of $F_A - F_B$. This represents the lag of detecting the faulty Node *B*. There is also an additional inconsistency cost by a false negative outcome during the time period $T - F_A$ that is also the maximum duration. This is because Node *A* becomes faulty and is unable to perform fault correction of Node *B*.

6. A, B: OFF – Node *A* and *B* become faulty at time $F_A = F_B$.

The inconsistency cost by a false positive outcome is

generated during the time period $F_B - d$, with a maximum duration of $F_B - t$. The inconsistency cost by a false negative outcome is generated during the time period $T - F_A$ that is also the maximum duration. This is because Node A becomes faulty and is unable to perform fault correction of Node B.

As proven below, these fault scenarios are *sufficient* to model the overall system health status:

Theorem 1. *The fault times F_A , F_B of each possible pair of a Node A monitoring the health status of Node B are sufficient to calculate the health status of a decentralized system of n nodes that arbitrary defect in m batches, each of size $k < n$.*

Proof: The size of the health status space in a decentralized system of n nodes, where each node monitors the health status of all other nodes, is $n^2 - n$ as each node monitors the health status of all other $n - 1$ nodes. The fault times F_A , $F_B \neq 0$ and the six fault scenarios they determine (outlined in Table II) are sufficient to calculate this space, if the number of node pairs determined at each fault scenario sum up to $n^2 - n$. For each fault scenario, the number of node pairs are determined as follows:

- A: ON, B: ON, $F_A = F_B = 0$

Each of the $n - mk$ healthy nodes monitors the health status of all other $n - mk - 1$ healthy nodes, i.e. nodes do not monitor their own health status. This is the number of healthy node pair variations without repetition:

$$\binom{n - mk}{2} 2! = \frac{(n - mk)!}{2!(n - mk - 2)!} = (n - mk)(n - mk - 1) \quad (2)$$

- A: ON, B: OFF, $F_A = 0 < F_B \leq T$ and
A: OFF, B: ON, $F_B = 0 < F_A \leq T$

Each of the $n - mk$ healthy nodes monitors the health status of all other mk faulty nodes and vice versa for these two fault scenarios. The total number of these node pairs for both fault scenarios are calculated as:

$$2mk(n - mk) \quad (3)$$

- A: OFF, B: OFF, $0 < F_A < F_B \leq T$ and
B: OFF, A: OFF, $0 < F_B < F_A \leq T$

These fault scenarios involve k^2 pairs of faulty Nodes A and B for each possible pair of different batches of faulty nodes. The total number of all possible batch pairs in which a pair of two faulty Nodes A and B reside is the number of batch pair variations without repetition. Therefore, the total number of these faulty node pairs for both fault scenarios is calculated as follows:

$$k^2 \binom{m}{2} 2! = k^2 \frac{m!}{2!(m - 2)!} 2! = mk^2(m - 1) \quad (4)$$

- A, B: OFF, $0 < F_A = F_B \leq T$

This fault scenario involves $k^2 - k$ pairs of faulty Nodes A and B defecting at the same time at each of the m batches of faulty nodes. This is the number of faulty node pair variations without repetition:

$$m \binom{k}{2} 2! = m \frac{k!}{2!(k - 2)!} 2! = mk(k - 1) \quad (5)$$

The number of node pairs from all fault scenarios sum up as follows:

$$\begin{aligned} & \text{Fault scenario 1, Eq. 2} \\ & \overbrace{(n - mk)(n - mk - 1)} + \\ & \text{Fault scenario 2 \& 3, Eq. 3} \\ & \overbrace{2mk(n - mk)} + \\ & \text{Fault scenario 4 \& 5, Eq. 4} \\ & \overbrace{mk^2(m - 1)} + \\ & \text{Fault scenario 6, Eq. 5} \\ & \overbrace{mk(k - 1)} = n^2 - n, \end{aligned} \quad (6)$$

which is the overall health status space of a decentralized system. ■

V. MODEL APPLICABILITY

This section illustrates the applicability of the fault scenarios for self-healing in decentralized in-network data aggregation. Fault detection, fault correction and fault tolerance are illustrated.

A. Fault detection via gossip-based communication

Gossip-based communication [6] is selected for fault detection given the following: (i) It is a communication protocol for large-scale and highly decentralized systems that falls within the scope of this paper. (ii) It is general-purpose and fundamental as it can be widely used for fast information dissemination, new information discovery, preserving network robustness by keeping the network connected, and other core operations required in decentralized systems [35], [6], [36]. (iii) It finds real-world applicability in several systems such as peer-to-peer networks [35], cloud computing [37], [38], Big Data systems [39], distributed ledgers [40], [41], middleware systems [42] etc. (iv) It is probabilistic in nature and as a result, fault detection based on gossiping communication comes with uncertainties within which dilemmas of fault correction vs. fault tolerance can be systematically studied.

Gossip-based communication realizes health status monitoring as illustrated in Figure 2c. Nodes execute a gossiping protocol such as the peer sampling service [35] that equips each node with a limited-size list of node descriptors, each containing the IP address, port number, timestamp and application information. This list is the partial view that nodes have of the system. It is periodically updated with new random node descriptors during peer-to-peer gossip exchanges with other random nodes selected from the partial view (the same list).

The health status of the parent node is locally determined by the time period passed since the last time the descriptor² of the parent node was present in the partial view of the node in which the self-healing agent resides. If the threshold t is surpassed, the parent node is considered faulty and fault

²Descriptors of the parent node with a timestamp value later than the migration time are the ones counted. Earlier descriptors of the parent node may be present and circulated in the network. They are eventually replaced with the latest one during the gossip exchanges [35].

correction is initiated. Otherwise, fault tolerance is performed by awaiting further for the parent node descriptor to arrive by relying on underlying gossip-based communication.

An effective choice of the threshold t depends on the system size and the internal configuration of the gossip-based communication protocol: (i) The size of the partial view. (ii) The execution period. (iii) The node and view selection policy that determine the level of randomization in the communication and exchange of node descriptors respectively. The threshold choice also depends on the external environment, e.g. latency, convergence speed of the communication model, bandwidth and load of the network [4]. Even if all these uncertainties that determine whether the parent node is truly faulty or not are controlled, the dilemma of the self-healing agent remains: is it fault tolerance or fault correction that results in lower inconsistency cost? Given that the inconsistency cost is context/application dependent, this paper introduces the computational case study of decentralized data aggregation within which inconsistency cost is assessed.

B. Computational case study: decentralized data aggregation

The computational problem of dynamic in-network data aggregation is studied [43]. More specifically, this paper studies how self-healing can improve the accuracy in decentralized computations of aggregation functions when nodes fail. The computational case study is the following: Each node in the network is a *data supplier* and *data consumer* (extreme performance benchmark). Data suppliers generate and share data (streams) with data consumers. Data consumers collect data (streams) from data suppliers and compute/update aggregation functions such as average, summation, count, maximum/minimum and other. When a data supplier disconnects from the network, data consumers need to update their aggregation function by performing a reverse computation, i.e. rollback, that removes the counted input data of the departing data supplier.

Preserving accurate estimations of aggregation functions in this computational case study is challenging given that (i) data suppliers and consumers need to discover each other in a decentralized unstructured network, (ii) data suppliers can change the input data of the aggregation functions, (iii) data consumers may compute any aggregation function given the input data of data suppliers and (iv) reverse computations are required when data suppliers leave the network. In contrast to earlier decentralized aggregation methodologies such as gossiping [44], [45], tree-based [46] or synopsis diffusion [47], DIAS³, the *Dynamic Intelligent Aggregation Service* [5], [48], [9] is a decentralized gossip-based aggregation system designed to meet all these requirements⁴ and therefore it is used

³Available at <http://dias-net.org> (last accessed: March 2021).

⁴This is made possible by using an efficient and scalable distributed memory system based on probabilistic data structures, the Bloom filters [49]. Based on Bloom filters, a data supplier can reason whether it has earlier communicated with a data consumer to share data and vice versa a data consumer can reason whether it has earlier communicated with a data supplier to aggregate data. Data suppliers and consumers can also reason about what data have been shared and aggregated, i.e. the most recent ones or outdated ones, so that aggregation inaccuracies are minimized, while unnecessary communication is limited. Further information about DIAS is out of the scope of this paper and can be found in earlier work [5], [48], [9].

to assess how well the inconsistency cost of the fault scenarios predicts the aggregation inaccuracies.

The inconsistency cost is measured by the average relative approximation error in the estimation of the aggregation functions among all data consumers in the network. In other words, the inconsistency cost measures how far the estimation of the aggregates is from the actual true values of the aggregates. Apparently, when nodes hosting data suppliers become faulty, reversed (rollback) computations are required by data consumers that have earlier aggregated data of these now faulty data suppliers. Without such computations, the estimations of the aggregates diverge from the actual ones generating inconsistency cost (false negative state in Figure 3c). However, inconsistency cost may also result by reversed (rollback) computations because of an erroneous gossip-based fault detection, e.g. a very low threshold value t that determines the node hosting the data supplier as faulty when actually it is not (false positive state in Figure 3d). Therefore, the self-healing dilemma is highly applicable in this computational case study and the rest of this section introduces the functionality of the fault correction and fault tolerance in DIAS.

C. Fault-corrective aggregation

This paper extends an earlier self-corrective aggregation mechanism [9] for nodes joining and leaving the network into a fault-correction mechanism when nodes arbitrary fail. The rationale of self-correction when a node with a data supplier leaves the network is the following: A self-healing agent creates a replica of the data supplier with which it migrates to a remote random neighboring host node (see Figure 2b) selected via the peer sampling service⁵ based on which DIAS operates. The migrated data supplier initiates corrective rollback operations with the data consumers in the network to update the aggregation functions. This process either completes or is interrupted when the self-healing agent detects⁶ via the peer sampling service that the parent node has joined again the network. In the latter case, the migrated self-healing agent together with the migrated data supplier return back to the parent node to continue their operations as before. Migrations can be consecutive if the migrated host node leaves the network as well. More information about the protocol specification and evaluation results can be found in earlier work [9].

The limitation of this mechanism is that self-corrective operations are initiated reactively by the parent node before leaving the network. This is not realistic in a scenario of arbitrary node failures that can terminate all local processes before self-corrective operations are initiated. This paper extends this model by proactively migrating each self-healing agent to a remote host, where it runs as a daemon monitoring the health

⁵Random selection of the migration host is performed for load-balancing. Without loss of generality, DIAS reuses the peer sampling service for the purpose of the migrations to limit the need for another such mechanism that comes with additional performance overhead, i.e. communication, processing and storage cost. Other methodologies for migration include random walks in the network or allocating dedicated nodes for redundancy [7].

⁶The returned parent node is detected when its descriptor appears in the partial view of the migrated node with a timestamp value later than the leave.

status of the parent node as shown in Figure 2c. Monitoring is performed by reusing the peer sampling service⁷ according to the fault-detection mechanism introduced in Section V-A so that no other performance overhead is introduced.

D. Fault-tolerant aggregation

The alternative to fault correction is fault tolerance that determines no corrective operations until the threshold t is reached. Fault tolerance eliminates inconsistency costs originated by false positive states (see Figure 3d). Moreover, fault tolerance is cost-effective when the faulty node can recover promptly, given the time required for corrective operations to complete. More specifically, fault tolerance eliminates inconsistency cost if it holds:

$$F_B + t + c > R_B + p \quad (7)$$

where F_B is the time when Node B becomes faulty, t is the fault-detection threshold and c is the duration for the corrective operations to restore a required aggregation accuracy level. On the other side of the inequality, R_B is the time when Node B recovers⁸ and p is the time required by Node A to detect the recovery, i.e. propagation time of the Node B descriptor by the peer sampling service. This inequality can be used to determine threshold values t for each node given empirical models for $R_B - F_B$, which are though not the focus of this paper. Instead, different threshold values and their influence on inconsistency cost are studied.

VI. EXPERIMENTAL METHODOLOGY

This study has the following three objectives: (i) Profiling of the inconsistency cost generated by the modeled fault scenarios under varying fault scales, fault profiles and fault-detection thresholds. (ii) Validation of whether the inconsistency cost of the modeled fault scenarios is a good general predictor of the accuracy observed in the application scenario of decentralized aggregation of real-world power consumption data. (iii) Comparison of different model calibrators for the prediction of aggregation accuracy. Table III outlines the experimental parameterization⁹. All studied systems are implemented with an improved version [51] of the Protopeer prototyping toolkit [52] for distributed systems.

The following scales of faulty nodes are studied: $\{10\%, 20\%, \dots, 80\%\}$ of the total number. Three fault profiles are introduced that come with 1, 2 and 4 batches of faulty nodes respectively: (i) *1st profile*: All faulty nodes defect in one batch on half of system runtime that is on the 1600th epoch. (ii) *2nd profile*: Faulty nodes defect in two batches, with half of

Table III
AN OVERVIEW OF THE EXPERIMENTAL PARAMETERIZATION.

System Parameter	Value	System Parameter	Value
ECBT data ¹⁰ [5]	Day 199 (January 4 th)	DIAS execution period	1s
Num. of nodes	3000	Num. of aggregation sessions [5]	4
Num. of epochs	3200	Partial view size [35]	50
Epoch duration	250ms	Swap parameter [35]	24
Fault scales	10%, 20%, ..., 80%	Healer parameter [35]	1
Fault profiles (Table IV)	1 st , 2 nd , 3 rd	Fault detection threshold	[100, 800] with step 25
Num of epochs for bootstrapping	400		

the faulty nodes defecting on the 1332nd epoch and the other half on the 2264th epoch. (iii) *3rd profile*: Faulty nodes defect in four batches of equal size on the 1060th, 1620th, 2180th and 2740th epoch. Such parameters can accurately model failures observed in real-world systems, i.e. failure bursts correlated in time/space [53], [54], [27], while the evaluated parameter space with extreme fault scales stretches the experimental evaluations. Table IV summarizes the applicability of the three fault profiles to the modeled fault scenarios. Note in particular that nodes can defect in any of these batches except the case of the 4th and 5th fault scenarios that determine faulty nodes in different batches, while either Node A or B defects first. In the 2nd profile, Node A cannot defect at the 2nd batch if it defects first and respectively, Node B cannot defect at the 1st batch if it defects second. Similarly in the 3rd profile, the node that defects first cannot defect at the 4th batch and the node that defects second cannot defect at the 1st batch.

Table IV
APPLICABILITY OF THREE FAULT PROFILES TO EACH FALSE POSITIVE (FP) AND FALSE NEGATIVE (FN) STATE OF THE FAULT SCENARIOS. THE FREQUENCIES OF FAULT-SCENARIOS SUM UP TO $n^2 - n$ (THEOREM 1), WHERE n IS NUMBER OF NODES IN THE NETWORK, k THE NUMBER OF FAULTY NODES AT EACH BATCH OF DEFECTED NODES OUT OF A TOTAL OF m BATCHES.

Health	Frequency	State	Node	Defect Batch IDs		
				1 st Profile $m = 1$	2 nd Profile $m = 2$	3 rd Profile $m = 4$
1. A: ON B: ON	$(n - mk)(n - mk - 1)$	FP	None	✗	✗	✗
2. A: ON B: OFF	$mk(n - mk)$	FP, FN	B	1	1, 2	1, 2, 3, 4
3. A: OFF B: ON	$mk(n - mk)$	FP	A	1	1, 2	1, 2, 3, 4
4. A: OFF B: OFF	$\frac{1}{2}mk^2(m - 1)$	FP, FN	A	✗	1	1, 2, 3
			B	✗	2	2, 3, 4
5. B: OFF A: OFF	$\frac{1}{2}mk^2(m - 1)$	FP, FN	B	✗	1	1, 2, 3
			A	✗	2	2, 3, 4
6. A, B: OFF	$mk(k - 1)$	FP, FN	A, B	1	1, 2	1, 2, 3, 4

To address the first objective, the fault profiles are applied to a decentralized network of 3000 nodes each running fault detection with the peer sampling service [35] as illustrated in Section V-A and with the respective parameters of Table III. The threshold values of $\{100, 125, 150, \dots, 775, 800\}$ epochs are evaluated. By knowing which nodes defect at which time point during system runtime, all false positive and false negative states in the six possible fault scenarios of Table II can be measured and analyzed. This analysis is performed exhaustively to profile the inconsistency cost across three dimensions: $8 \text{ fault scales} \times 3 \text{ fault profiles} \times 29 \text{ thresholds} = 696 \text{ experimental settings}$.

The finest-grain measurements of inconsistency cost are performed with size $3000^2 - 3000 = 8997000$ according to

⁷Other mechanisms such as heartbeat messages [50], [31] can be used.

⁸The scenario in which $F_B + t + c < R_B + p$ is more complex to determine whether fault tolerance or fault correction should be performed as it depends on the relation of t and p , the data consumers with which corrective operations have been performed and their aggregated data.

⁹The system parameterization of the peer sampling service and DIAS is chosen based on earlier experimental findings [35], [5], [9], [48] and on the rationale of a cost-effective operation of the decentralized data aggregation under no faulty nodes. In this way, the effect of the faults on the data aggregation and how this effect can be controlled via a self-healing tuning (choice of threshold) can be isolated and studied systematically.

Theorem 1: every node monitors the health status of every other node in the network. Given the fault scale (k) and fault profile (m), the health status of all node pairs is calculated according to the equations of Table IV (Theorem 1) and these calculations result in the relative frequencies of Figure 4.

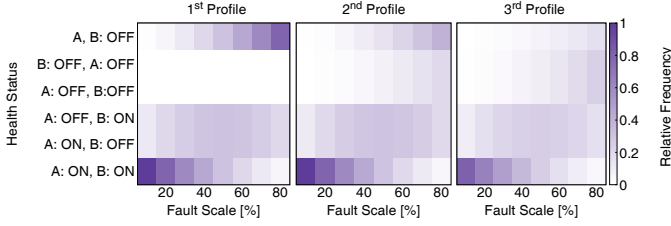


Figure 4. Relative frequencies of the health status among all node pairs for different fault scales and fault profiles.

To address the second objective, the modeled fault scenarios are evaluated by measuring how well they predict the inconsistency cost of a self-healing computational/application scenario with faulty nodes. This scenario is the decentralized aggregation of DIAS in which self-healing is performed in terms of fault correction (executing self-corrective operations, see Section V-C) and fault tolerance (postponing self-corrective operations, see Section V-D). The prediction of the inconsistency cost is the prediction of the aggregation accuracy measured by the *average relative approximation error* between the *estimated* aggregate values and the *actual* aggregates. In other words, this paper assesses for all 696 experimental settings how good predictor the total inconsistency cost (Equation 1) is of the average relative approximation error of DIAS measured over all nodes and throughout system runtime.

The experiments focus on the summation (total power load) of real-world power consumption data¹⁰ from ECBT, the *Electricity Customer Behavior Trial* during 2009-2010 in Ireland. They are collected from smart meters with a frequency of 30 minutes. The power records of the 199th day (4.1.2009) are used for the experiments that are 2 records/hour $\times$ 24 hours = 48 records uniformly distributed over the system runtime of 2800 epochs, plus 400 epochs for system bootstrapping. Out of the total of 6435 residential and small-medium enterprise consumers in the dataset, 3000 residential consumers are mapped to the 3000 nodes of the decentralized network. Each operates as both data supplier and consumer to evaluate the most demanding computational scenario in which every node shares and aggregates power consumption data.

Predicting the DIAS accuracy is highly challenging given that the modeled fault scenarios are totally agnostic of the applied (i) computational problem, i.e. aggregation, (ii) algorithm, i.e. DIAS and (iii) data, i.e. power consumption. As such, it is assumed that all fault scenarios l can generate during runtime a total (maximum) inconsistency cost of $\epsilon_s^- = \epsilon_s^+ = 1$. To improve prediction, three model calibration methods are applied that rely on the profiling of inconsistency cost calculated for the first objective of this study. All three calibration methods use application-independent features. One of these

methods is totally agnostic of any information about the aggregation problem or the DIAS algorithm, while the other two use DIAS performance target data for fitting a model. Therefore, significant comparisons can be made between a non-calibrated prediction vs. calibrated predictions as well as the application-agnostic calibrations vs. the ones that fit a model to the data.

1) *False negative calibration*: The fault scenarios of Table II with a false negative state given by $\frac{T-F_A}{T-F_A} = 1$ assume that the fault of Node A generates inconsistency cost throughout the time period $T - F_A$ as the fault of Node B cannot be anymore detected (and corrected) during this period. However, recovery may occur earlier, which means in practice that the inconsistency cost may be eliminated within a short period of time, for instance, self-corrective operations in DIAS converge in a finite time period [9]. Therefore, this calibration method introduces the calibration factor $\lambda \in [0, 1]$ as an additional coefficient for these fault scenarios with false negative state. For $\lambda = 1$, no calibration is performed. For each fault scale, the λ value with the lowest root mean square error between the predicted and the DIAS inconsistency cost is selected for the comparison with the other calibration methods as shown in Figure 5a.

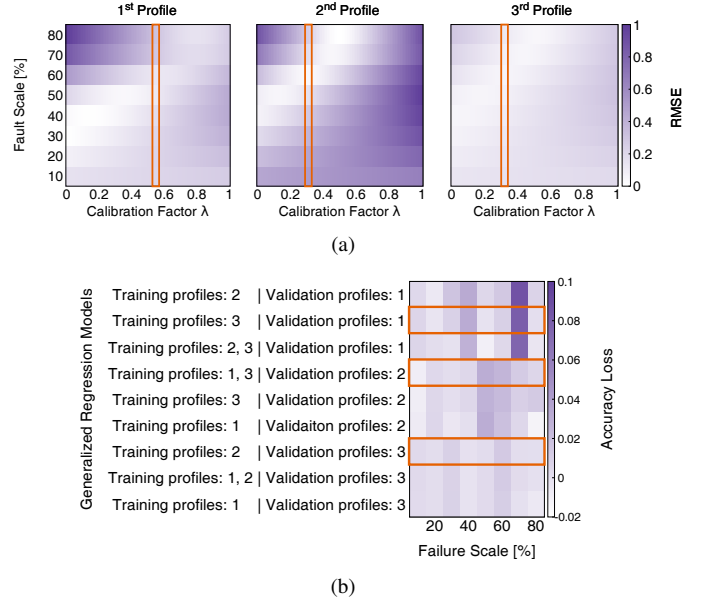


Figure 5. Calibration configurations and their prediction performance for two calibration methods: (a) False negative calibration. (b) Generalized regression. The best calibration configurations are marked for further comparison of the different methods.

The other two model calibration methods are designed as follows: For each experimental setting, a feature vector of size $12 \times 5 + 2 = 62$ is constructed. This vector contains 5 quantiles (10th, 30th, 50th, 70th, 90th) of inconsistency cost for each of the 12 calculations of the fault scenarios (Table II). These values are extracted from the fault profiles applied to the peer sampling service [35]. The feature vector also contains the respective relative (to the maximum of 800) threshold and the fault scale for each experimental setting. All values of the feature vector are in the range $[0, 1]$. Application-level data, i.e. the DIAS inconsistency cost, are used as target values

¹⁰ Available at <http://www.ucd.ie/issda/data/commissionforenergyregulationcer/> (last access: March 2021)

for training, while features are agnostic of DIAS. Regression relies on the ordinary least squares model and its Python implementation of the statsmodels¹¹. The prediction based on linear regression is validated with two schemes:

2) *Regression*: This second scheme uses all 696 experimental settings to train the linear regression model without regularization. It represents the best possible fit (intentional overfit) to the inconsistency costs observed in DIAS.

3) *Generalized regression*: In this third scheme, training is limited to certain fault profiles and validation is performed on profiles on which training is not performed, assuming that the inconsistency cost for different fault profiles is generated from the same distribution. Figure 5b illustrates the prediction performance of all possible combinations of training and validation fault profiles for a generalized regression, measured with the accuracy loss:

$$RMSE(C_{GR}, C_D) - RMSE(C_{GR}, C_R), \quad (8)$$

where $RMSE$ is the root mean square error, C_R, C_{GR} are the inconsistency cost of regression and generalized regression respectively and C_D is the predicted inconsistency cost of DIAS, i.e. the average relative approximation error of the summation. For each of the three validation fault profiles, the best fits observed in Figure 5b are selected to compare generalized regression to the other predictors. Generalized regression is performed with regularization¹².

VII. EXPERIMENTAL EVALUATION

This section illustrates the profiling of the inconsistency cost and how it can be used to improve the effectiveness of self-healing in decentralized data aggregation. It also shows a comparison of the calibration methods.

A. Profiling of inconsistency cost

For the first evaluation objective, the inconsistency cost is profiled as follows: The density of the inconsistency cost and the relative frequency of each fault scenario are measured under varying fault scales, fault profiles and fault-detection thresholds. Due to space limitations, Figures 6-8 focus on the fault scales of 20%, 50% and 80% that depict the overall trend.

The following observations can be made in Figures 6-8: (i) The inconsistency cost by false positive states has on average higher magnitude than the one of false negative states across the different fault profiles and scales. (ii) With an increasing fault scale, the inconsistency cost slightly increases, especially for the fault-scenario $A: ON, B: ON$, false positive. However, the relative frequency of the inconsistency cost for this fault scenario decreases, in exchange for an increase in fault scenarios $A: OFF, B: OFF$, false positive, $B: OFF, A: OFF$, false positive, $B: OFF, A: OFF$, false negative and $A, B: OFF$, false positive and negative. (iii) For each fault profile when nodes do not fail, the magnitude of the inconsistency cost

by false positives is respectively 22.37%, 17.2% and 16.07% higher on average than the one with defecting nodes. For fault scales of 20%, 50% and 80%, this difference is 13.0%, 20.99% and 22.63% higher on average when nodes do not fail. (iv) The inconsistency cost by false positives is minimized for middle threshold values, i.e. 450 epochs for 20% fault scale, 350 epochs for 50% and 80% fault scale. These thresholds though depend on the system size and parameters with which the peer sampling service is chosen to operate, i.e. partial view size, execution period, swap/healer parameters [35], [5]. In other words, different configurations of the underlying system yield to a different profiling of the inconsistency cost. (v) In the second profile, the density of the inconsistency cost for the fault scenario of $A: ON, B: OFF$, false negative, has two peaks that originate from the two different times in which the nodes defect (respectively three peaks at the 3rd profile). Larger thresholds shift the peaks to larger inconsistency costs ($d - F_B$ is maximized) and increase the distance between the peaks as also confirmed for the fault scenario $B: OFF, A: OFF$, false negative. (vi) The relative frequency of fault scenarios with a false positive state decreases for higher thresholds, while it increases or remains constant for a false negative state. All these observations confirm that the profiling of the inconsistency cost generated by the fault scenarios can provide a highly insightful analysis of the trade-offs involved in tuning fault-detection mechanisms in decentralized systems with uncertainties.

B. Self-healing decentralized data aggregation

More cost-effective self-healing mechanisms can be designed, tailored to minimize the predicted inconsistency cost of specific fault scenarios. Note for instance Figure 9 that illustrates the applicability of self-healing in DIAS in the three fault profiles and the fault scales of 20%, 50% and 80%. The actual aggregate of summation is compared to the faulty estimate (no corrective operations) and two corrective estimates (without any calibration): (i) A reference with a fixed threshold at 100 epochs. (ii) The one with the threshold that minimizes the inconsistency cost. Therefore, the profiling of the inconsistency cost provides the required tuning to fault detection to minimize the relative approximation error of the aggregation. The root mean square error between the actual sum and the faulty estimate (no self-corrective operations) is on average 28.17% higher than the DIAS estimate with the threshold resulting in minimal inconsistency cost. Across fault profiles, the corresponding errors are 4.29%, 34.83% and 34.72% higher for the fault scales of 20%, 50% and 80%, respectively. The DIAS corrective estimate with reference threshold $t = 100$ performs worse than the faulty estimate across all fault profiles and scales¹³, demonstrating the implication of an erroneous fault correction and, apparently, how dramatic can a misconfiguration of fault detection be for a decentralized application.

¹¹Available at <https://www.statsmodels.org/stable/index.html> (last access: March 2021).

¹²Elastic net is used with strength parameter of $\alpha = 0.07$ and $L_1 = 0.05$ representing the preference of LASSO regularization over the RIDGE one.

¹³On average, the root mean square error between the actual sum and the faulty estimate is 113.07% lower than the DIAS estimate with a reference threshold of $t = 100$

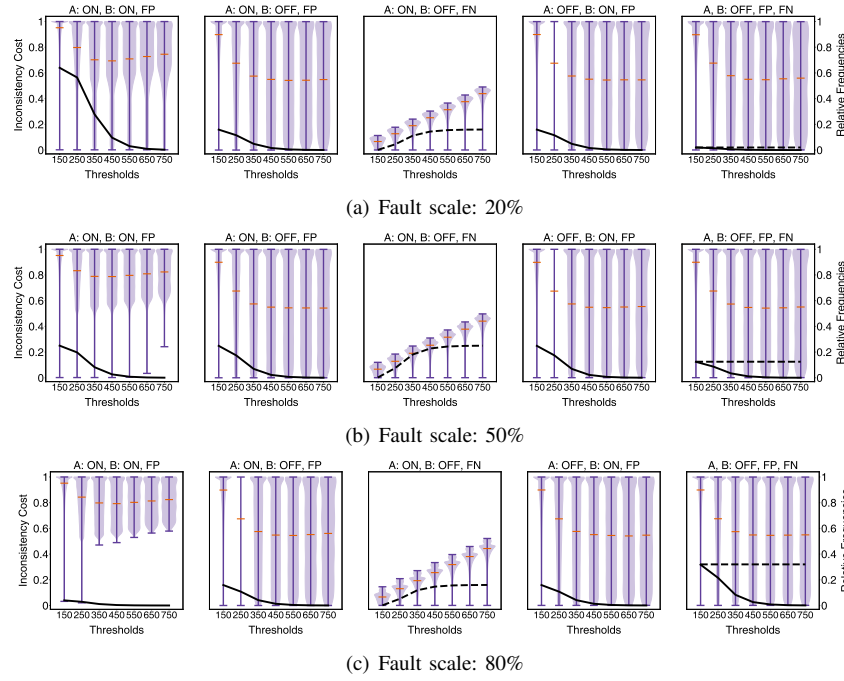


Figure 6. The inconsistency cost of the fault scenarios (violins with density values on the left Y-axis) and their relative frequency (lines with values on the right Y-axis) under a fault scale of 20%, 50% and 80% in the 1st fault profile. The solid lines depict the relative frequency for false positive (FP) states, while the dashed lines the one for false negative (FN) states.

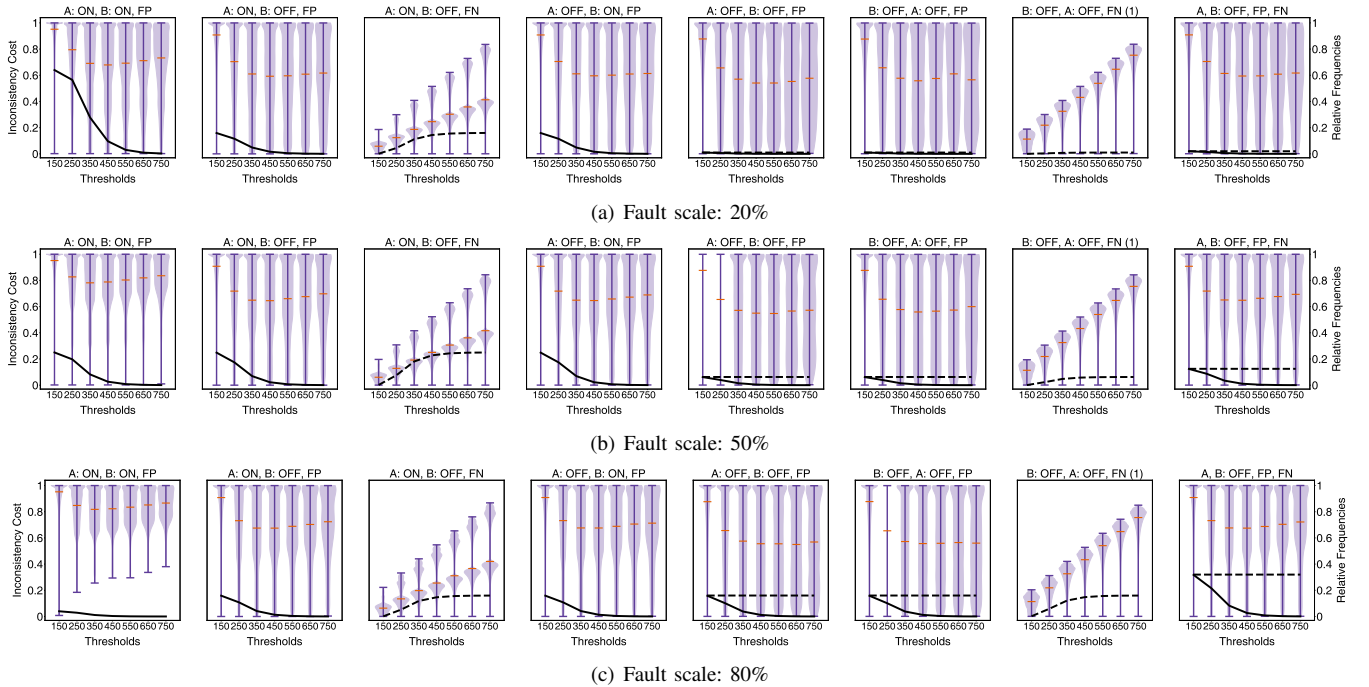


Figure 7. The inconsistency cost of the fault scenarios (violins with density values on the left Y-axis) and their relative frequency (lines with values on the right Y-axis) under a fault scale of 20%, 50% and 80% in the 2nd fault profile. The solid lines depict the relative frequency for false positive (FP) states, while the dashed lines the one for false negative (FN) states.

C. Evaluation of model calibration methods

Figure 10 addresses the second and third objective of the experimental evaluation that is the predictive performance of the inconsistency cost by the modeled fault scenarios. The average relative approximation error of the DIAS sum estimations is compared to the calibrated predictions made by

the modeled fault scenarios under different thresholds in the three fault profiles.

The followings observations are made: (i) The application agnostic calibration methods, i.e. non-calibrated prediction and false negative calibration, correlate well with DIAS, with correlation coefficient of 0.83 and 0.82, respectively across

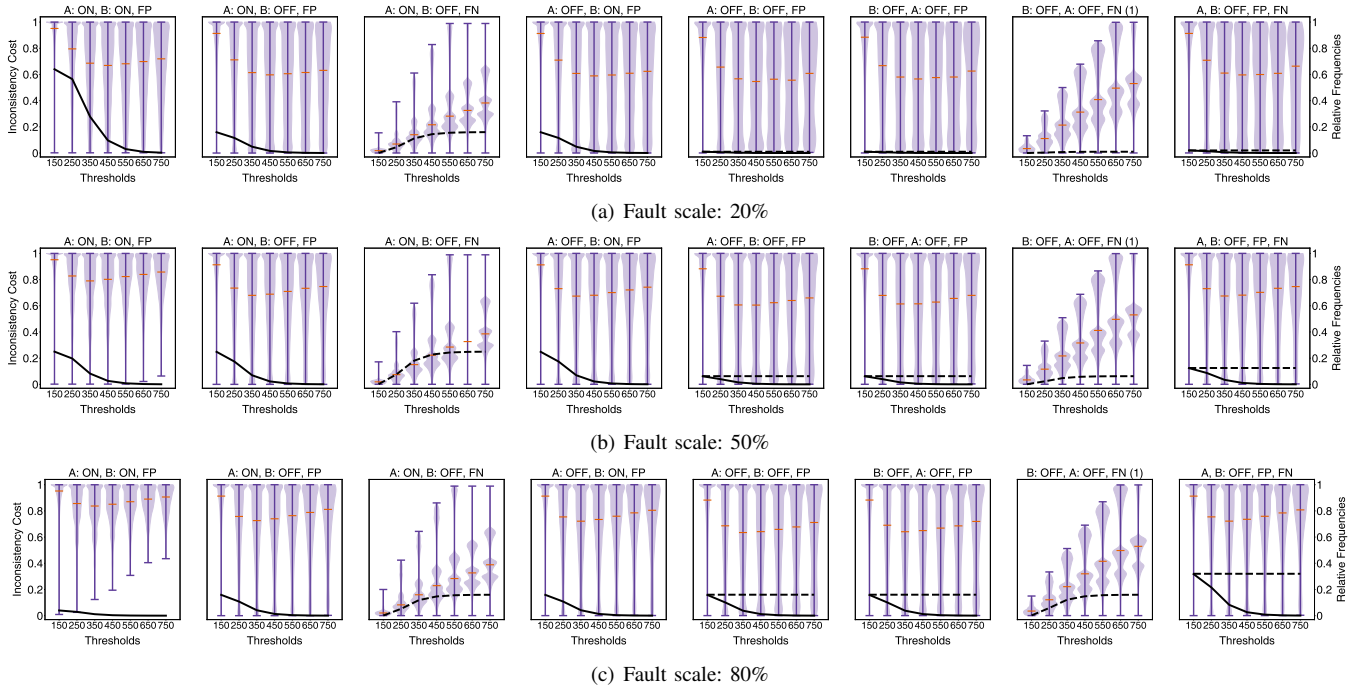


Figure 8. The inconsistency cost of the fault scenarios (violins with density values on the left Y-axis) and their relative frequency (lines with values on the right Y-axis) under a fault scale of 20%, 50% and 80% in the 3rd fault profile. The solid lines depict the relative frequency for false positive (FP) states, while the dashed lines the one for false negative (FN) states.

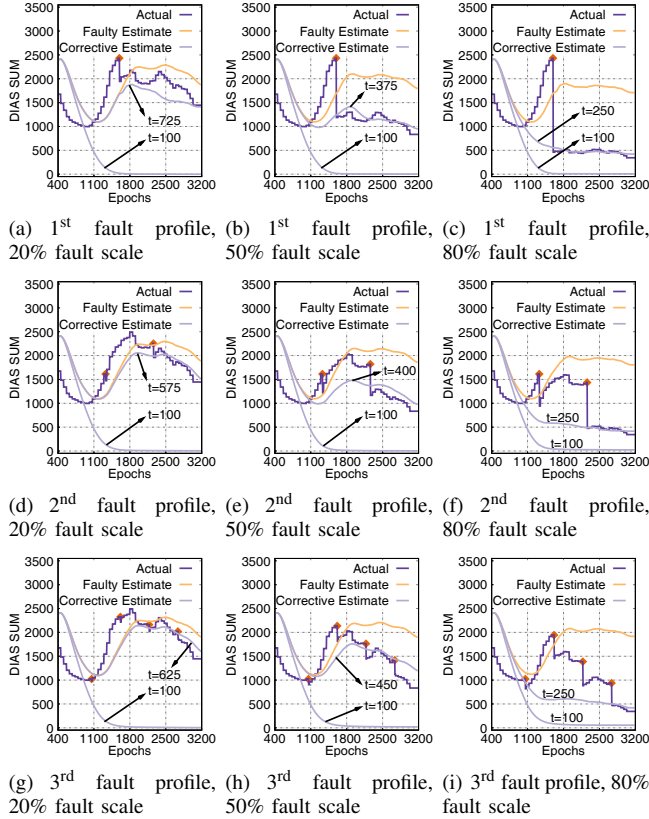


Figure 9. DIAS self-healing under 20%, 50% and 80% fault scales in the three fault profiles. In the corrective estimates, the threshold of $t = 100$ is shown as reference vs. the threshold that minimizes the inconsistency cost.

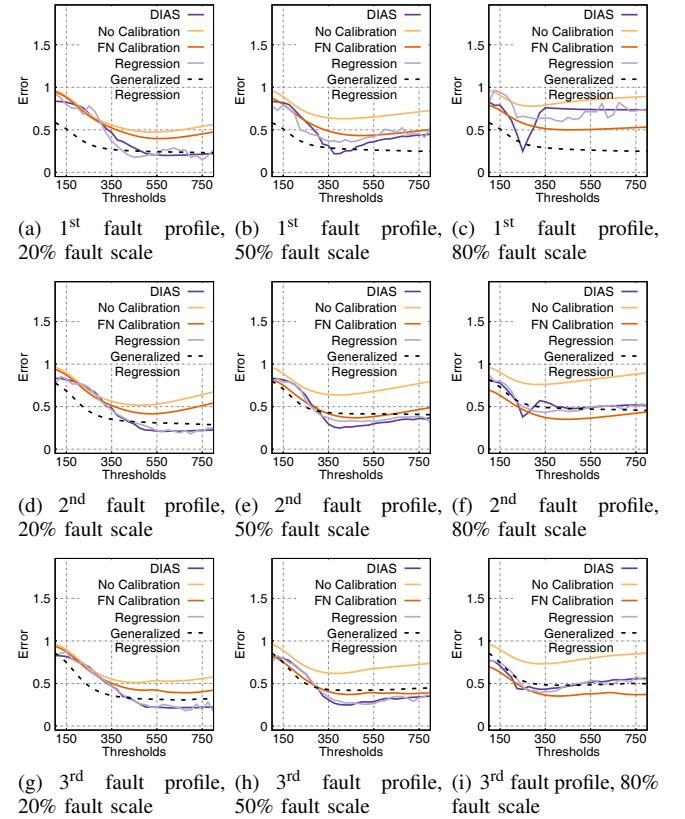


Figure 10. Prediction performance of the calibration methods. 20%, 50% and 80% fault-scale respectively for the three fault profile.

the fault profiles and scales. However, note the improvement

of the latter to match the magnitude of the DIAS errors. These correlation values cannot be improved by more than

7.8% on average with the linear regression calibration. This is an expected result as linear regression represents the best fit, i.e. linear transformation, of the application-independent features to the DIAS inconsistency cost data. On the contrary, the correlation coefficient between DIAS and generalized regression decreases to 0.79 on average. This confirms the feasibility of selecting effective thresholds for fault detection without information about the application that makes use of self-healing. (ii) Without calibration, the fault scenarios overestimate the magnitude of the DIAS errors, especially for large fault scales and thresholds. This is because the modeling of the total inconsistency cost assumes a uniform generation of inconsistency cost during runtime for each fault scenario. However, the magnitude of estimation errors in the summation aggregation function as well as how errors cancel out each other are highly dependent on the data. (iii) The hypothesis that the inconsistency cost in false negative states with the value of 1 (Table II) is a worst case scenario in practice is actually confirmed: Across all thresholds, the root mean square error between DIAS and the false negative calibration is on average 0.51, 2.27, and 0.72 times lower than no calibration for a fault scale of 20%, 50% and 80% respectively. These numbers are 0.44, 1.25, and 1.47 times lower across the fault scales for the 1st, 2nd and 3rd fault profile respectively.

VIII. CONCLUSION AND FUTURE WORK

This paper demonstrates how the performance of self-healing systems operating in decentralized asynchronous environments is significantly influenced by uncertainties in fault detection inherited from such systems. However, it also concludes that this influence can be accurately predicted and mitigated by modeling a number of fault scenarios that identify the origin of inconsistencies. This paper also shows how to minimize inconsistencies by tuning appropriately fault detection at the design phase. The significance of these findings stems from application-independence: A high prediction performance in the aggregation accuracy of real-world power demand data is confirmed under 696 experimental settings of different fault scales, fault profiles and fault detection thresholds.

Future work focuses on addressing some of the limitations of this study as well as unfolding some new promising research pathways: The prediction performance of the inconsistency cost of other distributed application scenarios is required for further validation. Other costs with more complex performance trade-offs can be modeled, e.g. inconsistency vs. communication cost. Comparing the inconsistency profiles of different decentralized systems with different size, fault profiles/models, connectivity and fault-detection mechanisms can provide further new insights on how to design, deploy and operate self-healing systems.

ACKNOWLEDGMENT

This study is supported by the Swiss National Science Foundation (SNSF) as part of the National Research Programme NRP77 Digital Transformation, project no. 187249.

REFERENCES

- [1] S. Mei, X. Zhang, and M. Cao, *Power grid complexity*. Springer Science & Business Media, 2011.
- [2] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 33–39, 2018.
- [3] R. Van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Middleware '98*. Springer, 1998, pp. 55–70.
- [4] A. Lavinia, C. Dobre, F. Pop, and V. Cristea, "A failure detection system for large scale distributed systems," *International Journal of Distributed Systems and Technologies (IJDSST)*, vol. 2, no. 3, pp. 64–87, 2011.
- [5] E. Pournaras, J. Nikolic, A. Omerzel, and D. Helbing, "Engineering democratization in internet of things data analytics," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2017, pp. 994–1003.
- [6] D. Shah *et al.*, "Gossip algorithms," *Foundations and Trends® in Networking*, vol. 3, no. 1, pp. 1–125, 2009.
- [7] M. Oyediran, T. Fagbola, S. Olabiyisi, E. Omidiora, and A. Fawole, "A survey on migration process of mobile agent," in *Proceedings of the world congress on engineering and computer science*, vol. 1, 2016.
- [8] D. Croce, F. Giuliano, I. Tinnirello, A. Galatioto, M. Bonomolo, M. Beccali, and G. Zizzo, "Overgrid: A fully distributed demand response architecture based on overlay networks," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 471–481, 2016.
- [9] E. Pournaras and J. Nikolić, "Self-corrective dynamic networks via decentralized reverse computations," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2017, pp. 11–20.
- [10] R. Stanković, M. Štula, and J. Maras, "Evaluating fault tolerance approaches in multi-agent systems," *Autonomous agents and multi-agent systems*, vol. 31, no. 1, pp. 151–177, 2017.
- [11] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, 2018.
- [12] B. E. Isong and E. Bekele, "A systematic review of fault tolerance in mobile agents," *American Journal of Software Engineering and Applications*, vol. 2, no. 5, pp. 111–124, 2013.
- [13] A. Fedoruk and R. Deters, "Improving fault-tolerance by replicating agents," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, 2002, pp. 737–744.
- [14] O. Marin, P. Sens, J.-P. Briot, and Z. Guessoum, "Towards adaptive fault-tolerance for distributed multi-agent systems," in *Proceedings of ERSADS*, 2001, pp. 195–201.
- [15] Y. Arfat and F. E. Eassa, "A survey on fault tolerant multi agent system," *IJ Inf. Technol. Comput. Sci*, vol. 9, pp. 39–48, 2016.
- [16] A. Luna-Almeida, S. Aknine, J.-P. Briot, and J. Malenfant, "Plan-based replication for fault-tolerant multi-agent systems," in *Proceedings of the 11th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS'06)*, 2006, pp. 413–418.
- [17] G. Jin, B. Ahn, and K. D. Lee, "A fault-tolerant protocol for mobile agent," in *International Conference on Computational Science and Its Applications*. Springer, 2004, pp. 993–1001.
- [18] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, and M. Völz, "Rollback-recovery without checkpoints in distributed event processing systems," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013, pp. 27–38.
- [19] K. Park, "A fault-tolerant mobile agent model in replicated secure services," in *International Conference on Computational Science and Its Applications*. Springer, 2004, pp. 500–509.
- [20] S. Kumar and P. R. Cohen, "Towards a fault-tolerant multi-agent system architecture," in *Proceedings of the fourth international conference on Autonomous agents*, 2000, pp. 459–466.
- [21] D. Terry, "Toward a new approach to IoT fault tolerance," *Computer*, vol. 49, no. 8, pp. 80–83, 2016.
- [22] D. Ratasich, M. Platzer, R. Grosu, and E. Bartocci, "Adaptive fault detection exploiting redundancy with uncertainties in space and time," in *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2019, pp. 23–32.
- [23] R. Sterritt, "Autonomic computing," *Innovations in systems and software engineering*, vol. 1, no. 1, pp. 79–88, 2005.
- [24] M. Panda and P. M. Khilar, "Distributed byzantine fault detection technique in wireless sensor networks based on hypothesis testing," *Computers & Electrical Engineering*, vol. 48, pp. 270–285, 2015.
- [25] A. Rahnema and P. J. Antsaklis, "Resilient learning-based control for synchronization of passive multi-agent systems under attack," *arXiv preprint arXiv:1709.10142*, 2017.

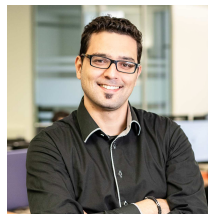
- [26] L. Su and N. H. Vaidya, "Non-bayesian learning in the presence of byzantine agents," in *International symposium on distributed computing*. Springer, 2016, pp. 414–427.
- [27] S. Ghahremani and H. Giese, "Evaluation of self-healing systems: An analysis of the state-of-the-art and required improvements," *Computers*, vol. 9, no. 1, p. 16, 2020.
- [28] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.
- [29] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 685–722, 1996.
- [30] N. Sridhar, "Decentralized local failure detection in dynamic distributed systems," in *2006 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*. IEEE, 2006, pp. 143–154.
- [31] K. S. Gyamfi, J. Brusey, E. Gaura, and R. Wilkins, "Heartbeat design for energy-aware IoT: Are your sensors alive?" *Expert Systems with Applications*, vol. 128, pp. 124–139, 2019.
- [32] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, 2018.
- [33] G. P. Bhandari and R. Gupta, "Fault analysis of service-oriented systems: a systematic literature review," *IET Software*, vol. 12, no. 6, pp. 446–460, 2018.
- [34] A. Gorbenko, A. Romanovsky, and O. Tarasyuk, "Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency," *Journal of Network and Computer Applications*, vol. 146, p. 102412, 2019.
- [35] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 3, pp. 8–es, 2007.
- [36] P. L. Snyder, G. Valetto, J. L. Fernandez-Marquez, and G. D. M. Serugendo, "Augmenting the repertoire of design patterns for self-organized software by reverse engineering a bio-inspired p2p system," in *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 2012, pp. 199–204.
- [37] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," in *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE, 2011, pp. 1–6.
- [38] J. Lim, K.-S. Chung, H. Lee, K. Yim, and H. Yu, "Byzantine-resilient dual gossip membership management in clouds," *Soft Computing*, vol. 22, no. 9, pp. 3011–3022, 2018.
- [39] X. Cao, S. Gao, and L. Chen, "Gossip-based load balance strategy in big data systems with hierarchical processors," *Wireless Personal Communications*, vol. 98, no. 1, pp. 157–172, 2018.
- [40] X. He, Y. Cui, and Y. Jiang, "An improved gossip algorithm based on semi-distributed blockchain network," in *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2019, pp. 24–27.
- [41] L. Baird, "The swirls hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirls, Inc. Technical Report SWIRLDS-TR-2016*, vol. 1, 2016.
- [42] T. Preisler, T. Dethlefs, and W. Renz, "Middleware for constructing decentralized control in self-organizing systems," in *2015 IEEE International Conference on Autonomic Computing*. IEEE, 2015, pp. 325–330.
- [43] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, 2007.
- [44] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.
- [45] D. Pianini, J. Beal, and M. Viroli, "Improving gossip dynamics through overlapping replicates," in *International Conference on Coordination Languages and Models*. Springer, 2016, pp. 192–207.
- [46] M. Ding, X. Cheng, and G. Xue, "Aggregation tree construction in sensor networks," in *EEE 58th Vehicular Technology Conference. VTC 2003-Fall*, vol. 4. IEEE, 2003, pp. 2168–2172.
- [47] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 2, pp. 1–40, 2008.
- [48] E. Pournaras and J. Nikolić, "On-demand self-adaptive data analytics in large-scale decentralized networks," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2017, pp. 1–10.
- [49] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [50] M. Hasan and M. S. Goraya, "Fault tolerance in cloud computing environment: A systematic survey," *Computers in Industry*, vol. 99, pp. 156–172, 2018.
- [51] F. Fanitabasi, E. Gaere, and E. Pournaras, "A self-integration testbed for decentralized socio-technical systems," *Future Generation Computer Systems*, vol. 113, pp. 541–555, 2020.
- [52] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, "Protopeer: a p2p toolkit bridging the gap between simulation and live deployment," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009, pp. 1–9.
- [53] 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.
- [54] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 398–407.



Jovan Nikolić is a Software Engineer at Google, Zürich, Switzerland. Since 2019, he holds a MSc in Computer Science from Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, and since 2015 a BSc degree in Electrical Engineering and Computer Science from University of Belgrade, Belgrade, Serbia. He was a member of Computational Social Science group, ETH Zürich from 2015 to 2019, conducting research focusing on intelligent multi-agent systems, combinatorial multi-objective optimization and machine learning.



Nursultan Jubatyrov is a Software Engineer at Facebook, London, UK. He obtained a BSc degree in Computer Science from Nazarbayev University, Nur-Sultan city, Kazakhstan in 2019. From 2017 to 2018, Nursultan was working as a Research and Engineering Assistant at Computational Social Science group, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland. During this time, he conducted research on the reliability of distributed systems.



Evangelos Pournaras is an Associate Professor at Distributed Systems and Services group, School of Computing, University of Leeds, UK. He is also currently a research associate at UCL Center of Blockchain Technologies. He has more than 5 years experience as senior scientist and postdoctoral researcher at ETH Zurich in Switzerland after having completed his PhD studies in 2013 at Delft University of Technology in the Netherlands. Evangelos has also been a visiting researcher at EPFL in Switzerland and has industry experience at IBM T.J. Watson Research Center in the USA. Evangelos has won the Augmented Democracy Prize, the 1st prize at ETH Policy Challenge as well as 4 paper awards and honors. He has published more than 75 peer-reviewed papers in high impact journals and conferences and he is the founder of the EPOS, DIAS, SFINA and Smart Agora projects featured at decentralized-systems.org. He has raised significant funding and has been actively involved in EU projects such as ASSET, SoBigData and FuturICT 2.0. Evangelos' research interest focus on distributed and intelligent social computing systems with expertise in the inter-disciplinary application domains of Smart Cities and Smart Grids.