ChaT: Evaluation of Reconfigurable Distributed Network Systems Using Metamorphic Testing

Alif Akbar Pranata, Olivier Barais, Johann Bourcier, Ludovic Noirie

▶ **To cite this version:**

Alif Akbar Pranata, Olivier Barais, Johann Bourcier, Ludovic Noirie. ChaT: Evaluation of Reconfigurable Distributed Network Systems Using Metamorphic Testing. GLOBCOM 2021 - IEEE Global Communications Conference, Dec 2021, Madrid, Spain. pp.1-6. hal-03379913

HAL Id: hal-03379913
https://hal.science/hal-03379913

Submitted on 15 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ChaT: Evaluation of Reconfigurable Distributed Network Systems Using Metamorphic Testing

Alif Akbar Pranata, Olivier Barais, Johann Bourcier
Inria, IRISA, Univ Rennes 1
Rennes, France
alif-akbar.pranata@inria.fr, olivier.barais@irisa.fr, johann.bourcier@inria.fr

Ludovic Noirie
Nokia Bell Labs
Nozay, France
ludovic.noirie@nokia-bell-labs.com

***Abstract*—Detecting faults in distributed network systems is challenging because of their complexity, but this is required to evaluate and improve their reliability. This paper proposes ChaT, a testing and evaluation methodology under system reconfigurations and perturbations for distributed network systems, to evaluate QoS reliability by discriminating safe and failure-prone behaviors from different testing scenarios. Motivated by metamorphic testing technique that removes the burden of defining software oracles, we propose some metamorphic relationships that correlate system inputs and outputs to find patterns in executions. Classification techniques based on machine learning (principal component analysis and support vector machine) are used to identify system states and validate the proposed metamorphic relationships. These metamorphic relationships are also used to help anomaly detection. We verify this with several anomaly detection techniques (isolation forest, one-class SVM, local outlier factor, and robust covariance) that categorize experiments belonging to either safe or failure-prone states. We apply ChaT to a video streaming application use case. The simulation results show the effectiveness of ChaT to achieve our goals: identifying execution classes and detecting failure-prone experiments based on metamorphic relationships with high level of statistical scores.**

***Index Terms*—distributed network systems, metamorphic testing, metamorphic relationships, classification techniques, anomaly detection techniques**

I. Introduction

Distributed network systems and functions are implemented as pieces of software (microservices) [1]. This increases the configurability of deployed solutions and the ability to implement autonomic techniques to optimize system performance or to support recovery strategies (e.g., see Tootaghaj et al. [2]). While this trend brings many benefits in terms of flexibility to deploy new services, it also significantly increases the state space of such distributed systems, forcing the community to revisit the techniques used to guarantee their reliability.

In the line of traditional software development, testing techniques are used to deal with the explosion of the state space of such systems, resulting from the high frequency of evolution of these services implementation. Unlike model-checking or software proving techniques, testing can detect the presence and the absence of errors. In standard testing, every software test case requires oracles (a mechanism found in software testing practice for matching the correct outputs with each given input) [3]. Defining a precise oracle for each test case is costly and complex (the observed system behavior and its performance are often non-deterministic). Among many testing practices, metamorphic testing (MT) [4] can minimize the effort of determining the oracle. In MT, the oracle problem can be mitigated by alleviating metamorphic relationships (MRs). MRs is a property to model the relationships between output and input data of the system under test (SUT). MRs tell that for each given input, then the observed outputs should be as what found in baseline information. Hereafter, we call these input data *execution classes*. MRs can also perform more tasks, such as failure and anomaly detection [5].

The intuition we explore in this work is the following: Machine learning techniques applied to a set of observable metrics of the distributed system can be used to define disjoint classes, allowing to classify the behaviors of the system. These classes can then be used as a basis for establishing the MRs necessary for the use of MT. Therefore, the execution of known test scenarios such as normal behavior, reconfiguration, perturbation test, etc., must be classified correctly. If not, the test scenario cannot be correctly analyzed.

We propose *ChaT*, a learning-based testing and evaluation methodology using MT with MRs formalization for detecting safe and failed executions in distributed network systems. ChaT performs execution classification using principal component analysis (PCA) and support vector machine (SVM) techniques. The classification result is used to check and compare the model of MRs, validating the accuracy of the modeling. ChaT also validate MRs usage with anomaly detection techniques: isolation forest (IF), one-class SVM (OCSVM), local outlier factor (LOF), and robust covariance (RC) to decide safe and failed scenario of each execution. We also compare the performance of these detection techniques. We apply ChaT to a small but representative distributed network system, a video streaming application with a set of nodes serving various network functions and roles. Our main contribution is twofold: 1) the formalization of MRs for testing distributed network systems and their validation by using classification techniques, 2) the evaluation of anomaly detection techniques on a representative use case to validate the usage of the MRs.

The rest of the paper is as follows. We discuss related works in section II and ChaT methodology in section III. We explain our use case for the MRs evaluation in section IV. Section V evaluates ChaT methodology in validating the MRs. Section VI concludes the paper and provides future works for ChaT.

II. Related Works

ChaT is motivated by MT that removes the burden of determining software oracles in various application domains [6]. Luo et al. proposed simple uses of the verification technique on microservices using MT and they proved the effectiveness of the technique in detecting failures using the defined MRs [7]. ChaT enhances the proposal by not only detecting and validating the failures using MRs, but also using some anomaly detection algorithms. TDD4Fog by Li et al. also addressed the challenge of testing and verifying software applications in distributed systems using MT [8]. Their approach used several testing techniques, one of them is MT, to test the microservices applications on specific fog computing technology. Consequently, they narrowed their methodology to follow fog computing approach, which is testing the system on bottom-up design fashion. Their approach was also limited to verifying software in its development platform. ChaT proposes not only testing and verifying the software, but also detecting failures in software affected by reconfigurations and perturbations in any execution platform with any network technology, for example, cloud networks.

Johnson et al. offered reconfigurations and perturbations for testing the system using delta debugging approaches [9], which is similar to ChaT. Another example is ConfAdvisor, a configuration tuning framework that works close to system reconfigurations and offers the oracles absence to test and verify the system in Kubernetes [10]. Yet, each of them tests different platforms and use cases. ChaT offers more flexibility in performing testing and verification as it could be applied to a variety of applications in numerous network platforms and environments by formalizing the correct MRs.

There are many applications applied by anomaly detection techniques in the literature, such as time series monitoring, fraud detection, and satellite image analysis [11], [12]. Among those techniques, unsupervised learning approaches are popular for detecting errors and failures in network monitoring and performance on large data set [13]. Tuan et al. implemented the detection scheme using local outlier factor in SDN [14]. In this paper, ChaT applies four popular automatic anomaly detection techniques: Isolation forest (IF) [15], one-class SVM (OCSVM) [16], local outlier factor (LOF) [17], and robust covariance (RC) [18], [19] algorithms. The algorithms have been applied for detecting anomalies in various domains, such as wind turbine monitoring [20] and crop classification data [21]. ChaT is close to Jin et al. [22], working on microservices architecture (MSA) as they used PCA to obtain specific, useful information about MSA before performing anomaly detection techniques, for example as mentioned above, to find anomalies in the microservices metrics performance.

To the best of our knowledge, the ChaT approach, described in the rest of this paper, is the first proposal of testing and verifying distributed systems with a representative use case using MT and the MRs and its validation using classification and anomaly detection techniques.

III. ChaT Methodology Using Metamorphic Testing

A. Metamorphic relationships (MRs) modeling for ChaT

Metamorphic testing (MT) is a testing technique which compares several executions to learn the system behavior by defining MRs in replace of the standard software oracles [6]. The MRs are the system properties represented by the relations among input and output data of the system under test (SUT). The goal of MT is to help verify the system correctness without the need for oracles found in traditional software testing.

The general concept of MRs is as follows. In standard software testing, we need to define oracles (a mechanism to test if the test case has passed or failed based on the relationships between output data for each input data). ; Based on the knowledge from this mechanism, we can decide if new execution is safe or erroneous scenario. In MT, we remove the oracles and instead rely on MRs that are able to assert the decision (safe or erroneous).

Our modeling for MRs is the following. There are some definitions *a priori* to explain each variable. We use our use scenario (video streaming application in section IV) to give a concrete example corresponding to each definition.

Definition III.1 (Baseline configuration)**.** B is the set of baseline configurations, i.e., $B = \{b_1, b_2, ..., b_{n_B}\}$.

We started our execution with default configurations (we assume the absence of errors in the execution). In our video streaming application on section IV, we have 4 video servers which store and relay video information to the clients. An identity manager sits in front of these servers to provide authentication and authorization, as well as a load balancer for balanced traffic distribution. We set some parameters in this configuration, and parameter variations give different baseline configurations $(b_1, b_2, ..., b_{n_B})$.

Definition III.2 (Reconfigurations)**.** R is the set of reconfigurations, i.e., $R = \{r_1, r_2, ..., r_{n_R}\}$.

A reconfiguration is an action on the SUT that changes execution parameters in the experiment during runtime. This action can take the form of CREATE, DELETE, MODIFY, REROUTE, ROLLBACK, and RESTART. Reconfiguration variations may take the same action above, but each should aim for different purposes.

Definition III.3 (Perturbations)**.** P is the set of perturbations, i.e., $P = \{p_1, p_2, ..., p_{n_P}\}$.

A perturbation is an action that alters the behavior of the system due to external injection of faults. Some examples are PACKET_LOSS, LATENCY, STRESS. In the same manner with reconfigurations, perturbation variation may take the same action above, but each should aim for different purposes.

Definition III.4 (Executions)**.** E is the set of executions, i.e., $E = B \times \text{Parts}(R) \times \text{Parts}(P) = \{e_1, e_2, ..., e_{n_E}\}$.

The selection of B, R, P is arbitrary in any E, e.g., $e_1 = (b_1, \{r_1, r_2\}, \{p_1\})$, where $b_1 \in B$, $\{r_1, r_2\} \subset R$, $\{p_1\} \subset P$.

An execution is a set of an experiments running in our simulation environment, using a baseline configuration in B from which it starts, none or some reconfigurations in R and none or some perturbations in P. For example, execution e_1 started the executions with baseline configuration b_1, then creates new clients (reconfiguration r_1), reroutes some traffic to another video server (another reconfiguration r_2), and finally injects stress to load balancer (perturbation p_1).

The set E has the following specific subsets, each of which we call execution class:

1) $E_B = B \times \{\emptyset\} \times \{\emptyset\} \subset E$ is the subset of baseline executions;
2) $E_R = B \times (\text{Parts}(R) - \{\emptyset\}) \times \{\emptyset\} \subset E$ is the subset of executions with reconfigurations only;
3) $E_P = B \times \{\emptyset\} \times (\text{Parts}(P) - \{\emptyset\}) \subset E$ is the subset of executions with perturbations only;
4) $E_{RP} = B \times (\text{Parts}(R) \times \text{Parts}(P) - \{(\emptyset, \emptyset)\}) \subset E$ is the subset of executions with reconfigurations and/or perturbations.

Definition III.5 (Vector space of metrics vectors)**.** M is the vector space of metrics vectors, i.e., $M = \mathbb{R}^{n_M}$ with n_M real metrics.

$M = \mathbb{R}^{n_M}$ is the vector space of metrics vectors containing the n_M real metrics that one may measure on the SUT, e.g., number of HTTP 400 error code, CPU load, memory consumption, etc.

Definition III.6 (Metric function)**.** m is the metric function such that $m : E \to M$ where $x \mapsto m(x)$.

This function gives for each execution in E the resulting metrics vector in M that we observed in the experiments. The metric function gives the following subsets of the metrics vector space M:

1) $M_B = m(E_B) \subset M$ is the set of baseline metrics;
2) $M_R = m(E_R) \subset M$ is the set of reconfiguration metrics;
3) $M_P = m(E_P) \subset M$ is the set of perturbation metrics;
4) $M_{RP} = m(E_{RP}) \subset M$ is the set of combined reconfiguration & perturbation metrics.

For our MRs formalization, we assume that the metrics of baseline execution and the ones for other execution classes should be separated: $M_B \bigcap M_R = \emptyset$, $M_B \bigcap M_P = \emptyset$ and $M_B \bigcap M_{RP} = \emptyset$. Then, based on the above definitions and assumptions, the following MRs should hold for ChaT:

MR_B: **if** $\forall x \in E_B$, $m(x) \in M_B$, **then** $\forall y \in E$, $m(y) \in M_B \Rightarrow y \in E_B$

MR_R: **if** $\forall x \in E_R$, $m(x) \in M_R$, **then** $\forall y \in E$, $m(y) \in M_R \Rightarrow y \in E_R$

MR_P: **if** $\forall x \in E_P$, $m(x) \in M_P$, **then** $\forall y \in E$, $m(y) \in M_P \Rightarrow y \in E_P$

MR_{RP}: **if** $\forall x \in E_{RP}$, $m(x) \in M_{RP}$, **then** $\forall y \in E$, $m(y) \in M_{RP} \Rightarrow y \in E_{RP}$

Our formalized MRs above are simple and understandable so as to clearly define each relations based on the execution classes applied to the SUT, replacing the need of determining software oracles in standard software testing.

B. Principal component analysis (PCA) and support vector machine (SVM) for MRs validation

To evaluate ChaT, we run the executions classes: baseline (E_B), reconfiguration (E_R), perturbation (E_P), and perturbation & reconfiguration (E_{RP}). This gave metrics vectors that we grouped in two data set matrices: D_{train} for training and D_{test} for testing. The dimension n_M of the metrics vectors is usually high, so we used PCA to reduce this dimension. Its effect can be modeled as a function $f_{PCA} : M \to M' = \mathbb{R}^{n_{M'}}$ with $n_{M'} << n_M$. With PCA, we obtained a lower number $n_{M'}$ of new metrics that are given by the composed metrics function $m' = f_{PCA} \circ m : E \to M'$. The data set matrix D_{train} was used to learn the parameters of f_{PCA}, which can then be applied on both D_{train} and D_{test} data sets.

SVM supports ChaT evaluation by setting decision boundaries between each execution class (for classification). In SVM, each data point contributes to the setting of the decision boundaries for the classification among classes. The best boundary is the one that maximized the margin from each class. For ChaT, SVM discriminates each execution class corresponding to E_B, E_R, E_P, and E_{RP}, using decision boundaries in the composed metrics vector space of dimension $n_{M'}$ identified by the PCA technique previously applied.

C. Anomaly detection exploiting MRs modeling in ChaT

Detecting anomalies in streaming data is a crucial problem in a wide range of real-world systems since it contains critical details, such as cybersecurity threats, fraud detection, and other real-time applications risks [11]. To detect anomalies, various approaches such as statistics-based, isolation-based, and clustering-based have been developed. ChaT uses several machine learning techniques for detecting anomalies: IF, OCSVM, LOF, and RC algorithms. The purpose of detecting anomalies by ChaT is to find which actions (reconfigurations and perturbations) may lead to system failures by proving and matching the anomalies finding and the error found in the execution set E. ChaT also aims to recognize patterns in large data sets as well as patterns in the system execution behavior.

The methodology of anomaly detection techniques by ChaT is as follows. After we had PCA calculation and SVM classification techniques above, we expanded our original training set D_{train} and testing set D_{test} with n_M by adding the two principal components from PCA calculation and the execution classes (E_B, E_R, E_P, and E_{RP}) inferred by SVM classification. We then performed 4 outlier detection algorithms mentioned above. The results consist of each algorithm findings with inliers (experiments that are considered safe, normal executions) and outliers (experiments that may lead to system/network failures) for each data set D_{train} and D_{test} plotted in 2D graph with $n_{M'} = 2$.

IV. Use Case to Validate ChaT Methodology

Video streaming applications can create service and delivery quality issues to clients, mostly when reconfigurations or perturbations occur during runtime. For example, when a service is down or changes some parameters, other dependant

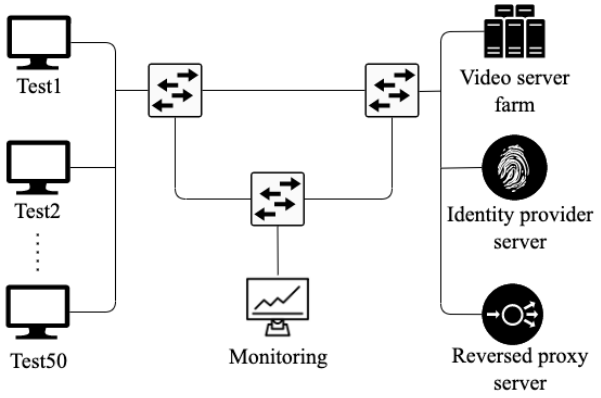


Fig. 1: The network topology of video streaming application.

services may lose their states to continue functioning correctly. We performed ChaT methodology on this use case to discover dynamic system misbehavior resulting from reconfigurations and perturbations and to find potential failures caused by them.

We set up the network architecture and ran our video streaming application consisting of nodes that run services in container applications. We then evaluated ChaT in GNS3 emulation environment by injecting reconfigurations and perturbations to the application. The GNS3 server ran in our private cloud lab with computing resources that can run thousands of microservice applications: Intel(R) Xeon(R) Gold 6238 CPU with 2.10GHz 88 Core, 187GB memory, and 11 TB disk storage size.

Figure 1 illustrates our video streaming application topology. We had objects in the SUT as a set of nodes: 4 video servers, an identity provider, a reversed proxy, 50 clients to send and receive traffics, and monitoring with all necessary tools to collect metrics information for evaluation and analysis. The collected metrics were mainly from the network properties: the number of requests (video server and load balancer), HTTP codes, CPU usage, traffic loads (received and sent), RSS memory, used memory & available memory. In total, the number of metrics used for PCA evaluation was $n_M = 12$.

Using this representative use case which is composed of few services and with nominal metrics, we argue that ChaT is adequate in any scale of distributed systems with a factual number of metrics. The plan for greater system scales is included in our future works.

For the evaluation in the next section, we emulated 1784 executions of this use case with the baseline (E_B), reconfiguration (E_R), perturbation (E_P), and perturbation & reconfiguration (E_{RP}) execution classes. We split the executions into a training data set D_{train} with 1338 experiments and a testing data set D_{test} with 446 experiments.

V. Evaluation of ChaT Methodology

A. Validation of the metamorphic relationships (MRs)

We first validate the MRs defined in subsection III-A with the PCA & SVM method of subsection III-B on the video streaming use case defined in the section IV.

Figure 2 visualizes our evaluation results using PCA and SVM techniques. In figure 2a, PCA was applied on the training data set D_{train} with reduced dimension $n_{M'} = 2$: the two main principal components found by PCA were enough to discriminate the points of different 4 classes represented by different colors. Figure 2b shows the boundaries obtained by SVM classification after dimension reduction by PCA, and the points corresponding to the testing data set D_{test}. We calculated the precision, recall, and F1-score of each execution class to support the visualization of SVM technique. Table I gives the obtained results. The overall accuracy, precision, and recall of our classification were 99%.

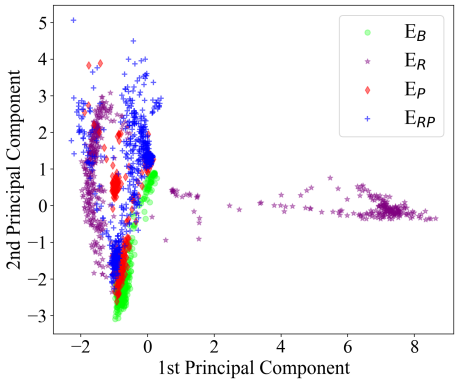
Using this classification, we can learn that new executions will be classified correctly by ChaT because the results above have a high level of accuracy of MRs defined in section III-A. For example, we can know from table I that the correctly classified execution class that belongs to baseline execution (E_B) is 91 (100% accuracy of true positive), meaning that all of 91 experiments has respected the definition of metamorphic relation E_B in section III-A, so any new E_B execution would have 100% accuracy to be classified as E_B execution. Thus, the formalized MRs in section III-A are validated, the assumptions are correct and our classification method automatically builds oracles to determine the execution classes.

B. Validation of the usage of MRs for anomaly detection

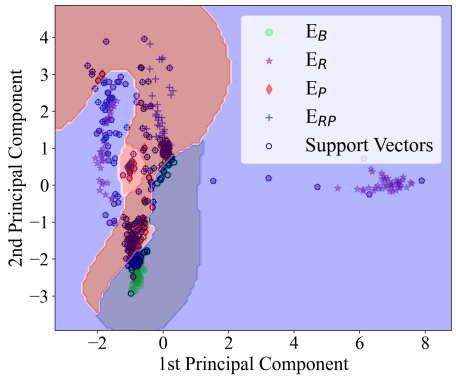
Then we validate the usage of MRs for anomaly detection with the method of subsection III-C on the same video streaming use case defined in the section IV, using isolation forest (IF), one-class SVM (OCSVM), local outlier factor (LOF) and robust covariance (RC) algorithms.

Figure 3 shows the inliers and outliers both in D_{train} and D_{test} sets for the 4 algorithms. Table II shows the classical accuracy, precision, recall, and F-measure scores as defined by Powers [23], from each machine learning algorithm for outliers and inliers and for both D_{train} and D_{test} sets. This shows that ChaT achieves higher accuracy with LOF and RC algorithm, with about 97% accuracy for both training and testing data sets, while IF and OCSVM have lower accuracy. Focusing on the most accurate algorithm (LOF and RC), they correctly predict inliers with precision and recall between 97% and 100%. About outliers, they succeed in detecting them with a good precision between 96% and 100% but there are about 20% of false alarms with a recall between 77% and 80%. Thus, our approach is good to detect alarms (outliers) with a rate of about 20% false alarms. This is acceptable if the objective is to detect alarms when they occur. By this results, we validated our MRs-based approach to detect anomalies.

The fact that the accuracy is high despite the low recall score (about 80%) is because our data set is imbalanced [24] with few outliers compared to inliers, and most outliers coming from E_R and some from E_{RP}, none in E_B as expected. This could be corrected by using resampling approaches (oversampling and undersampling) [25], even though it can be thought of as an inadequate practice in applying machine learning techniques.



(a) PCA applied on D_{train} with $n_{M'} = 2$.



(b) The classification using SVM on D_{test}.

Fig. 2: Executions classification using ML techniques.

TABLE I: Execution classification results for each execution class.

Execution class	Support	TP	TN	FP	FN	Precision	Recall	F1-score
Baseline (E_B)	91	91	355	0	0	100%	100%	100%
Reconfiguration (E_R)	135	135	310	0	1	100%	99%	100%
Perturbation (E_P)	65	63	379	2	2	97%	97%	97%
Reconfiguration, Perturbation (E_{RP})	155	153	290	2	1	98%	99%	99%

VI. Conclusion & Future Works

This paper proposes ChaT, a testing and evaluation methodology under reconfigurations and perturbations for distributed network systems to evaluate QoS reliability by finding safe and failure-prone system behaviors. ChaT performed metamorphic testing with our formalized metamorphic relation to find patterns of each input class with its expected outputs for the system under test, replacing the traditional known oracles in common software testing scenarios. ChaT has 4 execution classes: baseline (E_B), reconfiguration (E_R), perturbation (E_P), and reconfiguration and perturbation (E_{RP}). For execution classification, we used PCA and SVM techniques on metrics data we obtained on the video streaming application. PCA reduced the high dimensional data knowledge to lower dimension $n_{M'} = 2$ without losing the essential information, separating each execution class. SVM could then easily find the boundaries between these classes and classify different system behaviors corresponding to different execution classes with a high accuracy: 99% in our use case. ChaT then applied various anomaly detection techniques, analyzed and compared the system performance dynamics under the 4 execution classes that we could identify with the MRs. Among these techniques, local outlier factor (LOF) and robust covariance (RC) could detect potential failed experiments in the executions with a level of accuracy up to 98%.

For future work, we plan to apply ChaT in the software production phase with real-world use cases. In such use cases, PCA can learn the system performance and behavior by examining its historical data information. Thus, the data set D_{train} can be replaced by such data and the rest of the approach should follow ChaT methodology. After we collect metrics data and information of such implementation, we are interested in carefully examined the data in both D_{train} and D_{test} for any data sets imbalances. The goal is to improve the analysis of ChaT classification and anomaly detection, thus improving the evaluation of the system QoS reliability.

References

[1] T. Salah, M. Jamal Zemerly, Y. Chan Yeob, M. Al-Qutayri, and Y. Al-Hammadi. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 318–325. IEEE, dec 2016.

[2] D.Z. Tootaghaj, N. Bartolini, H. Khamfroush, T. He, N.R. Chaudhuri, and T.L. Porta. Mitigation and Recovery From Cascading Failures in Interdependent Networks Under Uncertainty. *IEEE Transactions on Control of Network Systems*, 6(2):501–514, jun 2019.

[3] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, may 2015.

[4] T.Y. Chen, S.C. Cheung, and S.M. Yiu. Metamorphic Testing: A New Approach for Generating Next Test Cases. feb 2020.

[5] O. Johnston, D. Jarman, J. Berry, Z.Q. Zhou, and T.Y. Chen. Metamorphic Relations for Detection of Performance Anomalies. In *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)*, pages 63–69. IEEE, may 2019.

[6] S. Segura, G. Fraser, A.B. Sanchez, and A. Ruiz-Cortes. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, sep 2016.

[7] G. Luo, X. Zheng, H. Liu, R. Xu, D. Nagumothu, R. Janapareddi, E. Zhuang, and X. Liu. Verification of Microservices Using Metamorphic Testing. pages 138–152. 2020.

[8] Rui Li, Xiao Liu, Xi Zheng, Chong Zhang, and Huai Liu. TDD4Fog: A Test-Driven Software Development Platform for Fog Computing Systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 673–676. IEEE, may 2020.

[9] J. De Bleser, D. Di Nucci, and C. De Roover. A Delta-Debugging Approach to Assessing the Resilience of Actor Programs through Run-time Test Perturbations. In *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test*, pages 21–30, New York, NY, USA, oct 2020. ACM.

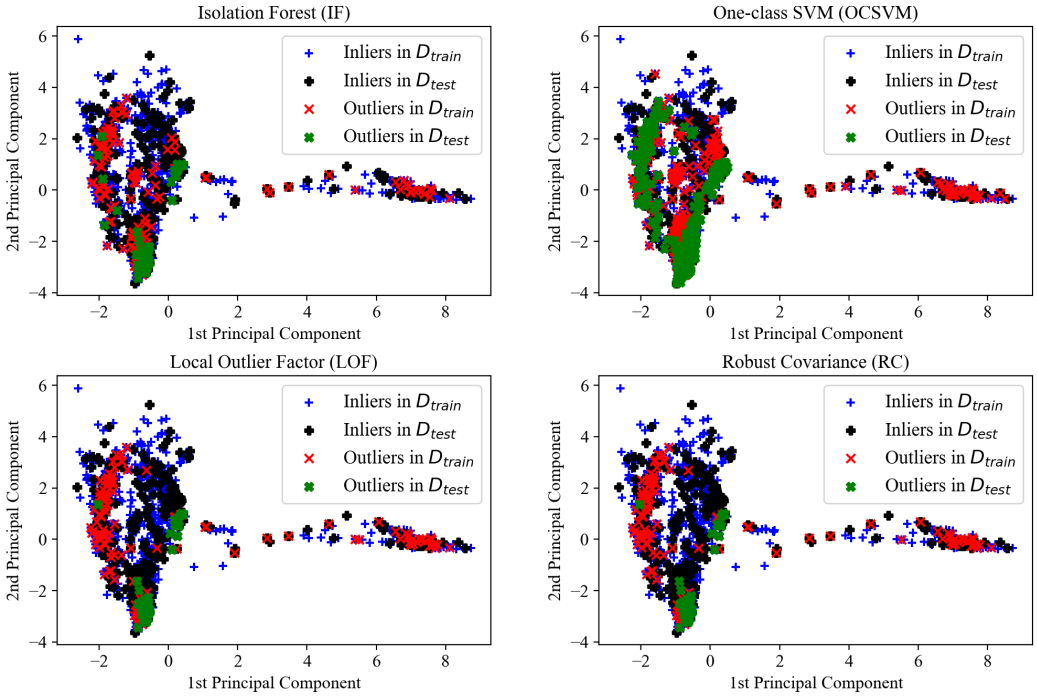


Fig. 3: Anomaly detection techniques for finding potential errors in system exections.

TABLE II: Anomaly detection results for each algorithm on training and testing data set.

Algorithm	Type	Training Set				Testing Set			
		Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy
IF	Outlier	78%	61%	69%	93%	84%	68%	75%	94%
	Inlier	95%	98%	96%		96%	98%	97%	
OCSVM	Outlier	43%	98%	60%	83%	26%	93%	41%	66%
	Inlier	100%	81%	90%		96%	98%	97%	
LOF	Outlier	99%	78%	87%	97%	100%	80%	89%	98%
	Inlier	97%	100%	98%		97%	100%	99%	
RC	Outlier	100%	78%	88%	97%	96%	77%	85%	97%
	Inlier	97%	100%	98%		97%	99%	98%	

[10] T. Chiba, R. Nakazawa, H. Horii, S. Suneja, and S. Seelam. ConfAdvisor: A Performance-centric Configuration Tuning Framework for Containers on Kubernetes. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 168–178. IEEE, jun 2019.

[11] V.J. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 22(2):85–126, oct 2004.

[12] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, feb 2018.

[13] Z. Cheng, C. Zou, and J. Dong. Outlier detection using isolation forest and local outlier. *Proceedings of the 2019 Research in Adaptive and Convergent Systems, RACS 2019*, pages 161–168, 2019.

[14] N.N. Tuan, N. Danh Nghia, P.H. Hung, D. Khac Tuyen, N.M. Hieu, N. Tai Hung, and N.H. Thanh. An Abnormal Network Traffic Detection Scheme Using Local Outlier Factor in SDN. In *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, pages 141–146. IEEE, jan 2021.

[15] F.T. Liu, K.M. Ting, and Z.-H. Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, dec 2008.

[16] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13(7):1443–1471, jul 2001.

[17] M.M. Breunig, H.-P. Kriegel, R.T. Ng, and J. Sander. LOF: identifying density-based local outliers. *ACM SIGMOD Record*, 29(2):93–104, jun 2000.

[18] P.J. Rousseeuw and K.V. Driessen. A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics*, 41(3):212–223, aug 1999.

[19] M. Hubert, M. Debruyne, and P.J. Rousseeuw. Minimum covariance determinant and extensions. *WIREs Computational Statistics*, 10(3), may 2018.

[20] C. McKinnon, J. Carroll, A. McDonald, S. Koukoura, D. Infield, and C. Soraghan. Comparison of New Anomaly Detection Technique for Wind Turbine Condition Monitoring Using Gearbox SCADA Data. *Energies*, 13(19):5152, oct 2020.

[21] L. Shumilo. Automatic Anomaly Detection Methodology for Crop Classification Data Using Morphological Features. In *2020 IEEE 5th International Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, pages 1–5. IEEE, sep 2020.

[22] M. Jin, A. Lv, Y. Zhu, Z. Wen, Y. Zhong, Z. Zhao, J. Wu, H. Li, H. He, and F. Chen. An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis. *IEEE Access*, 2020.

[23] D. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[24] Haibo, H. and E.A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, sep 2009.

[25] N.V. Chawla. Data Mining for Imbalanced Datasets: An Overview. *Data Mining and Knowledge Discovery Handbook*, pages 875–886, 2009.