DistFax: A Toolkit for Measuring Interprocess Communications and Quality of Distributed Systems

Xiaoqin Fu
xiaoqin.fu@wsu.edu
Washington State University
Pullman, WA, USA

Boxiang Lin
boxiang.lin@wsu.edu
Washington State University
Pullman, WA, USA

Haipeng Cai
haipeng.cai@wsu.edu
Washington State University
Pullman, WA, USA

Abstract

In this paper, we present DistFax, a toolkit for measuring common distributed systems, focusing on their interprocess communications (IPCs), a vital aspect of distributed system run-time behaviors. DistFax measures the coupling and cohesion of distributed systems via respective IPC metrics. It also characterizes the run-time quality of distributed systems via a set of dynamic quality metrics. DistFax then computes statistical correlations between the IPC metrics and quality metrics. It further exploits the correlations to classify the system quality status with respect to various quality metrics in a standard quality model. We empirically demonstrated the practicality and usefulness of DistFax in measuring the IPCs and quality of 11 real-world distributed systems against diverse execution scenarios. The demo video of DistFax can be viewed at https://youtu.be/VLmNiHvOuWQ online, and the artifact package is publicly available at https://tinyurl.com/zaz27ec8.

CCS Concepts

• **Software and its engineering → Maintaining software**.

Keywords

Distributed system, IPC, software measurement, quality

ACM Reference Format:
Xiaoqin Fu, Boxiang Lin, and Haipeng Cai. 2022. DistFax: A Toolkit for Measuring Interprocess Communications and Quality of Distributed Systems. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510454.3516859

1 Introduction

Most real-world, industry-scale software systems are distributed by design, in order to accommodate increasing demands for performance and scalability. However, due to their typically great sizes and complexities along with intrinsic concurrency and nondeterminism [6], distributed software systems are especially difficult to characterize (e.g., measure), understand, and analyze [10].

In this context, it is beneficial to have practical tooling support that exercises well-informed and principled measurement mechanisms for distributed software systems. In general, the components of a *common* distributed system (as defined in [6]) concurrently run on different machines (at physically different locations) and communicate with one another via message passing. Importantly, there does not exist a global timing mechanism (clock) that enables readily synchronizing the message passing and other execution events across the distributed processes of the running system. Thus, an important, distinct aspect of the run-time behaviors of distributed systems lies in those induced by interprocess communications (IPCs). A tool that characterizes IPCs would help distributed software developers understand the IPC-induced behaviors and the run-time quality of the systems related to those behaviors.

However, measuring the IPC-induced run-time behaviors of distributed systems is not trivial. For instance, based on the mechanism of IPCs, it would be intuitively useful to measure the cohesion of individual processes and coupling between different processes in distributed system executions. Yet traditional coupling metrics are based on explicit relations among program entities, and thus they are not suitable for measuring IPCs in distributed systems. Accordingly, existing tools that compute those metrics target single-process programs and do not properly apply to distributed software.

Jin et al. proposed three cohesion metrics (*CC*—short for cohesion metric at component level, *CCW*—cohesion metric at component level considering edge weights, *CHC*—cohesion factor of component) and one coupling metric *CPC* (coupling factor of component). These metrics were used to measure the cohesion and coupling for multi-process programs. And the authors used the Kieker monitoring framework [21] to probe the systems, monitor their run-time behaviors during system executions, and store execution traces. However, both the Kieker tool and these metrics were designed only for *specialized* distributed systems (e.g., RSS Reader applications, the distributed version of iBATIS JPetStore) [14, 15].

As it stands, current software metrics and measurement tools do not sufficiently support measurements of common distributed systems as described above, including the measurement of IPC-induced behaviors. Especially, there is a lack of tools that not only measure IPCs but also characterize how IPC-induced behaviors are related to the run-time quality of distributed software.

Therefore, we define a novel set of IPC metrics [9], as listed in Table 1, aiming to measure the communication structure, complexity, and reusability of common distributed systems at runtime. Based on the definitions, we developed DistFax (short for *Distributed software system's facts (sounding Fax)*), a toolkit for measuring the IPCs of distributed systems in relation to their quality, with respect to six IPC metrics about the coupling and cohesion of distributed processes. This paper describes DistFax and demonstrates its use. At its core, DistFax computes these IPC metrics by reasoning about interprocess dependencies [3] from the happens-before relations

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05.
https://doi.org/10.1145/3510454.3516859

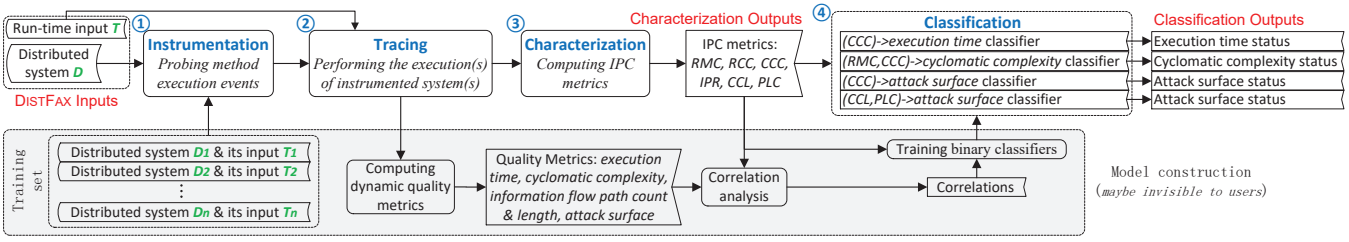

Figure 1: An overview of DistFax architecture.

among executing methods across processes, derived from a global partial ordering of execution (i.e., entry/returned-into [2]) events of the methods. Then, DistFax quantifies the statistical relationships between these IPC metrics and several quality metrics, as listed in Table 2. Finally, DistFax leverages the discovered relationships to classify the level/status (`anomalous` versus `normal`) of quality of the system with respect to those dynamic quality metrics through supervised or unsupervised learning based on the IPC metrics.

We have applied DistFax to 11 distributed systems of various architectures, domains, and sizes against diverse, large-scale execution scenarios (as reflected by different types of run-time inputs). DistFax exhibited practical capabilities of measuring IPC-induced behavioral characteristics of these systems and their executions in terms of the six IPC metrics. Based on the measurement results, DistFax also enabled the discovery of several moderate/strong correlations between IPC and run-time quality metrics. Our empirical results further demonstrated that the IPC-metrics-based quality classifiers offer practically useful effectiveness (about 74%∼98% F1 accuracy, on average) for the quality (level/status) classifications.

Novelty/contributions. To the best of our knowledge, DistFax offers the *first toolkit for characterizing IPC-induced behaviors in relation to various run-time quality metrics of distributed systems, while further providing capabilities (grounded upon those relationships) of helping users assess quality levels of the systems.* The main contributions are the design and implementation of DistFax and the empirical evaluation of it on industry-scale distributed systems against their executions driven by 16,880 run-time inputs.

Audience/usefulness. The envisioned users of DistFax include distributed software system developers, testers, maintainers, and end users. They can use the IPC measurement results to obtain a comprehensive understanding of relevant behaviors of the system execution(s) of interest. Like defect prediction and anomaly detection tools, from the quality classification results, *DistFax can tell whether the system quality is abnormal hence needs attention or not.* Thus, these users can *make decisions on various aspects of the system quality and take actions accordingly*, as how a defect/anomaly predictor helps users improve software quality.

2 Background

Interprocess communication (IPC) is the standard mechanism used in common distributed systems [6] through which the distributed processes communicate with each other and synchronize their actions. A primary way of realizing IPC is *message passing*, by which processes send and receive messages either synchronously or asynchronously via blocking or non-blocking I/Os.

Accordingly, *IPC metrics* [9] have been proposed as a kind of dynamic software metrics dedicated to measuring IPC related characteristics of distributed software. Intuitively, these metrics quantify *IPC-induced behaviors* of distributed systems—the run-time system behaviors due to the use of IPCs. Measuring IPCs is essential as they constitute a unique aspect of the overall run-time behaviors of distributed systems (e.g., concerning whole-system code dependencies [3] and the security of information flow across process boundaries [11]), as opposed to those of single-process software.

Yet different from *quality metrics*, IPC metrics do not immediately measure the quality of distributed systems. On the other hand, since it is aimed to help users understand distributed system's quality through IPC metrics (which are dynamic), DistFax considers dynamic quality metrics that are related to IPC-induced system behaviors. Moreover, we select such quality metrics from those defined in ISO/IEC 25010 [13], a standardized system and software quality model. DistFax is built on the statistical relationships between the IPC metrics and those selected quality metrics to help users understand and classify distributed systems' quality status.

3 DistFax Architecture

Figure 1 shows an overview of DistFax's architecture. In a typical usage scenario, DistFax Inputs consist of a distributed system D and its run-time input set T, which is a set of program inputs fed to D during its operation (execution). With these inputs, DistFax first instruments D in the **Instrumentation** phase and then executes the instrumented system to trace method execution events during the **Tracing** phase. From the execution traces, DistFax computes the IPC metrics of D in the third phase **Characterization**, resulting in the metric values as the Characterization Outputs. Lastly, in the **Classification** phase, DistFax uses the IPC metric values as features to classify the quality of D against T with respect to several statistically relevant quality metrics (e.g., execution time as a performance efficiency metric). Underlying these classifications are the pre-trained per-quality-metric classifiers. Each classifier is named in a form of $X \rightarrow Y$ where X is the set of IPC metrics used as features while Y is the quality metric whose status is classified as `anomalous` or `normal`. All such classification results constitute the Classification Outputs of DistFax.

These pre-trained classifiers were built via a *model construction* workflow which may be invisible to users unless they need to re-train the classifiers (by following a similar workflow). In this workflow, DistFax takes n distributed systems $D_1, D_2, ..., D_n$ and their respective input sets $T_1, T_2, ..., T_n$ as the training set. By reusing relevant phases in the typical usage scenario, it computes IPC metrics and several dynamic quality metrics for each training sample,

Table 1: Summary of IPC Metrics

Type	Metric	Definition / computation	Rationale
Coupling	RMC	interprocess message coupling; computed as #messages sent from one process to another	the extent of run-time interactions among processes
	RCC	class coupling across processes; computed as the ratio of #methods in the second class that are dependent on any method in the first class, to #methods in total which are dependent on any method in the second class in all processes but the first one	how methods from a class in one process access methods in other processes
	CCC	the aggregate coupling as regards an individual class executed in a local process with respect to classes in all remote processes; computed as the aggregate RCC metrics between the class and other classes in all remote processes	the importance of a class in terms of its influence on classes in remote processes
	IPR	interprocess coupling via common methods; computed as the size ratio of intersection between the local and remote dependence sets to the union set of methods executed	code overlapping and reuse across processes
	CCL	the communication load of an individual class communicating with others in all remote processes; computed as the sum of the sizes of remote dependence sets of the class's methods divided by the size of the set of all executed methods in the class	how much a class contributes to the communication load among processes
Cohesion	PLC	internal connections within an individual process; computed as the sum of the sizes of local dependence sets of all methods in the process, divided by the size of the set of all executed methods in the process	the degree to which all of the methods executed in a process belong together

Table 2: Summary of Quality Metrics

Metric	Description	Justification (w.r.t ISO/IEC 25010 [13])
Execution time	time duration of system execution	measure the quality characteristic *performance efficiency* and its sub-characteristic *time behavior*
Run-time cyclomatic complexity	cyclomatic complexity measured at runtime, computed as #independent paths covered at runtime + 1 [17]	measure the quality characteristic *maintainability* and its sub-characteristics *modifiability* and *testability*
Information flow path count	#information flow paths between all sources and sinks, computed by our dynamic taint analyzer [8]	measure the quality characteristic *security* and its sub-characteristic *confidentiality*
Information flow path length	the average length of all the information flow paths, computed by our dynamic taint analyzer [8]	measure the quality characteristic *security* and its sub-characteristic *confidentiality*
Attack surface	computed as the Euclidean distance between <0,0,0> and the vector <#methods that enclose a source/sink, #network ports used, #files read/written> [16]	measure the quality characteristic *security* and its sub-characteristics *confidentiality*, *integrity*, *non-repudiation*, *accountability*, and *authenticity*

and calculates the correlations between them. Finally, DistFax trains both supervised classifiers using the two types of metrics and unsupervised classifiers using the IPC metrics only.

The supervised classifiers apply to situations in which users can afford to compute all these dynamic quality metrics hence have labeled training dataset to enable potentially more accurate classifications. In contrast, the unsupervised classifiers may offer lower accuracy but accommodate scenarios where the users do not have enough labeled data for training. DistFax provides both options to meet diverse user needs. The key rationale is that computing most (if not all) of dynamic quality metrics (e.g., those based on attack surfaces) can be very expensive, which is also why we use IPC metrics to indirectly classify quality status rather than using the quality metrics directly in model training and inference.

4 Typical Workflow

We now elaborate on each of the key working phases of DistFax in its typical usage scenario (i.e., with classifiers trained already) to produce characterization and classification results as outputs.

Phase 1: Instrumentation. DistFax computes the IPC and quality metrics mainly based on dynamic dependencies at method level using our dependence analysis framework for distributed programs [3], for which method execution events need to be traced. To that end, it inserts probes for monitoring covered statements and the *entry* (i.e., program control entering a method) and *returned-into* (i.e., program control returning from a callee into a caller) events [4] of each executed method. This results in the instrumented version of D, reusing our Java dynamic analyzers [5, 7]. In addition, DistFax refers to the list of the most widely used APIs in the JDK for probing and then monitoring the message-passing (i.e., sending/receiving messages) events during the execution(s).

Phase 2: Tracing. DistFax executes the instrumented version of the distributed system D with its inputs T to generate method-execution/message-passing event and statement coverage traces, which will be used to compute the IPC metrics of the system.

Phase 3: Characterization. From the execution traces obtained in the previous phase, DistFax computes five IPC coupling metrics—RMC (short for run-time message coupling), RCC (run-time class coupling), CCC (class central coupling), IPR (interprocess reuse), and CCL (class communication load), and one IPC cohesion metric—PLC (process-level cohesion). The detailed definitions can be found in [9]. For this demo to be self-contained, we summarize them in Table 1. For a method m, the set of methods that depend on m and are in the same (*resp.*, any different) process as m is referred to as the local (*resp.*, remote) dependence set of m.

Phase 4: Classification. In this phase, DistFax feeds the trained per-quality-metric classifiers with the IPC metrics, which results in the quality status classified as `anomalous` or `normal` concerning the respective quality metrics. Each quality metric corresponds to a quality sub-characteristic that the metric measures. Thus, the classification results indicate the status of the distributed system quality with respect to the associated quality sub-characteristic.

5 Model Construction

While the IPC metric values as immediate characterization results help capture IPC-induced behaviors of distributed systems, DistFax goes further to inform the implications of those metrics to the run-time quality of the systems. To that end, DistFax first discovers with respect to which quality metrics can the IPC measures inform about the quality. After extensive investigation, we found five (dynamic) quality metrics significantly correlated to IPC metrics. Table 2 summarizes these quality metrics, of which some had moderate/strong correlations with some of the IPC metrics in terms of Spearman's coefficients [19]. The IPC metrics and correlated quality metrics are shown in the classifier names in Figure 1.

These quality metrics were chosen according to the standard quality model ISO/IEC 25010 [13]. Each quality metric directly measures quality sub-characteristics under a quality characteristic defined in the quality model. We normalized all these quality metrics by the system size in terms of logical source line of code (SLOC) using the LocMetrics tool [1] rather than its physical SLOC to reduce the biases caused by variations in programming formats/styles.

For supervised training a classifier named $X \rightarrow Y$, each sample is the X vector value computed from a system execution (D_i against T_i of Figure 1). The sample is labeled *normal* if the Y value for this sample is lower than the mean Y over all training samples, and *anomalous* otherwise. The rationale is that the quality metrics considered are all reversed measures: lower metric values indicate better quality in that metric (e.g., greater attack surface signifies lower security). DistFax uses the bagging algorithm by default, which was found to be more accurate than any other supervised learning algorithms in the latest Scikit-learn library [18].

When unsupervised learning the classifier $X \rightarrow Y$, Y values are not available in the training set where samples are not labeled. We used a k-means clustering algorithm (also implemented in Scikit-learn) with k=2 to cluster the training samples based on their X values. The mean of each resulting cluster is computed as the mean of the Euclidean lengths of all X vectors in the cluster. Then, a cluster with a higher (lower) mean is labeled as *anomalous* if each IPC metric in X was found positively (negatively) correlated with Y. Otherwise, the label *normal* is determined accordingly.

For each (supervised or unsupervised) classifier, X includes IPC metrics that are either all positively or negatively correlated with Y. When Y is found correlated with IPC metrics of which some are positively correlated and others negatively correlated, we split these IPC metrics into two groups and train two separate classifiers. This is why we had two classifiers for some of the quality metrics (e.g., for *attack surface* as shown in Figure 1).

Through these classifiers, DistFax enables assessing distributed system quality through IPC measurement. For example, when a system execution has a large (relative to the average case) IPC metric *CCC* value, its quality metric *attack surface* [16] would be relatively large (than normal), according to the positive correlation between these two metrics. Thus, DistFax alarms that the system would have relatively low quality (security) in terms of attack surface.

6 Applying DistFax

We have implemented DistFax using Java 8 and Python 3, targeting distributed systems written in Java. For supporting distributed systems in other programming languages, only the first phase **instrumentation** needs to be modified. We have successfully applied DistFax to 11 real-world distributed systems of various architectures, application domains, and sizes against a total of 16,880 execution scenarios. All subjects are industry-scale systems, including ZooKeeper used by Yahoo [12] and Voldemort by LinkedIn [20]. The run-time inputs included integration, system, and load tests.

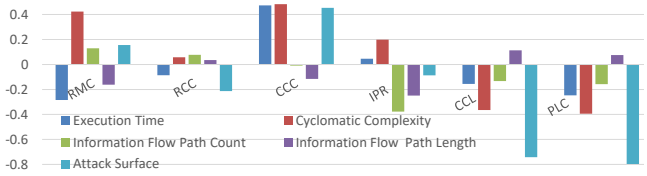

Figure 2: Spearman's correlation coefficients between IPC and quality metrics for 11 systems and 16,880 executions.

From these diverse and large-scale system executions, DistFax computed the IPC and quality metrics as described earlier. Then it computed correlations between the two kinds of metrics for all the 16,880 data points, and trained the per-quality-metric classifiers on 70% of the samples randomly selected while using the remaining 30% for testing for a hold-out validation. For evaluation purposes, we further performed a 10-fold cross-validation for each classifier. All experiments were performed on Ubuntu Linux 18.04.1 workstations, each with one 2.4GHz CPU, 512GB DRAM, and 2TB HDD. For the experiment for one subject, the average time cost was 17 minutes and the peak memory cost was 53GB in the worst case.

6.1 Results

Statistical correlations. Figure 2 depicts the Spearman's correlation coefficients between the six IPC metrics (each marking a bar group) and five dynamic quality metrics considered (each represented by a color bar). On the particular datasets DistFax was applied to, four of the IPC metrics (RMC, CCC, CCL, and PLC) were found significantly and moderately/strongly correlated to three of the quality metrics (execution time, dynamic cyclomatic complexity, and attack surface) with varying strengths.

In particular, the positive correlations between CCC and these three quality metrics indicate that a higher degree of class central coupling implies longer execution time, higher run-time complexity, and greater attack surface. We also observed that higher interprocess message coupling (RMC) was significantly associated with higher cyclomatic complexity. Also, CCL and PLC are significantly and negatively correlated to attack surface. Thus, if a distributed system has a larger internal class-level communication load (CCL) measure and/or a higher process-level cohesion (PLC) measure than others, the communications throughout the system executions tend to be within individual processes, hence the attack surface that outer attackers can reach may be smaller.

In sum, there are three significantly positive correlations (i.e., CCC with execution time, RMC/CCC with dynamic cyclomatic complexity, and CCC with attack surface) and one significantly negative correlation (i.e., CCL/PLC with attack surface) computed by DistFax. These IPC metrics and quality metrics are used as the features and classification outcomes, respectively, of the binary classifiers in the last phase of DistFax.

Quality classification. Figure 3 shows the effectiveness (precision, recall, and F1 accuracy) of our unsupervised learning (k-means clustering) and supervised learning (bagging) classifiers built based on the statistical correlations.

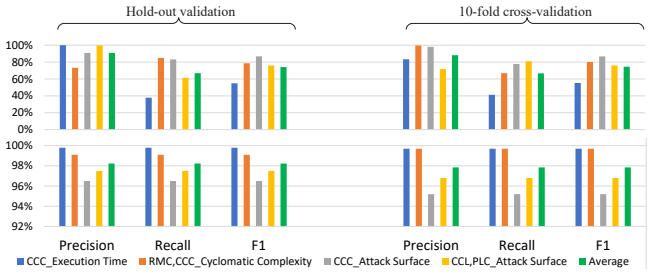

Figure 3: The precision, recall, and F1 accuracy of k-means clustering (y axis, top) and bagging (y axis, bottom) of DistFax per classifier (x axis), for both validation schemes.

DistFax achieved generally high effectiveness for classifying the three quality metrics (execution time, dynamic cyclomatic complexity, and attack surface) based on the IPC metrics (RMC/CCC/CCL/PLC) that are significantly correlated to those quality metrics (see Figure 2). As expected, the supervised classifiers were generally more effective than unsupervised ones according to our results of both the hold-out validations and 10-fold cross-validations.

In particular, with the unsupervised classifiers, the least effective result (55% F1) was for classifying execution time from CCC, while the most effective was for classifying attach surface from CCC (87% F1). In contrast, the supervised classifiers are commonly highly performant, offering promising (95+% F1) accuracy for any of the quality metrics considered. Finally, it is noteworthy that for both supervised and unsupervised classifiers, the results from the two validation schemes were quite consistent, consolidating the validity of the quality classifications in DistFax.

6.2 Use Scenario

In our demo, we use Apache Thrift to illustrate how DistFax works. First, we instrument Thrift and execute the instrumented system against given inputs. Second, we compute the six IPC metrics and the five dynamic quality metrics from the executions. Third, we compute Spearman's rank correlation coefficients [19] between these IPC and quality metrics. Then, we train the unsupervised and supervised classifiers for each of the quality metrics that are significantly correlated with some IPC metrics. Lastly, we apply the classifiers to predict the quality status for two quality metrics.

6.3 Limitations

DistFax is subject to several limitations. First, DistFax inserts probes into the system's bytecode in its first phase. If the bytecode is not allowed to be modified, the first phase **instrumentation** cannot run (hence the toolkit cannot work). Second, DistFax does not address all of the quality metrics that may be related to IPC-induced system behaviors. It also only focuses on dynamic quality metrics at this point, with static quality metrics (e.g., bugs and vulnerabilities reported) not considered. Third, quality metrics in DistFax do not cover all of the quality characteristics (e.g., compatibility, portability) and sub-characteristics (e.g., interoperability, adaptability) defined in the underlying quality model ISO/IEC 25010 [13]. Finally, through its binary classifiers, DistFax currently only supports quality classifications at a relatively coarse and high level.

7 Conclusion

We presented DistFax, a toolkit that systematically measures a given distributed system for its IPC-induced behaviors in (statistical) relation to various aspects of its quality at runtime. It further leverages such relations to enable users to assess those quality aspects with respect to a range of quality metrics that would otherwise be very expensive to measure directly.

We have applied DistFax to 11 real-world distributed systems against 16,880 execution scenarios in total. DistFax exhibited practical capabilities in measuring IPCs, run-time quality, and their correlations in these systems. Our evaluation also demonstrated that DistFax achieved high effectiveness with its unsupervised and supervised classifiers. In sum, DistFax offers the first toolkit for measuring IPCs and their quality implications in common distributed systems, hence for understanding IPC-relevant run-time behaviors of these systems, with promising merits.

Acknowledgments

This work was supported by NSF through grant CCF-1936522.

References

[1] S Aswini and M Yazhini. 2017. An Assessment Framework of Routing Complexities Using LOC Metrics. In *2017 Innovations in Power and Advanced Computing Technologies*. 1–6.
[2] Haipeng Cai. 2017. Hybrid Program Dependence Approximation for Effective Dynamic Impact Prediction. *TSE* (2017).
[3] Haipeng Cai and Xiaoqin Fu. 2021. D^{2}ABS: A Framework for Dynamic Dependence Analysis of Distributed Programs. *TSE* (2021).
[4] Haipeng Cai and Raul Santelices. 2014. DIVER: Precise Dynamic Impact Analysis Using Dependence-based Trace Pruning. In *ASE*. 343–348.
[5] Haipeng Cai and Douglas Thain. 2016. DistIA: A Cost-Effective Dynamic Impact Analysis for Distributed Programs. In *ASE*. 344–355.
[6] George Coulouris et al. 2011. *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley Publishing Company.
[7] Xiaoqin Fu et al. 2020. Dads: Dynamic Slicing Continuously-Running Distributed Programs With Budget Constraints. In *ESEC/FSE*. 1566–1570.
[8] Xiaoqin Fu and Haipeng Cai. 2019. A Dynamic Taint Analyzer for Distributed Systems. In *ESEC/FSE*. 1115–1119.
[9] Xiaoqin Fu and Haipeng Cai. 2019. Measuring Interprocess Communications in Distributed Systems. In *ICPC*. 323–334.
[10] Xiaoqin Fu and Haipeng Cai. 2020. Scaling Application-Level Dynamic Taint Analysis to Enterprise-Scale Distributed Systems. In *ICSE-Companion*. 270–271.
[11] Xiaoqin Fu and Haipeng Cai. 2021. FlowDist: Multi-Staged Refinement-Based Dynamic Information Flow Analysis for Distributed Software Systems. In *USENIX Security*. 2093–2110.
[12] Patrick Hunt et al. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX ATC*.
[13] ISO/IEC. 2011. ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.
[14] Wuxia Jin et al. 2016. Dynamic Cohesion Measurement for Distributed System. In *SCDTCP*. 20–26.
[15] Wuxia Jin et al. 2018. Dynamic Structure Measurement for Distributed Software. *Software Quality Journal* (2018).
[16] Pratyusa K Manadhata et al. 2010. An Attack Surface Metric. *TSE* (2010).
[17] Thomas J McCabe. 1976. A Complexity Measure. *TSE* (1976).
[18] Fabian Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of Machine Learning research* (2011).
[19] Philip Sedgwick. 2014. Spearman's rank correlation coefficient. *Bmj* (2014).
[20] Roshan Sumbaly et al. 2012. Serving Large-scale Batch Computed Data with Project Voldemort. In *FAST*, Vol. 12. 18–18.
[21] André Van Hoorn et al. 2012. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *ICPE*. 247–248.