Hindawi
Mobile Information Systems
Volume 2021, Article ID 2912240, 13 pages
https://doi.org/10.1155/2021/2912240

Research Article

CIPC: A Change Impact Propagation Computing Based Technique for Microservice Regression Testing Prioritization

Lizhe Chen,[1] **Xiang Yu**,[2,3] **Ji Wu**,[1] **and Haiyan Yang**[1]

[1]*School of Computer Science and Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China*
[2]*College of Computer Science, National University of Defense Technology, Changsha, Hunan 410073, China*
[3]*College of Electronic Engineering, National University of Defense Technology, Hefei, Anhui 230037, China*

Correspondence should be addressed to Xiang Yu; yuxiang17@nudt.edu.cn

Received 24 September 2021; Accepted 29 October 2021; Published 23 November 2021

Academic Editor: Fazlullah Khan

Regression testing is the optimal technique that can be used in each iteration of microservice systems. However, regression testing prioritization is the only main method that gives better results. These techniques directly involve the processes of artifacts, data acquisition, analysis, and maintenance. The microservice systems have input data, which are difficult to obtain and control, while such processes are of high costs with impractical design. This paper gives a detailed study on testing prioritization technique, which is referred to as CIPC. As there are dependencies between services from API gateway logs, a novel CIPC algorithm is proposed, which is based on belief propagation. There are some rules that are directly affected by service changes. Therefore, the higher execution order of test case prioritizes CIPC, which is based on impact changes. Multiobjective prioritization algorithm is based on heuristic searching, in which sequence test cases are done by coverage. By evaluating the effectiveness of CIPC, the empirical study presents five microservice systems and four different techniques. The results describe that CIPC has improved fault detection rate with acceptable time and cost. The technique is more practical than typical artifacts, which are based on increments of system scales.

1. Introduction

Lightweight communication mechanism utilizes microservice due to its widespread design. Especially, this architecture pattern is widely preferred for cloud applications [1–3]. Additionally, the entire procedure assists in constructing and deploying microservice [4]. Through this mechanism, requirements of business expansion are completely satisfied.

Furthermore, regression test is conducted in each cycle of microservice systems. Such test verifies whether former technical issues are fixed or not. Also identify any most probable obstacle during working or fixing [5]. However, a common approach for regression test is to rerun previously used cases, where it illustrates costs. Moreover, rapid repetition with numerous amounts of service is established, whereas the entire costs of repeated trial are assessed through this process. For instance, WeChat (social microservice system) offers thousands of amenities. Hence, this particular system might consume several months to perform the entire sort of regression assessments along with retest-all strategy [6]. Therefore, numerous techniques are practiced in order to optimize the system. Predominantly, to enhance trails efficiency and decrease overall expenses, several approaches are adopted. Such methodologies include prioritization, selection, and test suite minimization [7].

Firstly, regression testing prioritization (RTP) deals with adjusting test cases in proper order. Through this procedure, a common goal is to evaluate desirable properties and improve fault detection rate [5]. Therefore, earliest spotted issue is kept on priority and run through whole system whether this is likely to be happen in other parts or not. As a result, a relationship is established between test cases and historical information. Moreover, artifacts or history data is incorporated in building relationship. Artifacts include specifications, strategy models, and code files. On the other

hand, historical data compromised previous failure. Additionally, RTP procedure is composed of two-stage analysis and prioritization. In the analysis phase, historical data or artifacts are managed to retrieve evidence such as coverage, critical-rank, and change impacts, whereas prioritization phase and test cases are linked with information and arranged through former data. However, RTP practices consider that historical data is broad, reliable, and accessible. Furthermore, correlation amongst them is sustained through availability in continuous integration environments [8, 9].

Although RTP based acquisition on data is difficult to initiate, still it is implemented for better response. Thus, a specific approach is applied to microservice regression testing. Practically, the system encounters three challenges while performing.

(a) Data Attainment

As the microservice system works independently, it faces several issues while achieving artifacts and historical information from multiple teams. Thus, extra communication is acquired for gathering, which results in elevating the total cost. Additionally, the expense is increased through obedience with numerous security strategies.

(b) Data Integration

Having developments in methodologies, the procedure becomes more complex to integrate the data. Therefore, integrity, comprehensibility, and consistency of information are difficult. Consequently, the performance of RTP is severely affected.

(c) Sustaining Relationship

With growth of system scale in uninterrupted environments, association amongst artifacts, history data, and test cases might become difficult to sustain. Due to large set of information, various issues hinder from building association between them.

Hence, the above three main issues, which exist in RTP based techniques, are not practical in microservice regression testing.

Furthermore, API gateway layer is a significant element of microservice systems, where it is utilized for centralized distribution of service requests [4]. Practically, in API mode, the gateway logs record the entreaties. However, requests from clients include applicant information, responder, time, and status code. For performance, bundles of statistical data are collected for further assessment. Additionally, attained requests submitted in API gateway logs are executed individually to prove service scheme dependencies. Moreover, regarding the system that is primarily based on dependencies, impact investigation is carried out, where the impact analysis depends on belief propagation strategy. Incase test cases are ranked on the basis of final output, and the above challenges can be governed easily. Thus, this motivates the approach, which is incorporated in paper. Change impact propagation computing mechanism is primarily taken from microservice regression test strategy CIPC.

Initially, CIPC creates request-directed-graph (RDG) through proper counting frequency of invocations from API gateway logs. Secondly, change impact propagation is based on belief propagation algorithm. Additionally, the proposed algorithm is utilized to measure change impacts quantitatively from the RDG [10]. Thirdly, microservice experiment is represented as test path [11, 12], whereas test paths are matched in pairs based on results. Such comparison determines execution orders. Furthermore, a multiobjective prioritization algorithm based on heuristic searching is proposed. Thus, test path is prioritized by both service coverage and change impacts. Moreover, to estimate effectiveness of CIPC, experimental revisions are carried out. However, five systems are utilized to measure average percentage of fault detection (APFD) and time costs [5]. Beside this, four typical RTP techniques of three groups are nominated to match with CIPC such as artifacts-RTP, history-RTP, and control-RTP. Consequently, outcomes illustrate that CIPC enhances fault detection rate as compared with former procedures. Especially, less prioritization is observed in time-cost scenario by two magnitudes. Finally, measure escalates during continuous integration environment, where time-cost shows preferable level. This ensures system with reliable incomes of data.

To sum up the introduction, leading contributions of this paper are as follows.

(1) Data utilized to prioritize test-cases are extracted from API gateway logs, while regression assessment evaluation phase information is collected from microservice.

(2) Change impacts are quantitatively calculated through mathematical approach.

(3) Single-objective and multiobjective prioritization are proposed to meet practical requirements.

2. Related Methodologies

2.1. Microservice Testing. Microservice testing comprises unit-testing, service-testing, and end-to-end testing [4]. However, unit testing is utilized to identify culpabilities in functions or classes. Additionally, this process is sustained by two experimental tools such as xUnit [13] and mockito [14]. In order to bypass user interference and rapid assessment, this service-testing is preferred. However, end-to-end testing focuses on behaviors of entire system to improve efficiency. Although several serious challenges were faced by this autonomous system, still service and end-to-end testing are designed to tackle them effectively. Particularly, this approach is concerned about researches and practices of microservice testing [4, 8], whereas test-cases in this research work include service and end-to-end based assessment.

Furthermore, both procedures at the same interval involve multiple services and their invocations. So, high efficiencies are obtained through service and end-to-end analysis. For instance, consumer-driven based evaluation includes consumer, target, and stubbed-services [4]. Therefore, test cases of these two assessment possibilities are

abstracted as self-possessed test paths of services, which are purely based on prioritization criteria [11, 12, 15–18].

2.2. Test Case Prioritization. Formal definition of regression testing prioritization problem is as follows:

Definition 1 (RTP issue).

Given: T, test-suite, PT, f, and set of all possible permutations of T, whereas f determines performance from PT to real numbers.

Problem: Find $T' \in PT$:

$$\text{s.t. } (\forall T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]. \quad (1)$$

Definition 1 illustrates main points of RPT. Objectives of equation focus on degree of qualitative model, where test cases satisfy prioritization strategies through designing function f, whereas "f, T" comparison-based sorting, heuristic searching, and algorithms approach exist [19].

Furthermore, RTP is emerging area in regression testing based research. However, artifacts procedures are distributed into two categories such as code and model based approaches [20, 21]. Firstly, code-based approaches principally deal with coverage information of test cases such as statement, branch, and path coverage [17, 22]. Such approaches involve source codes availability of system during analysis. Though multiple complex programming languages create challenges for microservice system, therefore, coding system depends on specific language to resolve the complexity of computing. Secondly, model-based methodologies establish association information amongst test-cases and models, whereas models are further integrated into system structures or system behaviors. This approach is utilized to prioritize test cases based on criticality, complexity, and defect density of models [23–25]. Hence, model-based methodology is more preferred than code-based techniques due to its practical viabilities. Beside these feasibilities, it also incorporates microservice system to be more dynamic, integrated, and consistent throughout the procedure.

In addition, several research works primarily focus on historical information specific scenarios. Research conducted by Azizi and Do [26] elucidates that test cases were ranked by changing historical data of artifacts in dynamic environment. However, sustaining changing trend of historical information is a difficult task in microservice systems, where location-based RTP procedure is recommended for embedded environment [27]. This technique emphasizes gravitation laws for high reliability after modification. Also, similarity-based RTP methodology is proposed for product line testing scenario [28]. Revision evaluated effectiveness of real and seeded fault detection in Ouriques [29]. Diverse prevailing RTP methods are compared to monitor the real-time efficiency of approach. Therefore, model-based analysis is compared to investigate influence of test-case on fault-detection rate. However, this approach is only preferable for specific circumstances. Generally, it is not applicable for all regression techniques in microservice regression testing.

With the advent of machine and deep learning approaches in recent years, numerous researchers applied these tactics to RTP [20]. The typical approaches are neural network based RTP approaches such as genetic algorithm and Bayesian network based RTP attitudes. Amongst them, methods based on neural network mainly extract features from system specifications [30]. Beside this, historical analysis records data to prioritize test cases [31]. Additionally, genetic algorithm approaches build some features such as module failure rate, code coverage, and optimum orders, whereas RTP tactics based on Bayesian network mainly combine white box information such as code-changes and code-coverage [32]. Generally, technological methodologies still count on codes, conditions, replicas, and historical statistics as inputs. Therefore, the system has numerous obstacles while performing acquisition, investigation, and sustaining as mentioned above.

2.3. Belief propagation. Belief propagation algorithm (BP) is a repetitive process for estimated interpretation based on graph structure. There are numerous applications of BP, which include forward propagation algorithm, Viterbi algorithm, decoding algorithms of low density parity check (LDPC), and turbo codes. Such methodologies are utilized for different sort of situation and scenario [10]. However, the core idea of BP algorithm is to transmit message in order to maintain communication through repeated iterations. Thus, it keeps updates confidence values of nodes. Generally, BP algorithm is as follows:

(1) Initialization: set the initial value of each node.

(2) Propagation: update all message values and node confidence values.

(3) Determining values are convergent. Incase standards are convergent, and inference results are obtained according to confidence values. Otherwise, we will jump back to step (2) and propagate iteratively.

In recent years, studies on BP algorithm comprise application and optimization. According to its application, scholars predominantly focus on communication coding and signal processing. In order to reduce complexity of SCMA, dynamic edge assortment procedure based on BP algorithm is introduced. Through iterative calculation, range boundaries of nodes are detected [33]. However, nonlinear equalization method utilized neural network, where BP algorithm is applied to remove signal noises [34]. Additionally, for large-scale MIMO channel detection, belief propagation is used, which is purely based on deep neural network [35]. In optimization aspect, investigators primarily focus on implementation and convergence condition. Furthermore, LDPC along computational process assists in parallelization and merging memory access [36]. Beside this, convergence problem of BP and numerical polynomial-homotopy-continuation method revealed influence of structures. Therefore, parameters of graph models solved through fixed points [37].

Literature study reveals that BP algorithm is not applied in microservice RTP, though the proposed work acquires to analyze impacts based on service dependencies from API gateway logs. Additionally, when it is transformed to

directed graph, the process of impact analysis propagation from changed services to nodes of unchanged services. Thus alteration investigation is converted into graph based on intellectual problem.

3. Methodologies

CIPC resolves issue as given in Definition 1 for microservice systems. This problem is tackled through two stages: change impact analysis and test path prioritization as displayed in Figure 1. Furthermore, inputs primarily comprise API gateway logs and test suite, while output is queue of test paths, which determine testing orders.

The above illustration includes two activities: request directed graph generation and change impact propagation computing, where the model based on graph from API gateway logs is generated in earlier stage, while change impacts are created in the second phase through analysis.

3.1. Request Directed Graph Generation. In this approach, each record of API gateway logs comprises data such as requestor, responder, and tracing identifier. Through Service Registry, identifiers of requestor and responder are achieved [4]. Additionally, service status is properly observed through microservice frame, which includes Eureka of Netflix, ZooKeeper of Apache, and Nacos of Alibaba, while Service-Chain-Monitor key function is tracing the identity [4]. Besides that, alternative setup of microservice is utilized to monitor user session such as Open Zipkin of Twitter, CAT of VWAP, and Naver Pinpoint, where services request call by specific user sharing identical trace identity.

Through accumulation records of logs, probability of interservice dependencies is calculated. However, service supplication is estimated, which is purely based on large number theorem. Statistical analysis of this approach is defined as follows.

Definition 2. (Service Dependencies)

Given a set S as all services of microservice system, $\forall s_i$, $s_j \in S$ $(i \neq j)$. Suppose inv_{ij} symbolizes invocation from s_i to s_j, and $\text{count}(\text{inv}_{ij})$ denotes the number of tracing identifiers of all inv_{ij} in API gateway logs. Incase $\text{count}(\text{inv}_{ij}) > 0$, probability of s_i depending on s_j is defined as follows:

$$w \approx \frac{\text{count}\left(\text{inv}_{ij}\right)}{\sum_{i \neq k}\left(\text{count}\left(\text{inv}_{ik}\right) + \text{count}\left(\text{inv}_{ki}\right)\right)}. \tag{2}$$

Based on formula (2), RDG can be defined as follows:

Definition 3. (Request directed graph, RDG)

RDG is a tuple (N, E), where $N \subseteq S$ represents set of nodes. $E = \{e_{ij} | w(e_{ij}) > 0, 1 \leq i, j \leq n \wedge i \neq j\}$ represents set of directed edges. eij symbolizes directed edge from node s_i to node s_j with the weight $w(e_{ij})$, which is computed by formula (2).

From Definition 3, RDG is generated from API gateway logs with Algorithm 1, where the algorithm scans logs for each record. Besides this evaluation, it assists in cross-examinations of service ID, updating nodes, and directed edges of RDG as given in lines 2 to 11, though function query_services recovers requester and responder info from service registries. Additionally, function find_or_new finds node or edge in directed graph, whereas function update is used to count the trace identifiers related to invocations or services with logs. Finally, at completion of browsing, weights of edges are updated through the approach based on formula (2) (lines 12 to 15) (Algorithm 1).

3.2. Change Impact Propagation Computing. Given service set S and modified version S', it is convenient to achieve list of changed services. Service registries are represented as S'-S. Moreover, to measure impacts of nodes S'-S on other nodes, BP based algorithm is preferred, where updating and message propagation approach for nodes is proposed. Also, junction of repetitive process is analyzed as follows:

(1). Node Updating. Meanwhile, there is no limit to sum of change impacts on all service nodes. Therefore, standard BP algorithm is inadequate in such cases. This approach defines "impact degree" to measure variation on service nodes, where impact degree of nodes S'-S is defined as integer 1, while upper limit value and influence level are not affected by fluctuations. At final stage, its impact degree is updated through messages from directed edge. However, proficient value should be maximal value of all messages. The formal definition is as follows:

Definition 4 (Impact Degree)

Assume 3 as definition of RDG. Suppose N_i symbolize neighborhood of s_i and m_{ji} represents message from s_j to s_i, where t denotes iteration cycle of message propagation. Then, impact degree $p_t(s_i) \in [0, 1]$ of s_i is recursively computed.

$$p_0\left(s_i\right) = \begin{cases} 1, s_i \in S' - S \\ 0, s_i \notin S' - S \end{cases}, \tag{3}$$

$$p_{t+1}\left(s_i\right) = \max\left(p_t\left(s_i\right), \max_{j \in N_i}\left(m_{ji}\right)\right). \tag{4}$$

Apparently, impact degree after message propagation must not be less than earlier message propagation, which is expressed as $p_{t+1}(s_i) \geq p_t(s_i)$.

(2) Message propagation. In case of graph, directed edge is denoted as e_{ij}, where impact degree s_i will be propagated to s_j based on weight $w(e_{ij})$. So, message is well defined in the following way.

Definition 5. (Message)

Definition for RDG is 3, where $e_{ij} \in$ RDG.E, and message m_{ij} propagated from s_i *to* s_j:

$$m_{ij} = p_t\left(s_i\right) \times w\left(e_{ij}\right). \tag{5}$$

Since $w(e_{ij}) \leq 1$, therefore, the message value will always be smaller than the existing impact level. This correlation is denoted as $m_{ij} \leq p_t(s_i)$.

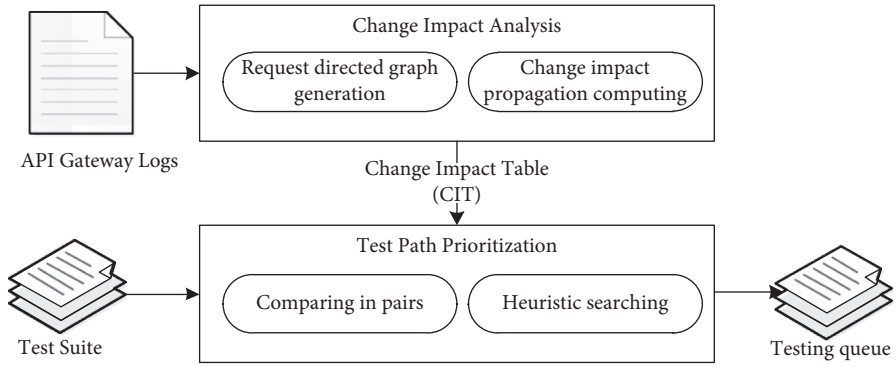

Figure 1: Complete flowchart of prioritization analysis.

(3) Convergence Analysis. Classifications 3.3 and 3.4 indicate computing process of impact propagation. When there is no loop in directed graph, then RDG is acyclic display. When propagation sequences do not exceed the number of edges in the longest RDG path, then calculation is convergent. Additionally, there are some other types of loops in RDG. Therefore, updating series is unsatisfactory. In such case, it is categorized into two possible circumstances: changed services and static services in loops, respectively, discussed as follows:

(i) When there is node $s \in S'\text{-}S$ in the loop, then $p_0(s) = 1$, where $p_{t+1}(s) \geq p_t(s)$, and $p_t(s) \leq 1$, then $p_t(s) = p_0(s) = 1$, impact amount of s is not rationalized through message propagations, whereas directed edge s at final phase does not subsidize to computing process and measured as disturbed. Meanwhile, loop is directly disconnected and transformed into directed acyclic graph. Hence, computing process converges, which is presented in Figure 2.

(ii) For node s of loop, $s \notin S'\text{-}S$, then $p_0(s) = 0$. According to Definition 4, messages propagated on loop are not synchronized until loop collects peripheral messages. When there are multiple inputs, then output function Max relies on parameters value. Therefore, multiple input messages reach the loop in the same round through aligning iteration. However, computing process of loop is distributed into three phases:

① **Computing all input messages:** At this phase, message propagations in loop are not updated. So, it is transformed into directed acyclic graph, where computing process converges in this stage.

② **Computing the message propagations in loop**: As the input reaches loop in the same round, impact degrees are assigned according to formula (4). Through mathematical assessment, node s_m with maximal impact degree is achieved, which is denoted by $p_t(s_m)$. Since messages propagated in loop satisfy $m_{ij} < p_t(s_i)$, maximal value of messages in loop is less than $p_t(s_m)$. Therefore, the impact level will not be synchronized with regard to how much messages propagate in loops. Furthermore, loop is disconnected from directed edge with s_m and loop is removed, which leads to computing convergence.

③ **Computing all output messages**: According to stage 2 outcomes, output messages in loop are computed directly based on formula (5). Schematic diagram of calculation-process is shown in Figure 3.

To summarize, impact degrees are computed through Definitions 4 and 5, which are convergent. For the sake of simplicity, outcomes of variation propagation computing are directly placed in dictionary structure, where it is referred to as the change-impact-table (CIT). Also, pseudocodes CIT is created from RDG as presented in Algorithm 2. Firstly, impact degrees RDG for nodes (lines 1 to 3) are initialized according to $S'\text{-}S$. Secondly, line 4 is used for CIT. Thirdly, iterative computing iteration is used (lines 6 to 8). Moreover, lines 9 to 13 are utilized to update impact degree through messages. Finally, repetitive processes, where impact degrees are, keep constant for all nodes in CIT (line 14).

3.3. Test Path Prioritization. Test path prioritization (TPH) is utilized to arrange test-paths based on change impact analysis (CIA) outcomes. However, TPH assessment is further categorized into two main activities.

(1) **Comparing in pairs:** this approach focuses on single-objective prioritization, where variation in test-path is compared in pair form.

(2) **Heuristic searching:** this methodology is purely based on multiobjective prioritization. Test-route along with impact variation and service coverage is performed through heuristic based searching.

3.3.1. Comparing in Pairs. Single-objective prioritization is utilized to exploit impacts variation on test paths. Therefore, execution is performed earlier, because variation strongly influences test paths. Additionally, TPH procedure is considered to produce analysis queue, where test paths queue comprises two steps.

(i) **Set the head of queue:** test-path with numerous services is considered as head of queue.

```
    Declaration: generateRDG(logs, RDG).
    Parameters: logs(in); RDG(out).
(1) RDG.N, RDG.E←{}
(2) for each l ∈ L do
(3)    s_i, s_j ← query_services(S, l)
(4)    if s_i ≠ null then
(5)       RDG.N.find_or_new(s_i)
(6)    end if
(7)    RDG.N.find_or_new(s_j)
(8)    update(s_j.Count)
(9)    e_ij ←RDG.E.find_or_new(s_i, s_j)
(10)   update(e_ij.Count)
(11) end for
(12) for each r ∈ RDG.E do
(13)   s_i←r.Sender
(14)   e.w←e.Count/s_i.Count
(15) end for
```

Algorithm 1: RDG generation algorithm.

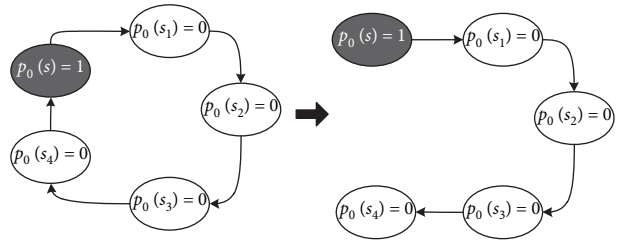

Figure 2: Loop-wise illustration when node services variates.

(ii) **Update tail of queue:** test-path is usually closed to existing tail. Change impact is attached to queue as new tail, where it is determined through computing variation on test paths in pairs.

Particular test suite T and service set S, for test path $tp \in T$. Services contain tp which is composed of subset S_{tp} of S. Through probing output CIT of change impact examination, impact level is achieved of service $s \in S_{tp}$, denoted as CIT(s). Therefore, impact degrees of services in tp are composed of set $V_{tp} = \{CIT(s), s \in S_{tp}\}$, whereas this approach emphasis is on test path tp variation, which depends on services. However, two test-paths tpa, tpb are compared in order to remove services with the same impact degree. Furthermore, based on set-operation, the proposed model is utilized to measure variations of impacts fluctuation as shown in Definition 6.

Definition 6. (Dissimilarity of change-impacts dc)

Given two test paths tpa and tpb, suppose (S_{tpa}, S_{tpb}) symbolize services tpa and tpb, respectively, where $V_{tpa} = \{\text{CIT(s)}, s \in S_{tpa}\}$ *and* $V_{tpb} = \{\text{CIT(s)}, s \in S_{tpb}\}$, denote difference of change-impacts, which is defined as follows:

$$\text{dc}(\text{tpa}, \text{tpb}) = |\text{elementmax}(V\text{tpa} - (V\text{tpa} \cap V\text{tpb})) - \text{elementmax}(V\text{tpb} - (V\text{tpa} \cap V\text{tpb}))|, \quad (6)$$

where $\text{element}_{\max}$ is an operator to recover maximal element of set V.

$$\text{element}\max(V) = \begin{cases} v, V \neq \varnothing, \forall vi \in V, v \geq vi \\ 0, V = \varnothing \end{cases} \quad (7)$$

Since impact degree CIT(s)$\in$[0, 1], it is concluded that $\text{element}_{\max}(V) \in [0, 1]$ and dc(tpa, tpb)$\in$[0, 1]. Additionally, pseudocodes are proposed for single-objective prioritization as shown in Algorithm 3. However, inputs of Algorithm 3 include CIT and test-suite, whereas the output comprises mainly assessment-based queue of test paths. Moreover, set of impact-degrees is initialized for every single test-route, and queue head is figured out through traversing test suite (lines 1 to 7). Finally, queue tail is updated consistently until the complete test route is added (lines 8 to 18).

3.3.2. Hermitic Searching. Single-objective limits the relevancy of prioritization techniques, which are used to ignore the practical constraints, while regression testing multi-objective approaches are utilized in recent research [19].

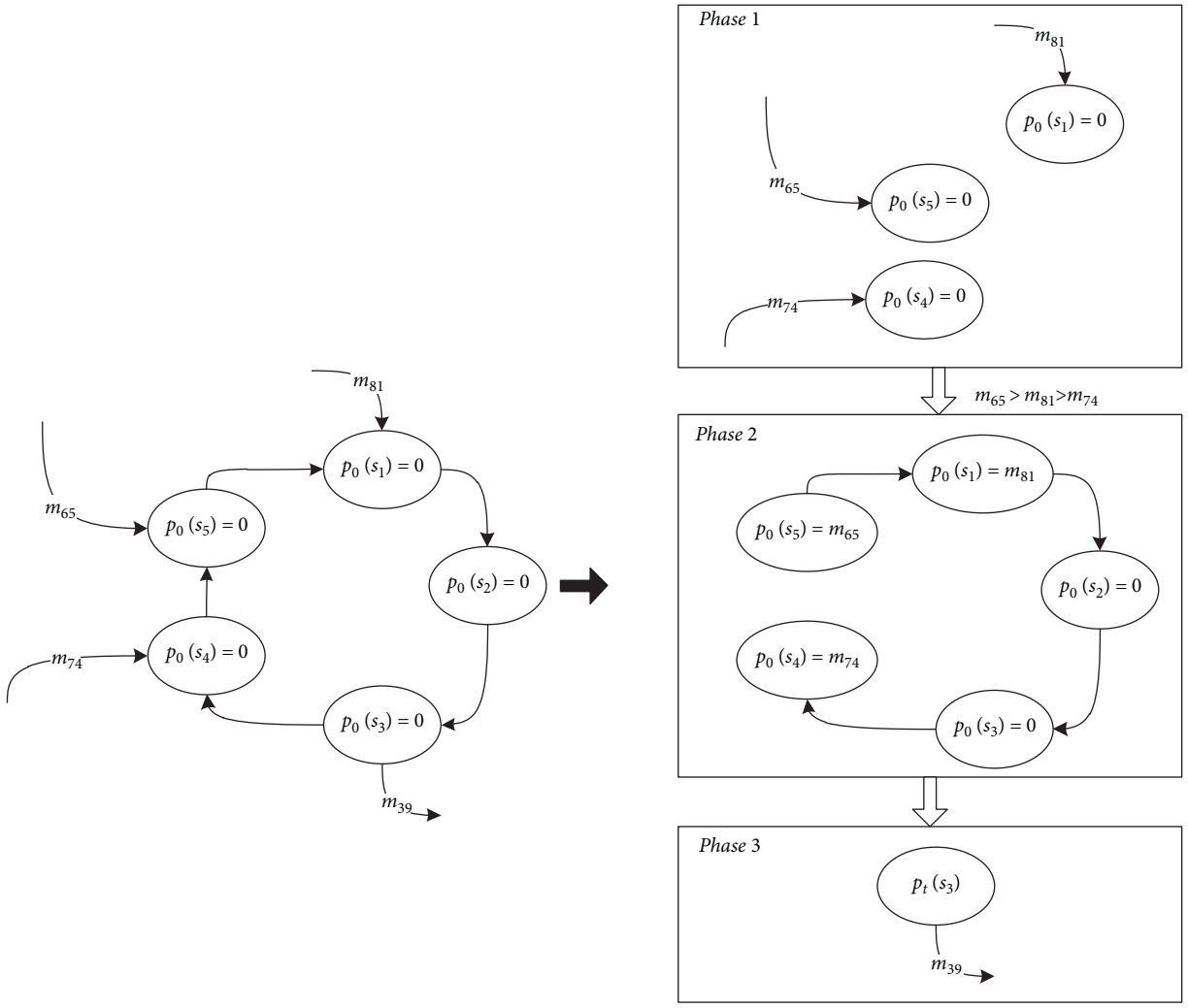

Figure 3: No nodes of the loop are changed services.

Early maximization of structural coverage needs to increase the fault detection rate, and also prioritization techniques aim to optimize statement, branch, and path coverage [5]. Microservice regression testing will provide the coverage to use structural objectives, which should be applied on the objectives. Multiobjective prioritization of CIPC is considered to maximize the metrics, which include simplicity and service coverage.

However, multiobjective prioritization is the two-step process, which is used to generate testing on queue. Also, every step needs to be regulated, while, in the first step, two options directly determine the head of testing queue. Therefore, selection of test route will maximize service coverage. Also, the increase has high impact on the related selected test path. In heuristic searching technique, the mentioned options are utilized. The second step will update test path to fix the heuristic search, which is based on quantitative measurement.

According to Definition 6, dc is applied to measure the altered impact, although the most suitable candidate test path increases the service coverage, which is used to test the queue. To find out the test path tp with its service set S_{tp}, suppose the services of testing queue are denoted as S_{TQ} by using Jaccard distance [38] in between S_{TQ} and S_{tp}. The maximization is as follows:

Definition 7. (Jaccard distance between two service sets, *ds*)

Give two service sets S_a, S_b, where the diversity between S_a and S_b can be computed as follows:

$$ds(S_a, S_b) = \frac{|S_a \cup S_b| - |S_a \cap S_b|}{|S_a \cup S_b|}. \quad (8)$$

Based on Definition 7, $ds(S_a, S_b) \in [0, 1]$, which has the specified value range of dc and *ds*, while dc and ds, the quantitative measurement, must need to increase heuristic searching, which can be computed as $ds(S_{TQ}, S_{tp})$-dc(TQ.tail, tp). The TQ.tail represents tail of the testing queue. However, Algorithm 4 is proposed for multiple objective prioritizations.

Algorithm 4 is having similar metrics like input, output, and control structures in comparison with Algorithm 3. The main difference in Algorithm 4 is the head of testing queue, which selects the test path. This process increases the service

```
    Declaration: computeImpactPropagation (RDG, S', CIT).
    Parameters: RDG (in); S'(in); CIT(out).
(1) for each s ∈ RDG.N do
(2)   s.p←1 if s ∈ S'-RDG.N else 0
(3) end for
(4) CIT←Table(RDG.N)
(5) do
(6)   for each e ∈ RDG.E do
(7)     e.m←e.w•e.start.p
(8)   end for
(9)   for each s ∈ RDG.N do
(10)    for each e ∈ RDG.E and e.end = s do
(11)      s.p←e.m if e.m > s.p
(12)    end for
(13)  end for
(14) while CIT.update(RDG.N)
```

Algorithm 2: Impact propagation algorithm.

```
    Declaration: singlePrioritization (CIT, T, TQ).
    Parameters: CIT (in); T, test suite (in); TQ, testing queue(out).
(1) TQ←Queue()
(2) for each tp ∈ T do
(3)   tp.V←{CIT(s), s ∈ tp.S}
(4)   if element_max(tp.V) > element_max(TQ.head.V) then
(5)     TQ.head ← tp
(6)   end if
(7) end for
(8) T←T - {TQ.tail}
(9) while T≠∅ do
(10)  tc'←T[0]
(11)  for each tc ∈ T do
(12)    if dc(TQ.tail, tc) > dc(TQ.tail, tc') then
(13)      tc'←tc
(14)    end if
(15)  end for
(16)  TQ.tail←tc'
(17)  T←T - {TQ.tail}
(18) end while
```

Algorithm 3: Single-objective prioritization algorithm.

coverage and tail of the queue continuously by using the rules from line 1 to 9, while the queue directly updates the value of ds–dc (lines 10 to 19).

4. Empirical Study

The experimentation is performed on Python 3.5, which accesses CIPC to collect five microservice systems for analysis. The subsections are used to describe case introduction, techniques, and evaluation metrics, while the empirical study investigates the below research questions:

RQ Can CIPC improve the effectiveness of microservice regression testing prioritization?

Moreover, CIPC is a more practical approach than typical artifacts, which is based on history data like RTP methods for microservice regression testing?

4.1. Case Introduction. The following five microservice systems need to be utilized in the empirical study:

(1) m-Ticket: a multi-end ticket system based on SpringBlade is an open source microservice framework, which is available at this link: https://github.com/chillzhuang/SpringBlade. These ticket services can be used in various fields such as transportation, accommodation, tourist attractions, movies, supporting service management, monitoring, and tracing.

(2) z-Shop: this is basically an oriented mall system based on Zheng, which is an open-source framework having availability at the address, https://github.com/shuzheng/zheng. However, the mentioned system provides one-stop management services for

```
    Declaration: multiplePrioritization (CIT, T, TQ).
    Parameters: CIT (in); T, test suite (in); TQ, testing queue(out).
(1) TQ←Queue()
(2) for each tp ∈ T do
(3)    tp.V←{CIT(s), s ∈ tp.S}
(4)    if element_max(tp.V) > element_max(TQ.head.V) then
(5)       if tp.S.count() >TQ.head.S.count() then
(6)          TQ.head ← tp
(7)       end if
(8)    end if
(9) end for
(10) T←T - {TQ.tail}
(11) while T≠∅ do
(12)    tc'←T[0]
(13)    for each tc ∈ T do
(14)       if ds(TQ.S, tp.S)-dc(TQ.tail, tp) > ds(TQ.S, tp'.S)-dc(TQ.tail, tp') then
(15)          tc'←tc
(16)       end if
(17)    end for
(18)    TQ.tail←tc'
(19)    T←T - {TQ.tail}
(20) end while
```

Algorithm 4: Multiobjective prioritization algorithm.

goods, stores, content promotion, orders, and logistics.

(3) Need: knowledge graph system needs to work on spring cloud, which provides functions like data collection, auxiliary analysis, information extraction, intelligent query, and other life cycle management services.

(4) KAS: this system checks data collection, processing services for domain literatures, importing, persistence, analysis, and also duplicate checking.

(5) JOA: the OA system primarily forms spring cloud, which comes up with functions like comprehensive information display, document circulation, process approval, plan management, organization personnel control, and fund services for the organization with multiple departments and secret levels.

Table 1 describes the number of logs, services, test cases, and faults of all mentioned cases. Faults of respective versions are collected from corresponding test reports. The a posteriori method should be utilized to adopt the experiment setup, where execution results and testing time are the main activities to prioritize test cases, while Table 1 represents m-ticket, z-shop, need, KAS, and JOA, which are relatively uncomplicated. The design documents, logs, and testing data of each version are collected, which meet the prerequisites of artifacts based on RTP techniques. In addition, JOA have relatively huge number of services, which involve data access permission and joint development of multiple teams. The artifacts directly based on RTP method cannot be applied on JOA.

4.2. Techniques for Comparison. CIPC is compared with typical RTP methods, which consider technical support for three different groups of artifacts, while the artifacts are based on RTP and on control, respectively.

(1) WS-BPEL analysis (WA) [12]: this method relies on flow graph models, which are based on Web Service for Business Process Execution Language (WS-BPEL) [39]. The model's construction has WS-BPEL dependence analysis to retrieve information between services to impact analysis. This approach computes the weights, where modification-affected services prioritize test paths accordingly having more alteration in accordance with highest weight.

(2) Accumulation of failures with weights from statistical analysis and correlation data (AF) [40]: this technique prioritizes test paths based on the weighted graphs from the statistical analysis of failure history and the correlation data among test paths. In the integrated environment, all the data need to be maintained.

(3) Service coverage (SC): these methods prioritize test paths, which has a total number of services. Therefore, multiple test paths cover the similar number of services.

(4) Changed service coverage (CC): the technique normally prioritizes test paths, which are based on the total number of services. These resources directly need to be covered. However, multiple test paths cover the same number of changed services, which are ordered randomly.

Table 1: Properties of each case.

Subject	No. of services	Logs	No. of versions	No. of test cases	No. of faults
m-Ticket	61	140,000	5	1182	539
z-Shop	43	520,000	4	913	427
Need	169	2,000,000	4	847	781
KAS	207	16,000,000	6	4558	1096
JOA	605	50,000,000	9	13356	4783

In RTP methods, the SC and CC integrate to control the benchmark techniques. Empirical study will focus on comparing hermitic search of CIPC, which must adopt each case like CIPC-CP and CIPC-HS, respectively. The same computing resources are utilized for all RTP techniques to ensure the fairness of experimentation.

4.3. Evaluation Metrics. RTP evaluation metrics are as follows [5, 8].

(1) Average Percentage of fault detection (APFD): this technique needs to measure the efficiency of testing sequence T. Let n be the size of T and let m represent the size of fault set F. TF_i describes the order value of test case, by which the i-th fault is found. Therefore, APFD is computed in equation (9). Although the APFD higher value easily detects faults, however, the more efficient prioritization test cases are as follows:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} \text{TF}_i}{(\text{mn})} + \frac{1}{(2n)}. \tag{9}$$

(2) Prioritization time costs (PTC): this approach measures the time costs of RTP technique to make use of prioritization process. Normally, PTC and APFD are both considered as NA. Since the intention of RTP is to reduce testing time costs, which contain prioritization time costs. However, RTP technique more practically spends less time cost.

5. Results and Discussion

5.1. Data and Analysis. The data of the experimentation is presented in Table 2. While APFD values range from 0.70 to 0.93 for CIPC-CP with having the mean value of 0.82, 0.83 up to 0.98, however, CIPC-HS mean value is formulated as 0.89, where CIPC achieves high fault detection rate in microservice regression testing. By comparing RTP techniques, where boxplots are drawn to represent the value distribution of APFD, therefore, Figure 4 apparently has smaller values of SC and CC in comparison with WA, AF, and CIPC. This means that RTP control methods are very less efficient in identifying faults. Coverage relationships are considered in between test paths and services to control RTP techniques. However, the artifacts and history based RTP methods are used to retrieve more information, which controls faults and change in impact. Information with the high probability of detecting faults need to be found by improving effectiveness of prioritization. In addition, RTP methods significantly improve efficiency of microservice regression testing.

Meanwhile, APFD, WA, AF, and CIPC values are similar, which means a direct change in impact propagation computing. However, regression testing prioritization must achieve high quality of effectiveness to detect faults, while the box height of CIPC-HS is much less than other contemporary techniques. Moreover, the fluctuation of APFD values of CIP-HS contains lower value in comparison with other methods. CIPC-HS try to adopt multiple objectives, which prioritize test paths to achieve adaptability.

PTC values are given properly in Table 2. Although PTC must have values of WA, AF, which is apparently much larger than SC, CC, and CIPC, the former are almost two orders of magnitude higher than the others. Therefore, results present WA, AF, where costs are higher than those of SC, CC, and CIPS.

The control flow graph models are used to process, establish, analyze, and maintain the time cost for WA. However, AF also analyzes the relationships between fault history and test paths. The artifacts and history data seriously affect the processing performance. On the other hand, SC, CC, and CIPC need to prioritize time cost, which mainly covers the related queries. These services are used to take very small amount of time in every RTP technique. The artifacts, API gateway logs are compared with commonly structural approach to automate the CIPC. Therefore, data relationship cannot maintain and balance the whole process. Fault detection rate, SC, and CC are having vulnerabilities during implementation. In addition, achieving the expected fault detection rate, CIPC saves much more time in comparison with WA and AF after analyzing the different trends of prioritization time cost, which easily scales up the system. Different RTP methods represent various results, which are shown in Figure 5. The horizontal axis directly arranges 5 cases according to the number of services from small to large value, while the vertical axis describes the mean value of PTC in every case. The numerical diversities of PTC methods having different values in the mentioned metrics like SC, CC, CIPC, WA, and AF. Two-line charts having various value ranges from vertical axis are drawn to facilitate the process. In Figure 5, the trends of prioritization time costs relatively need to increase with the passage of time. The proposed method increases the system scales, which is a very tough process. The JOA approach is having multiple teams, which must be engaged to obtain the related data of experimentation. The RTP techniques like WA and AF will never be applied to enhance the capabilities. However, the API gateway logs easily collect the information from evolution of microservice systems. The performance of CIPC is mainly determined to scale up the logs, which optimize the algorithms and computing resources, while the CIPC method is

Table 2: APFD and PTC (minutes) of each RTP technique in each case.

Subject & versions		SC		CC		WA		AF		CIPC-CP		CIPC-HS	
		APFD	PTC	APFD	PTC	APFD	PTC	APFD	PTC	APFD	PTC	APFD	PTC
m-Ticket	v2	0.33	0.01	0.36	0.01	0.76	37.29	0.65	15.52	0.70	0.18	0.84	0.18
	v3	0.35	0.01	0.41	0.01	0.78	30.71	0.67	15.21	0.78	0.20	0.85	0.20
	v4	0.38	0.01	0.35	0.01	0.76	31.25	0.68	15.10	0.75	0.22	0.88	0.22
	v5	0.46	0.01	0.52	0.01	0.90	27.63	0.74	15.89	0.75	0.25	0.89	0.25
z-Shop	v2	0.37	0.01	0.30	0.01	0.73	20.22	0.62	12.35	0.72	0.11	0.83	0.11
	v3	0.42	0.01	0.35	0.01	0.79	17.82	0.65	12.50	0.75	0.14	0.87	0.14
	v4	0.51	0.01	0.56	0.01	0.89	19.03	0.84	12.63	0.88	0.15	0.89	0.15
Need	v2	0.19	0.02	0.22	0.02	0.78	97.22	0.73	24.32	0.70	0.30	0.83	0.30
	v3	0.45	0.02	0.43	0.02	0.88	79.32	0.82	25.84	0.86	0.38	0.85	0.38
	v4	0.42	0.03	0.31	0.03	0.92	65.24	0.85	26.20	0.87	0.42	0.90	0.42
KAS	v2	0.43	0.04	0.34	0.04	0.75	124.10	0.75	40.21	0.74	0.35	0.83	0.35
	v3	0.25	0.04	0.31	0.04	0.89	109.75	0.83	42.50	0.81	0.49	0.87	0.49
	v4	0.26	0.04	0.29	0.04	0.90	85.68	0.69	45.81	0.90	0.60	0.91	0.60
	v5	0.54	0.05	0.48	0.05	0.94	80.10	0.67	46.05	0.90	0.65	0.95	0.65
	v6	0.25	0.05	0.53	0.05	0.92	74.95	0.80	46.90	0.93	0.84	0.95	0.84
JOA	v2	0.28	0.09	0.30	0.09	*NA*	*NA*	0.70	95.30	0.75	0.62	0.85	0.62
	v3	0.44	0.09	0.47	0.09	*NA*	*NA*	0.68	92.25	0.78	0.65	0.83	0.65
	v4	0.45	0.11	0.47	0.11	*NA*	*NA*	0.85	96.58	0.79	0.85	0.90	0.85
	v5	0.53	0.11	0.53	0.11	*NA*	*NA*	0.83	90.14	0.84	0.92	0.90	0.93
	v6	0.54	0.12	0.58	0.12	*NA*	*NA*	0.84	90.95	0.90	1.25	0.93	1.26
	v7	0.24	0.12	0.30	0.12	*NA*	*NA*	0.76	95.61	0.92	1.40	0.96	1.42
	v8	0.48	0.13	0.32	0.13	*NA*	*NA*	0.82	96.20	0.92	1.45	0.96	1.46
	v9	0.29	0.13	0.40	0.13	*NA*	*NA*	0.66	95.45	0.92	1.57	0.98	1.59

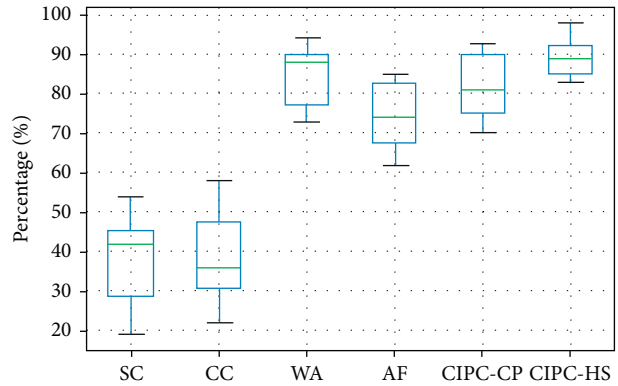

Figure 4: The boxplot of the APFD values of each RTP technique.

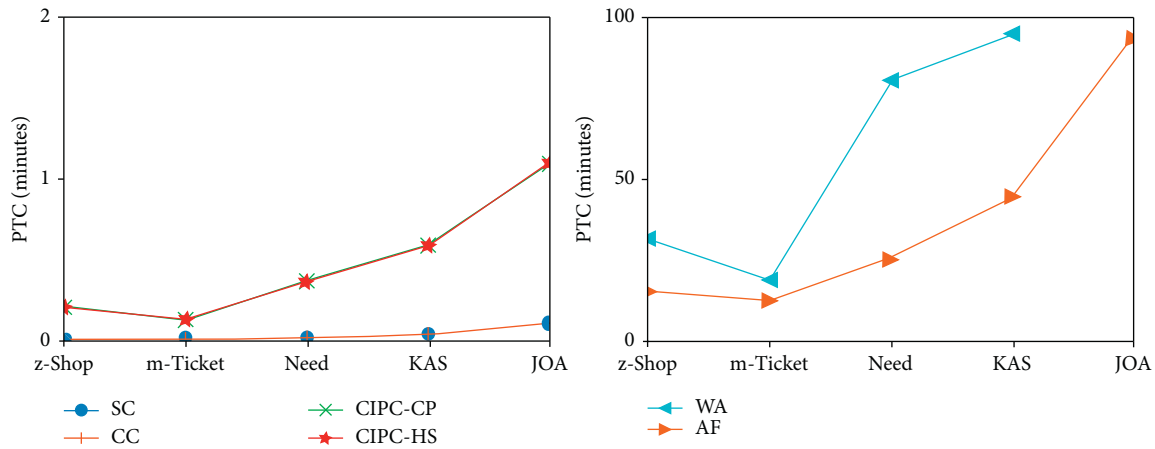

Figure 5: The changing trends of prioritization time costs with the system scales.

more practical than artifacts and history processes, which are based on RTP techniques like WA and AF. This approach uses to enhance the microservice regression testing.

The CIPC improve the effectiveness of microservice in regression testing. In addition, the above-mentioned challenges need to be solved with acceptable time cost.

5.2. Threats to Validity. In the empirical studies, there are some risks, which need to be tested on different experimentations, which mainly validate the two main aspects:

(1) Cases selection: during the practical experimentation factors like size, complexity will have limitations, which do not cover all types of microservice systems, while, at the same time, test case generation will deal with fault records. But the behavior will be comprehensive and will not affect the results. Therefore, different areas need to validate the scales and distribution characteristics.

(2) Comparison RTP techniques selection: RTP techniques are compared with the existing work. Therefore, validity risks must be introduced in the experimentation. However, WA and AF are typical RTP methods. In future, more contemporary techniques should be compared with the proposed solution.

6. Conclusions and Future Work

This research study explains microservice regression testing technique, which is referred to as CIPC. The whole process of CIPC is discussed in detail, which verifies the effectiveness of the empirical study. The proposed solution retrieves service dependencies from API gateway layer logs and impact propagation calculation with directed graph. Also, prioritizing test paths needs to be compared in pairs or hermitic search. The process must be fully automatic and applicable for microservice system regression testing. In future, the two aspects, which include granularity of CIPC and service dependencies from API gateway logs, must give insight overview to improve accordingly. Also, CIPC is used in different fields and patterns like mesh service to collect more cases for empirical study.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no competing interest.

References

[1] C. Y. Fan and S. P. Ma, "Migrating monolithic mobile application to microservice architecture: An experiment report," in *Proceedings of the 2017 IEEE International Conference On AI & Mobile Services (AIMS)*, pp. 109–112, Honolulu, HI, USA, June 2017.

[2] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," 2014, http://martinfowler.com/articles/microservices.html.

[3] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Software*, vol. 35, no. 3, pp. 96–100, 2018.

[4] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, Sebastopol, CF, USA, 2015.

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[6] C. Gao, W. Zheng, Y. Deng, and D. Lo, "Emerging app issue identification from user feedback: experience on WeChat," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019*, Montreal, QC, Canada, May, 2019.

[7] P. Roberto, "On the testing resource allocation problem: Research trends and perspectives," *Journal of Systems and Software*, vol. 161, 2020.

[8] Q. Dong, B. Li, S. Ji, and H. LEUNG, "Regression Testing of web service: a systematic mapping study," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–46, 2014.

[9] Z. Qian, "Test case Generation and Optimization for user session-based web application testing," *Journal of Computers*, vol. 5, no. 11, pp. 1655–1662, 2010.

[10] J. S. Yedidia, W. T. Freeman, and Y. Weiss, *Understanding belief Propagation and its Generalizations*, Morgan Kaufmann, Burlington, MA, USA, 2002.

[11] G. Canfora and M. Di Penta, "Service-oriented architectures testing: a survey," *Software Engineering*, Springer, Berlin, Germany, pp. 78–105, 2009.

[12] L. Chen, Z. Wang, L. Xu, H. Lu, and B. Xu, "Test case prioritization for web service regression testing," in *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE'10)*, pp. 173–178, Oxford, UK, August 2010.

[13] G. Meszaros, *xUnit: Test Patterns Refactoring Test Code*, Addison-Wesley, Boston, MA, USA, 2007.

[14] T. Kaczanowski, "Practical unit testing with JUnit and mockito," 2013, https://site.mockito.org/.

[15] Z. J. Li, H. F. Tan, H. H. Liu, J. Zhu, and N. M. Mitsumori, "Business-Process-Driven gray-box SOA testing," *IBM Systems Journal*, vol. 47, no. 3, pp. 457–472, 2008.

[16] T. A. Khan and R. Heckel, "On model-based regression testing of web-services using dependency analysis of visual contracts," in *Proceedings of the 14th International Conference on Fundamental Approaches To Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE'11/ETAPS'11)*, pp. 341–355, Saarbrücken, Germany, March 2011.

[17] B. Li, D. Qiu, H. Leung, and D. Wang, "Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1300–1324, 2012.

[18] H. Liu, Z. Li, J. Zhu, and H. Tan, "Business process regression testing," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, pp. 157–168, Vienna, Austria, September 2007.

[19] M. Azizi and H. D. Graphite, "A greedy graph-based Technique for regression test case prioritization," in *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 245–251, Memphis, TN, USA, October 2018.

[20] E. K. Mece, K. Binjaku, and H. Paci, "The application of machine learning in test case prioritization - a review," *European Journal of Electrical and Computer Engineering*, vol. 4, no. 1, pp. 1–9, 2020.

[21] M. L. M. Shafie and W. M. N. W. Kadir, "Model-based test case prioritization: a systematic literature review," *Journal of Theoretical and Applied Information Technology (JATIT & LLS)*, vol. 96, no. 14, pp. 4548–4573, 2018.

[22] F. Horváth, T. Gergely, Á. Beszédes, D. Tengeri, G. Balogh, and T. Gyimóthy, "Code coverage differences of Java bytecode and source code instrumentation tools," *Software Quality Journal*, vol. 27, no. 1, pp. 79–123, 2019.

[23] A. Mahdian, A. A. Andrews, and O. J. Pilskalns, "Regression testing with UML software designs: a survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 4, pp. 253–286, 2009.

[24] B. Korel, L. H. Tahat, and M. Harman, "Test prioritization using system models," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pp. 559–568, Budapest, Hungary, September 2005.

[25] B. Korel, G. Koutsogiannakis, and L. H. Tahat, "Application of system models in regression test suite prioritization," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, pp. 247–256, Beijing, China, September 2008.

[26] M. Azizi and H. Do, "A collaborative filtering recommender system for test case prioritization in web applications," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing-SAC'18*, pp. 1560–1567, Pau, France, April 2018.

[27] X. Wang, H. Zeng, H. Gao, H. Miao, and W. Lin, "Location-based test case prioritization for software embedded in mobile devices using the law of gravitation," *Mobile Information Systems*, vol. 2019, Article ID 9083956, 14 pages, 2019.

[28] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Software and Systems Modeling*, vol. 18, no. 1, pp. 499–521, 2019.

[29] J. F. S. Ouriques, E. G. Cartaxo, and P. D. L. Machado, "Test case prioritization techniques for model-based testing: a replicated study," *Software Quality Journal*, vol. 26, no. 4, pp. 1451–1482, 2018.

[30] N. Gökçe and M. Eminli, "Model-based test case prioritization using neural network classification," *Computer Science and Engineering: International Journal*, vol. 4, no. 1, pp. 15–25, 2014.

[31] H. Spieker, A. Gotlieb, D. Marijan, and D. Marijan, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12–22, Santa Barbara, CA, USA, July 2017.

[32] X. Zhao, Z. Wang, X. Fan, and Z. Wang, "A clustering-bayesian network based approach for test case prioritization," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference*, Taichung, Taiwan, July 2015.

[33] L. I. Mao, Z. Zhi-Gang, and W. Tao, "Multiuser detection scheme for scma systems based on stability of belief propagation," *Computer Science*, vol. 46, no. 1, pp. 138–142, 2019.

[34] E. Yamazaki, N. Farsad, and A. Goldsmith, "Low noise non-linear equalization using neural networks and belief propagation," ArXiv, 2019.

[35] X. Tan, W. Xu, and Y. Be'Ery, "Improving massive MIMO belief propagation detector with deep neural network," ArXiv, 2018.

[36] B. Shan and Y. Fang, "GPU accelerated parallel Algorithm of sliding-window belief Propagation for LDPC codes," *International Journal of Parallel Programming*, vol. 48, no. 3, pp. 566–579, 2020.

[37] C. Knoll, D. Mehta, T. Chen, and F. Pernkopf, "Fixed points of belief propagation-an analysis via polynomial homotopy continuation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 9, pp. 2124–2136, 2018.

[38] J. A. Prado Lima and S. R. Vergilio, "Test case prioritization in continuous integration environments: a systematic mapping study," vol. 121, Information and Software Technology, 2020.

[39] A. Alves, A. Arkin, and S. Askary, *OASIS Standard, "Web Services Business Process Execution Language Version 2.0"*, http://docs.oasisopen.org/wsbpel/2.0/, 2007.

[40] Y. Cho, J. Kim, and E. Lee, "History-based test case Prioritization for failure information," *IEEE*, vol. 16, pp. 1530–1362, 2016.