Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH **Open Access**

A microservice regression testing selection approach based on belief propagation

Li-zhe Chen, Ji Wu*, Hai-yan Yang and Kui Zhang

Abstract

Regression testing is required to assure the quality of each iteration of microservice systems. Test case selection is one of main techniques to optimize regression testing. Existing techniques mainly involve artifacts acquisition, processing and maintenance, thus hard to apply in microservice regression testing since it is difficult to obtain and process required artifacts from multiple development teams, which is normal in cases of microservice systems. This paper proposes a novel approach, namely MRTS-BP, which takes API gateway logs instead of artifacts as inputs. By mining service dependencies from API gateway logs, MRTS-BP analyzes service change impacts based on a propagation calculation, and selects test cases affected by changes based on impact degree values. To evaluate the effectiveness of MRTS-BP, empirical studies based on four real deployed systems are presented. Retest-all strategy and a regression testing selection approach based on control flow graphs called RTS-CFG are compared with MRTS-BP. The results show that, MRTS-BP can significantly reduce both the number of test cases and overall time cost while maintaining the fault detection capability of selected test suite, and that MRTS-BP can save more time cost than RTS-CFG with the similar safety and precision.

Keywords Microservice, Regression testing selection, Log mining, Belief propagation

Introduction

Microservice architecture is an effective architecture pattern popularly used in developing current cloud applications, for which services are built independently and integrated at run time by using container technique such as docker [1–3]. Microservice architecture well supports frequent business expansion and smooth upgrading by facilitating independent development and deployment of services [4]. Whenever service modification happens, regression testing is required to detect any potential faults introduced in modifications [5]. Common strategy employed in regression testing is to rerun previously used test cases (referred to as original test suite), namely retest-all strategy. The cost of retest-all strategy, however, might be not acceptable in the case of microservice system with a large amount of services deployed and rapid iterations. For example, WeChat, a social microservice system, has tens of thousands of services deployed and takes several months normally to conduct regression testing with retest-all strategy [6]. To reduce testing cost, many techniques such as test case prioritization, test suite minimization and testing selection are proposed and applied [7].

Regression testing selection (RTS) reduces testing cost by selecting a subset, selected test suite, from original test suite to intentionally cover modules which are changed or affected by other changed modules introduced in last iteration [5]. Many researches at first identify targeted modules by change impact analysis and then produce selected test suite [8–11]. Artifacts such as requirement specification, design model and code file are required usually to conduct change impact analysis effectively. For microservice testing, covering scenarios of service invocations is one of main test objectives [4, 8, 11], and thus a

*Correspondence:
Ji Wu
maple_clz@163.com
School of Computer Science and Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China

service invocation chain is usually set as a test case, also called a test path. With such test paths, artifact based RTS extracts service dependencies from artifacts to analyze which invocation chains may be affected by service modifications such that corresponding test paths can be identified and selected. In addition, safety and precision are also considered while evaluating RTS approaches [5]. A RTS approach is safe if it will contain all test cases that can reveal faults in regression testing, and a RTS approach is precise if it will not contain any unnecessary test cases. To assure the safety, RTS approaches normally expand selected test suite, but that would decrease the precision. Usually, we want to improve precision (i.e. to reduce testing cost) while maintain the safety of test selection at an accepted level.

However, challenges might arise when artifact-based RTS approaches applied in microservice regression testing: (1) Artifacts acquisition. When the microservice system under testing is developed by multiple teams, it faces several issues while achieving artifacts. Extra communication is acquired for gathering which results into elevating total cost. Additionally, expense is increased through obedience with numerous security strategies [8]; (2) Artifacts processing. Diverse development approaches (e.g. different modeling methods and coding frameworks) applied [12–14] increase the difficulty to agree on the integrity, comprehensibility, and consistency of artifacts, which will seriously hinder the performance of artifacts processing; (3) Artifacts maintaining. With growths of system scale in uninterrupted environments, maintenance of artifact versions will cost extra. Especially when multiple versions of a service need to run together, the integration of artifacts might bring confuses. Hence, above three main challenges indicate that artifact-based RTS approaches are not suitable for microservice regression testing.

Furthermore, API gateway layer of microservice system logs every API invocation at runtime for quality of service investigation, including requester, responder, time, status code, etc. [4]. Given a large amount of API gateway logs collected, frequent collaboration patterns among services can be mined to indicate service dependencies. Based on service dependencies, change impact analysis can be conducted based on belief propagation [15], which can conquer above challenges, this motivates the approach presented in this paper, microservice regression testing selection technique based on belief propagation (MRTS-BP).

Initially, MRTS-BP generates service dependency matrix (SDM) by mining service dependencies at business level from API gateway layer logs. Secondly, a directed graph of dependencies among services is established from SDM, and is inputted to a change impact propagation algorithm to measure change impacts quantitatively. Thirdly, existent satisfaction, complete satisfaction and *k*-existent satisfaction are adopted for different cases to generate selected test suite. Moreover, to evaluate the effectiveness of MRTS-BP, we conduct empirical studies on four real deployed systems to measure reduction rate of testing cost, recall, precision and F-measure. Retest-all strategy and a typical artifact-based RTS called RTS-CFG [16] are compared with MRTS-BP. The results show that, MRTS-BP can significantly reduce both the number of test cases and overall time cost while maintaining fault detection capability of selected test suite, and that MRTS-BP can save more time cost than RTS-CFG with similar safety and precision.

Main contributions of this paper are: (a) data utilizes to select test cases are extracted from API gateway logs; (b) change impacts are quantitatively calculated through mathematical approach; (c) three selection strategies are proposed to meet practical testing scenarios.

The rest of this paper is structured as follows: Related work section presents related work on microservice testing, regression testing selection and belief propagation. Methodology section presents MRTS-BP in detail, Empirical study and Results and discussion sections present empirical analysis, and Conclusions and future work section discusses conclusions and future work.

Related work

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 [17] and mockito [18]. In order to bypass user interference and rapid assessment, the service-testing is preferred. End-to-end testing focuses on behaviors of entire system. Due to challenges by service autonomy, dynamic binding and access restrictions [8, 11], service testing and end-to-end testing are very different from traditional software, thus leading to many researches and practices [4]. Whereas, test cases in this paper cover both services and end-to-end behaviors of the microservice system under testing.

Furthermore, both procedures at same interval involve multiple services and their invocations. For instance, consumer-driven based evaluation includes consumer, target, and stubbed-services [4]. Therefore, test cases of these procedures are abstracted as test paths [19–22], defined as follows:

Definition 2.1*test path*

Let $<s_i, s_j>$ represent an invocation between two services, a test path is a sequence of $<s_i, s_j>$ and each of elements in the sequence can be a single invocation or a sequence composed of multiple invocations. A test path can be formally defined as a recursive regular expression $tp = <<s_i, s_j > (,<s_i, s_j>)^* > | < (tp,)^+ >$.

In our work, since logs of microservice systems collected present interfaces exposed by services, the granularity of "service" in Definition 2.1 is a service interface.

Regression testing selection

Formal definition of regression testing selection problem is follows as [5]:

Definition 2.2*Regression Testing Selection Problem, RTS issue.*

> ***Given***: The program, P, the modified version of P, P' and a test suite, T.
> ***Problem***: Find a subset of T, T_s, with which to test P'.

Most of existing RTS approaches mainly concentrate on formal presentation of change impact scopes from P to P' and searching which test cases cover these scopes [5]. Due to close relationship between RTS approaches and system architectures, with continuous development of architecture paradigms, various RTS approaches for different architecture patterns are proposed, which can be divided into two categories: independent program oriented RTS and web service oriented RTS.

Early application systems have relatively small set of functionalities, which are mainly in forms of independent programs. RTS researches concentrate on change impact analysis with code files, such as data flow analysis approach, graph traversal approach, firewall approach, etc. Data flow analysis approach extracts information detailing locations of definitions and uses, which is needed by an inter-procedural data flow tester to guide the selection and execution of test cases [23]. Though applied in regression testing for spreadsheet programs [24], such approach is difficult to conducted for codes that does not cover data flows. Graph traversal approach relies on graph models such as control dependency graph [25], program dependency graph [26], system dependency graph [26], control flow graph [27, 28]. This approach usually includes two phases: analysis and selection. In analysis phase, different granularities of graph models are established from P to P' to identify change impact scopes; in selection phase, relationships between such scopes and test cases are established, and then, test cases that relates to change scopes are selected. Integration scopes affected by changes are defined as "firewalls" in firewall approach, which is proposed for module integration testing [29, 30]. Based on firewalls figured out from code files, test cases covering firewalls are selected.

With the wide deployment of web applications, a large number of RTS researches concentrate on the issue of web services regression testing selection. Since web service testing concentrates on service compositions [8, 11], change impact analysis are conducted based on specifications and behavior models of web services. Treating web services testing as black box testing, web service description language (WSDL) based specifications for functions from an end-users point of view, are required as inputs to figure out change scopes to select test cases [31, 32]. In analogy to graph traversal approach, a two-stage RTS based on control flow graphs is proposed for web service regression testing selection [16] (referred to as RTS-CFG in the following). Such approach includes initialization stage and key stage: in initialization stage, artifacts of system under testing are collected and control flow graphs are established to represent service invocation logic; in critical stage, test cases are selected according to dangerous edges of control flow graphs. Considering different granularities of services, various RTS approaches are proposed. For service endpoints, a RTS approach is presented based on path analysis [21]. This approach compares the invocation path changes between service endpoints before and after iteration, and then selects test cases covering such changes. For service interfaces, a RTS approach based on service interface contract analysis is proposed [20]. In this approach, conflicts caused by contract changes are figured out to select test cases covering such conflicts. For services as a whole, a RTS approach is presented based on business process modeling [22]. Such approach requires structured business logic specifications as inputs and follows the graph traversal method. For large scale systems, a RTS approach needs to inject additional codes into services to collect relationship data between JAVA codes and test cases [14], but its implementation depends on Google's infrastructure, which makes it not universal.

Compared with program oriented RTS, web service oriented RTS approach tends to select test cases based on change analysis of artifacts such as specifications and models. When such RTS approaches are applied in microservice regressing testing, artifacts collection, consistency checking, information extraction and modeling are necessary, but require substantial efforts to implement since microservice systems are usually developed by multiple teams and using different techniques. Whereas, MRTS-BP is proposed to replace artifacts processing with extracting service dependencies from API gateway logs based on frequent pattern mining [33].

Frequent pattern mining

General process of frequent pattern mining is: given a frequent threshold, when the frequency of an item set in transaction set exceeds the threshold, the item set is considered as a frequent item set, which can be used to generate association rules [33]. The frequency of an item set in the transaction set is called "support", which is computed as the ratio of the number of transactions containing the item set to the size of the transaction set. Frequent item sets with *k* elements are called *k*-frequent item sets. Non-empty subsets of a frequent item set must be frequent item sets. Therefore, the support of a frequent item set must be less than or equal to that of its non-empty subsets.

There are many types of frequent pattern mining algorithms, such as candidate set based algorithms, tree based algorithms and recursive suffix based algorithms [33], which are customized according to meet practical requirements of mining problems. In our approach, the mining problem is to extract service dependencies as a basis for change impact analysis.

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, the Viterbi algorithm, decoding algorithms of low density parity check (LDPC) and turbo codes. Such methodologies are utilized for different scenarios [15]. Generally, BP algorithm is follows as.

(1) Initialization: setting initial value of each node.
(2) Propagation: update all message values and node confidence values.
(3) Determining whether node confidence values are convergent. Incase convergent, inference results obtained according to confidence values. Otherwise, it 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 sparse code multiple access, dynamic edge assortment procedure based on BP algorithm is introduced. Through iterative calculation, range boundaries of nodes are detected [34]. However, nonlinear equalization method utilized neural network where BP algorithm is applied to remove signal noises [35]. Additionally, for massive multiple-input multiple-output channel detection, BP is used purely based on deep neural network [36]. In optimization aspect, investigators primarily focus on implementation and convergence condition. Furthermore, LDPC along computational process assists in parallelization and merging memory access [37]. Beside this, convergence problem of BP algorithm and numerical polynomial-homotopy-continuation method revealed influence of structures. Therefore, parameters of graph models solved through fixed points [38].

Literature study reveals BP algorithm is not yet applied in microservice regression testing selection. Proposed work acquires to analyze impacts based on service dependencies from API gateway logs. Additionally, when service dependencies are transformed to a directed graph, impact analysis can be translated into impact propagation from some nodes to others, which can be addressed by BP-like methods.

Methodology

MRTS-BP resolves issue as given in Definition 2.2 for microservice systems. This problem is tackled through three steps: service dependency mining, change impact analysis and test case selection as displayed in Fig. 1. Furthermore, Inputs primarily comprise API gateway logs and original test path set. While, the output is selected test path set to be re-tested.

Service dependency mining

Microservice systems provide user accessible functions through service cooperating, leading to data exchanges and service invocations, called service dependencies [4]. API gateway logs record requests among services. One can see business flow and data flow triggered by users. Thus, our approach mines service dependencies from API gateway logs to generates service dependency matrix (SDM). This step mainly contains two activities: data preprocessing and service dependency matrix generation. The former establishes user request chains from logs and generates a transaction set, while the latter generate a SDM from the transaction set.

Data preprocessing

To facilitate log mining, raw data should be preprocessed to remove irrelevant items and to form into structured data [39]. API gateway logs may contain requester address, service name, service address assigned by load balancer, status code and so on, though its concrete structure varies from system to system. The first step in data preprocessing is to remove irrelevant items such as self-checking records from API gateway logs. Only required data fields such as requester address, service name and service address will be retained to form a structured data set.

Next, with cleaned data, our approach takes user requests as starting point to search for associated service invocations and then formulates into service invocation chains to represent a user session, called as user session extraction. Extraction process is implemented based on another key component of microservice systems called

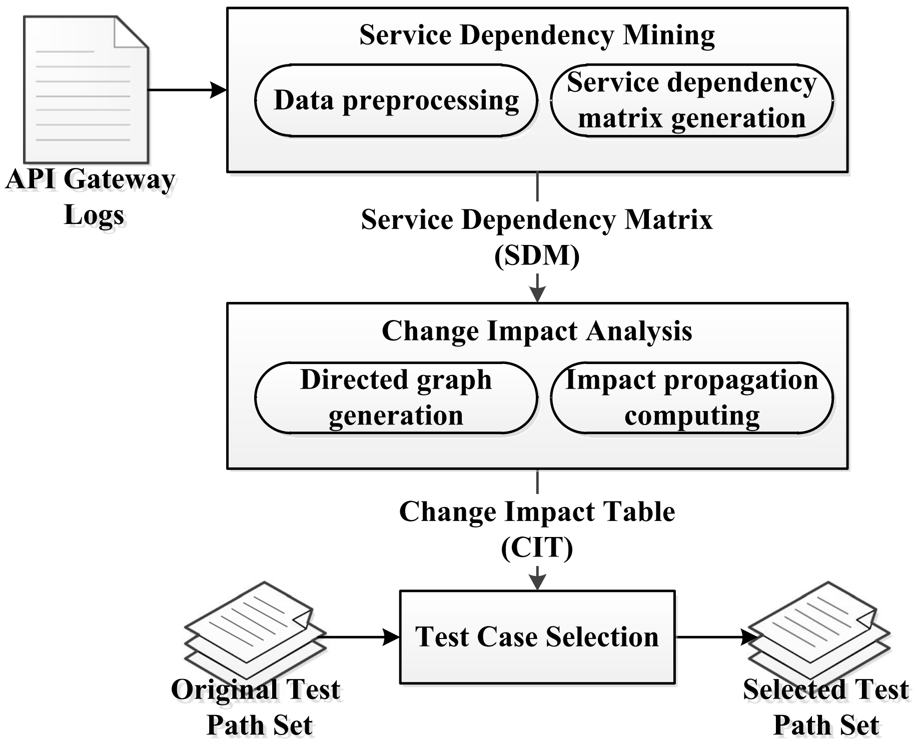
Service Dependency Mining
Data preprocessing
Service dependency matrix generation
API Gateway Logs
Service Dependency Matrix (SDM)
Change Impact Analysis
Directed graph generation
Impact propagation computing
Change Impact Table (CIT)
Test Case Selection
Original Test Path Set
Selected Test Path Set

Fig. 1 Flowchart of MRTS-BP

"service chain monitoring". Service chain monitoring mainly collects, analyzes and displays service invocations while microservice systems running, which supports for fault diagnosis and performance optimization, e.g. Open Zipkin of Twitter, CAT of Dianping.com, and Naver Pinpoint etc. Service invocations associated with the same user request share the same tracing ID. Therefore, a list of service invocations representing a user session can be formed by determining the consistency of tracing ID.

Then, a transaction set for mining is generated from service invocation chains. Considering that service invocations can directly represent dependencies between services, our approach takes an invocation as an item, and takes a service invocation chain as a transaction. Formally, let $S=\{s_j|0<j\leq n\}$ represent the service set of a microservice system and n denote the number of services, then:

Definition 3.1*Global Item Set.*

A global item set $I_a=\{<s_j, s_k>|s_j, s_k \in S \wedge j \neq k\}$, where item $i_{jk}=<s_j, s_k>$ represents s_j invocates s_k.

Definition 3.2*Transaction Set.*

A transaction set $D=\{T|T\subseteq I_a\}$, where T represents a transaction.

According to Definitions 3.1 and 3.2, transaction set generation can be implemented by traversing the service invocation chains once, which is as follows: (1) initialize a transaction set as an empty set; (2) traverse each invocation of each service invocation chain, and denote each invocation as an item. After removing duplicate items, an item set is generated as a transaction and appended to the transaction set; (3) after traversing all service invocation chains, output the transaction set.

Service dependency matrix generation

Service dependencies are mainly derived from two type of sources:

(1) Request flows

Request flows are represented as invocation chains of services, which can be decomposed into invocations between services. If an invocation occurs frequently, it can be inferred that there may be a dependency between corresponding two services, which is defined as "request dependency" in our approach. Since an invocation is defined as an item in Definition 3.1, request dependencies are represented as *1*-frequent item sets.

(2) Data flows

On the one hand, data flows may directly occur with a single invocation between services, which can be also considered as the category of request dependency. On the other hand, data flows may occur indirectly through multiple invocations, two basic cases of which are shown in Fig. 2. In Fig. 2, invocations between *service 1* and *service 3* do not exist. In the left case, *service 2* is invocated through *Req12* and *Req32* respectively. When *Req12* changes some persistent data in *service 2* and *Req32* needs to query such data, *service 3* indirectly exchanges the data with *service 1*. A typical example is data subscription with data decoupling patterns proposed in [4], which includes customer management service (*service 1*), subscription management service (*service 2*) and report service (*service 3*). Customer management service sends new customer data incrementally to subscription management service, and report service queries subscription data from subscription management service, including customer data. In this example, report service does not directly interacts with customer management service, but the former relies on the latter indirectly through the customer data. When customer management service changes (for example, the structure of customer table is changed), such changes may also affect report presentation of report service, which is needed to verify in regression testing. Similarly, in the right case of Fig. 2, *service 2* invocates *service 1* and *service 3* through *Req21* and request *Req23* respectively. When parameters of *Req23* include some data returned by *Req21*, *service 3* may indirectly rely on *service 1* through such data. Service dependency generated by indirect data exchange is defined as "data dependency" in our approach. From the perspective of frequent patterns, data dependencies are expressed as *2*-frequent item sets in the transaction set, and requestors or responders of two items are the same.

Through the analysis above, it is concluded that the request flows may lead to request dependencies, while data flows may lead to both request dependencies and data dependencies. In order to measure the possibility of a service dependency quantitatively, confidence value is defined as follow:

Definition 3.3*Confidence of service dependencies.*

Given frequent threshold *c*, let F_1, F_2 respectively represent the set of *1*-frequent item sets and the set of *2*-frequent item sets in transaction set *D*, *count(I)* represent the number of transactions containing item set *I* in *D*. Then the confidence value of service s_i depending on s_j is given by equations in Formula 1:

$$conf\left(s_i,s_j\right)=\begin{cases}\frac{count\left(\{<s_i,s_j>\}\right)}{|D|}, & \left\{<s_i,s_j>\right\}\in F_1\\ \frac{\left(count\left(<s_i,s_k>,<s_j,s_k>\right)\right)^2}{|D|\times count\left(<s_i,s_k>\right)}, & \left\{<s_i,s_j>\right\}\notin F_1\wedge\left\{<s_i,s_k>,<s_j,s_k>\right\}\in F_2\\ \frac{\left(count\left(<s_k,s_i>,<s_k,s_j>\right)\right)^2}{|D|\times count\left(<s_k,s_i>\right)}, & \left\{<s_i,s_j>\right\}\notin F_1\wedge\left\{<s_k,s_i>,<s_k,s_j>\right\}\in F_2\\ 0, & otherwise\end{cases}\tag{1}$$

Formula 1 divides confidence degree of service dependency into four situations: the first equation computes the confidence of service dependency between two services in the same invocation, corresponding to request dependency, which is measured by the support of *1*-frequent item set; the second and the third equations compute the confidence of service dependency between two services in different invocations that belong to a same data flows, corresponding to data dependency, which are measured by the product of the support of *2*-frequent item sets and the confidence of association rules from corresponding tuples [33] (the second equation corresponds to data dependency in left case of Fig. 2, while

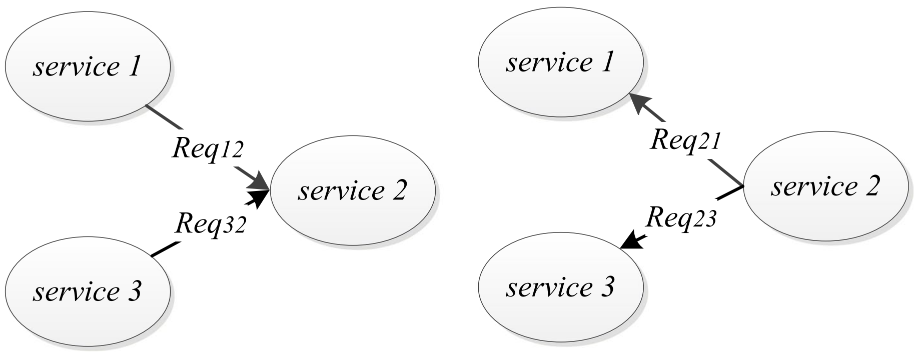

Fig. 2 Two basic cases of indirect data exchange

the third equation corresponds to right one). Except for above situations, it is considered that there is no dependency between services, and the confidence is defined as 0.

For example, in Fig. 2, supposing that *Req12* appears *500* times in logs and the log size is *2000*, given $c = 0.2$, then $<s_1,s_2> \in F_1$ and $conf(s_1,s_2)$ is 0.25 according to the first equation of Formula 1. Supposing that *Req13* does not appear anywhere and *Req12*, *Req32* appear together in the same transaction *400* times in logs, them $\{<s_1,s_2>, <s_3,s_2>\} \in F_2$ and *service 1* may depend on *service 3* via data flow with global probability $conf(s_1,s_3) = 0.25 \times 0.8 = 0.2$ according to the second equation of Formula 1. Meanwhile, *service 3* may also depend on *service 1* via data flow based on the confidence of *Req32.*

It is noted that the support of *2*-frequent item sets is less than or equal to the support of *1*-frequent item sets, and this method does not consider service dependencies whose confidence is less than *c*. Then, the definition of service dependency matrix is as follows:

Definition 3.4*Service Dependency Matrix, SDM*

Given a service set $S = \{s_i | 0 < i \le n\}$, SDM is a *n*-order square matrix, and its element a_{ij} in row *i* and column *j* is defined as follows:

$$a_{ij} = \begin{cases} conf(s_i, s_j), & i \neq j \wedge conf(s_i, s_j) \geq c \\ 0, & otherwise \end{cases} \qquad (2)$$

Based on Definition 3.3 and Definition 3.4, an algorithm for generating SDM from *D* can be proposed as Algorithm 1. Firstly, the algorithm constructs *1*-frequent item sets, and generates candidate sets by Cartesian product, and then removes infrequent item sets by *c* to obtain *2*-frequent item sets (line 1 to 4). Secondly, SDM is initialized as an *n*-order zero square matrix (line 5). Based on the first equation of Formulas 1 and 2, the element of SDM corresponding to *1*-frequent item is set as the support value (line 6 to 8). Thirdly, by traversing *2*-frequent item sets, when tails of two invocations are the same, corresponding element of SDM is updated as the maximal value of its current value and the result computed with the second equation of Formula 1 (line 10 to 18); when heads of two invocations are the same, the third equation of Formula 1 is adopted (line 19 to 28).

```
Declaration: generateSDM(D, c, SDM) .
Parameters: D (in); c (in); SDM (out).
1  F← new Frequent(c)
2  F_1←F.constructOneFrequents(D)
3  F_2←F.constructCandidateFrequents(D,F_1,F_1)
4  F_2←F_2.removeNotFrequents(c)
5  SDM←zeroMatrix(D. Services)
6  for each f={<s_i,s_j>} ∈F_1 do
7      SDM.element(s_i,s_j) ←f.support
8  end for
9  for each f={<s_i,s_j>,<s_k,s_m>} ∈F_2 do
10     if j= m then
11         conf←(f.support)^2/(|D|•<s_i,s_j>.support)
12         if conf ≥ c and conf > SDM.element(s_i,s_k) then
13             SDM.element(s_i,s_k) ←conf
14         end if
15         conf←(f.support)^2/(|D|•<s_k,s_m>.support)
16         if conf ≥ c and conf > SDM.element(s_k,s_i) then
17             SDM.element(s_k,s_i) ←conf
18         end if
19     else if i= k then
20         conf←(f.support)^2/(|D|•<s_i,s_j>.support)
21         if conf ≥ c and conf > SDM.element(s_j,s_m) then
22             SDM.element(s_j,s_m) ←conf
23         end if
24         conf←(f.support)^2/(|D|•<s_k,s_m>.support)
25         if conf ≥ c and conf > SDM.element(s_m,s_j) then
26             SDM.element(s_m,s_j) ←conf
27         end if
28     end if
29 end for
```

Algorithm 1 Service dependence matrix generation algorithm

For example, a SDM of 9 services generated from logs is shown in Table 1. As Table 1 shows, elements on the diagonal of the SDM are all 0, and elements above 0 indicate dependencies where the confidence level exceeds given threshold. According to Algorithm 1, non-zero elements of SDM are related to F_1, F_2 and *c*,

Table 1 An example of SDM

Service	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
s_1	0	0	0	0	0.25	0	0	0.20	0
s_2	0.50	0	0	0	0	0	0	0	0
s_3	0	0.22	0	0	0	0	0	0	0
s_4	0	0	0.29	0	0	0	0.45	0	0
s_5	0	0	0	0.30	0	0.32	0	0	0
s_6	0	0	0	0	0	0	0	0	0
s_7	0	0	0	0	0	0	0	0	0
s_8	0	0	0	0	0	0	0	0	0
s_9	0	0	0.26	0	0	0	0	0	0

and therefore to the scale of logs and frequency threshold. Theoretically, the scale of logs is larger, more dependencies among services are covered, and there will be more SDM non-zero elements. Given logs, frequency threshold is smaller, more SDM non-zero elements will be generated. More non-zero elements indicate that SDM is more complete, which can affect the safety of test case selection, and it will be discussed in Empirical study section further.

Change impact analysis

Change impact analysis also including two activities: directed graph generation and impact propagation computing. The former builds a directed graph model from SDM, while the latter measures impacts by an impact propagation algorithm.

Directed graph generation

Directed graph is used to represent impact propagation network of services. Nodes of the graph represent services and directed edges represent propagation paths among services. Since each element of SDM represents the confidence of corresponding service dependency, the weight of each directed edge can be initialized. Therefore, based on Definition 3.4, a directed graph can be defined as follows:

Definition 3.5*Directed Graph, DG*

A directed graph for impact propagation is a tuple $DG=(N,E)$, where node set $N=S$, edge set $E=\{e_{ij}| a_{ji}\in SDM \land a_{ji}>0\}$, e_{ij} represents a directed edge from s_i to s_j, and $w(e_{ij})=a_{ji}$ represents the weight of e_{ij}.

Based on this definition, an algorithm of directed graphs generation from SDM is shown in Algorithm 2. Node set of DG (line 1) is established based on service set. And then, all elements of SDM are traversed (line 2 to 12) while directed edges are established between nodes with confidences greater than 0. The direction of a directed edge is opposite to the direction of corresponding service dependency (lines 5 to 8). For example, a DG corresponding to Table 1 is shown in Fig. 3. It can be seen that directed edges on DG are inverted with respect to non-zero elements in SDM.

```
Declaration: generateDG(SDM, DG).
Parameters: SDM (in); DG (out).
1   DG.N←SDM.S
2   i←0, j←0
3   while i < SDM.numberOfServices do
4       while j < SDM.numberOfServices do
5           if i ≠ j and SDM.element(s_i,s_j) >0 then
6               e ←DG.E.create(s_j, s_i)
7               e.w←SDM.element(s_i,s_j)
8           end if
9           j←j+1
10      end while
11      i←i+1
12  end while
```

Algorithm 2 Directed graph generation algorithm

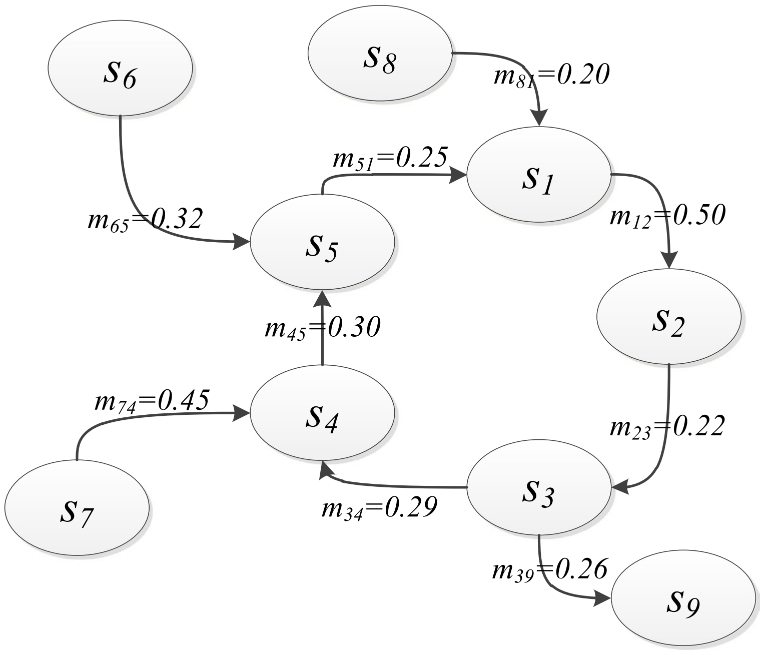

Fig. 3 An example of DG

Impact propagation calculation

Given a service set S and its modified version S', it is easy to obtain a list of changed services with service registries [4], which is denoted as ΔS. Since services affect each other through service dependencies during changing, based on DG, change impact analysis can be translated into a quantitative assessment of impact propagation from some nodes in ΔS to others, which can be addressed by BP-like methods. The difference is that messages in belief propagation are used to calculate the probability of edge distribution of nodes, while messages in impact propagation are used to calculate the probability of nodes being affected by changes. Referring to the framework of BP algorithm, an algorithm of node updating and message propagation is proposed, and its convergence of iterative calculation process is also analyzed.

Node updating Since there is no limit to the sum of change impacts on all service nodes, standard BP algorithm can not be applied. Our approach defines "impact degree" to measure change impacts of service nodes. Impact degree of nodes in ΔS is defined as 1, as the upper limit value, and impact degree of nodes not affected by changes is defined as 0, as the lower limit value. When a node acts as the end node of a directed edge, its impact degree may be updated with the message passing from the directed edge. The updated value of this node should be the maximum value of all messages sent to it and its current impact degree. The formal definition is as follows:

Definition 3.6*Impact Degree*

Given a directed graph DG of Definition 3.5, let N_i represent the neighborhood of s_i, m_{ji} represent the message from s_j to s_i, t represent iteration rounds of message propagation, then impact degree $p_t(s_i)\in[0,1]$ of s_i is recursively calculated as follows:

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

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

Apparently, impact degree of a node after message propagation is not less than that before message propagation, that is, $p_{t+1}(s_i) \geq p_t(s_i)$.

Message propagation In a directed graph, through directed edge e_{ij}, impact degree of s_i can be propagated to s_j based on weight $w(e_{ij})$, so the message is defined as follows:

Definition 3.7*Message*

Given a directed graph DG of Definition 3.5, when $e_{ij}\in DG.E$, message m_{ij} propagated from s_i to s_j is:

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

Apparently, since $w(e_{ij})<1$, message value is always less than current impact degree of the sender node, that is, $m_{ij}<p_t(s_i)$.

Convergence analysis Definitions 3.6 and 3.7 show iterative computing process of change impact propagation. When there are no loops in DG, that is, a directed acyclic graph, propagation rounds of each node do not exceed the number of edges contained in the longest path of DG, so calculation process must be convergent. When there are some loops in DG, updating rounds are uncertain. In this case, it can be divided into two situations to analyze, changed services in the loops and not in the loops, respectively discussed as follows:

(i) When there is a node $s\in\Delta S$ in a loop, then $p_0(s)=1$. Because $p_{t+1}(s)\geq p_t(s)$, and $p_t(s)\leq 1$, so $p_t(s)=p_0(s)=1$, that is, impact degree of s will not be updated by message propagation. The directed edge with s as the end node does not work in computing process and can be considered as interrupted. At this time, the loop can be directly disconnected and transformed into a directed acyclic graph, so computing process converges, which can be shown schematically in Fig. 4.

(ii) For any node s of the loop, $s\notin \Delta S$, then $p_0(s)=0$. According to Definition 3.7, messages propagated on the loop is always 0 until the loop receives external input messages. When there are multiple external input messages, because the output of function *Max* only depends on values of parameters, but has nothing to do with the number of parameters, so we can suppose multiple external input messages reach the loop at the same time by aligning iteration rounds. Then computing process of the loop can be divided into three phases:

① Computing all input messages. At this phase, because message propagations in the loop are not considered, it is transformed into a directed acyclic graph, and computing process converges in this phase;

② Computing message propagations in the loop. After external input messages enter the loop at the

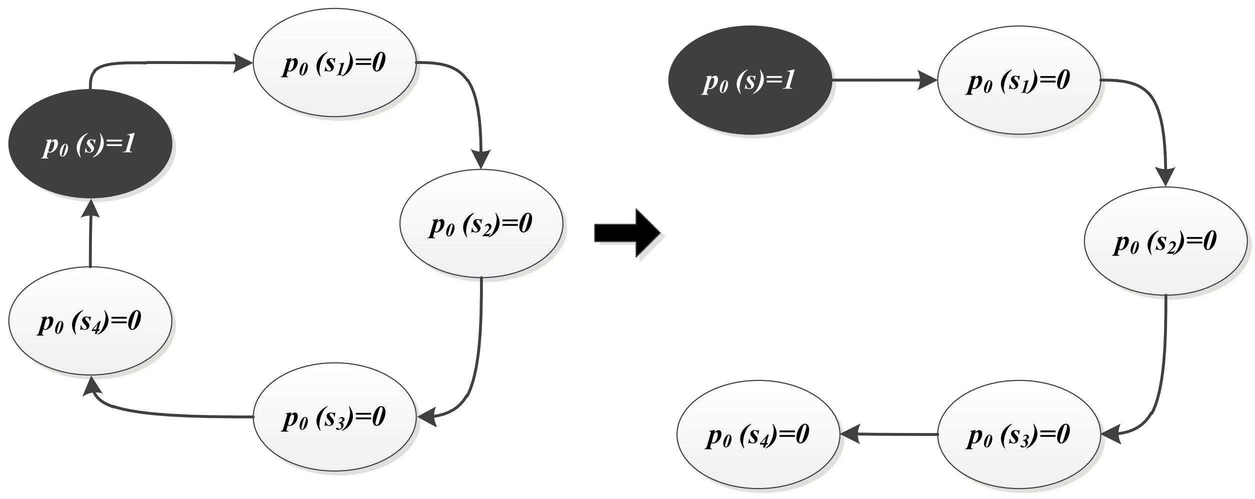

Fig. 4 A node of a loop is changed services

same time, impact degrees of nodes are assigned with Formula 4. Through numerical comparison, node s_m with maximal impact degree can be obtained, which is denoted as $p_t(s_m)$. Since messages propagated in the loop satisfy $m_{ij}<p_t(s_i)$, maximal value of messages in the loop is less than $p_t(s_m)$. Therefore, no matter how many rounds messages propagates in the loop, $p_t(s_m)$ does not changes. That is, the loop can be disconnected from the directed edge with s_m as the end node, and the loop can be removed, which means calculation process converges.

③ Computing all output messages. According to the calculation results of phase 2, output message of each node in the loop is calculated directly with Formula 5.

The schematic diagram of calculation process in case (ii) is shown in the Fig. 5.

To summarize, computing process of impact degrees determined by Definition 3.6 and Definition 3.7 is convergent, that is, impact degree of each node can converge to a stable value in finite iteration rounds. The results of impact propagation computing can be put in a dictionary structure called change impact table (CIT) to access conveniently in our approach. The pseudo codes for generating a CIT from a DG is shown in Algorithm 3. Firstly, impact degrees of all nodes in the DG (line 1 to 3) are initialized according to changed service list, and the CIT (line 4) is also initialized correspondingly. Secondly, iterative computing process (line 5 to 14) is started. Message values (line 6 to 8) are computed with current impact degrees of nodes in each iteration, while impact degrees (line 9 to 13) are updated with the message values. The exit condition of iteration process is that impact degrees of all nodes in the CIT no longer change (line 14). For the DG in Fig. 3, supposing that $p_0(s_6)=1$, $p_0(s_7)=1$ and $p_0(s_8)=1$, 4 iterations of impact propagation will be performed based on Algorithm 3. And the output CIT = {s_1:0.2, s_2:0.1, s_3:0.022, s_4:0.45, s_5:0.32, s_6:1, s_7:1, s_8:1, s_9:0.0057}.

```
Declaration: computeImpactPropagation (DG, S', CIT).
Parameters: D (in); S'(in); CIT(out).
1   for each s ∈DG.N do
2      s. p←1 if s ∈S'∩DG.N else 0
3   end for
4   CIT←Table(DG.N)
5   do
6      for each e ∈DG.E do
7         e.m←e.w •e.start.p
8      end for
9      for each s ∈DG.N do
10        for each e ∈DG.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(DG.N)
```

Algorithm 3 Impact propagation algorithm

Test case selection

This step selects a subset T_s from test path set T based on a CIT. For any test path $tp \in T$, a service set S_{tp} can be derived from the invocations which make up tp with service registries. By querying a CIT, we can obtain impact degree of any service $s \in S_{tp}$, denoted as $CIT(s)$.

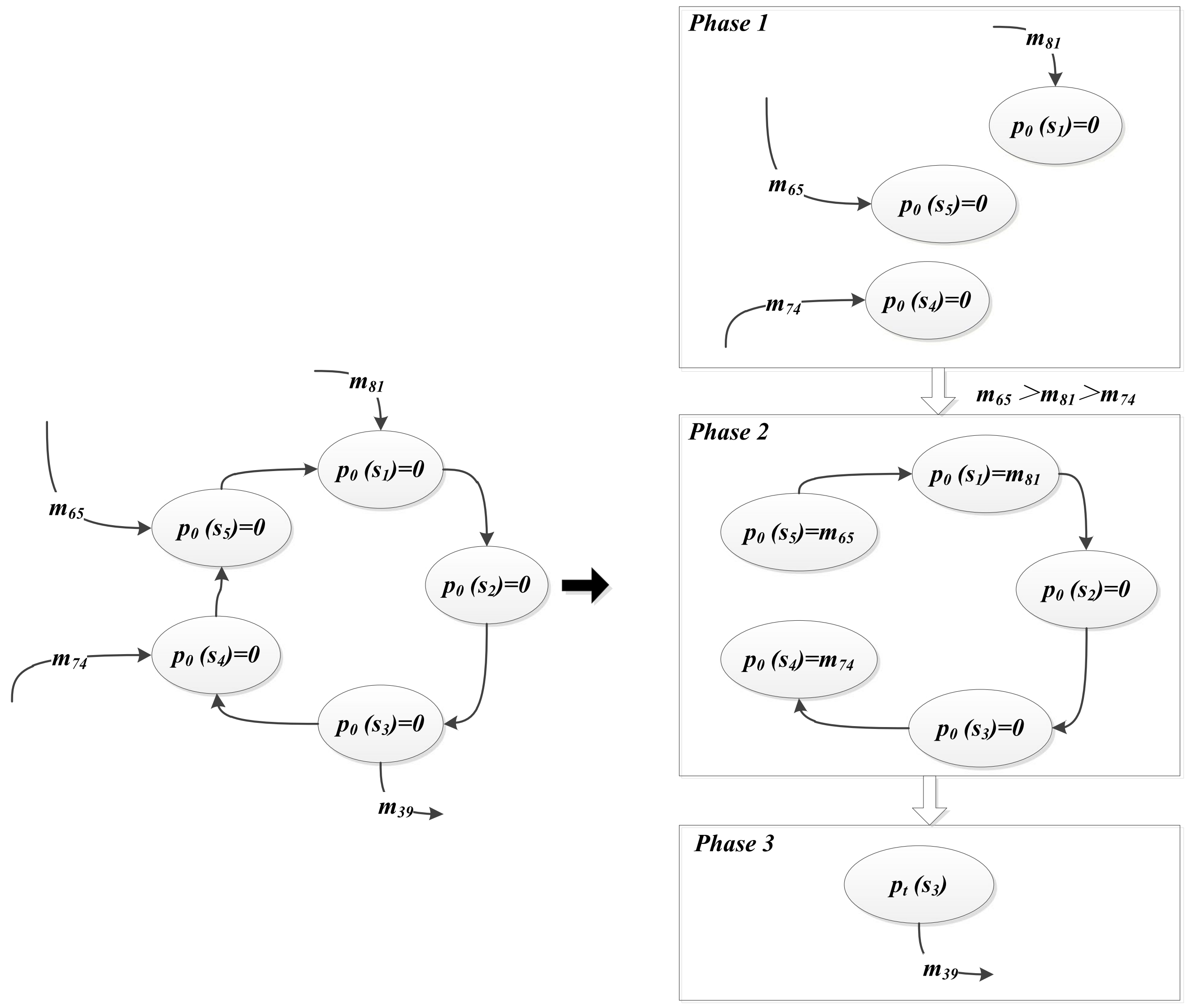

Fig. 5 No nodes of the loop are changed services

Then, to select test paths which are affected by changes, a threshold p can be set to determine whether *CIT(s)* of service s contained in the test path exceeds p. To avoid ignoring test paths affected by changes, p is set the minimum non-zero value of elements in a CIT. To adapt to different cases such as high safety, tight schedule etc., three selection strategies are proposed as follows according to the number of services that meet the conditions:

(1) Existent satisfaction strategy. Given a filtering threshold p, if a test path $tp \in T$ contains services satisfies $CIT(s) \geq p$, then the test path is appended to T_s. That is,

$$T_s = \{tp \mid \exists s \in S_{tp}, CIT(s) \geq p\} \quad (6)$$

(2) Complete satisfaction strategy. Given a filtering threshold p, for a test path $tp \in T$, if every service in S_{tp} satisfies $CIT(s) \geq p$, then the test path is appended to T_s. That is,

$$T_s = \{tp \mid \forall s \in S_{tp}, CIT(s) \geq p\} \quad (7)$$

(3) The k-existent satisfaction strategy. Given a filtering threshold p, for a test path $tp \in T$, if the number of services in S_{tp}, which satisfies $CIT(s) \geq p$, exceeds k, then the test path is appended to T_s. That is,

$$T_s = \left\{ tp \mid \underset{k}{\exists} s \in S_{tp}, CIT(s) \geq p \right\} \quad (8)$$

From above strategies, existent satisfaction strategy is the most relaxed strategy, and the scale of selected test path set is largest. Especially when p is the minimum non-zero value in *CIT(s)*, it means that as long as a test path contains services affected by changes, such test path is selected. The complete satisfaction strategy is the most strict strategy, and the scale of selected test path set is smallest. The k-existent satisfaction strategy is between the two, which can be used to adjust the scale of selected test path set as needed.

Corresponding test case selection algorithms can be proposed. The pseudo codes of existent satisfaction strategy are shown in Algorithm 4, in which original test path set is traversed once (line 2 to 9). By querying a CIT, whether the current test path is selected into the T_S (lines 3 to 8) can be determined. The pseudo codes of the other two strategies are similar and will not be described further.

```
Declaration: testingSelection($T$, $p$, $CIT$, $T_s$).
Parameters: $T$ (in); $p$ (in); $CIT$ (in); $T_s$ (out).
1 $T_s \leftarrow \emptyset$
2 for each $tp \in T$
3     for each $s \in tp.S$
4         if $CIT(s) \geq p$ then
5             $T_s \leftarrow T_s \cup \{tp\}$
6             break
7         end if
8     end for
9 end for
```

Algorithm 4 Test case selection algorithm based on the existent satisfaction strategy

Empirical study

In order to evaluate MRTS-BP and analyze the influence of process parameters on testing selection, we implemented whole process based on Python 3.5, and collected testing data of four microservice systems for experimental analysis. Our empirical study investigates four research questions as follow:

RQ1 Whether MRTS-BP is safe or not, and how the value of the frequent threshold c affects its safety.

An RTS technique is safe if it will contain all test cases revealing faults in regression testing [5]. Safety determines the availability of RTS techniques. In MRTS-BP, the frequent threshold c is directly related to the number of mined frequent patterns, and then affects network structure of directed graph, which has a great influence on the results of change impact propagation. Therefore, the number of test cases selected by MRTS-BP is related to the value of c. It is necessary to analyze the relationship between c and the safety of MRTS-BP, and find out the range of c that can ensure the safety.

RQ2 On the premise of guaranteeing safety, whether MRTS-BP can save testing cost of microservice regression testing.

Compared with retest-all strategy, RTS techniques save testing cost by reducing the number of test cases, but extra time consumption is caused for selecting. When time cost saved are greater than extra time consumption, RTS techniques can save time cost of regression testing overall. Through the experiments, the reduction of the number of test cases and testing time cost are both counted to determine whether MRTS-BP can save testing cost of microservice regression testing.

RQ3 Compare MRTS-BP with a typical artifacts based RTS approach.

Theoretically, MRTS-BP does not rely on artifacts such as specifications, design models and code files. It is completely decoupled from techniques for constructing of the systems under testing, that is, the scalability of MRTS-BP is obviously better than artifacts based RTS approaches. In order to make a more comprehensive comparison, an RTS approach based on control flow analysis (RTS-CFG) [16] is chosen to compare with MRTS-BP to reveal the practicability of the two in microservice regression testing.

RQ4 How to choose selection strategies of MRTS-BP to optimize time cost.

Existent satisfaction strategy, complete satisfaction strategy and k-existent satisfaction strategy are proposed in our approach to meet different testing requirements. To analyze the influence of selection strategies on the efficiency of MRTS-BP, experiments are needed to clarify how the selection strategies affect the number of test cases selected, the safety and the precision of MRTS-BP, which will be helpful to select appropriate strategies in practice.

Case introduction

The following four microservice systems are adopted in our empirical study:

(1) m-Ticket: a multi-end ticket system based on SpringBlade (an open source microservice framework available at https://github.com/chillzhuang/

SpringBlade), provides ticket services in various fields such as transportation, accommodation, tourist attractions and movies, supporting service management, monitoring and tracing.

(2) z-Shop: a mobile oriented mall system based on Zheng (an open source microservice framework available at https://github.com/shuzheng/zheng), provides one-stop management services for goods, stores, content promotion, orders, logistics, etc.
(3) Need: a knowledge system based on Spring Cloud, provides data collection, auxiliary analysis, information extraction, knowledge graph construction, intelligent query and other knowledge graph management services.
(4) JOA: an office automation system based on Spring Cloud, provides comprehensive information display, document circulation, process approval, plan management, organization personnel management, contract management, fund management and material management services for the organization with multiple departments and secret levels.

Table 2 shows the numbers of logs, services, versions, test cases and faults of all cases above. The number of faults is collected from corresponding testing reports where all faults found in testing are reported. Based on these data, a posteriori method is adopted to setup experiments, that is, execution results of test suite are known before, and main activities are to select test cases and to perform statistical analysis. As shown in Table 2, m-Ticket, z-Shop and Need are relatively small systems, but JOA have much more services respectively. To compare with artifacts based approaches, we also collect design documents, logs and testing data. Since multiple teams developed JOA and they did not agree to grant the access permission, we failed to collect corresponding artifacts for JOA. Artifacts based RTS approaches can not be applied in JOA.

Evaluation metrics

According to common RTS evaluation metrics [11–14], considering problems in experiments, 3 metrics are proposed as follows:

(1) Testing time cost saving rate (*ET*) is used to measure the extent to which RTS approaches reduce regression testing time cost. Let *TO* represent total execution time cost of original test suite, *TR* represent execution time cost of selected test suite based on RTS approaches, and *TS* represent the time cost of selection process, then *ET* is given by equation in (9):

$$ET = \frac{TO - TR - TS}{TO} \times 100\% \tag{9}$$

(2) Percentage reduction of the number of test cases (*EN*) is used to measure the ability of RTS techniques save testing cost only in terms of reducing the number of test cases [12, 13]. Let *NO* represent the number of original test cases, *NR* represent the number of selected test suite, then *ET* is given by equation in (10):

$$EN = \frac{NO - NR}{NO} \times 100\% \tag{10}$$

(3) Recall (*R*) indicates the percentage of selected test cases relative to all failed test cases [11–14], which is used to measure the safety of RTS techniques. Let *NOF* represent the number of test cases revealing faults in original test suite, *NSF* represent the number of test cases revealing faults in selected test suite based on RTS techniques, then *R* is given by equation in (11):

$$R = \frac{NSF}{NOF} \times 100\% \tag{11}$$

(4) Precision (*P*) indicates the accuracy with which test cases were selected to be rerun [11–14]. Since *NR* represents the number of selected test suite, then *P* is given by equation in (12):

$$P = \frac{NSF}{NR} \times 100\% \tag{12}$$

(5) F-measure (*F*) is a combination of both *P* and *R* [11–14], which indicates the combination of safety and accuracy of RTS approaches, and *F* is given by equation

Table 2 Properties of each case

Subject	No. of services	No. of logs	No. of versions	No. of test cases	No. of faults
m-Ticket	61	40,000	5	1182	539
z-Shop	43	20,000	4	913	427
Need	169	2,000,000	4	847	781
JOA	605	50,000,000	9	13,356	4783

in Formula 13. It can be seen that the lager F is, the better the combination of safety and accuracy is.

$$F = \frac{2 \times P \times R}{P + R} \tag{13}$$

Experiments setup

According to research problems above, four experiments are setup as follows:

- Experiment 1: determine the safety of MRTS-BP and its relationship with frequent threshold c

  For each case, from version v (v ≥ 2), following steps are carried out.

  (1) Taking logs of version *v-1* as input, global item set and transaction set are generated respectively. After frequencies of all items obtained, the minimum value and maximum value are taken as the lower bound and upper bound of frequent threshold c respectively. Then divide the range of c into ten equal parts, and take the lower bound of each equal part as a value of c, which are denoted as c_i ($i = 1,2...10$). The ten values of c are used respectively to generate SDMs through transaction set mining.
  (2) Based on the ten SDMs generated in step 1, ten CITs are generated respectively. For each CIT, the minimum non-zero element in CIT is taken as selection threshold p, and test cases are selected with existent satisfaction strategy.
  (3) *NOF* is counted from testing report of corresponding version. For each test suite selected, *NSF* and recall R are also counted.

- Experiment 2: compare the ability to save testing cost of MRTS-BP and RTS-CFG
  For each case, from version v ($v \geq 2$), following steps are carried out.

  (1) Find out selected test suite in Experiment 1 with smallest number of test cases and $R = 100\%$ (set *NA* when there is no $R = 100\%$). Collect time cost to select such test suite, including the whole process of MRTS-BP, which is *TS* of MRTS-BP. Then, *TO* and *TS* are computed from testing report of corresponding version, and then *ET* is computed.
  (2) Apply RTS-CFG on each version with available artifacts and tools [16] to select a test suite with $R = 100\%$ (set *NA* when there is no available artifacts or no $R = 100\%$). Collect time cost to select such test suite, including the whole process of RTS-CFG, which is *TS* of RTS-CFG. Similarly, count *TO*, *TS* and compute *ET*.
  (3) For MRTS-BP and RTS-CFG, count the number of test cases selected and compute *EN* respectively.
  (4) Compare *ET*, *EN* of MRTS-BP and RTS-CFG.

- Experiment 3: compare the effectiveness of MRTS-BP and RTS-CFG
  For each case, from version v ($v \geq 2$), following experimental steps are carried out.

  (1) With the results of step 1 in Experiment 2, compute recall R, precision P and F-measure F of MRTS-BP.
  (2) With the results of step 2 in Experiment 2, compute recall R, precision P and F-measure F of RTS-CFG (set *NA* when there is no available artifacts).
  (3) Compare R, P, F of MRTS-BP and RTS-CFG.

- Experiment 4: analyze changing trends of *EN*, R, P and F with different selection strategies of MRTS-BP. For the last version of each case, following steps are carried out.

  (1) Find out the CIT of step 2 in Experiment 1 with smallest number of test cases and $R = 100\%$ (set *NA* when there is no $R = 100\%$). Test cases are selected respectively with complete satisfaction strategy and k-existent satisfaction strategy ($k = 2,3,4$), if the maximal number of services contained in test paths is smaller than four, then the upper of k is set as such maximal number.
  (2) For each test case selected, compute *EN*, R, P and F.
  (3) Compare values of *EN*, R, P and F of different strategies by bar charts.

Results and discussion

Data and analysis

Data of Experiment 1 are shown in Table 3. Let R_i ($i = 1,2...10$) represent the value of R corresponding to c_i of step 1, it can be seen that R can reach 100% with frequent threshold c assigned an appropriate value in each version of each case. That is because existent satisfaction strategy means that test cases including any services affected by changes will be selected. So it indicates that MRTS-BP can ensure safety when

Table 3 Data of Experiment 1

Subject & versions	***c_i/ R_i***									
	i=1	***i=2***	***i=3***	***i=4***	***i=5***	***i=6***	***i=7***	***i=8***	***i=9***	***i=10***
m-Ticket										
v2	0.005	0.026	**0.048**	0.069	0.091	0.113	0.134	0.156	0.177	0.199
	100%	100%	**100%**	92.3%	91.2%	75.6%	61.3%	50.6%	32.4%	9.35%
v3	0.008	0.030	0.053	0.076	**0.099**	0.122	0.145	0.168	0.191	0.214
	100%	100%	100%	100%	**100%**	78.2%	74.6%	49.1%	27.2%	5.62%
v4	0.003	0.040	0.077	**0.114**	0.151	0.189	0.226	0.263	0.300	0.337
	100%	100%	100%	**100%**	93.6%	75.2%	69.3%	55.1%	38.7%	7.20%
v5	0.003	0.038	0.074	**0.110**	0.145	0.181	0.217	0.252	0.288	0.324
	100%	100%	100%	**100%**	95.5%	73.2%	71.6%	43.3%	29.6%	8.13%
z-Shop										
v2	0.019	0.051	0.084	**0.117**	0.150	0.183	0.216	0.249	0.283	0.315
	100%	100%	100%	**100%**	97.2%	84.2%	55.9%	52.1%	31.6%	14.5%
v3	0.014	0.057	0.101	0.145	**0.189**	0.233	0.277	0.321	0.365	0.409
	100%	100%	100%	100%	**100%**	92.1%	65.3%	57.9%	33.6%	11.2%
v4	0.017	0.060	0.103	**0.146**	0.189	0.232	0.275	0.318	0.361	0.404
	100%	100%	100%	**100%**	93.2%	86.7%	59.2%	49.2%	28.7%	7.52%
Need										
v2	0.001	0.020	0.039	**0.059**	0.078	0.098	0.117	0.136	0.156	0.175
	100%	100%	100%	**100%**	95.6%	91.5%	67.4%	59.2%	29.4%	4.30%
v3	0.001	0.021	0.042	**0.062**	0.083	0.104	0.124	0.145	0.165	0.186
	100%	100%	100%	**100%**	89.2%	85.6%	59.7%	55.3%	26.7%	3.55%
v4	0.002	0.023	0.044	0.065	0.086	**0.108**	0.129	0.150	0.171	0.192
	100%	100%	100%	100%	100%	**100%**	71.3%	64.8%	21.5%	11.7%
JOA										
v2	0.001	0.028	0.056	0.083	**0.111**	0.139	0.166	0.194	0.221	0.249
	100%	100%	100%	100%	**100%**	85.3%	80.6%	56.2%	53.4%	21.4%
v3	0.001	0.030	**0.059**	0.088	0.117	0.147	0.176	0.205	0.234	0.263
	100%	100%	**100%**	97.2%	92.5%	86.2%	79.4%	63.1%	57.1%	19.2%
v4	0.002	0.037	0.072	**0.107**	0.142	0.177	0.212	0.247	0.282	0.317
	100%	100%	100%	**100%**	90.8%	83.6%	77.5%	58.2%	50.6%	24.7%
v5	0.003	0.038	0.074	0.109	**0.145**	0.181	0.216	0.252	0.287	0.323
	100%	100%	100%	100%	**100%**	86.2%	81.5%	49.6%	43.2%	19.8%
v6	0.003	0.042	0.081	**0.120**	0.159	0.198	0.237	0.276	0.315	0.354
	100%	100%	100%	**100%**	93.7%	87.9%	79.6%	53.7%	47.7%	18.9%
v7	0.003	0.040	0.078	0.116	**0.154**	0.192	0.229	0.267	0.305	0.343
	100%	100%	100%	100%	**100%**	95.1%	83.5%	69.2%	46.6%	22.5%
v8	0.019	0.059	0.100	0.141	**0.182**	0.222	0.263	0.304	0.345	0.386
	100%	100%	100%	100%	**100%**	95.6%	89.7%	70.4%	47.5%	20.6%
v9	0.008	0.044	0.081	0.118	0.155	**0.192**	0.229	0.266	0.303	0.340
	100%	100%	100%	100%	100%	**100%**	91.6%	68.3%	56.6%	14.1%

c is set appropriate with existent satisfaction strategy. To show change trend of R with c intuitively, the line chart Fig. 6 is drawn from the last version of each case. From Fig. 6, when c changes from minimum value to maximum value, R will gradually decrease. This is because when c becomes larger, less frequent patterns are mined, that is, less possible service dependencies are obtained, which leads to more zero elements in SDM, and further leads to less directed edges of directed graph. Since less edges of directed graph has, less nodes will be considered in change propagation computing, which may lead to that less services are

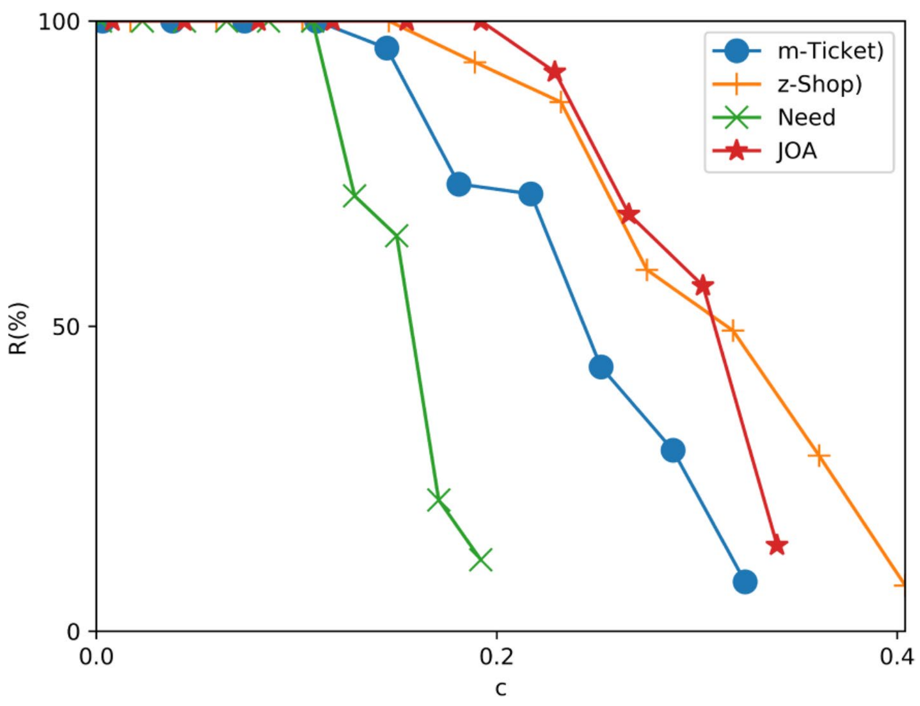

Fig. 6 The change trends of *R* with *c* in Experiment 1

Table 4 Data of Experiment 2

Subject & versions	Original		MRTS-BP					RTS-CFG				
	NO	*TO*	*NR*	*EN*	*TR*	*TS*	*ET*	*NR*	*EN*	*TR*	*TS*	*ET*
m-Ticket												
v2	932	521.24	410	56%	223.55	0.13	57%	437	53%	240.12	190.94	17%
v3	1057	654.43	507	52%	295.59	0.17	55%	540	49%	337.31	82.83	36%
v4	1094	725.98	612	44%	366.99	0.19	49%	625	43%	409.50	34.07	39%
v5	1182	743.11	591	50%	334.72	0.21	55%	631	47%	401.72	13.40	44%
z-Shop												
v2	772	388.40	355	54%	204.37	0.09	47%	370	52%	223.13	124.48	11%
v3	869	533.93	521	40%	258.63	0.10	52%	568	34%	338.81	68.02	24%
v4	913	598.42	456	50%	221.50	0.12	63%	483	47%	320.35	15.71	44%
Need												
v2	737	534.66	316	57%	241.40	4.41	54%	322	56%	243.65	273.64	3%
v3	805	685.31	362	55%	367.82	5.12	46%	394	51%	395.14	102.18	27%
v4	847	734.50	406	52%	389.96	5.53	46%	421	50%	395.08	59.34	38%
JOA												
v2	10,742	7305.56	4726	56%	2996.86	42.34	58%	*NA*	*NA*	*NA*	*NA*	*NA*
v3	11,593	7752.70	6608	43%	3791.62	57.15	50%	*NA*	*NA*	*NA*	*NA*	*NA*
v4	11,989	7917.07	6833	43%	3935.13	63.33	49%	*NA*	*NA*	*NA*	*NA*	*NA*
v5	12,374	8253.80	6434	48%	3494.98	75.22	57%	*NA*	*NA*	*NA*	*NA*	*NA*
v6	12,507	8327.07	7003	44%	4109.38	90.32	49%	*NA*	*NA*	*NA*	*NA*	*NA*
v7	12,887	8569.12	5928	54%	3185.81	99.54	62%	*NA*	*NA*	*NA*	*NA*	*NA*
v8	13,252	9112.34	5963	55%	3204.49	118.21	64%	*NA*	*NA*	*NA*	*NA*	*NA*
v9	13,356	9257.31	6678	50%	3886.37	137.36	57%	*NA*	*NA*	*NA*	*NA*	*NA*

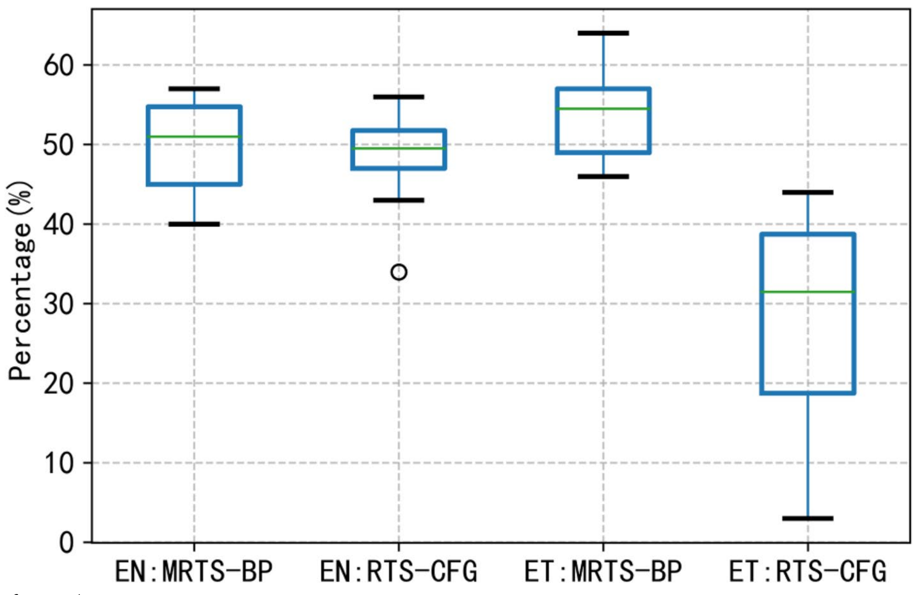

Fig. 7 The boxplot of *EN* and *ET* in Experiment 2

appended to CIT. Then, less test paths will be selected with existent satisfaction strategy, which leads to the smaller value of *R*. Therefore, in order to ensure the safety of MRTS-BP, the value of *c* should be close to its lower bound value.

Data of Experiment 2 are shown in Table 4. From *EN* of MRTS-BP, value ranges from 40% to 57%, and mean value is 50%, that is, the number of test cases is apparently reduced by applying MRTS-BP. Similarly, *EN* of RTS-CFG ranges from 34% to 56% with mean value 48%.

Table 5 Data of Experiment 3

Subject & versions	MRTS-BP			RTS-CFG		
	R	*P*	*F*	*R*	*P*	*F*
m-Ticket						
v2	100%	38.2%	0.55	100%	35.8%	0.53
v3	100%	13.8%	0.24	100%	12.9%	0.23
v4	100%	11.9%	0.21	100%	11.6%	0.21
v5	100%	6.59%	0.12	100%	6.20%	0.12
z-Shop						
v2	100%	37.4%	0.54	100%	35.9%	0.53
v3	100%	12.2%	0.21	100%	11.2%	0.20
v4	100%	6.57%	0.12	100%	6.20%	0.12
Need						
v2	100%	86.7%	0.93	100%	85.1%	0.92
v3	100%	42.2%	0.59	100%	38.8%	0.56
v4	100%	37.9%	0.55	100%	36.5%	0.53
JOA						
v2	100%	40.1%	0.57	*NA*	*NA*	*NA*
v3	100%	5.00%	0.09	*NA*	*NA*	*NA*
v4	100%	1.30%	0.03	*NA*	*NA*	*NA*
v5	100%	0.90%	0.02	*NA*	*NA*	*NA*
v6	100%	3.50%	0.07	*NA*	*NA*	*NA*
v7	100%	1.00%	0.02	*NA*	*NA*	*NA*
v8	100%	0.60%	0.01	*NA*	*NA*	*NA*
v9	100%	0.90%	0.02	*NA*	*NA*	*NA*

It can be seen that, in terms of percentage reduction of test suite scale, MRTS-BP can save testing cost like the typical RTS technique. However, from values of *ET*, testing time cost saving rates of the two approaches are different. Since MRTS-BP has more data than RTS-CFG, boxplots in Fig. 7 are drawn to show the comparison of the two. From Fig. 7, MRTS-BP and RTS-CFG have similar mean values, standard deviation values and extreme values for *EN*. But for *ET*, mean value and extreme values of the former are apparently larger than those of the latter (the difference is at least 20%), that is, MRTS-BP can save more testing time cost than RTS-CFG. This is because RTS-CFG spends large amounts of time on artifacts processing, while MRTS-BP spends much less time on process log mining and compute change propagation. Consistency, comprehensibility, integrity, granularity other characteristics of artifacts seriously affect processing performance of RTS-CFG, while logs are structural and the process of MRTS-BP can be easy to automate. Therefore, MRTS-BP spends much less time than RTS-CFG during the selection phase. To RQ2, on the premise of guaranteeing safety, MRTS-BP can reduce testing cost of regression testing not only in the number reduction of test cases but also in time cost saving.

The Data of Experiment 3 are shown in Table 5. From values of *R*, MRTS-BP and RTS-CFG can ensure all test cases revealing faults are selected for each version of each case, that is, MRTS-BP and RTS-CFG are both safe. To intuitively compare *P* and *F* of the two techniques, line charts are drawn for each case in Fig. 8. From the line charts of case m-Ticket, z-Shop and Need, for *P* and *F*, it is can be seen that values and their change trends of MRTS-BP and RTS-CFG are almost the same. This is due to similar abilities of the two approaches to reduce the number of test cases and cover impact scopes of changes. Essentially, MRTS-BP and RTS-CFG both identify change impact scopes based on service dependencies. The difference lies in that, the former adopts impact propagation calculation while the latter adopts edge analysis based on control flow models. And it also indicates that BP-like algorithm worked in regression testing selection. On the other hand, since artifacts based RTS approaches are not adapted to the case which artifacts are difficult to obtain, such as JOA, the scalability of MRTS-BP is obviously better than that of RTS-CFG in practice.

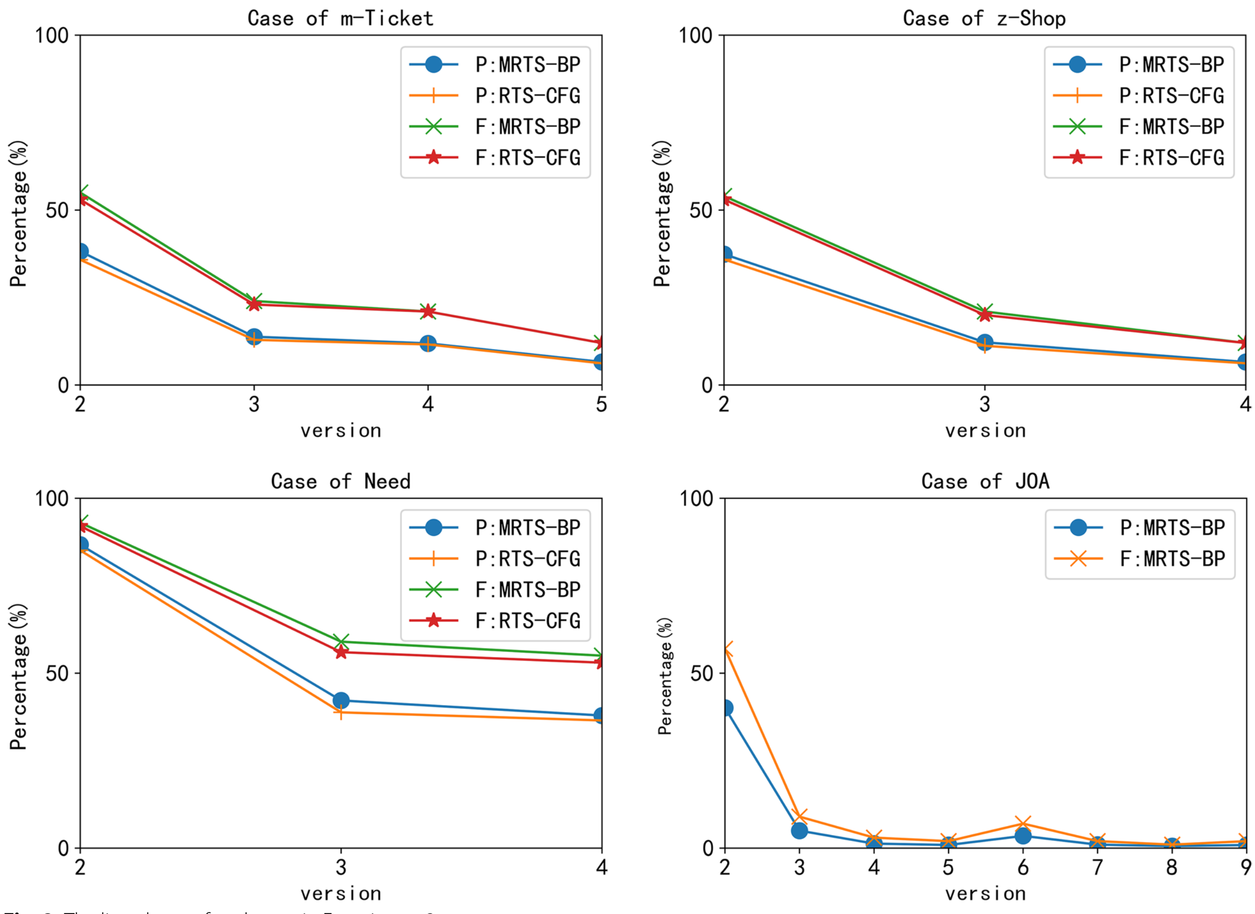

Fig. 8 The line charts of each case in Experiment 3

Table 6 Data of Experiment 4

Subject & metrics	Selection strategy				
	Existent satisfaction	**Complete satisfaction**	**2-existent satisfaction**	**3-existent satisfaction**	**4-existent satisfaction**
m-Ticket					
EN	50%	82%	63%	95%	*NA*
R	100%	52.9%	87.2%	37.5%	*NA*
P	6.59%	9.91%	7.78%	24.7%	*NA*
F	0.12	0.16	0.14	0.30	*NA*
z-Shop					
EN	50%	89%	68%	94%	*NA*
R	100%	66.7%	100%	43.6%	*NA*
P	6.57%	20%	10.3%	23.9%	*NA*
F	0.12	0.31	0.19	0.31	*NA*
Need					
EN	52%	92%	67%	89%	96%
R	100%	43.5%	100%	55.4%	17.3%
P	37.9%	98.1%	55.1%	91.6%	78.6%
F	0.55	0.60	0.71	0.69	0.28
JOA					
EN	50%	96%	58%	66%	98%
R	100%	40.0%	100%	82.3%	36.0%
P	0.9%	4.50%	1.07%	1.09%	8.09%
F	0.02	0.08	0.02	0.02	0.13

The Data of Experiment 4 are shown in Table 6, and corresponding bar charts are shown in Fig. 9. From values of *EN* with different selection strategies in each case, it can be seen that more strict selection strategy is, less test cases are selected, and more testing cost are saved. However, that less test cases are selected means more test cases affected by changes are ignored, which can make MRTS-BP be not safe, as shown by the values of *R* in each cases. That is, *EN* and *R* are a pair of trade-off with different selection strategies, and one should choose strategies according to the actual case. From Fig. 9, existent satisfaction strategy can always ensure the safety of MRTS-BP, but *EN* and *F* with such strategy are not the best, which means MRTS-BP can be applied in the case of high safety. Complete satisfaction strategy save more testing cost than others, but it is far less safe than others, which means such strategy can be applied in the case of tight schedule. The efficiency of *k*-existent satisfaction strategy is determined by the value of *k*. When *k* is larger, the number of selected test cases will be significantly reduced, but the safety will also be worse, or even unavailable. When *k* is smaller, MRTS-BP has better safety, but testing cost reduction rate becomes lower. It is worth noting that when *k* is 2, *R* reaches 100% in three cases, and *EN* and *F* are better compared with those of complete satisfaction strategy. This indicates that *2*-existent satisfaction strategy can be applied and may bring more efficiency.

Threats to validity

As with most empirical studies, there are some risks to apply the conclusions directly of experiments above, and threats to validity mainly are manifested in two aspects:

(1) Cases selection. Although our experiments consider factors such as size, complexity, and domain when selecting cases, there may be some limitations that do not cover all types of microservice systems. At the same time, methods of test cases generation, and whether faults records are comprehensive or not may also affect experimental results. It needs to be validated by more cases in different areas, different scales and different data distribution characteristics.

(2) Comparison RTS techniques selection. The method for comparison, RTS-CFG, comes from related work, which also has validity risks that will be introduced into our experiments. At the same time, though RTS-CFG is a typical RTS technique relying on artifacts, it does not represent all artifacts based RTS techniques and our approach need to be validated against with more different artifacts based RTS techniques.

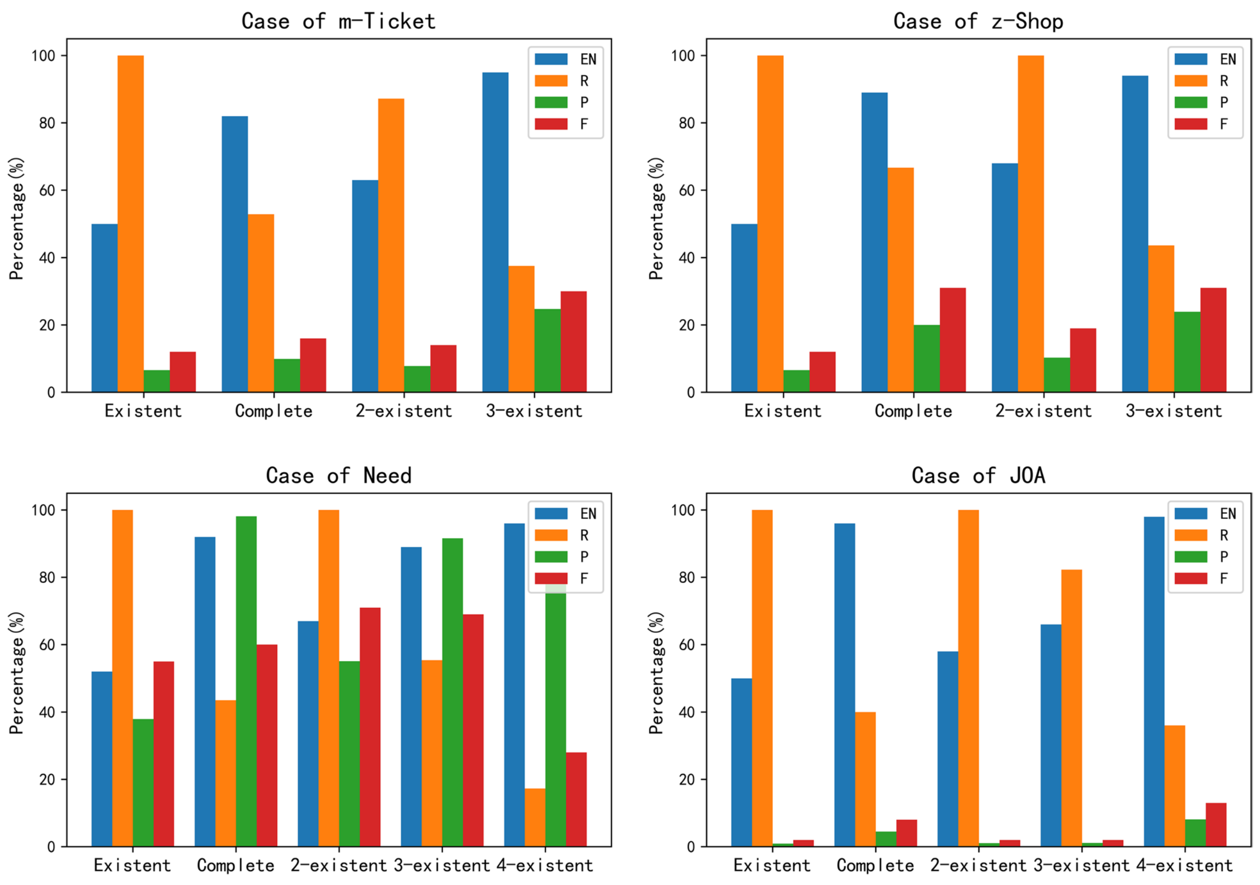

Fig. 9 The bar charts of each case in Experiment 4

Conclusions and future work

This paper proposes a microservice regression testing selection approach MRTS-BP, describes the whole process in detail, and verifies its effectiveness through experiments. MRTS-BP conquer the challenges of artifacts based RTS approaches by processing API gateway logs instead of artifacts. For acquisition issue, API gateway logs are automatically and centrally recorded without business-specific data, which avoids additional communication costs and security risks. For processing issue, API gateway logs are structural and consistent, which leads to processing automatically with MRTS-BP. For maintaining issue, API gateway logs are identified clearly and accurately correspond to each version of services. Thus, MRTS-BP can be fully automated and is applicable to microservice systems regression testing in practice.

In future, two aspects which include granularity of MRTS-BP and service dependencies from API gateway logs must give insight overview to improve accordingly. Also, MRTS-BP in different fields and patterns like mesh service to collect more cases for empirical study.

Acknowledgements
All those who have contributed to this paper have been included in the list of authors.

Authors' contributions
LIZHE CHEN contributed to the conception of the study. JI WU contributed significantly to analysis and manuscript preparation. HAIYAN YANG performed the experiments. KUI ZHANG performed the data analyses and wrote the manuscript. The author(s) read and approved the final manuscript.

Authors' information
LIZHE CHEN received his master degree in computer application technology from National Defense University of Science and Technology, Hefei, China, in 2010. He is currently a Ph.D candidate in the oftware Engineering Institute (SEI) at Beihang University. His research interests include model driven engineering, data mining, machine learning, model base safety analysis, and software testing.
JI WU is associate professor of software engineering at Beihang University. He received his PhD degree from Beihang University in 2003 and MS degree from the Second Research Institute of the China Aerospace Science and Industry Group in 1999. His research interests include embedded system and software modeling and verification, software requirement and architecture modeling and verification, safety and reliability assessment, and software testing.
HAIYAN YANG is lecture of software engineering at Beihang University. She received her master degree in Computer Software and Theory at Beihang University in 2000. Her research interests include software modeling and verification， software testing, software measurement.
KUI ZHANG received his master degree in information management and information system from Beijing Institute of Technology, Beijing, China, in 2010.

He is currently a Ph.D candidate in the oftware Engineering Institute (SEI) at Beihang University. His research interests include model driven engineering, model based real time analysis, airworthiness certification, model base safety analysis, and general model based software engineering.

Funding
This paper is supported by the scholarship project set up in Beihang University.

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

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
All authors approved the final manuscript and the submission to this journal.

Competing interests
We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of.

Received: 17 May 2021 Accepted: 27 January 2023
Published online: 13 February 2023

References
1. Fan CY, Ma SP (2017) Migrating monolithic mobile application to microservice architecture: an experiment report. In: Proceedings of 2017 IEEE International Conference on AI & Mobile Services (AIMS), pp 109–112
2. Lewis J, Fowler M (2014) Microservices: a definition of this new architectural term. http://martinfowler.com/articles/microservices.html
3. Larrucea X, Santamaria I, Colomo-Palacios R, Ebert C (2018) Microservices. IEEE Softw 35(3):96–100
4. Newman S (2015) Building microservices: designing fine-grained systems. O'Reilly Media, Sevastopol
5. Yoo S, Harman M (2012) Regression testing minimization, selection and prioritization: a survey. Softw Test Verif Reliab 22(2):67–120
6. Gao C, Zheng W, Deng Y, Lo D (2019) 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 25-31
7. Roberto P (2020) On the testing resource allocation problem: research trends and perspectives. J Syst Softw 161:110462
8. Qiu D, Li B, Ji S, Leung H (2014) Regression testing of web service: a systematic mapping study. ACM Comput Surv 47(2):1–46
9. Kazmi R, Jawawi DNA, Mohamad R, Ghani I (2017) Effective regression test case selection: a systematic literature review. ACM Comput Surv 50(2):29
10. Gligoric M, Eloussi L, Marinov D (2015) Practical regression test selection with dynamic file dependencies, ISSTA. ACM, New York, pp 211–222
11. Ali S, Hafeez Y, Hussain S, Yang S (2019) Enhanced regression testing technique for agile software development and continuous integration strategies. Softw Qual Control. https://doi.org/10.1007/s11219-019-09463-4
12. Rothermel G, Harrold MJ, Dedhia J (2015) Regression test selection for C++ software. Softw Test Verif Reliab 10(2):77–109
13. Spoon SA, Jones JA, Li T (2001) Regression test selection for Java software. ACM Sigplan Not 36(11):312–326
14. Zhong H, Zhang L, Khurshid S (2019) TestSage: regression test selection for large-scale web service testing. In: 12th IEEE Conference on Software Testing, Validation and Verification (ICST), pp 430–441. https://doi.org/10.1109/ICST.2019.00052
15. Yedidia JS, Freeman WT, Weiss Y (2002) Understanding belief propagation and its generalizations. Morgan Kaufmann, San Mateo
16. Ruth M, Oh S, Loup A (2007) Towards automatic regression test selection for web services. In: Computer software and applications conference. COMPSAC, New York
17. Meszaros G (2007) xUnit: test patterns refactoring test code. Addison-Wesley, Boston
18. Kaczanowski T. Practical unit testing with JUnit and Mockito. https://site.mockito.org/, 2013
19. Li ZJ, Tan HF, Liu HH, Zhu J, Mitsumori NM (2008) Business-process-driven gray-box SOA testing. IBM Syst 47(3):457–472
20. Khan TA, Heckel R (2011) 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
21. Li B, Qiu D, Leung H, Wang D (2012) Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. Syst Softw 85(6):1300–1324
22. Liu H, Li Z, Zhu J, Tan H (2007) Business process regression testing. In: Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07), pp 157–168
23. Harrold MJ, Soffa ML (1989) Interprocedural data flow testing. In: Proceedings of the Symposium on Software Testing, Analysis, and Verification, pp 158–167
24. Fisher M, Jin D, Rothermel G (2002) Test reuse in the spreadsheet paradigm. In: Proceedings of the International Symposium on Software Reliability Engineering, pp 257–268
25. Rothermel G, Harrold MJ (1993) A safe, efficient algorithm for regression test selection. In: Proceedings of International Conference on Software Maintenance, pp 358–367
26. Rothermel G, Harrold MJ (1994) Selecting tests and identifying test coverage requirements for modified software. In: Proceedings of International Symposium on Software Testing and Analysis, pp 169–184
27. Rothermel G, Harrold MJ (1997) A safe, efficient regression test selection technique. ACM Trans Softw Eng Methodol 6(2):173–210
28. Beydeda S, Gruhn V (2001) Integrating white-and black-box techniques for class-level regression testing. In: Proceedings of the International Computer Software and Applications Conference, pp 357–362
29. White L, Robinson B (2004) Industrial real-time regression testing and analysis using firewalls. In: Proceedings of the International Conference on Software Maintenance, pp 18–27
30. White L, Jaber K, Robinson B (2008) Extended firewall for regression testing: an experience report. J Softw Maint Evol 20(6):419–433
31. Paul R, Yu L, Tsai WT, Bai X (2001) Scenario-based functional regression testing. In: Proceedings of the International Computer Software and Applications Conference, (COMPSAC 2001), pp 496–501
32. Tarhini A, Fouchal H, Mansour N (2006) Regression testing web services-based applications. In: Proceedings of the IEEE International Conference on Computer Systems and Applications, (COMPSAC 2006), pp 163–170
33. Aggarwal CC, Han J (2014) Frequent pattern mining. Springer International Publishing, Switzerland, pp 19–36
34. Mao LI, Zhi-Gang Z, Tao W (2019) Multiuser detection scheme for SCMA systems based on stability of belief propagation. Computer Science, Beijing
35. Yamazaki E, Farsad N, Goldsmith A (2019) Low noise non-linear equalization using neural networks and belief propagation
36. Tan X, Xu W, Be'Ery Y (2018) Improving massive MIMO belief propagation detector with deep neural network
37. Shan B, Fang Y (2020) GPU accelerated parallel algorithm of sliding-window belief propagation for LDPC codes. Int J Parallel Prog 48(3):566–579
38. Knoll C, Mehta D, Chen T (2018) Fixed points of belief propagation -- an analysis via polynomial homotopy continuation. IEEE Trans Pattern Anal Mach Intell 9:2124–2136
39. Lu Z et al (2003) Web Log Mining. Web Intelligence. Springer Berlin, Heidelberg, pp 173–194