Formulating Criticality-Based Cost-Effective Fault Tolerance Strategies for Multi-Tenant Service-Based Systems

Yanchun Wang, Qiang He, Dayong Ye, and Yun Yang

Abstract—The proliferation of cloud computing has fueled the rapid growth of multi-tenant service-based systems (SBSs), which serve multiple tenants simultaneously by composing existing services in the form of business processes. In a distributed and volatile operating environment, runtime anomalies may occur to the component services of an SBS and cause end-to-end quality violations. Engineering multi-tenant SBSs that can quickly handle runtime anomalies cost effectively has become a significant challenge. Different approaches have been proposed to formulate fault tolerance strategies for engineering SBSs. However, none of the existing approaches has sufficiently considered the service criticality based on multi-tenancy where multiple tenants share the same SBS instance with different multi-dimensional quality preferences. In this paper, we propose Criticality-based Fault Tolerance for Multi-Tenant SBSs (CFT4MTS), a novel approach that formulates cost-effective fault tolerance strategies for multi-tenant SBSs by providing redundancy for the critical component services. First, the criticality of each component service is evaluated based on its multi-dimensional quality and multiple tenants sharing the component service with differentiated quality preferences. Then, the fault tolerance problem is modelled as an Integer Programming problem to identify the optimal fault tolerance strategy. The experimental results show that, compared with three existing representative approaches, CFT4MTS can alleviate degradation in the quality of multi-tenant SBSs in a much more effective and efficient way.

Index Terms—Cloud computing, criticality, fault tolerance, multi-tenancy, redundancy, service-based system

1 Introduction

The service-oriented paradigm has become an effective way to engineer service-based systems (SBSs) by composing network-accessible (and often distributed) services [1], [2], which collectively offer the functionality of the SBS and fulfil its quality requirements [3]. The development and popularity of e-business, e-commerce, especially the pay-as-you-go business model promoted by cloud computing, have fueled the rapid growth of services and SBSs, which is indicated by the statistics published by ProgrammableWeb, an online web service and web API directory [4]. More and more functionally equivalent services are emerging, characterised by different multi-dimensional quality, such as cost, response time, and throughput [5]. In the cloud environment, SBS vendors can employ these existing services to build SBSs that fulfil tenants' multi-dimensional quality requirements, and achieve their own optimisation goals [6]. However, such an SBS is not failure free due to the fact that the services selected to build the SBS (named *component services*) are usually offered by independent (and often distributed) service providers in the cloud. Thus, the environment in which the SBS operates tends to be volatile. Unexpected runtime anomalies may occur to its component services [7], [8]. If a runtime anomaly cannot be fixed in a timely manner, it often leads to degradation and end-to-end violation in multi-dimensional system quality, incurring penalties and losses caused by Service Level Agreement (SLA) breach. Amazon reports that by adding only 100 ms in response time, sales drop by 1 percent [9]. An e-commerce system can lose thousands of dollars for every minute of its unavailability [10]. Thus, an SBS must be highly fault tolerant and it has become a significant challenge for system engineers to build SBSs that can handle runtime anomalies effectively and efficiently.

To ensure that SBSs remain operational and available upon runtime anomalies, a number of approaches has been proposed in recent years, such as fault removal [11], [12], [13] and fault tolerance techniques [14], [15], [16]. Fault removal reduces the impacts of anomalies through runtime service adaptation, e.g., service reselection, resource reconfiguration, etc., which, however, may still not be able to handle the failures sufficiently. For example, service reselection can be carried out when a system change is detected by selecting alternative services with equivalent functionalities to replace the anomalous ones. This approach is not ideal in most cases because it often causes unacceptable business interruptions due to the fact that solving the NP-complete

- *Y. Wang and Q. He are with the State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China, and the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Vic 3122, Australia. E-mail: {yanchunwang, qhe}@swin.edu.au.*
- *D. Ye and Y. Yang are with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Vic 3122, Australia. E-mail: {dye, yyang}@swin.edu.au.*

Manuscript received 16 May 2016; revised 10 Feb. 2017; accepted 5 Mar. 2017. Date of publication 12 Mar. 2017; date of current version 21 Mar. 2018. Recommended for acceptance by A. Zisman.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2681667

service selection problem satisfactorily is often very time consuming [17], especially in large-scale scenarios. Under this circumstance, fault tolerance methods, by offering service redundancy with functionally equivalent services, become an intuitive and promising way to ensure performance and availability of the SBS in a volatile environment. However, constrained by cost, it is often impractical to allocate service redundancy for every component service of an SBS due to the potentially excessive cost, especially for large-scale SBSs. Therefore, the identification of critical component services is an important issue in the formulation of cost-effective fault tolerance strategies for SBSs.

Several approaches have been proposed for the identification of critical services in an SBS [3], [11], [18]. However, those approaches can handle only one quality dimension (e.g., reliability), which is insufficient in real-world scenarios where multiple quality dimensions need to be taken into account. In addition, these approaches are not capable of capturing the important characteristic of cloud computing: *multi-tenancy*, which refers to a software architecture that uses a single software instance to serve multiple tenants. A tenant of a multi-tenant system is an organisational entity that hosts a number of end users and subscribes to the multi-tenant system according to the pay-as-you-go business model. For example, a company that subscribes to a multi-tenant Customer Relationship Management (CRM) system provided by Saleforce[1] is one of the tenants that shares the system. The employees of that company are the end users of the system. A multi-tenant SBS provides multiple tenants with similar and yet customised functionalities with potentially different quality values [19]. Multi-tenancy introduces significant challenges to the formulation of cost-effective fault tolerance strategies for SBSs. On one hand, the tenants of a multi-tenant SBS often have differentiated quality preferences [20]. For example, a tenant may have strong preference for response time, while another may prefer high system throughput. On the other hand, the number of tenants that a component service serves simultaneously may vary dramatically. It is possible that one component service in an SBS is shared by a large number of tenants while another service in the same SBS only serves for a few tenants. In this context, a component service critical to some tenants may not be of the same criticality to the others. When formulating a fault tolerance strategy for a multi-tenant SBS, the system engineer must consider the diversity in tenants' differentiated quality preferences and endeavour to reduce the risk of quality violations for all tenants. Therefore, multi-tenancy significantly complicates the identification of critical services and the formulation of fault tolerance strategies for SBSs.

To address the above issues, this paper proposes Criticality-based Fault Tolerance for Multi-Tenant SBSs (CFT4MTS), a novel approach for formulating cost-effective fault tolerance strategies for multi-tenant SBSs with service redundancy based on *service criticality*, i.e., the severity of the impact of an anomalous component service on the SBS. As a premise, the system engineer of an SBS must have first selected appropriate component services to build the SBS. After that, CFT4MTS identifies the most critical component services, and then formulates fault tolerance strategies for SBSs considering multiple tenants' multi-dimensional quality preferences. CFT4MTS is designed to be used at design time or in an offline manner at runtime. Based on the formulated fault tolerance strategy, redundant services can replace anomalous component services at runtime to minimise the risk of system quality violation. The major contributions of this paper are as follows:

1) We propose a novel method for identifying critical component services in a multi-tenant SBS based on service criticality, which is calculated by considering the following three metrics: a) quality-based criticality, which is evaluated with sensitivity analysis techniques based on the impacts caused by service anomalies on the multi-dimensional quality of the SBS, b) tenant-based criticality, which is evaluated based on the impacts caused by service anomalies on multiple tenants in the SBS, and c) tenants' differentiated multi-dimensional quality preferences for the SBS.
2) We propose a Constraint Optimisation Problem (COP) model based on service criticality for formulating cost-effective fault tolerance strategy for a multi-tenant SBS. A fault tolerance strategy specifies the allocation of redundant services for the critical component services and their redundancy mode. The Integer Programming technique is employed to find the optimal solution to the COP.

In addition, we conduct comprehensive evaluation of CFT4MTS with extensive experiments. CFT4MTS is compared with other existing representative approaches, which are adjusted for a fair comparison in the context of this research. The experimental results show that, compared with three existing representative approaches, CFT4MTS significantly improves the effectiveness and efficiency of fault tolerance in protecting the multi-dimensional system quality of a multi-tenant SBS.

The rest of this paper is organised as follows. The next section motivates this research with an example. Section 3 introduces the compositional quality model adopted in this research. Section 4 presents the proposed approach for formulating fault tolerance strategies for multi-tenant SBSs. Section 5 presents the experimental evaluation. Section 6 reviews related work. Finally, Section 7 concludes the paper and points out our future work.

2 Motivating Example

This section presents an example multi-tenant SBS, namely *Versatile Online Video Studio (VOVS)* to motivate this research. As shown in Fig. 1, this SBS consists of nine tasks. Similar to [11], [12], [21], [22], we assume in this research that a group of functionally equivalent candidate services with differentiated quality values are available for each of the tasks of *VOVS*. The system engineer has selected component services s_1 to s_9 from the corresponding set of candidate services for each task to compose the SBS, which provides three types of online video services to multiple tenants. The notation summary for this paper can be found in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2017.2681667.

In *VOVS*, s_1 provides the Access Control Service (ACS) to authenticate tenants when their end users try to gain access to

1. https://www.salesforce.com

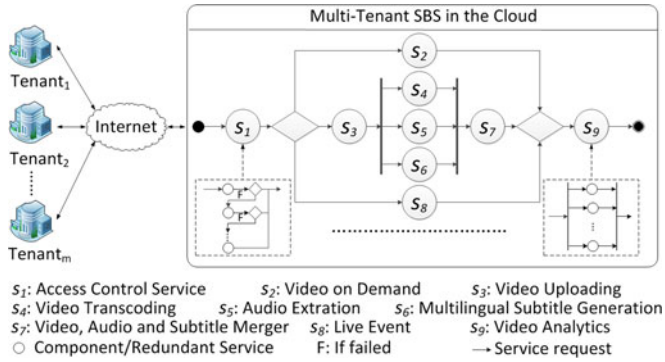


Fig. 1. Business process of Versatile Online Video Studio (*VOVS*).

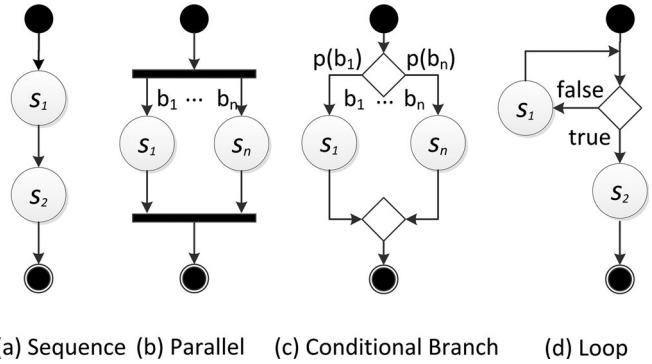


Fig. 2. Composition patterns.

the SBS. After the tenants are authenticated, the end users can use different categories of video services: s_2 is used to view videos on demand (VoD); s_8 offers the live streams services, which are used to broadcast or watch live videos; while s_3 to s_7 provide services for uploading, editing, and publishing the video clips; s_3 hosts the uploaded videos; s_4 transcodes the videos to different formats and resolutions compatible with various end devices; s_5 extracts audio streams from the videos, with which multilingual subtitles are generated by s_6; s_7 merges the videos, audios and subtitles, and publishes the synthetic video clips; s_9 performs video analytics, which carries out measurement and analysis of the videos viewed and uploaded online for user experience optimisation.

This SBS has the following characteristics:

1) It must process the video stream timely and continuously to ensure low-latency streaming of high quality videos. The system must be able to handle anomalies in a dynamic and volatile environment to ensure the quality of its video streaming, such as response time, throughput, etc.
2) It needs to achieve high fault tolerance to minimise the risk of system quality degradation caused by runtime anomalies. Service redundancy can be used to realise this goal: first, selecting redundant services for the component services from the corresponding group of candidate services, then, running the selected redundant services with the corresponding component services in a certain redundancy mode, such as sequence or parallel. In this way, the redundant services can immediately replace the faulty component services at runtime. It is theoretically ideal to prepare redundancy for all component services of the SBS. However, this is impractical because it often results in excessive cost. Deploying redundant services for more critical component services is a more cost-effective and practical solution.
3) Fault tolerance of the SBS should be able to cater for tenants' diverse multi-dimensional quality preferences and different business scenarios. First, the tenants may contain end users that access the SBS with different kinds of end devices, and thus have different concerns on quality of the SBS. For example, when viewing videos on demand, tenants that offer services for end users using laptops on a home network may prefer high resolution and low latency without worrying about cost incurred by data traffic, while tenants that access the SBS with a mobile network may prefer relatively low resolution to limit the cost incurred by data traffic and be endurable to long buffering time. In addition, the tenants may subscribe to *VOVS* for different purposes, e.g., watching videos on demand or live videos, and thus anomalies of some services may not affect the tenants who do not use them. Therefore, the severity of the impact of an anomalous component service on the SBS, i.e., the criticality of a component service, is varied to different tenants.

Given the above characteristics, in order to achieve cost-effective fault tolerance with service redundancy for *VOVS*, the criticalities of the component services of *VOVS* must be evaluated based on not only its multi-dimensional quality but also the multiple tenants' service sharing and their quality preferences. Component services with higher criticalities should be given higher priorities in the allocation of service redundancy.

3 Composition Quality Model

Quality evaluation is the basis for quality-aware service composition for SBSs. In this section, we introduce the composition quality model adopted in this research, including composition pattern, execution path, execution plan, characteristics of numerical quality, and utility evaluation.

3.1 Composition Pattern

A system engineer selects services from a number of sets of candidate services (also known as service classes) to compose an SBS with certain composition patterns. Composition patterns describe the order in which the services are executed in the service composition of an SBS. Similar to other work [5], [11], as Fig. 2 shows, four basic composition patterns are included in our model: *Sequence*, *Parallel*, *Conditional Branch* and *Loop*.

1) *Sequence*: The services are executed sequentially.
2) *Parallel*: All the branches $\{b_1, b_2, \ldots, b_n\}$ are executed at the same time.
3) *Conditional Branch*: There is a set of branches $\{b_1, b_2, \ldots, b_n\}$ in this structure, and only one of them can be selected to execute at one time with a probability $p(b_i)(0 \leq p(b_i) \leq 1, \sum_{i=1}^{n} p(b_i) = 1)$.
4) *Loop*: In this structure, every loop iterates for i times with a probability $p(l_i)$, where $0 \leq i \leq MNI$, $\sum_{i=0}^{MNI} p(l_i) = 1$ and MNI is the expected maximum number of iterations for the loop.

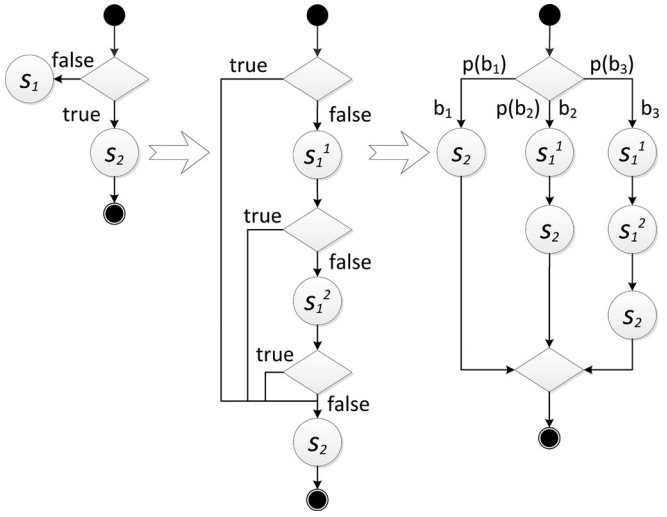


Fig. 3. A loop peeling example.

From the example SBS presented in Fig. 1, we can identify three types of composition patterns in *VOVS*: *Sequence*, *Parallel*, and *Conditional Branch*. For example, in response to each service request, s_1, s_2, s_9 are executed in sequence; s_4, s_5 and s_6 are performed in parallel; s_2, s_3 and s_8 are located on conditional branches, only one of which is selected each time with a probability that depends on the distribution of tenants' functional requirements.

In this paper, we mainly focus on *Sequence*, *Parallel* and *Conditional Branch*. The loop structure is transformed to the conditional branch structure with the loop peeling method described in [5]. Fig. 3 shows an example of the loop peeling process where the expected maximum number of iterations is 2. The loop structure is transformed into a conditional branch structure that contains three branches b_1, b_2 and b_3, which are selected to execute with the probabilities of $p(b_1)$, $p(b_2)$ and $p(b_3)$ respectively.

3.2 Execution Path and Execution Plan

Due to the branches in a service composition, execution paths and execution plans can be identified, which are defined as follows:

Definition 1 (Execution Path). *Denoted by* ep, *an execution path is a flattened sequence of services from the initial service to the final service in a service composition. It does not contain conditional branch and parallel structures. A service can belong to multiple execution paths.*

Definition 2 (Execution Plan). *Denoted by* epl, *an execution plan is a combination of execution paths with conditional branch or parallel structures, which accomplishes certain functional requirements. It serves multiple tenants in an SBS. A service can also belong to multiple execution plans.*

As depicted in Fig. 4, five execution paths and three execution plans are identified for *VOVS*, where epl_1 is used for Video on Demand service, epl_2 is for the tenants to upload, edit and publish their video clips, and epl_3 streams live videos.

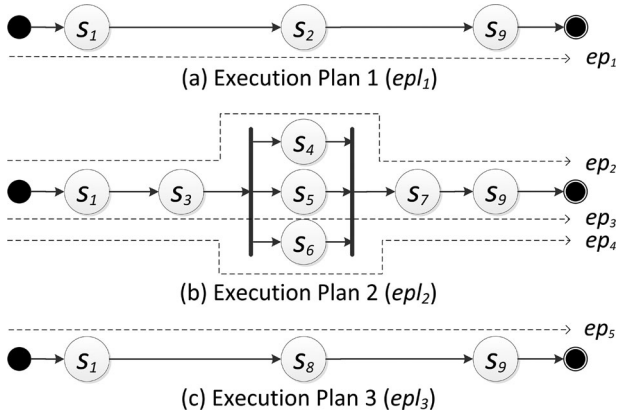


Fig. 4. Execution paths and execution plans of *VOVS*.

3.3 Quality Characteristics

The quality of an SBS is evaluated based on all its possible execution plans according to their execution probabilities, which is calculated based on the execution probabilities of their execution paths [5]. Table 1 shows the aggregation functions for evaluating the overall quality of an SBS $\mathbb{S}$ with four typical numerical quality parameters: cost, response time, reliability and throughput, where the quality values are denoted by q_{ct}, q_{rt}, q_{re} and q_{tp}, respectively, and $s_{i,j}$ represents the jth service in the ith service class. ep_k is the kth execution path and $p(epl_e)$ is the execution probability of the eth execution plan. Take response time of *VOVS* as an example, it can be calculated by $q_{rt}(VOVS) = p(epl_1) \times q_{rt}(epl_1) + p(epl_2) \times q_{rt}\ \ (epl_2) + p(epl_3) \times q_{rt}(epl_3)$, and the response time of each execution plan is determined by its execution path with the longest execution time.

The numerical quality parameters in Table 1 can be divided into two categories: positive and negative, defined as follows:

Definition 3 (Positive Quality Parameter). *If the quality evaluation increases along with the increase of quality value, the quality parameter is regarded as positive, such as throughput and reliability.*

Definition 4 (Negative Quality Parameter). *A quality parameter is negative if its evaluation will decrease as its value increases, such as cost and response time.*

TABLE 1
Quality Aggregation Functions

Quality Parameter	Aggregation Function
Cost	$q_{ct}(\mathbb{S}) = \sum_{s_{i,j} \in epl_e epl_e \in \mathbb{S}} p(epl_e) \times q_{ct}(s_{i,j})$
Response Time	$q_{rt}(epl_e) = \max_{ep_k \in epl_e} \left(\sum_{s_{i,j} \in ep_k} q_{rt}(s_{i,j}) \right)$ $q_{rt}(\mathbb{S}) = \sum_{epl_e \in \mathbb{S}} p(epl_e) \times q_{rt}(epl_e)$
Reliability	$q_{re}(\mathbb{S}) = \prod_{s_{i,j} \in \mathbb{S}} q_{re}(s_{i,j})$
Throughput	$q_{tp}(epl_e) = \min_{ep_k \in epl_e} (\min_{s_{i,j} \in ep_k} q_{tp}(s_{i,j}))$ $q_{tp}(\mathbb{S}) = \sum_{epl_e \in \mathbb{S}} p(epl_e) \times q_{tp}(epl_e)$

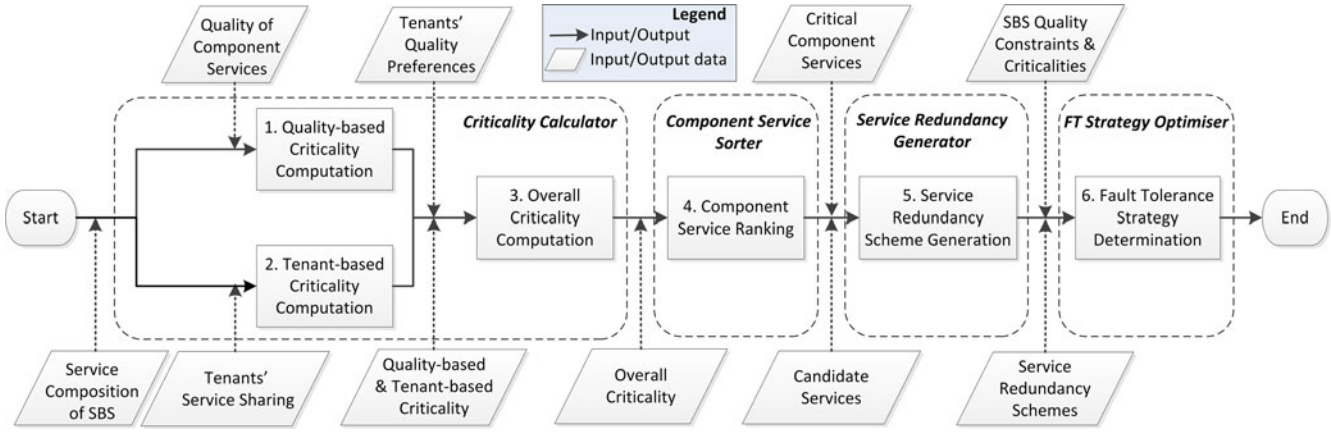


Fig. 5. System architecture and procedure of CFT4MTS.

The model and experiments in this paper are mainly based on the quality parameters presented in Table 1. The quality parameters introduced in other literatures can be included as added dimensions in our model.

3.4 Utility Evaluation

Services in the same service class are functionally equivalent but usually different in their multi-dimensional quality values. As shown in Fig. 1, the Video on Demand service in *VOVS* provided by one service provider may have better performance as well as higher price than a service with the same functionality provided by another provider. Which service is better in service selection depends on the quality preferences of the tenants sharing the service. In this paper, we rank the services in a service class based on their *utility*, which is defined as follows:

Definition 5 (Service Utility). *The utility of a service indicates tenants' generalised preferences for the service. It is calculated based on the quality values of the service and the tenants' quality preferences by applying the Simple Additive Weighting (SAW) technique [23].*

Different quality parameters can be adopted in the calculation of service utility, depending on the scenarios.

In order to remove the incomparability between the units of measurement for different quality parameters, the original numerical quality values are normalised first using (1) for positive quality parameters and (2) for negative quality parameters with widely used Min-Max normalisation technique [5] [11]

$$Q_p(s_{i,j})=\begin{cases}\dfrac{q_p(s_{i,j})-q_p^{min}(sc_i)}{q_p^{max}(sc_i)-q_p^{min}(sc_i)} & \text{if } q_p^{max}(sc_i)\neq q_p^{min}(sc_i)\\ 1 & \text{if } q_p^{max}(sc_i)=q_p^{min}(sc_i)\end{cases} \quad (1)$$

$$Q_p(s_{i,j})=\begin{cases}\dfrac{q_p^{max}(sc_i)-q_p(s_{i,j})}{q_p^{max}(sc_i)-q_p^{min}(sc_i)} & \text{if } q_p^{max}(sc_i)\neq q_p^{min}(sc_i)\\ 1 & \text{if } q_p^{max}(sc_i)=q_p^{min}(sc_i)\end{cases}, \quad (2)$$

where $q_p^{max}(sc_i)$ and $q_p^{min}(sc_i)$ are the maximum and minimum values, respectively, for the pth quality parameter in the ith service class; $q_p(s_{i,j})$ is the pth dimensional quality value of the jth service in the ith service class; and $Q_p(s_{i,j})$ is the normalised pth quality value of service $s_{i,j}$. In this paper, a specific quality parameter can be indicated by both "pth" and the abbreviations of their names, e.g., "rt" for response time, "tp" for throughput, etc.

Similar to [6], [20], for a multi-tenant SBS, we use (3) to calculate the average preference for each quality parameter of a service across all the tenants

$$w_p^{ave}=\frac{1}{\tau}\times\sum_{t=1}^{\tau}w_{t,p}, p=1,\ldots,d, \quad (3)$$

where τ is the number of tenants that share a given service, and a system engineer can usually obtain tenants' quality preferences from their SLAs. Parameter $w_{t,p}$ indicates the tth tenant's preference for the pth quality parameter, $\sum_{p=1}^{d}w_{t,p}=1$ where d is the number of quality parameters.

Then, the utility of a given service $s_{i,j}$ with d quality parameters across τ tenants is calculated by

$$u(s_{i,j})=\sum_{p=1}^{d}w_p^{ave}\times Q_p(s_{i,j}). \quad (4)$$

The utility of an execution plan epl_e can be obtained by summing up the utilities of the services that belong to epl_e, and the utility of an SBS $\mathbb{S}$ can be calculated by weighting the utilities of its execution plans with their corresponding execution probabilities. In general, an SBS with higher utility is more preferable by the tenants.

4 CFT4MTS Approach

Fig. 5 shows the system architecture of CFT4MTS, which includes four functional components. Their roles are as follows:

1) *Criticality Calculator*: This component calculates service criticality based on quality of service, tenants' quality preferences and their service sharing.
2) *Component Service Sorter*: This component ranks component services according to their criticalities.
3) *Service Redundancy Generator*: This component generates service redundancy schemes for component services based on redundant services and redundancy mode.
4) *FT Strategy Optimiser*: This component determines the optimal fault tolerance strategy of an SBS.

CFT4MTS consists of six phases, as shown in Fig. 5. In Phase 1, we calculate the service criticality for each dimension of quality with the Sensitivity Analysis (SA) technique. In Phase 2, the criticality based on multi-tenancy is evaluated by analysing tenants' multi-dimensional quality preferences and service sharing. Based on the outcomes of Phase 1 and Phase 2, in Phase 3, the overall criticality of each service is calculated, by which the services in the composition are ranked in Phase 4. Redundancy schemes are then generated in Phase 5. Finally, in Phase 6, formulation of cost-effective fault tolerance strategy with service redundancy based on criticality is modelled as an optimisation problem, and the Integer Programming (IP) technique is used to find the solution. These six phases are detailed one by one in this section.

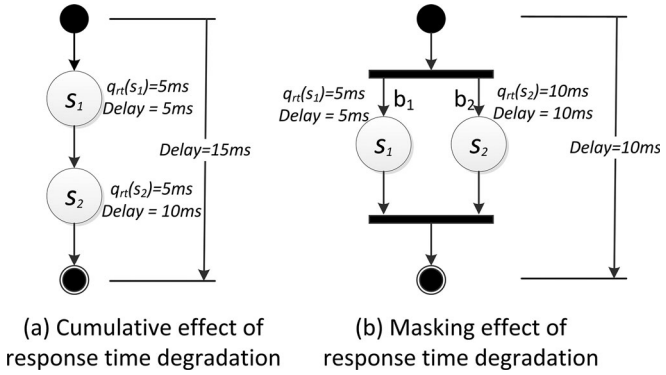


Fig. 6. Effects of multiple service anomalies on SBS-level degradation in response time.

4.1 Quality-Based Criticality Computation

In a dynamic cloud environment, various changes can occur to the component services of an SBS. The changes can be positive or negative, causing quality upgradation or degradation to the services, which often impacts the quality of the SBS. In this paper, we consider only the negative changes, i.e., quality degradation incurred by runtime anomalies occurring to the services, as they can result in end-to-end system quality violation.

The sensitivities of the quality degradation of the SBS with respect to the quality degradation of different services may vary. Take Fig. 4 for example, there is a parallel structure composed by s_4, s_5 and s_6 in the execution plan epl_2 of *VOVS*. The execution time of the parallel structure depends on the maximum execution time of s_4, s_5 and s_6. Assume that the execution time of s_4, s_5 and s_6 are 5, 10 and 15 ms respectively, when a delay of 10 ms occurs to each service respectively, the corresponding delays caused to epl_2 by s_4, s_5 and s_6 are 0, 5 and 10 ms, which thereby may have different impacts on the end-to-end response time of the SBS. In this paper, we adopt the Sensitivity Analysis method, which is widely used in many areas [24], [25], [26], to quantify and analyse the impacts of quality degradation of component services on the end-to-end system quality.

We measure the criticality of a given service by the sensitivity of the quality degradation of an SBS $\mathbb{S}$ in comparison to the quality degradation of that service. In order to quantitively evaluate the quality degradation of a component service, we first define *quality degradation coefficient* as follows:

Definition 6 (Quality Degradation Coefficient). *Denoted by q_p^{dc}, the quality degradation coefficient indicates the maximum quality degradation level of the pth quality parameter of a component service, represented by the ratio of the variation in the pth quality value upon anomaly to the original quality value.*

Take s_1 in *VOVS* for example, if the quality degradation coefficients for response time and throughput are 200 and 80 percent respectively, the corresponding quality value of s_1 will increase by 200 to 300 percent of its original value and decrease by 80 to 20 percent of its original value respectively upon anomalies.

As shown in formula (5), we then degrade the pth quality value of component service s_i by q_p^{dc} in steps of Δq_p percentage for r iterations, e.g., increasing response time or decreasing throughput of s_i by 20 percent each time. In the $k\text{th}(0 \leq k \leq r)$ iteration, we investigate the variation percentage of pth quality value of $\mathbb{S}$ relative to its original value, denoted by $\Delta q_{k,p}^{vp}(\mathbb{S}, s_i)$, and divide it by the variation percentage of the quality of s_i which is represented by $\Delta q_{k,p}^{vp}(s_i)$. The results are summed and averaged as the criticality of the pth quality parameter of s_i, denoted by $cr_p^Q(s_i)$.

$$cr_p^Q(s_i) = \frac{\sum_{k=1}^{r}\left(\frac{\Delta q_{k,p}^{vp}(\mathbb{S}, s_i)}{\Delta q_{k,p}^{vp}(s_i)}\right)}{r}, r = \frac{q_p^{dc}}{\Delta q_p}. \quad (5)$$

When employing (5) in an SBS containing n component services, only the quality value of one service is varied at a time. In reality, however, the other $n-1$ component services may fail at the same time. According to the quality aggregation functions in Table 1, the quality degradations caused by multiple anomalous services can cumulate (as shown in Fig. 6a) or mask (as shown in Fig. 6b) one another. However, the effects of different anomalies on the quality of the SBS are independent, which therefore does not affect the criticalities of the component services.

In real-world scenarios, the quality degradation coefficient varies from service to service and thus must be determined empirically on a case-by-case basis. This is because there are many uncertain factors that may impact the actual quality degradation caused to a service by a runtime anomaly, e.g., the time taken to diagnose the anomaly, the difficulty of fixing the anomaly, etc. For the sake of simplicity, in this paper we use a unified q_p^{dc} across all the services without affecting the effectiveness of the proposed approach. Alternatively, the maximum quality degradation level of a specific service can be evaluated by inspecting the service' past executions, clients' feedbacks, service providers' profile, the SLA, etc. In addition, formula (5) does not handle the varied failure probabilities of different component services and the scenario where one quality degradation level is likely to occur more than another. For example, s_1 in *VOVS* may fail more frequently than s_2 when processing a same amount of service requests, and a 20 percent response time degradation more likely to occur in s_1 than a 40 percent degradation. However, the probability distributions of service failure and quality degradation are domain specific. The approach proposed in this paper can be adjusted to deal with specific requirements, e.g., using a weighting process based on probabilities to differentiate the contribution of each service failures to the system quality degradation, which does not influence the effectiveness of our approach either.

TABLE 2
Criticality Computation and Service Ranking in *VOVS*

Component Services	Response Time (ms)	Throughput (req/sec)	cr^Q_{rt}	crn^Q_{rt}	cr^Q_{tp}	crn^Q_{tp}	$\tau(s_i)/\tau(\mathbb{S})$	$\xi(s_i)/\xi(\mathbb{S})$	cr^T	cr^O	Rank
s_1	90.7	40	0.334	1	0.838	0.918	1	1	1	0.959	1
s_2	69.8	22	0.023	0.055	0.054	0	0.35	0.09	0.032	0.001	9
s_3	56.6	92	0.153	0.449	0.092	0.044	0.75	0.73	0.548	0.134	5
s_4	40.0	46	0.020	0.045	0.339	0.334	0.75	0.73	0.548	0.104	6
s_5	35.9	59	0.011	0.018	0.212	0.185	0.75	0.73	0.548	0.056	7
s_6	83.5	33	0.225	0.667	0.908	1	0.75	0.73	0.548	0.457	2
s_7	73.8	63	0.199	0.588	0.193	0.163	0.75	0.73	0.548	0.206	4
s_8	7.4	57	0.005	0	0.282	0.267	0.60	0.18	0.108	0.014	8
s_9	16.3	68	0.060	0.167	0.300	0.288	1	1	1	0.228	3

In this paper, we adopt response time and throughput to calculate the quality-based criticality of a component service. In terms of reliability, all the component services are equally important in contributing to the overall system quality according to the quality aggregation functions presented in Table 1. The cost of a service does not change upon anomalies. However, reliability and cost can play important roles in criticality calculation in scenarios where more complicated reliability model and pricing model are expected, such as the reliability based on the structure of an SBS, the consideration of the cost for fixing a service anomaly, etc. Please note that some criticality measures other than the one based on sensitivity analysis in this paper, such as reliability-based criticality discussed in [27], [28], [29], can be used for service criticality calculation in a similar manner.

4.2 Tenant-Based Criticality Computation

As described in Section 2, *VOVS* contains multiple execution plans that serve different tenants with different functional requirements. All the tenants can access the SBS through service requests simultaneously, and the SBS is executed upon the receipt of each service request. However, some component services are not shared by all the tenants, such as the Video on Demand service and Live Event service. From the SBS vendor's perspective, these services are not equally critical and resources should be allocated to the services that serve more tenants and respond to more service requests. Given the fact that the quality degradation of a component service may only affect the tenants sharing the service, the tenant-based criticality of a service is calculated by

$$cr^T(s_i) = \frac{\tau(s_i)}{\tau(\mathbb{S})} \times \frac{\xi(s_i)}{\xi(\mathbb{S})}, \tag{6}$$

where $\tau(s_i)$ is the number of tenants sharing service s_i, $\tau(\mathbb{S})$ is the number of all tenants served by the SBS, $\xi(s_i)$ is the average number of service requests that s_i processes per unit of time, e.g., per second, and $\xi(\mathbb{S})$ is the average number of service requests that SBS $\mathbb{S}$ processes within the same unit of time. For example, we assume that $\tau(VOVS)$ is 100, $\tau(s_2)$ is 35, $\xi(VOVS)$ is 100 req/s, and $\xi(s_2)$ is 9 req/s, then we can calculate $cr^T(s_2)$ to be 0.032. The numbers of tenants and service requests in (6) can be obtained by the system engineer through the analysis of tenants' business requirements as well as the corresponding SLA.

4.3 Overall Criticality Computation

A component service may not be of the same criticality according to different criteria. In this paper, we aim at providing an integrated solution to the formulation of fault tolerance strategy across multiple tenants of the SBS. Therefore, we calculate the overall criticality of a component service in this phase by combining its multi-dimensional quality-based criticality and tenant-based criticality, considering tenants' quality preferences.

First, due to the different methods that can be adopted when calculating quality-based criticality, we normalise the criticality value with (7) to eliminate the impact of different measurement units, where cr^{QX}_p and cr^{QN}_p are the maximum and minimum criticality values, respectively, of the component services for the pth quality parameter in the SBS $\mathbb{S}$, and $cr^Q_p(s_i)$ is the pth quality-based criticality value of the ith component service in the composition of $\mathbb{S}$

$$crn^Q_p(s_i) = \begin{cases} \dfrac{cr^Q_p(s_i) - cr^{QN}_p(\mathbb{S})}{cr^{QX}_p(\mathbb{S}) - cr^{QN}_p(\mathbb{S})} & \text{if } cr^{QX}_p(\mathbb{S}) \neq cr^{QN}_p(\mathbb{S}) \\ 1 & \text{if } cr^{QX}_p(\mathbb{S}) = cr^{QN}_p(\mathbb{S}) \end{cases}. \tag{7}$$

Having been normalised, the multi-dimensional quality preferences of the tenants that share s_i, with the tenant-based criticality of s_i, are used to calculate the overall criticality of s_i with the following formula:

$$cr^O(s_i) = \left(\sum_{p=1}^{d} crn^Q_p(s_i) \times w^{ave}_p \right) \times cr^T(s_i), \tag{8}$$

where w^{ave}_p is calculated with (3) and denotes the average preferences for each quality parameter of s_i across all the tenants sharing s_i. Now the criticalities of the component services of an SBS have been calculated, which are used to rank those component services in the next phase.

4.4 Component Service Ranking

Based on service criticality, the component services of an SBS can be ranked and the top k $(1 \leq k \leq n)$ most critical services can be identified. Table 2 shows the criticality computation and service ranking for *VOVS*. For demonstration purpose, we use q^{dc}_{rt}=200 and Δq_{rt}=50 percent for response time, and use q^{dc}_{tp}=80 and Δq_{tp}=20 percent for throughput. Take s_1 for example, its response time-based and throughput-based criticalities are 0.334 and 0.838 respectively which are calculated with (5) based on the quality values in Table 2. Then the

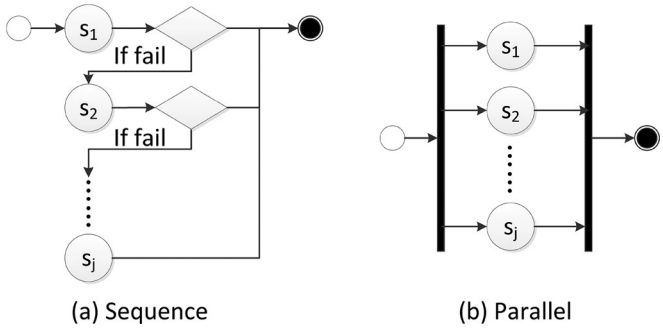


Fig. 7. Service redundancy modes.

quality-based criticalities of s_1 are normalised to 1 and 0.918 respectively with (7). The tenant-based criticality of s_1 is 1 which is calculated with (6) based on the service sharing and service requests distribution in *VOVS*, represented by $\tau(s_i)/\tau(\mathbb{S})$ and $\xi(s_i)/\xi(\mathbb{S})$ in Table 2. The overall criticality of s_1 is 0.959 which is calculated using (8) with equal preferences for response time and throughput across all tenants.

For all the component services in *VOVS*, the phase-by-phase outcomes of criticality calculation and service ranking from Phase 4.1 to Phase 4.4 are presented in Table 2. The ranking results show that in this scenario s_1 has the highest criticality and should be given the highest priority in the allocation of service redundancy.

4.5 Service Redundancy Scheme Generation

In this phase, CFT4MTS generates the *service redundancy scheme*, which is defined as follows:

Definition 7 (Service Redundancy Scheme). *Denoted by SRS, service redundancy scheme is the combination of its member services (denoted by MS, including the component service and redundant services) and the corresponding redundancy mode (denoted by RM), such as sequence or parallel.*

Various redundancy modes have been proposed to improve the reliability of critical applications in conventional software system [30], [31], [32]. In this paper we consider two modes widely adopted: *Sequence* and *Parallel*, as shown in Fig. 7. Other modes can be included in CFT4MTS in a similar manner. The two modes function as follows:

1) *Sequence*. The services are executed sequentially from the first service until one of them completes successfully.
2) *Parallel*. The services are executed concurrently, and the first service that completes successfully determines the final result.

In both cases, a redundancy mode fails only if all the services fail. Similar to other work [5], [11], [21], in this paper we assume that a task can be bound to any functionally equivalent candidate service in its corresponding service class, which means that these services are idempotent and the service requests are handled in an idempotent manner.

In general, service redundancy schemes with *Sequence* mode have lower cost but may suffer from poor response time performance in volatile environments, while *Parallel* schemes can obtain faster response time at the cost of more resources consumed. In this case, the service redundancy scheme with *Parallel* mode is a better choice for the Live Event service in *VOVS* if the quality constraint on response time is severe and the budget for fault tolerance is sufficient.

Let $SRS_{i,j}$ be the jth service redundancy scheme of service class sc_i containing m_i candidate services. For $SRS^{sq}_{i,j}$ with $RM_{i,j}=Sequence$, and $MS_{i,j}=(s_{i,j_0}, s_{i,j_1}, \ldots, s_{i,j_m})$, $s_{i,j_0}=s_i, 1\le m\le m_i$, the quality aggregation functions are as follows:

$$\begin{aligned}
q_{ct}(SRS^{sq}_{i,j}) &= \sum_{f=0}^{m}\left(\sum_{r=0}^{f} q_{ct}(s_{i,j_r}) \times \prod_{r=0}^{f-1}\big(1-q_{re}(s_{i,j_r})\big)\right)\\
q_{rt}(SRS^{sq}_{i,j}) &= \sum_{f=0}^{m} q_{rt}(s_{i,j_f}) \times \prod_{r=0}^{f-1}(1-q_{re}(s_{i,j_r}))\\
q_{re}(SRS^{sq}_{i,j}) &= 1-\prod_{r=0}^{m}(1-q_{re}(s_{i,j_r}))\\
q_{tp}(SRS^{sq}_{i,j}) &= \sum_{f=0}^{m} q_{tp}(s_{i,j_f}) \times \prod_{r=0}^{f-1}(1-q_{re}(s_{i,j_r})),
\end{aligned} \tag{9}$$

where $q_{re}(s_{i,j_r})$ is the reliability of the rth service in $SRS^{sq}_{i,j}$, and service s_{i,j_r} will be executed only when all the preceding services in the same scheme have failed. Hence, the cost, response time and throughput of all the services in $SRS^{sq}_{i,j}$ should be summed, and each weighted by the probability that the service is executed.

The quality of service redundancy scheme $SRS^{pl}_{i,j}$ with the *Parallel* mode is calculated as follows:

$$\begin{aligned}
q_{ct}(SRS^{pl}_{i,j}) &= \sum_{r=0}^{m} q_{ct}(s_{i,j_r})\\
q_{rt}(SRS^{pl}_{i,j}) &= \sum_{\lambda=1}^{2^m-1} \min_{s_{i,j_r}\in MS_\lambda} q_{rt}(s_{i,j_r}) \times \prod_{s_{i,j_r}\in MS_\lambda} q_{re}(s_{i,j_r})\\
&\quad \times \prod_{s_{i,j_r}\notin MS_\lambda}(1-q_{re}(s_{i,j_r}))\\
q_{re}(SRS^{pl}_{i,j}) &= 1-\prod_{r=0}^{m}(1-q_{re}(s_{i,j_r}))\\
q_{tp}(SRS^{pl}_{i,j}) &= \sum_{\lambda=1}^{2^m-1} q_{tp}(\mathrm{SMRT}(s_{i,j_r}))\times \prod_{s_{i,j_r}\in MS_\lambda} q_{re}(s_{i,j_r})\\
&\quad \times \prod_{s_{i,j_r}\notin MS_\lambda}(1-q_{re}(s_{i,j_r})),
\end{aligned} \tag{10}$$

where MS_λ is the λth non-empty subset of $MS_{i,j}$. Since all the services in $SRS^{pl}_{i,j}$ are performed simultaneously, the costs of all services in the scheme are summed, while the response time is calculated by summing the minimum response time of all MS_λ, which is weighted by the probability that only the services in MS_λ are executed successfully. In *Parallel* mode the throughput is determined by the first service in MS_λ that completes successfully. The operator *SMRT* returns the service in MS_λ with minimum response time.

In both redundancy modes, the reliability of a service redundancy scheme is the probability that at least one service completes successfully.

In order to capture the requirements of formulating cost-effective fault tolerance strategy for an SBS, we introduce *Fault Tolerance Budget* and *Fault Tolerance Cost* in this paper.

Fault tolerance budget indicates how much the SBS vendor is willing to pay for the deployment of fault tolerance, while fault tolerance cost represents how much the SBS vendor has to pay for that. The cost of the jth service redundancy scheme in the ith service class can be calculated as follows:

$$q_{ft}(SRS_{i,j}) = q_{ct}(SRS_{i,j}) - q_{ct}(s_i), \tag{11}$$

where $q_{ct}(s_i)$ is the cost of component service s_i in the ith service class.

The fault tolerance cost of an SBS $\mathbb{S}$ can be calculated by

$$q_{ft}(\mathbb{S}) = \sum_{\substack{SRS_{i,j} \in epl_e \\ epl_e \in \mathbb{S}}} p(epl_e) \times q_{ft}(SRS_{i,j}), \tag{12}$$

where $p(epl_e)$ is the execution probability of the eth execution plan of $\mathbb{S}$.

4.6 Fault Tolerance Strategy Determination

Fault tolerance strategy determination is the process of allocating service redundancies for some or all of the component services in an SBS. To determine the fault tolerance strategy for the SBS, we formulate a Constraint Optimisation Problem. Considering an SBS $\mathbb{S}$ with n component services $\{s_1, s_2, \ldots, s_n\}$, and each of them is associated with a service class sc_i containing m_i ($m_i \geq 1$) candidate services $\{s_{i,1}, s_{i,2}, \ldots, s_{i,m_i}\}$. The system engineer needs to determine the fault tolerance strategy based on the criticalities of component services under the fault tolerance budget. The objective of optimisation is to maximise the sum of criticalities of component services to be protected, which means the impacts on the system quality upon service anomalies can be minimised under a limited budget.

Fault tolerance strategy determination is a complex decision making process, which is NP-complete and often very time-consuming given a large number of service redundancy schemes. In order to find a solution rapidly, CFT4MTS first ranks the candidate services within each service class by their utilities, which are calculated with the approach introduced in Section 3.4, and then use a greedy method that increases the number of redundant services gradually in a multi-round manner. Given a service class containing m_i candidate services, in each round, the top k_r candidate services with the highest utilities in each service class are selected to generate service redundancy schemes, where k_r is incremental in each round. A solution to the optimisation model can be found more rapidly with a larger k_r. In this paper we use $k_r = 2\nu$ ($1 \leq \nu \leq \lceil m_i/2 \rceil$) and ν is the round in which the optimisation is performed. k_r can be bounded by the system engineer by specifying a maximum round number ν^{max}, which can reduce the complexity of fault tolerance strategy determination. The impact of k_r is analysed in detail in Section 5.

In each round of fault tolerance strategy determination, we create a set of 0-1 integer variables $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,l}\}$ for service class sc_i, where l is the number of service redundancy schemes in the ith service class. $x_{i,j}$ with value of 1 means that the jth service redundancy scheme in the ith service class is selected for the formulation of fault tolerance strategy for SBS $\mathbb{S}$. We also create a 0-1 integer variable y_i for component service s_i, which is 1 if service redundancy is to be allocated for the component service s_i, and 0 otherwise. The process of finding a solution to the COP model is to assign a value of 0 or 1 to each variable and ensure that all constraints can be satisfied, including the quality constraints. This is equivalent to a knapsack problem, thus is a NP-complete problem, which makes it computationally expensive, if not impractical, to find an optimal solution in large-scale scenarios.

The COP model is formulated as follows:

$$Objective(\mathbb{S}): \left\{ Max \left(\sum_{i=1}^{n} cr^{O}(s_i) \times y_i \right) \right\} \tag{13}$$

$$\begin{aligned} q_p^{pos}(\mathbb{S}) &\geq C_p^{pos} \\ q_p^{neg}(\mathbb{S}) &\leq C_p^{neg} \end{aligned} \tag{14}$$

$$q_{ft}(\mathbb{S}) \leq C_b \tag{15}$$

$$y_{i+1} \leq y_i, \ i = 1, \ldots, n-1 \tag{16}$$

$$\sum_{j=1}^{l} x_{i,j} = y_i, \ x_{i,j}, y_i \in \{0, 1\}, \tag{17}$$

where constraints family (14) ensures that the multi-dimensional quality constraints for the SBS can be satisfied, where q_p^{pos} and q_p^{neg} are the pth end-to-end quality of $\mathbb{S}$, which may be positive or negative quality parameter. C_p^{pos} and C_p^{neg} are the pth quality constraints over $\mathbb{S}$. Constraints family (15) ensures that fault tolerance cost cannot exceed the budget C_b. Constraints family (16) ensures that only most critical component services can have redundant services. Constraints family (17) guarantees that at most one service redundancy scheme can be selected if the ith component service is to be protected. Formula (13) shows the optimisation objectives for the COP model: maximising the sum of overall criticalities of component services that will be allocated service redundancy.

After the COP model is created, the Integer Programming technique is employed to find the optimal solution. Once the fault tolerance strategy is determined, the redundant services in the selected redundancy schemes can be deployed in the SBS.

The computational overhead introduced by CFT4MTS includes two main parts: the computation time of generating service redundancy schemes from candidate services, and the computation time of determining the fault tolerance strategy based on the COP model. As discussed in Section 4.5, the increase in the number of redundant services can result in a rapid increase in the number of service redundancy schemes, which may make it more computationally expensive to find the solution based on the COP model. However, the fault tolerance strategy is formulated at design time or in an offline manner at runtime, and thus does not cause extra computational overhead at runtime. In very large scenarios, the redundancy schemes can be enumerated in parallel by different machines with distributed computing techniques, such as MapReduce. On the other hand, service recommendation or filtering approaches, e.g., skyline techniques [22], and clustering-based service recommendation [20], can be used on candidate services as well as service redundancy schemes to further improve the

efficiency of CFT4MTS. In overall terms, the computational overhead caused by CFT4MTS is much less significant than that introduced by the conventional runtime adaptation approaches, especially in extremely volatile environments where frequent runtime adaptation is needed.

5 Evaluation

We have conducted a range of experiments in a simulated volatile environment, aiming at evaluating CFT4MTS in terms of effectiveness and efficiency in fault tolerance by comparing it with other representative approaches. The effectiveness is measured by three quantitative metrics: success rate, quality degradation and affected tenant percentage.

1) Success rate of fault tolerance is defined as the percentage of successful fault tolerance in all test instances given the same fault tolerance budget. If anomalies occur and the component service has service redundancy in position, the fault tolerance is regarded as successful unless the component service and all redundant services fail at the same time.
2) Quality degradation, including that in response time and throughput, is represented by the average variation percentage of the end-to-end quality value of an SBS upon runtime anomalies.
3) Affected tenant percentage is the proportion of tenants affected by anomalies to all tenants that share the SBS.

The impacts of tenants' quality preferences and the environment volatility are also analysed. Efficiency of all approaches is evaluated based on the computational overhead, which is the average computation time needed for formulating a fault tolerance strategy for an SBS.

Section 5.1 describes the setup of the experiments. Sections 5.2 and 5.3 evaluate the effectiveness and efficiency of CFT4MTS respectively. Section 5.4 discusses the threats to the validity of our evaluation.

5.1 Experimental Setup

CFT4MTS is implemented in Java with JDK 1.6.0 and Eclipse Java EE IDE. IBM CPLEX v12.6, a widely used linear programming tool, is employed to solve the COPs. We have also implemented three representative and state-of-the-art approaches for comparison in the context of this research. These approaches, namely Quality-Optimal, FTCloud and Random, are described as follows:

1) *Quality-Optimal*: Originated from the work presented in [33], this approach formulates fault tolerance strategies without taking service criticality into account. All component services are considered equally important. COP models are created only based on the quality performance, and the optimisation goal is to maximise the system utility. For the sake of fair comparison, Quality-Optimal in our experiments uses the same quality parameters and constraints as FTCloud, Random and CFT4MTS.
2) *FTCloud*: This approach is originated from [18], which evaluates the criticalities of component services of an SBS based on the invocation structure and invocation frequencies of the component services. A component service invoked more frequently by other services is considered more important. Service redundancy schemes are generated accordingly. Based on the service redundancy schemes, a COP model is created to formulate the fault tolerance strategies aiming at reducing the system failure probability. Local rather than global quality constraints, i.e., the constraints on individual component service instead of the service composition of SBS, are adopted in the COP. However, for this approach in the experiments, FTCloud employs the global quality constraints, the same as other approaches, for a fair and objective comparison.
3) *Random*: A service redundancy scheme is created for a given component service by choosing redundancy mode and redundant services randomly. The component services are also selected in a random manner and each of them can be selected once only. Same quality parameters as CFT4MTS are adopted and the quality constraints on the SBS must not be violated in this process. This approach is also used in [18] as an existing approach to compare with.

In order to simulate a volatile cloud environment, we adopt the similar approach as described in [18]. Each component service has a failure probability. The more service requests it receives, the more failures may occur. In our series of experiments, same as works in [3], [18], [34], we set the failure probability at a fixed value, such as 1 percent, and then inject service anomalies to the SBS at runtime. In order to create different levels of volatility in the experimental environment, we vary the number of anomalies injected at one time and the quality degradation coefficient (see Definition 6) of service, and their impacts are analysed. For the purpose of simplicity and consistency without losing generality, we assume that when anomalies occur to a component service without service redundancy, the quality degrades according to a unified quality degradation coefficient, which is calculated by $q_p^{dc} = 1-(2)^{-h}$ and $q_p^{dc} = (2)^h$, $h \geq 1$ for positive and negative quality parameters respectively. For example, when h is 1, the quality degradation coefficients for throughput and response time are 50 and 200 percent respectively.

In all the experiments conducted, we use fault tolerance budgets as a critical constraint in the formulation of fault tolerance strategies. In this way, we evaluate the effectiveness of CFT4MTS in formulating cost-effective fault tolerance strategy by comparing with the Quality-Optimal, FTCloud and Random.

The services used in the experiments are generated based on QWS [35], a publicly available Web service dataset that consists of more than 2,500 real-world Web services with multi-dimensional quality. For each service, the cost and throughput are generated randomly based on normal distribution.

Though we conducted experiments on the motivating example *VOVS* as a case study, in order to compare CFT4MTS with existing approaches more comprehensively, we have mimicked SBSs with different numbers of tasks n (up to 100). The findings are consistent with those from the experiments on *VOVS*. For each test instance, a business process consisting of n tasks is constructed first with

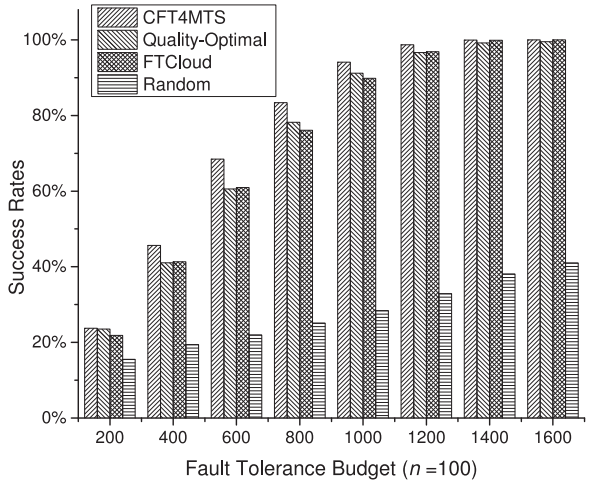


Fig. 8. Comparison in success rates.

randomly generated composition patterns introduced in Section 3.1. After that, 100 candidate services are created for each service class, based on which a service composition is formulated with IP techniques.

For each test instance with n service classes, $10 \times n$ tenants are created and distributed randomly when conditional branches exist in the service composition. Tenants' quality preferences are also generated randomly.

All the experiments were conducted on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. The experimental results are collected, averaged and compared from 100 test instances.

In order for other researchers to repeat the experiments in this paper, we have published our implementation of CFT4MTS and other three approaches used for comparison on Github, which is available at https://github.com/swincloud/CFT4MTS.

5.2 Effectiveness Evaluation

5.2.1 Comparison in Success Rates

In order to examine the effectiveness of CFT4MTS compared to Quality-Optimal, FTCloud and Random, we compare the success rates of fault tolerance obtained by these approaches. For a fixed number of service classes, we fix ν^{max} at 2 and increase the fault tolerance budget in steps of 200. One anomaly is injected to the SBS in each test instance. The experimental results show that, when budget increases, the success rates obtained by all approaches increase, and CFT4MTS outperforms other approaches in all scenarios. We use test instances with n = 100 as samples to demonstrate the results. As shown in Fig. 8, FTCloud and Quality-Optimal achieve similar performance and are beaten by CFT4MTS by approximately 5 percent on average. Random obtains the lowest success rates among all the approaches as expected.

5.2.2 Comparison in Quality Degradation

In order to evaluate the effectiveness of CFT4MTS in alleviating system quality degradation upon runtime anomalies against existing approaches, we conduct a set of experiments, where different approaches in comparison are employed, to

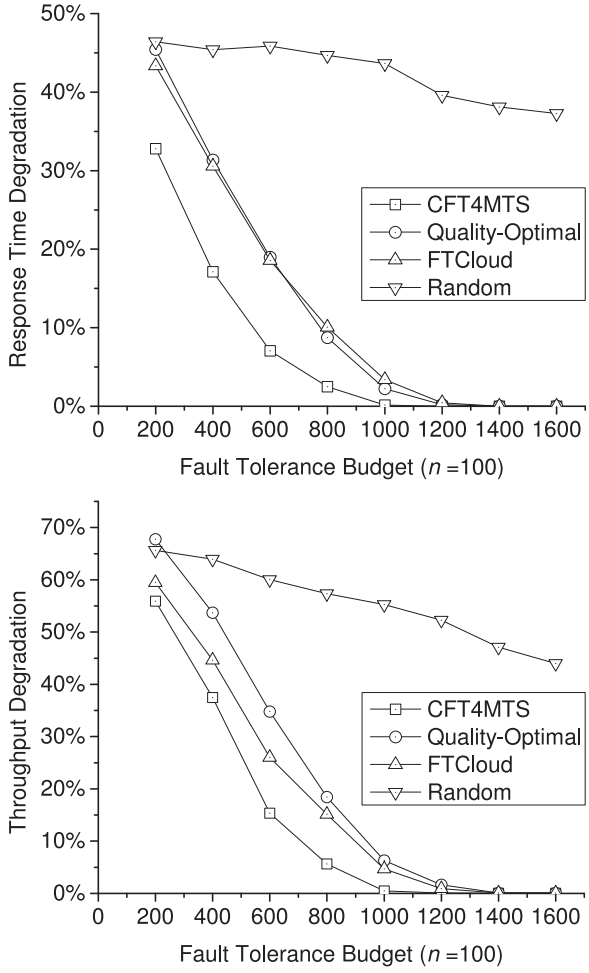


Fig. 9. Comparison in SBS-level quality degradation. (a) Degradation of response time; (b) Degradation of throughput.

observe the quality degradations in response time and throughput caused by runtime anomalies. In this set of experiments, the quality of a faulty service degrades to the maximum degradation level, which is represented by the quality degradation coefficient (see Definition 6). For a fixed number of service class n, the fault tolerance budget is increased in steps of 200. One anomaly is injected to the SBS in each test instance. The experimental results show that quality degradation of both quality parameters can be alleviated with all approaches when the budget increases, while CFT4MTS outperforms other three approaches significantly. Again, we use test instances with n = 100 as examples to demonstrate the results. As shown in Fig. 9a, in terms of response time degradation, Quality-Optimal gains a similar performance with FTCloud and outperforms it slightly when the budget exceeds about 600. CFT4MTS outperforms Quality-Optimal and FTCloud by 10 percent on average. Random, as expected, is beaten by other three counterparts significantly. From the perspective of throughput degradation, a similar result can be seen in Fig. 9b, where CFT4MTS shows the best performance and outperforms FTCloud and Quality-Optimal by 4.5 and 8.3 percent on average respectively.

The reason for the noticeable advantage of CFT4MTS over other approaches under a constrained fault tolerance budget is that with CFT4MTS, the component service that may cause more severe quality degradation has a higher

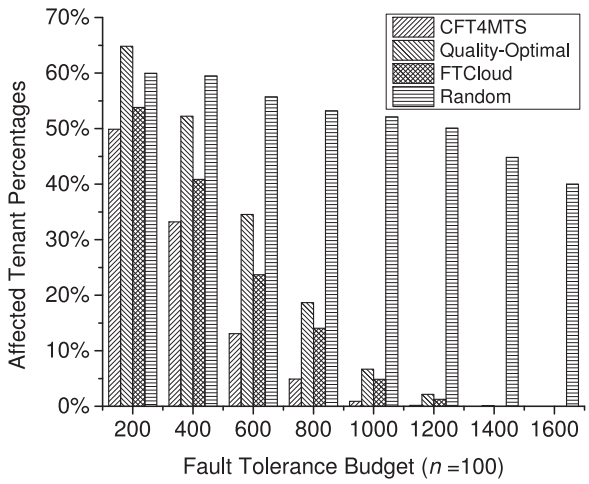


Fig. 10. Comparison in affected tenant percentages.

criticality, and thus is given higher priority in the allocation of service redundancy. As a result, the severity of system quality degradation caused by runtime anomalies can be alleviated. When the budget is high enough, all approaches except Random can achieve similar and satisfactory results.

5.2.3 Comparison in Affected Tenant Percentage

When an anomaly occurs to a component service of a multi-tenant SBS without a proper fault tolerance strategy, all the tenants sharing that component service may experience unexpected quality degradation. When quality degradation is unavoidable, the common practice is to limit the degradation to as few tenants as possible. With the same setup of the experiments in Section 5.2.2, we conduct experiments, where different approaches in comparison are employed, to observe the affected tenant percentage. The results are presented in Fig. 10. As demonstrated, CFT4MTS prevents significantly more tenants from being affected by runtime anomalies than the other approaches. When fault tolerance budget varies from 200 to 1,600, the affected tenant percentages obtained by all approaches decrease gradually and drop more quickly with CFT4MTS and FTCloud. Due to the consideration of service sharing across the tenants when calculating service criticality, CFT4MTS gains the most obvious advantage (over 20 percent on average) when budget is between 200 and 1,000.

5.2.4 Impact of Tenants' Quality Preferences

Playing an important role in service criticality and utility calculation, tenants' quality preferences impact the formulation of fault tolerance strategies. A set of experiments is conducted to evaluate this impact. Different from the experiments introduced in Section 5.2.2 where tenants' quality preferences were distributed normally and averagely, we first randomly generate a group of tenants with strong preferences for response time, e.g., $w_{rt} = 90\%$, and then execute the experiments and investigate the degradation in system response time, which is shown in Fig. 11a. After that, another group of tenants with weak preferences for response time, e.g., $w_{rt} = 10\%$, is generated, and the same experiments are conducted and results are presented in Fig. 11b. As demonstrated, CFT4MTS is able to capture tenants' various quality preferences for SBSs.

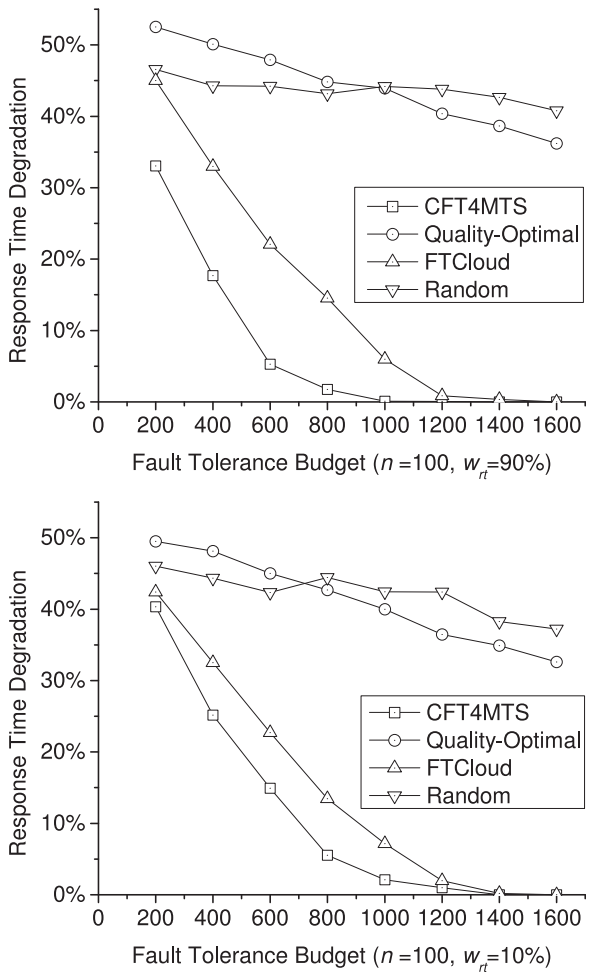


Fig. 11. Impact of tenants' quality preferences for response time. (a) Degradation of response time when $w_{rt} = 90\%$; (b) Degradation of response time when $w_{rt} = 10\%$.

Fig. 11a presents that the advantage of CFT4MTS remains and is even more obvious (approximately 10 percent on average higher than FTCloud) compared with Fig. 9a. Quality-Optimal is outperformed by Random when the fault tolerance budget is less than 1,000, and both of them are beaten by other two approaches significantly. Fig. 11b shows that the advantage of our CFT4MTS still exists but becomes less significant (approximately 5 percent on average higher than FTCloud), while other approaches achieve the similar performance compared with Fig. 11a. The more significant advantage of CFT4MTS in Fig. 11a is caused by tenants' higher preferences for response time, which prioritises response time in the allocation of service redundancy.

5.2.5 Impact of Environment Volatility

In this series of experiments, we evaluate the impacts of environment volatility on the quality of the SBS from two perspectives: the impacts of the number of anomalies occurred to the SBS concurrently and the quality degradation severity of a failed component service. The latter depends on the quality degradation coefficient (see Definition 6) in this paper. The experimental results show that when the operating environment is extremely volatile (e.g., large-scale service unavailability in the event of natural disaster), severe quality degradation is inevitable. In such

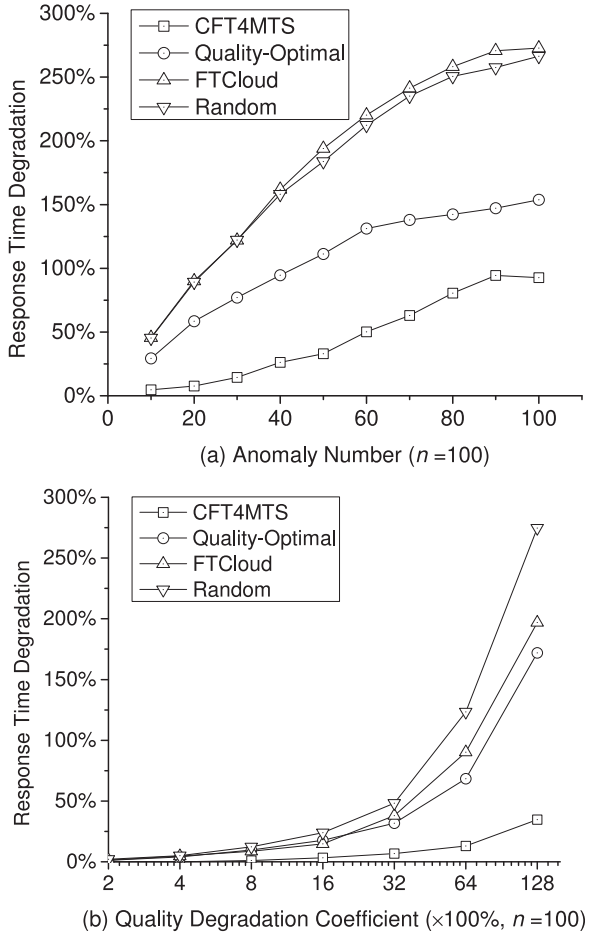


Fig. 12. Impact of environment volatility. (a) Impact of the number of anomalies that occurred to the SBS concurrently; (b) Impact of quality degradation coefficient.

an operating environment, CFT4MTS outperforms other approaches by relatively larger margins when the operating environment becomes more volatile.

A sample is shown in Fig. 12a, where we vary the number of anomalies in each test instance from 10 to n $(n = 100)$, fix fault tolerance budget at 500 and the quality degradation coefficient at 2, and observe the degradation (increase) in system response time. As depicted in Fig. 12a, when the number of concurrent anomalies rises, the response time degradation shown by other three approaches increases more rapidly than that presented by CFT4MTS, which shows a significant advantage over Quality-Optimal with a response time degradation margin of approximately 50 percent on average. With FTCloud and Random, which are much more vulnerable to the anomalies among the four approaches in this experiment, the response time degrades dramatically (roughly from 47 to 270 percent on average) along with the growth of the number of anomalies that occurred to the SBS concurrently.

We conduct another set of experiments with regards to the impact of quality degradation severity, which is determined by the quality degradation coefficient. As described in Section 5.1, the quality degradation coefficients for response time and throughput are defined by $(2)^h$ and $1-(2)^{-h}$ respectively. As an example, we use 100 service classes, fix fault tolerance budget at 500, and vary quality degradation coefficient of response time by increasing h from 1 to 7 in steps of 1. One anomaly is injected to the SBS in each test instance. Fig. 12b compares the increases in system response time when the four approaches are adopted. When the quality degradation coefficient varies from 200 to 12,800 percent, the increases in response time obtained by the four approaches are approximately 34.6 percent (CFT4MTS), 172.1 percent (Quality-Optimal), 196.8 percent (FTCloud), and 274.7 percent (Random) on average. Fig. 12b illustrates that quality degradations occurred and worsened regardless of the adopted approach. However, CFT4MTS has an obvious advantage over the other three.

TABLE 3
t-Test Results with Critical Value of 2.365
(95 Percent Confidence Interval)

Sample Data	t-values	p	Reject/Accept Null Hypothesis
Success Rates	3.254	0.0140	Reject
Response Time Degradation	2.991	0.0202	Reject
Throughput Degradation	3.014	0.0195	Reject
Affected Tenant Percentage	3.085	0.0177	Reject

5.2.6 Analysis of Statistical Significance

In order to assess the statistical significance of the advantages held by CFT4MTS over other approaches, we conduct t-test based on the results collected in the experiments in Sections 5.2.1 to 5.2.3 respectively. The t-test results are shown in Table 3, where all the null hypothesises that CFT4MTS has no difference to the second best approach in each set of experiments are rejected. Thus we can conclude that CFT4MTS produces significant advantages in effectiveness over other three approaches.

5.3 Efficiency Evaluation

In order to evaluate the efficiency of CFT4MTS, we conduct a set of experiments to assess its computational overhead. The impacts of two parameters are evaluated: the maximum number of redundant services for a component service and the number of service class. These two parameters have important influences on the complexity of the fault tolerance strategy formulation, which, as described in Section 4.6, is a NP-complete problem. The results show that with the increase in complexity, the computational overhead introduced by all approaches increases. However, CFT4MTS shows a better or similar performance compared with other approaches except Random, which is a naive and non-optimal approach.

We first evaluate the impacts of maximum redundant service number on the computational overhead. In this set of experiments, we fix n at 10, fault tolerance budget at 500 and change ν^{max} from 1 to 4 to vary the value of k_r $(k_r = 2\nu, 1 \leq \nu \leq \nu^{max})$, and then collect the average computation time achieved by the four approaches for finding solutions successfully. In order to examine the performance of these approaches in the worst-case scenarios and compare the efficiency on an equal basis, we let $\nu = \nu^{max}$ in each round. Fig. 13a shows that the computation times presented by all approaches grow rapidly along with the increase of ν^{max}. This is because the scale of service redundancy scheme grows with the increase of ν^{max} (here $\nu = \nu^{max}$), which

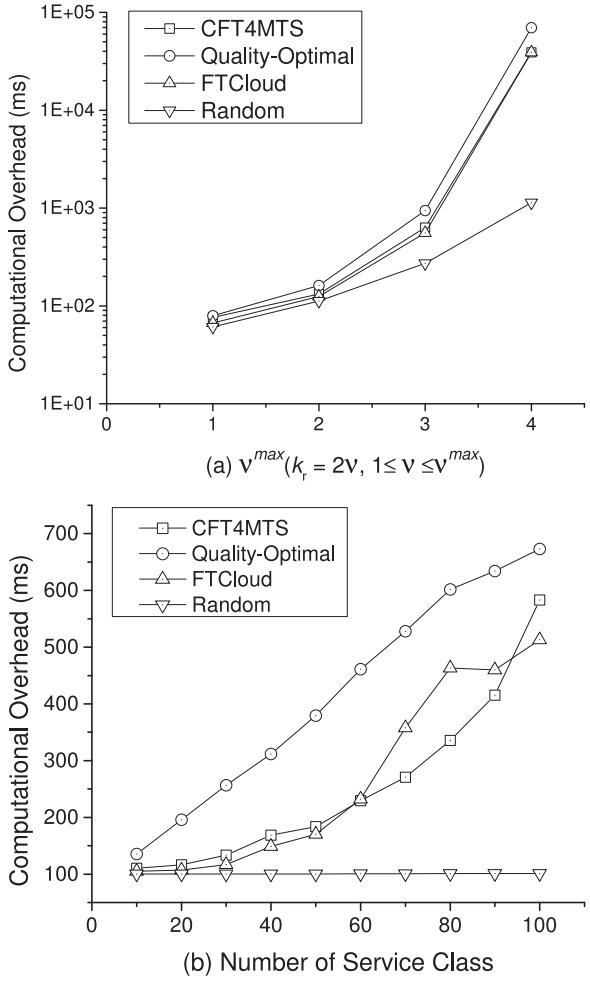


Fig. 13. Comparison in computational overhead. (a) Impact of ν^{max}; (b) Impact of the number of service class.

consequently increases the complexity of the optimisation problem and makes it increasingly difficult to find an optimal solution. However, a large ν^{max} is not necessary in practice considering the overhead it introduces and the improvement it brings to the system quality, which is analysed in Section 5.4.1.

Fig. 13b shows the impacts of service class number n on the computational overhead. We fix ν^{max} at 2 and fault tolerance budget at 500, and vary n from 10 to 100. The results show that the computation times consumed by all four approaches increase when n rises. The computational overheads introduced by CFT4MTS, Quality-Optimal and FTCloud increase much more significantly (from about 130 ms to approximately 600 ms on average) than that of Random, which stays stably at around 100 ms.

5.4 Threats to Validity

Here we discuss the threats to the validity of the evaluation on CFT4MTS.

5.4.1 Threats to Internal Validity

The main threat to the internal validity of our evaluation is the comprehensiveness of our experiments. In Section 5.2.4, we conducted experiments where two factors (fault tolerance budget and w_{rt}) changed simultaneously, but in other experiments, we simulated scenarios where the factors varied individually. More sophisticated scenarios could have been simulated, e.g., those factors also change at the same time. In those scenarios, we can predict the experimental results in general based on the results obtained. For instance, if the number of concurrent anomalies and the quality degradation coefficient increase at the same time, the rising trend of system response time would be more significant due to the cumulative effect caused by the simultaneous increases in these two factors. k_r is a factor that may impact the experimental comprehensiveness in this paper. It represents the maximum number of redundant services determined by ν^{max} in a service redundancy. we use a relatively small k_r in the experimental setup, because on one hand a large k_r would inevitably cause heavy computational overhead. On the other hand, deploying more services for fault tolerance introduces higher cost but may not yield significant improvement in system quality. For example, from the reliability point of view, the probability of that the component service and all redundant services fail at the same time is very small. We assume there are 3 redundant services deployed for a component service in a service redundancy and each service has a reliability of 90 percent, then theoretically the overall reliability of formulated service redundancy can reach 99.9999 percent, which is very high. Therefore, we believe that k_r is not a major limitation in practice.

The adjustments to existing approaches may also introduce threats to the internal validity of our evaluation. There may be other ways to adjust those approaches. The adjustments we made might not be most suitable. However, there are inherent limitations to existing approaches because they are designed for single tenant without considering service criticality in multi-tenant SBSs. This is shown by their obvious disadvantages compared with CFT4MTS in the experiments. Therefore, the experiments in this paper are able to demonstrate the inherent differences between existing approaches and CFT4MTS.

Another threat to the internal validity of our evaluation is whether the results are not biased by the use of experimental setup, such as the parameters selected and their variation, which may match the assumptions that CFT4MTS relies on (more than the alternative approaches). In the experiments, we used two representative quality parameters (response time and throughput). As we discussed in Section 4.1, other quality parameters can be used in the similar manner. In addition, these two parameters are also adopted as the quality constraints in the evaluation of other approaches. Other experimental setups, such as service fault injection method and quality degradation coefficient, affect all four approaches in the same way, which does not jeopardise the validity of the evaluation on our approach. Therefore, we can conclude that the experimental results in this paper are not biased.

5.4.2 Threats to External Validity

The main threats to the external validity of our evaluation are the representativeness of generated service compositions and candidate services in the experiments. In this paper, we randomly generated service compositions with the widely used composition patterns introduced in Section 3.1, which are also selected randomly. To evaluate

the scalability of CFT4MTS, we extended the motivating example by increasing the number of service classes in the SBS. In our experiments, we generated services randomly based on the QWS dataset, which has been widely used in the research on service-based systems. The cost and throughput of services are generated by following normal distribution. In fact, CFT4MTS does not rely on the absolute quality values of the services and their distributions. Thus, the experimental results presented in this paper are promising in demonstrating the advantages of CFT4MTS.

Another potential threat to external validity is that the non-linear quality constraints and parameters are not considered in the COP model. For example, a quadratic expression can be used as quality constraint or optimisation objective of the COP, and the values of a quality parameter can be nominal, such as the reputation of a service. However, this threat is marginal. On one hand, linear quality constraints and parameters have been the same premise of many other research efforts, such as the work presented in [5], [11], [17], [21], [33]. Considering only the linear quality constraints and parameters does not influence the comparison in the experiments between CFT4MTS and existing approaches. On the other hand, non-linear quality constraints and parameters can be handled with various techniques [36], e.g., Sequential Quadratic Programming [37], or transformed into the linear expressions [38]. Therefore, the consideration of linear quality constraints and parameters is not a significant limitation to our approach.

6 Related Work

Improving the ability of software systems to accommodate runtime anomalies has become a critical concern in software engineering. Various modeling and analysis techniques have emerged in recent years, such as the self-adaptive software systems [39], [40], [41], [42] and software fault tolerance [30], [32]. Service-oriented architecture (SOA) and cloud computing have raised new challenges to SBSs in tackling unexpected failures cost-effectively through service selection, service adaptation, and fault tolerance using functionally equivalent services.

6.1 Service Selection for SBSs

Over the past decade, quality-aware service selection for SBSs has been an active topic in service-oriented software engineering. In [11] the authors present AgFlow, a middleware platform for Web service composition, which can maximise client satisfaction with utility over quality attributes, while meeting clients' multi-dimensional quality requirements. Integer Programming is used to find the solution to the optimisation problem. The authors propose in [21] an architecture in which quality-aware service selection is modelled respectively as a 0-1 knapsack problem (MMKP) in the combinatorial model and a multi-constrained optimal path (MCOP) problem in the graph model. Heuristic algorithms are adopted to find near optimal solutions for different composition structures in polynomial complexity time. The work in [43] proposes a heuristic approach to find the close-to-optimal solution for service selection by decomposing global quality constraints into local quality constraints, which aims at applications with dynamic changes and real-time requirements. In their work [22], Skyline techniques are applied to quality-aware service composition to reduce search space of candidate services. In [5] we present CASS, a model that selects service based on iterative multi-attribute combinatorial auction. The complementarities between services are taken into account. Aiming at achieving multi-tenancy at different maturity levels, we also propose in [6] several approaches to quality-aware service selection for multi-tenant SBS by modelling the problem as different constraint optimisation problems, one for each maturity level of multi-tenancy.

6.2 Service Adaptation for SBSs

The research in service selection has promoted the exploration in quality-aware service adaptation for SBSs. In [11], the middleware platform AgFlow is designed with the ability of adaptation. When exceptions occur at runtime in an execution plan built with global selection, replanning procedure may be triggered to ensure that the end-to-end quality remains optimal. The authors in [33] propose a framework named MOSES that supports quality-driven adaptation of a service-oriented system at runtime. The optimal service adaptation problem is formulated as a Linear Programming (LP) problem. The Quality-Optimal approach in this paper is originated from MOSES. In [12] the authors present an optimisation approach for the composition of Web services, and adaptive reoptimisation is adopted to fulfil the variable quality constraints at runtime. The authors of [13] proposes a decentralised self-adaptation mechanism using market-based heuristics for service-based applications in the cloud. Continuous double auction is used to select services for composition to meet the changing quality requirements. The authors of [2] proposed QoSMOS, a tool-supported framework for the development of adaptive service-based systems. QoSMOS first translates high-level QoS requirements for the SBS to probabilistic temporal logic formulate, which are then analysed to identify and enforce optimal system configurations. Three mapping patterns of abstract to candidate services are studied: single, sequential one-to-many, and parallel one-to-many mapping.

The above approaches may be unsuitable for the time-critical scenarios due to their high computational cost, especially in large-scale scenarios. One promising way to tackle this issue is to build an SBS with fault tolerance capability, which has been attracting great interest in recent years.

6.3 Fault Tolerance for SBSs

In order to improve the efficiency in fixing runtime anomalies in SBSs, some researchers have looked into the use of service redundancy for fault tolerance to mitigate the impact of service anomalies.

Inspired by the success of hardware redundancy for tolerating hardware failures, software redundancy by using multiple versions of independently developed software has become a widely accepted means to improve the reliability and availability of a software system [44]. Service redundancy usually uses services with similar or identical interfaces as redundant replicas aiming at fault tolerance and performance improvement of SBSs. In [15], the authors present a distributed replication strategy evaluation and selection

framework for fault tolerant Web services. Time redundancy, space redundancy and the combinations of the two are studied. Similar strategies are used in [33] for service adaptation to obtain quality levels that could not be achievable by single service. The work presented in [45] investigates the inherent redundancy and diversity of services, base on which, the authors propose solutions to improve the dependability of SBSs and two invocation strategies of redundant services are employed: sequential and simultaneous. In [46], a framework is proposed for selecting the optimal fault tolerance strategy for an SBS, which is modelled as an optimisation problem with user requirements as local and global constraints, and a heuristic algorithm is used to find the solution. These works and the like all assume that all the services are equally important to the service providers, and thus probably make them overpay for the services which are not critical, or cannot mitigate the system failures effectively because of the absence of protection to critical services under a budget limit.

In order to solve the abovementioned problem, some approaches used for measuring the importance of a service have been studied. Recent years, in the area of software engineering, various reliability-based and structure-based importance measures have been proposed [27], [28], [29], which can be used to identify and rank the important components in a software system. These works have shed light on the issue of important service identification in the SBSs. In [11] the authors use the critical path in terms of execution duration to identify the critical services in a service composition. In [18] the authors present FTCloud, a component ranking framework for fault-tolerant cloud applications. Algorithms based on system structure information and prior knowledge are employed to identify and rank the significant components in a cloud application. In [34] and [3], the authors propose CriMon, an approach that evaluates the criticalities of both execution paths and component services based on the concept of probabilistic critical path. However, the existing approaches mostly assess service criticality from one dimension of quality, such as failure rate or response time, which is not suitable for the cloud environment characterised by multiple tenants with preferences for multiple dimensional quality.

In order to tackle above issues, we propose in this paper an approach for fault tolerance strategy formulation for a multi-tenant SBS based on criticality. We evaluate the criticality of a component service by combining two criticalities: quality-based criticality and tenant-based criticality, based on which, the component services are ranked. Then the redundant services are selected for critical component services and run with them in certain redundancy mode, which facilitates fault tolerance of the SBS at runtime.

7 Conclusion and Future Work

Quality-aware cost-effective fault tolerance for multi-tenant SBS is a critical issue in the dynamic and volatile cloud environments. This paper proposes a novel approach named CFT4MTS that formulates fault tolerance strategies for SBSs based on the service criticality, which is evaluated by analysing the quality of component services, multiple tenants' preferences for multi-dimensional system quality, and the service sharing across the tenants in the SBS. Component services are ranked by their criticalities and critical component services are given the priorities in the allocation of service redundancy. By doing so, the fault tolerance capabilities of SBSs can be improved significantly. To evaluate the effectiveness and efficiency of CFT4MTS, we have conducted a series of experiments, where we compared CFT4MTS with three representative approaches and analysed the impacts of various factors. The results show that CFT4MTS can formulate fault tolerance strategies for SBSs under a limited budget, which effectively and efficiently reduce the risk of system quality degradation for multi-tenant SBSs.

As our future work, we will investigate the runtime adaptation of the formulated fault tolerance strategy. We will also consider the handling of non-linear quality constraints and non-linear quality parameters in the COP model, with which the formulation and adaptation of fault tolerance strategies become a non-linear programming problem.

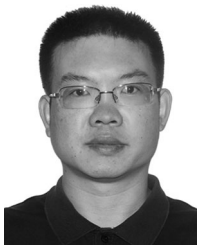
Acknowledgments

This work is partly supported by Australian Research Council Discovery Project DP150101775. Qiang He is the corresponding author.

References

[1] L. Baresi and S. Guinea, "Self-supervising BPEL processes," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 247–263, Mar./Apr. 2011.
[2] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, May/Jun. 2011.
[3] Q. He, J. Han, Y. Yang, H. Jin, J.-G. Schneider, and S. Versteeg, "Formulating cost-effective monitoring strategies for service-based systems," *IEEE Trans. Softw. Eng.*, vol. 40, no. 5, pp. 461–482, May 2014.
[4] ProgrammableWeb, "Growth in web APIs from 2005 to 2013," 2014. [Online]. Available: http://www.programmableweb.com/api-research
[5] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 2, pp. 192–215, Feb. 2014.
[6] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin, "QoS-driven service selection for multi-tenant SaaS," in *Proc. 5th IEEE Int. Conf. Cloud Comput.*, 2012, pp. 566–573.
[7] M. Armbrust, et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[8] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proc. 5th Int. Joint Conf. INC IMS IDC*, 2009, pp. 44–51.
[9] N. Poggi, D. Carrera, R. Gavalda, and E. Ayguadé, "Non-intrusive estimation of QoS degradation impact on e-commerce user satisfaction," in *Proc. 10th IEEE Int. Symp. Netw. Comput. Appl.*, 2011, pp. 179–186.
[10] J. Barr, A. Narin, and J. Varia, "Building fault-tolerant applications on AWS," 2011. [Online]. Available: http://media.amazonwebservices.com/AWS_Building_Fault_Tolerant_Applications.pdf
[11] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.
[12] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.
[13] V. Nallur and R. Bahsoon, "A decentralized self-adaptation mechanism for service-based applications in the cloud," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 591–612, May 2013.
[14] K. Birman, R. van Renesse, and W. Vogels, "Adding high availability and autonomic behavior to web services," in *Proc. 26th Int. Conf. Softw. Eng.*, 2004, pp. 17–26.

[15] Z. Zheng and M. R. Lyu, "A distributed replication strategy evaluation and selection framework for fault tolerant web services," in *Proc. 6th IEEE Int. Conf. Web Serv.*, 2008, pp. 145–152.
[16] W. Zhao, P. Melliar-Smith, and L. E. Moser, "Fault tolerance middleware for cloud computing," in *Proc. 3rd IEEE Int. Conf. Cloud Comput.*, 2010, pp. 67–74.
[17] P. A. Bonatti and P. Festa, "On optimal service selection," in *Proc. 14th Int. Conf. World Wide Web*, 2005, pp. 530–538.
[18] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 540–550, Oct.–Dec. 2012.
[19] Q. He, et al., "QoS-aware service selection for customisable multi-tenant service-based systems: Maturity and approaches," in *Proc. 8th IEEE Int. Conf. Cloud Comput.*, 2015, pp. 237–244.
[20] Y. Wang, Q. He, and Y. Yang, "QoS-aware service recommendation for multi-tenant SaaS on the cloud," in *Proc. 12th IEEE Int. Conf. Service Comput.*, 2015, pp. 178–185.
[21] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end QoS constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007, Art. no. 6.
[22] M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for QoS-based web service composition," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 11–20.
[23] M. Zeleny and J. L. Cochrane, *Multiple Criteria Decision Making*. Columbia, SC, USA: Univ. South Carolina Press, 1973.
[24] A. Saltelli et al. *Global Sensitivity Analysis: The Primer*. Hoboken, NJ, USA: Wiley, 2008.
[25] G. N. Rodrigues, D. S. Rosenblum, and S. Uchitel, "Sensitivity analysis for a scenario-based reliability prediction model," in *Proc. Workshop Archit. Depend. Syst.*, 2005, vol. 30, no. 4, pp. 1–5.
[26] M. Harman, J. Krinke, I. Medina-Bulo, F. Palomo-Lozano, J. Ren, and S. Yoo, "Exact scalable sensitivity analysis for the next release problem," *ACM Trans. Softw. Eng. Methodology*, vol. 23, no. 2, 2014, Art. no. 19.
[27] F. C. Meng, "Comparing the importance of system components by some structural characteristics," *IEEE Trans. Rel.*, vol. 45, no. 1, pp. 59–65, Mar. 1996.
[28] J. Freixas and M. Pons, "Identifying optimal components in a reliability system," *IEEE Trans. Rel.*, vol. 57, no. 1, pp. 163–170, Mar. 2008.
[29] H. Peng, D. W. Coit, and Q. Feng, "Component reliability criticality or importance measures for systems with degrading components," *IEEE Trans. Rel.*, vol. 61, no. 1, pp. 4–12, Mar. 2012.
[30] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1491–1501, Dec. 1985.
[31] T. J. Shimeall and N. G. Leveson, "An empirical comparison of software fault tolerance and fault elimination," *IEEE Trans. Softw. Eng.*, vol. 17, no. 2, pp. 173–182, Feb. 1991.
[32] B. Randell, "System structure for software fault tolerance," *ACM SIGPLAN Notices*, vol. 10, no. 6, pp. 437–449, 1975.
[33] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A framework for QoS driven runtime adaptation of service-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1138–1159, Sep./Oct. 2012.
[34] Q. He, J. Han, Y. Yang, J.-G. Schneider, H. Jin, and S. Versteeg, "Probabilistic critical path identification for cost-effective monitoring of service-based systems," in *Proc. 9th IEEE Int. Conf. Service Comput.*, 2012, pp. 178–185.
[35] E. Al-Masri and Q. H. Mahmoud, "Discovering the best web service," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 1257–1258.
[36] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[37] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.
[38] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[39] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adaptive Syst.*, vol. 4, no. 2, 2009, Art. no. 14.
[40] G. Cugola, L. S. Pinto, and G. Tamburrelli, "QoS-aware adaptive service orchestrations," in *Proc. 19th IEEE Int. Conf. Web Serv.*, 2012, pp. 440–447.
[41] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, 2012.
[42] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *Proc. 10th Joint Meet. Found. Softw. Eng.*, 2015, pp. 1–12.
[43] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 881–890.
[44] D. E. Eckhardt, et al., "An experimental evaluation of software redundancy as a strategy for improving reliability," *IEEE Trans. Softw. Eng.*, vol. 17, no. 7, pp. 692–702, Jul. 1991.
[45] A. Gorbenko, V. Kharchenko, and A. Romanovsky, "Using inherent service redundancy and diversity to ensure web services dependability," in *Methods, Models and Tools for Fault Tolerance*. Berlin, Germany: Springer, 2009, pp. 324–341.
[46] Z. Zheng and M. R. Lyu, "Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 219–232, Jan. 2015.



Yanchun Wang received the MEng degree in computer science from Beihang University, Beijing, China, in 2006. He is currently working toward the PhD degree at Swinburne University of Technology, Australia. His research interests include services computing and cloud computing.

Qiang He received the first PhD degree from Swinburne University of Technology (SUT), Australia, in 2009 and the second PhD degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2010. He is a lecturer with Swinburne University of Technology. His research interests include software engineering, cloud computing, services computing, big data analytics, and green computing. More details about his research can be found at https://sites.google.com/site/heqiang/.

Dayong Ye received the MSc and PhD degrees both from the University of Wollongong, Australia, in 2009 and 2013, respectively. He is a research fellow with Swinburne University of Technology, Australia. His research interests focus on service-oriented computing, self-organisation, and multi-agent systems.

Yun Yang received the PhD degree from the University of Queensland, Australia, in 1992. He is a full professor with Swinburne University of Technology. His research interests include software engineering, cloud computing, workflow systems, and service-oriented computing. More details about his research can be found at http://www.ict.swin.edu.au/personal/yyang/.