Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 407267, 12 pages
http://dx.doi.org/10.1155/2013/407267

Research Article

A Fast Optimization Method for Reliability and Performance of Cloud Services Composition Application

Zhao Wu,[1] Naixue Xiong,[2] Yannong Huang,[1] Qiong Gu,[1] Chunyang Hu,[1] Zhongbo Wu,[1] and Bo Hang[1]

[1] *School of Mathematics and Computer Science, Hubei University of Arts and Science, Xiangyang 441053, China*
[2] *School of Computer Science, Colorado Technical University, Colorado Springs, CO 80907, USA*

Correspondence should be addressed to Naixue Xiong; nxiong@coloradotech.edu

Received 15 April 2013; Accepted 13 September 2013

Academic Editor: Rung Ching Chen

At present the cloud computing is one of the newest trends of distributed computation, which is propelling another important revolution of software industry. The cloud services composition is one of the key techniques in software development. The optimization for reliability and performance of cloud services composition application, which is a typical stochastic optimization problem, is confronted with severe challenges due to its randomness and long transaction, as well as the characteristics of the cloud computing resources such as openness and dynamic. The traditional reliability and performance optimization techniques, for example, Markov model and state space analysis and so forth, have some defects such as being too time consuming and easy to cause state space explosion and unsatisfied the assumptions of component execution independence. To overcome these defects, we propose a fast optimization method for reliability and performance of cloud services composition application based on universal generating function and genetic algorithm in this paper. At first, a reliability and performance model for cloud service composition application based on the multiple state system theory is presented. Then the reliability and performance definition based on universal generating function is proposed. Based on this, a fast reliability and performance optimization algorithm is presented. In the end, the illustrative examples are given.

1. Introduction

Cloud computing is an emerging trend for the provision of IT infrastructure as services, with the potential of transforming the way of offering business services [1]. Based on cloud computing platform, software development becomes prominent and accessible for all without the expensive investing in hardware resources and the managing and maintaining costs.

On cloud computing platform, the cloud services composition (CSC) is a fashionable approach of software development based on cloud services [2–4]. In the framework of CSC, cloud services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the web.

How to select and integrate cloud services to satisfy user's functional requirements is an important issue, which has widely attracted attention of researchers [5]. Great progress has been made in this field [6–8]. However, little research focused on reliability model and simulation for CSC. Recently, there has been growing interest in this field. Methods and technologies related to reliability model and simulation for CSC have attracted attention because they can forecast the QoS that users will obtain from CSC [9–11]. In addition, it is helpful to analyze whether there are some reliability bottlenecks within CSC applications. Thus, reliability prediction is the basis of reliability optimization for the CSC applications.

The service-oriented architecture (SOA) is the most representative technological architecture to build the cloud services application on cloud computing platform [12–14]. However, because SOA supposed by services composition technique is of dynamic and cooperative essential characteristic, the traditional software reliability prediction methods are not suitable to the cloud services application based on SOA.

From the aspect of software architecture, cloud services application is a kind of Internetware based on cloud services,

which is built by cloud services composition technique [15, 16]. As a kind of abstract of distributed software system running on the Internet which is opened, dynamic, and difficult to control, there are many differences between the Internetware and traditional software system, such as structure, operation mechanism, correctness guarantees, development method, and life cycle. Due to the static, closed, and controllable running environment, the traditional software model is of finite autonomy, fixed encapsulation, monotonic interaction, tightly coupled structure, and offline evolution. Being different from the traditional software model, the cloud services application, as a kind of Internetware, exists in each node on the cloud service platform with a subjective software service form. In the running environment, which is opened, dynamic, and difficult to control, the cloud services application has some new characters, such as flexible evolution, continuous reaction, and multitarget self-adaption. Due to being difficult to adapt to these new characters, traditional software reliability assurance methods cannot be adopted for the cloud services application which is built based on service composition technique. Quite different to traditional software reliability assurance technique, the reliability assurance method for the cloud services application pays more attention to the mechanism of flexible reliability measure, predication and self-adapting based on summative evaluation of operation information in opened running environment [17, 18]. So, the fast reliability prediction method for the cloud services application has great theory research value.

From the aspect of software online evolution, cloud services application confronts fast and continuous change of user's requirement and running environment. So, cloud services application must have the ability to apperceive the changes in outrunning environment and dynamically evolve according to functionality and performance requirement with this kind of change. In order to provide better QoS to users, cloud services application must have more adaptability to collect various changes realtimely and adjust oneself online according to preestablished strategies in runtime [19]. However, with the closed, controllable, and static user's requirement in the background, traditional software reliability prediction methods lack the ability to dynamically adapt themselves to the changes of running environment and user's requirement. Therefore, it cannot be employed in the reliability prediction for cloud services application. So, the fast reliability prediction method for the cloud services application has important realistic technology requirement.

At the present time, the researches on reliability prediction for cloud services application are still just starting Due to the opened and dynamic running environment, continuous variable user's requirement, randomly selected member services and its own characters of loose coupling and long transaction, the severe challenges are confronted the reliability prediction for cloud services application, which is seriously restricting the further development, application, and extension of cloud services application. In the face of urgent demands of high reliable cloud services application from many government, economy, and commerce fields such as e-government, e-commerce, and e-bank, the fast reliability optimization becomes the key to promote the successful development, application, and extension of the cloud services application.

Facing the challenge, this paper researches the reliability model of cloud services application. On this basis, a fast reliability optimization for cloud services application is presented. The paper is organized as follows. Section 2 presents the reliability model for cloud service application based on the multiple state system (MSS) theory. The reliability and performance of cloud service and cloud services composition application are defined in Section 3. Section 4 presents a reliability and performance model for cloud services composition application based on universal generating function (referred to as UGF) technique. A fast reliability optimization algorithm by using the UGF technique is presented in Section 5. Section 6 provides some illustrative examples.

2. Reliability Model for Cloud Service Composition Application

2.1. Multiple State System Theory. The MSS was introduced in the middle of the 1970s in [20–23]. In these works, the basic concepts of MSS reliability were primarily formulated, the system structure function was defined, and its properties were initially studied. The notions of minimal cut set and minimal path set were introduced in the MSS context, as well as the notions of coherence and element relevancy.

Some systems can perform their tasks with various distinguished levels of efficiency usually referred to as performance rates. A system that can have a finite number of performance rates is called a multistate system. Any system consisting of different units that have a cumulative effect on the entire system performance has to be considered as a MSS [24]. So the cloud service application can be regard as a multiple state system.

MSS reliability analysis relates to systems for which one cannot formulate an "all or nothing" type of failure criterion. Such systems are able to perform their task with partial performance (intensity of the task accomplishment). Failures of some system elements lead only to the degradation of the system performance.

2.2. Reliability and Performance Definition for Cloud Services Composition Application. From the aspect of users, the reliability of cloud service application can be defined as the probability that its performance rates satisfy user's requirements, described as a vector pair $(\mathbf{w}, \mathbf{q})$, where $\mathbf{w} = \{w_1, w_2, \ldots, w_M\}$ is a vector of user's requirement rates w_j, $(j = 1, \ldots, M)$, and $\mathbf{q} = \{q_1, q_2, \ldots, q_M\}$ is the vector of steady state probability $q_j = \Pr\{W = w_j\}$, $(j = 1, \ldots, M)$, according to a certain user's requirement rate, where W is a random variable that represents the performance rates of cloud service application.

Based on the above definition, the reliability function of cloud service application under steady state can be defined as

$$R(t) = \Pr\left\{T_f \geq t \mid F(G(0), W(0)) \geq 0\right\}. \quad (1)$$

And the one under transient state can be defined as

$$R(t) = \Pr\left\{F(G(t), W(t)) \geq 0\right\}, \quad (2)$$

where $G(t)$ is the integral performance rate of cloud service application.

In the interval $[0, T]$, the reliability function of cloud service application can be defined as

$$R_T = \frac{1}{T}\int_0^T 1\left(F\left(G\left(t\right), W\left(t\right)\right) \geq 0\right) dt. \tag{3}$$

Based on the above definition, the reliability function of cloud service application under dynamically changing user's requirements can be defined as

$$\begin{aligned} R\left(\mathbf{w}, \mathbf{q}\right) &= \sum_{m=1}^{M} R\left(w_m\right) q_m \\ &= \sum_{m=1}^{M} q_m \sum_{k=1}^{K} p_k 1\left(F\left(g_k, w_m\right) \geq 0\right). \end{aligned} \tag{4}$$

In order to calculate the probability distribution of reliability, failure time T_f, time between failures T_b, and failure number N_T are defined.

2.3. Probability Distribution of Performance Rates for Cloud Service. Furthermore, the performance rates for cloud service can be defined. According to its performance rates, the cloud service j to build a cloud service application can be of k_j kinds of various states, described by $\mathbf{g}_j = \{g_{j1}, g_{j2}, \ldots, g_{jk_j}\}$, where g_{ji} is the performance rate of cloud service j under the state i, $i \in \{1, 2, \ldots, k_j\}$. The performance rate $G_j(t)$ corresponding to the cloud service j in any time $t \geq 0$ is a random variable that gets the value from $\mathbf{g}_j$: $G_j(t) \in \mathbf{g}_j$. The probability of performance rates of the cloud service j under various states in any time t can be described as a set,

$$\mathbf{p}_j\left(t\right) = \left\{p_{j1}\left(t\right), g_{j2}\left(t\right), \ldots, g_{jk_j}\left(t\right)\right\}, \tag{5}$$

where $p_{ji}(t) = \Pr\{G_j(t) = g_{ji}\}$. Because cloud service j is in one and only one of k_j kinds of various states in any time t, these states form a mutual exclusion events complete set. Therefore, the formula $\sum_{i=1}^{k_j} p_{ji}(t) = 1, (0 \leq t \leq T)$ is satisfied. In the end, the set of value pairs $\langle g_{ji}, p_{ji}(t)\rangle$ completely determines the probability distribution of performance rates corresponding to a cloud service j in any time t.

2.4. Structure Function of Performance Rates for Cloud Service Application. Based on the above definition of the performance rates of cloud service application and cloud service, the structure function of cloud service application can be defined. Let

$$L^n = \left\{g_{11}, \ldots, g_{1k_1}\right\} \times \left\{g_{21}, \ldots, g_{2k_2}\right\} \times \cdots \times \left\{g_{n1}, \ldots, g_{nk_n}\right\} \tag{6}$$

be the possible combinations of performance rates of all cloud services and $M = \{g_1, \ldots, g_k\}$ the possible values range of performance rates of cloud service application. Then the transform function $\phi(G_1(t), \ldots, G_n(t)) : L^n \rightarrow M$, called the structure function of cloud service application, can map the performance rates space of cloud services into one of cloud service applications. Hence, a general reliability model of cloud service application can be defined as

$$\mathbf{g}_j, \mathbf{p}_j\left(t\right), \quad 1 \leq j \leq n, \quad \phi\left(G_1\left(t\right), \ldots, G_n\left(t\right)\right). \tag{7}$$

The structure function of cloud service application establishes a feasible way to calculate the reliability of cloud service application using one of cloud services.

3. Reliability and Performance Definition for Cloud Service Composition Application Based on UGF

3.1. UGF Technique. The methods of MSS reliability assessment are based on four different approaches: (1) an extension of the Boolean models to the multivalued case; (2) the stochastic process (mainly Markov and semi-Markov) approach; (3) the Monte-Carlo simulation technique; and (4) the UGF approach.

The approach based on the extension of Boolean models is historically the first method that was developed and applied for the MSS reliability evaluation. It is based on the natural expansion of the Boolean methods to the multistate systems.

The stochastic process methods that are widely used for the MSS reliability analysis are more universal. The method can be applied only to relatively small MSS because the number of system states increases dramatically with the increase in the number of system elements.

Even though almost every real world MSS can be represented by the Monte-Carlo simulation for the reliability assessment, the main disadvantages of this approach are the time and expenses involved in the development and execution of the model.

The computational burden is the crucial factor when one solves optimization problems where the reliability measures have to be evaluated for a great number of possible solutions along the search process. This makes using the three above-mentioned methods in reliability optimization problematic. On the contrary, the UGF technique is fast enough. This technique allows one to find the entire MSS performance distribution based on the performance distribution of its elements by suing a fast algebraic procedure. An analyst can use the same recursive procedures for MSS with a different physical nature of performance and different types of element interaction.

For the above reasons, we choose UGF technique to study a fast reliability optimization method for cloud services composition network. The UGF generalizes the technique that is based on using a well-known ordinary generating function. The basic ideas of the method were introduced by Ushakov [25]. The approach proved to be very convenient for numerical realization. It requires relatively small computational resources for evaluating MSS reliability indices and, therefore, can be used in complexes reliability optimization algorithms. Because the relationship between system state probability and system output performance rates can be expressed definitely by UGF, and the u-function of system can be obtained by calculating the u-function of components simply, UGF is

approved as an efficient reliability assessment approach that is suitable to various MMS. Therefore, UGF can be successfully applied for the reliability assessment and optimization of MMS.

In most studies on the prediction and optimization of system reliability based on the UGF, the system structure and composite form of research object are relatively simple, such as electric power system and mechanical system. The presented design methods and calculation methods of the u-function composite operators are only applicable to some simple structure forms, such as series, parallel, series parallel hybrid, and bridge structure, which limits the application range of the UGF method. Different from the above research objects, the cloud services composition is of complex, flexible and dynamic structure form. The adaptability to complex, flexible and dynamic system, such as the cloud services composition application, becomes advantage and characteristic of our presented method.

3.2. Reliability Definition of Cloud Service Composition Application Based on UGF. Based on the reliability model for cloud service application described in Section 2, the reliability of cloud service composition can be defined by UGF. The general form of the definition is as follows.

The reliability of a cloud service composition (or a cloud service) is a random variable X. Therefore, the corresponding u-function can be defined as

$$u(z) = \sum_{k=1}^{K} p_k \cdot z^{X_k}, \tag{8}$$

where the discrete variable X has K possible values and p_k is the reliability when X is in the state X_k. Based on this definition, the reliability of a cloud service composition (or a cloud service) can be expressed as

$$U(t,z) = \sum_{k=1}^{K} p_k(t) \cdot z^{G_k}. \tag{9}$$

Because $U(z)$ is correlative with the state probability p_k and the reliability rate G_k, which correspond to the cloud service composition (or a cloud service), describes the reliability of cloud service composition (or a cloud service). On this basis, we can define related performance operators furthermore, such as usability operator δ_A, output performance operator δ_G and unfinished performance operator δ_U, to describe related reliability indexes.

Based on the above performance operators, the related reliability indexes for cloud service composition (or a cloud service) can be defined as follows.

(i) The usability is defined as

$$E_A = E_A(W,q) = \sum_{m=1}^{M} q_m \cdot \delta_R(U(z), F, W_m). \tag{10}$$

(ii) The output performance expectation is defined as

$$\begin{aligned} E_G &= \delta_G(U(z)) = \delta_G\left(\sum_{k=1}^{K} p_k \cdot z^{G_k}\right) \\ &= \frac{dU}{dz}(1) = \sum_{k=1}^{K} p_k \cdot G_k. \end{aligned} \tag{11}$$

(iii) The unfinished performance requirement is defined as:

$$E_U(W,q) = \sum_{m=1}^{M} q_m \cdot \delta_U(U(z), F, W_m), \tag{12}$$

where

$$\begin{aligned} &\delta_U(U(z), F, W_m) \\ &\quad = \delta_U\left(\sum_{k=1}^{K} p_k \cdot z^{G_k}, F, W_m\right) \\ &\quad = \sum_{k=1}^{K} p_k \cdot \max\{-F(G_k, W_m), 0\}. \end{aligned} \tag{13}$$

3.3. Composite Operators of Reliability and Performance Indexes Based on UGF. Based on the above reliability definition expressed by UGF for cloud services, the u-function composite operators Ω can be designed for various performance indexes of the diverse composition patterns. The reliability of cloud service composition can be worked out based on the Ω calculation of cloud services' reliability.

Two rules must be satisfied in the design of u-function composite operators Ω as follows:

$$\begin{aligned} &(1)\quad \Omega(U_1(z), \ldots, U_k(z), U_{k+1}(z), \ldots, U_n(z)) \\ &\qquad = \Omega(U_1(z), \ldots, U_{k+1}(z), U_k(z), \ldots, U_n(z)); \\ &(2)\quad \Omega(U_1(z), \ldots, U_k(z), U_{k+1}(z), \ldots, U_n(z)) \\ &\qquad = \Omega(\Omega(U_1(z), \ldots, U_k(z)), \Omega(U_{k+1}(z), \ldots, U_n(z))). \end{aligned} \tag{14}$$

The generic form of composite operators Ω can be expressed as

$$\Omega\left(\sum_{\forall k} p_k \cdot z^{G_k}, \sum_{\forall l} p_l \cdot z^{G_l}\right) = \sum_{\forall k}\sum_{\forall l} p_k \cdot p_l \cdot z^{f(G_k, G_l)}, \tag{15}$$

where $f(G_k, G_l)$ can be defined according to the performance indexes and composition structures of the cloud service application.

4. Reliability and Performance Model of Cloud Services Composition Application Based on UGF

4.1. Fault Tolerant Model of Cloud Services Composition Application. To strengthen the capability of fault tolerant of cloud

service composition applications to improve its reliability and performance, the component duplication technique has been introduced into the design of cloud service composition applications. By deploying a number of functionally equivalent software versions for each cloud service, the cloud service composition applications can avoid the global failure due to the fault of one cloud service as far as possible. We assume that n_c functionally equivalent software versions are available for each cloud service c. Each software version i has an estimated reliability r_{ci} and response time τ_{ci} (it includes the execution time of software version and the network transmission time used transfer computing results to other software version, end users, etc.). Failures of software versions in each cloud service are statistically independent, as well as the total failures of the different cloud services.

In many cases, the information about the software version's reliability and the response time is available from separate testing and/or reliability prediction models. This information can be incorporated into a fault-tolerant program model in order to obtain an evaluation of its reliability and performance.

According to the generally accepted model, the cloud service composition application consists of C cloud services. Each cloud service performs a subtask and the sequential execution of the cloud services performs a major task.

To assure that all of the computing tasks can correctly be executed by cloud service, the cloud services broker (referred to as CSB) and the check mechanism are established in the cloud service composition application. The check mechanism presumes that software versions send their computing results to the CSB. Then the CSB compares received computing results with each other. The CSB sends the computing results to the next cloud service in the service flow, if at least k out of n computing results agree. Otherwise, the CSB discards these received computing results and recalls the cloud services. If the consistent results cannot be obtained after trying a certain number of times, the CSB will stop the execution of the cloud services composition application.

The software versions in each cloud service c run on parallel hardware units. The total number of units is h_c. The units are independent and identical. The availability of each unit is a_c. The number H_c of units available at the moment determines the amount of available computational resources and, therefore, the number of software versions that can be executed simultaneously $L_c(H_c)$. No hardware unit can change its state during the execution.

The software versions of each cloud service c start their execution in accordance with a predetermined ordered list. L_c first software versions from the list start their execution simultaneously (at time zero). If the number of terminated software versions is less than k_c, after termination of each software version a new software version from the list starts its execution immediately. If the number of terminated software versions is not less than k_c, after termination of each software version the CSB compares the outputs. If k_c outputs are identical, the CSB terminates its execution (terminating all the software versions that are still executed), otherwise a new software version from the list is executed immediately.

If after termination of n_c software versions the number of identical outputs is less than k_c then the entire cloud services application fail.

The execution time of the CSB includes the execution time and data transmission time spent by all cloud services invoked by the CSB and itself. In the case that CSB gets not less than k_c consistent results successfully, the time of the entire CSB execution T_c is equal to the termination time of the software version that has produced the k_cth correct output (in most cases, the time needed by the CSB to make the decision can be neglected). It can be seen that the CSB execution time is a random variable depending on the reliability and the response time of the software versions and on the availability of the hardware units.

The sum of the random execution times of each CSB gives the random task execution time for the entire system T. In order to estimate both the system's reliability and its performance, different measures can be used, depending on the application.

In cloud service applications where the response time of each task is of critical importance, the system's acceptability function is defined as $F(T, w) = 1(T < w)$, where w is a maximal allowed system response time. The system's reliability $R(w) = E(F(T, w))$ in this case is the probability that the correct output is produced in time less than w. The conditional expected system response time $\widetilde{\varepsilon}(w) = E(T \times 1(T < w))/R(w)$ is considered to be a measure of the system's performance. This index, defined according to (16), determines the system's expected response time given that the system and network do not fail:

$$\widetilde{\varepsilon}(w) = \frac{E\left(\widetilde{G}\right)}{\Pr\{F(G, W) = 1\}} = \frac{E(GF(G, W))}{E(F(G, W))}. \quad (16)$$

In cloud service applications where the system's average productivity (the number of executed tasks) over a fixed mission time is of interest, the system's acceptability function is defined as $F(T) = 1(T < \infty)$, the system's reliability is defined as the probability that it produces correct outputs regardless of the total response time (this index can be referred to as $R(\infty)$), and the conditional expected system response time $\widetilde{\varepsilon}(\infty)$ is considered to be a measure of the system's performance.

4.1.1. Number of Software Versions That Can Be Simultaneously Executed. The number of available hardware units in cloud service c can vary from 0 to h_c. Given that all of the units are identical and have availability a_c, one can easily obtain probabilities $Q_c(x) = \Pr\{H_c = x\}$ for $0 \le x \le h_c$:

$$Q_c(x) = \Pr\{H_c = x\} = \binom{h_c}{x} a_c^x (1 - a_c)^{h_c - x}. \quad (17)$$

The number of available hardware units x determines the number of software versions that can be executed simultaneously: $l_c(x)$. Therefore,

$$\Pr\{L_c = l_c(x)\} = Q_c(x). \quad (18)$$

The pairs $Q_c(x)$, $l_c(x)$ for $0 \le x \le h_c$ determine the probability mass function (referred to as p.m.f. as follows) of the discrete random value L_c.

4.1.2. Termination Times of Software Version. In each cloud service c, a sequence where each software version starts its execution is defined by the numbers of software versions. This means that each software version i starts its execution not earlier than software versions $1,\dots,i-1$ and not later than software versions $i+1,\dots,n_c$. If the number of software versions that can run simultaneously is l_c, then we can assume that the software versions run on l_c independent processors. Let α_m be the time when processor m terminates the execution of a software version and is ready to run the next software version from the list of not executed software versions. Having the response time of each software version $\tau_{ci}(1 \le i \le n_c)$, one can obtain the termination time $t_{ci}(l_c)$ for each software version i using the following simple algorithm.

(1) Assign $\alpha_1 = \cdots = \alpha_{l_c} = 0$ (all of the units are ready to run the software versions at time 0).

(2) For $i = 1,\dots,n_c$ repeat the following:

(a) find any m $(1 \le m \le l_c) : \alpha_m = \min\{\alpha_1,\dots,\alpha_{l_c}\}$ (m is the number of the earliest processor that is ready to run a new software version from the list),

(b) obtain $t_{ci}(l_c) = \alpha_m + \tau_{ci}$ and assign $\alpha_m = t_{ci}(l_c)$.

Times $t_{ci}(l_c)$, $(1 \le i \le n_c)$, correspond to intervals between the beginning of cloud service execution and the moment when the software versions produce their outputs. Observe that the software versions that start execution earlier can terminate later: $j < y$ does not guarantee that $t_{cj}(l_c) \le t_{cy}(l_c)$. In order to obtain the sequence, in which the software versions produce their outputs, the termination times should be sorted in increasing order $t_{cm_1}(l_c) \le t_{cm_2}(l_c) \le \cdots \le t_{cm_{n_c}}(l_c)$ which gives the order of software versions $m_1, m_2, \dots, m_{n_c}$ corresponding to times of their termination.

The ordered list $m_1, m_2, \dots, m_{n_c}$ determines the sequence of software version outputs. Now one can consider the cloud service c as a system in which the n_c software versions are executed consecutively according to the order $m_1, m_2, \dots, m_{n_c}$ and produce their outputs at times $t_{cm_1}(l_c)$, $t_{cm_2}(l_c), \dots, t_{cm_{n_c}}(l_c)$.

4.2. Definition of Reliability and Performance of the Cloud Service and the Cloud Services Composition Application. Let r_{cm_i} be the reliability of the software version that produces ith output in cloud service c (r_{cm_i} is equal to the probability that this output is correct). Consider the probability that k out of n first software versions of cloud service c succeed. This probability can be obtained as

$$R_k = \left[\prod_{i=1}^{n}\left(1 - r_{cm_i}\right)\right] \times \left[\sum_{i_1=1}^{n-k+1} \frac{r_{cm_{i_1}}}{1 - r_{cm_{i_1}}} \sum_{i_2=i_1+1}^{n-k+2} \frac{r_{cm_{i_2}}}{1 - r_{cm_{i_2}}} \cdots \sum_{i_k=i_{k-1}+1}^{n} \frac{r_{cm_{i_k}}}{1 - r_{cm_{i_k}}}\right]. \tag{19}$$

The cloud service c produces the correct output directly after the end of the execution of j software versions $(j \ge k_c)$ if the m_jth software version succeeds and exactly $k_c - 1$ out of the first executed $j-1$ software versions succeed.

The probability of such event $p_{cj}(l_c)$ is

$$p_{cj}(l_c) = r_{cm_j}\left[\prod_{i=1}^{j-1}\left(1 - r_{cm_i}\right)\right] \times \left[\sum_{i_1=1}^{n-k_c+1} \frac{r_{cm_{i_1}}}{1 - r_{cm_{i_1}}} \sum_{i_2=i_1+1}^{n-k_c+2} \frac{r_{cm_{i_2}}}{1 - r_{cm_{i_2}}} \cdots \sum_{i_{k_c-1}=i_{k_c-2}+1}^{j-1} \frac{r_{cm_{i_{k_c-1}}}}{1 - r_{cm_{i_{k_c-1}}}}\right]. \tag{20}$$

Observe that $p_{cj}(l_c)$ is the conditional probability that the cloud service response time is $t_{cm_j}(l_c)$ given that l_c software versions can be executed simultaneously:

$$p_{cj}(l_c) = \Pr\left\{T_c = t_{cm_j}(l_c) \mid L_c = l_c\right\}. \tag{21}$$

Having the p.m.f. of L_c we can now obtain for $1 \le x \le h_c$

$$\begin{aligned}\Pr\left\{T_c = t_{cm_j}(l_c(x))\right\} &= \Pr\left\{T_c = t_{ck_j}(l_c(x)) \mid L_c = l_c(x)\right\} \Pr\left\{L_c = l_c(x)\right\} \\ &= p_{cj}(l_c(x))\, Q_c(x).\end{aligned} \tag{22}$$

The pairs $t_{cm_j}(l_c(x))$, $p_{cj}(l_c(x))Q_c(x)$, obtained for $1 \le x \le h_c$ and $k_c \le j \le n_c$, determine the p.m.f. of software version response time T_c.

Since the events of successful cloud service execution termination for different j and x are mutually exclusive, we can express the probability of cloud service c success as

$$R_c(\infty) = \Pr\{T_c < \infty\} = \sum_{x=1}^{h_c}\left[Q_c(x) \sum_{j=k_c}^{n_c} p_{cj}(l_c(x))\right]. \tag{23}$$

Since failure of any cloud service constitutes the failure of the entire application, the application's reliability can be expressed as

$$R(\infty) = \prod_{c=1}^{C} R_c(\infty). \tag{24}$$

For cloud services, there are four kinds of execution patterns in cloud services composition application: sequence, parallel, split, and loop. From the p.m.f. of response times T_c for each cloud service c one can obtain the p.m.f. of the response time of the entire application in accordance with

the composition structures and execution logics of the cloud services composition application:

$$T = \begin{cases} \sum_{c=1}^{C} T_c & \text{for sequence structure,} \\ \max\left(T_1, \ldots, T_c\right) & \text{for parallel structure,} \\ \sum_{c=1}^{C} p_c T_c & \text{for split structure,} \\ \sum_{l=1}^{L} \sum_{c=1}^{C} T_c & \text{for loop structure,} \end{cases} \quad (25)$$

where the p_c is the probabilities that the cth split is chosen to be executed. The L is the number of times of loop that C cloud services are executed repeatedly. p_c and L can be obtained from separate testing and/or prediction models.

5. Fast Optimization Algorithm of Reliability and Performance for Cloud Services Composition Application Based on UGF and GA

5.1. Using UGF to Evaluate the Response Time Distribution of Cloud Services. In order to obtain the response time distribution for a cloud service c for a given l_c in the form $p_{cj}(l_c)$, $t_{cm_j}(l_c)$ ($k_c \le j \le n_c$) one can determine the realizations $t_{cm_j}(l_c)$ of the response time $T_c(l_c)$ using the algorithm presented in Section 4.1.2 and the corresponding probabilities $p_{cj}(l_c)$ using (20). However, the probabilities $p_{cj}(l_c)$ can be obtained in a much simpler way using a procedure based on the UGF technique.

Let the random binary variable s_{cm_i} be an indicator of the success of software version m_i in cloud service c such that $s_{cm_i} = 1$ if the software version produces the correct output and $s_{cm_i} = 0$ if it produces the wrong output. The p.m.f. of s_{cm_i} can be represented by the u-function

$$u_{cm_i}(z) = r_{cm_i} z^1 + \left(1 - r_{cm_i}\right) z^0. \quad (26)$$

It can be easily seen that using the operator $\otimes_+$ we can obtain the u-function

$$U_{cj}(z, l_c) = \bigotimes_{+} \left(u_{cm_i}(z), \ldots, u_{cm_j}(z)\right) \quad (27)$$

that represents the p.m.f. of the number of correct outputs in cloud service c after the execution of a group of first j software versions (the order of elements $m_1, m_2, \ldots, m_{n_c}$ and, therefore, $U_{cj}(z, l_c)$ depend on l_c). Indeed, the resulting polynomial relates the probabilities of combinations of correct and wrong outputs (the product of corresponding probabilities) with the number of correct outputs in these combinations (the sum of success indicators). Observe that after collecting the like terms (corresponding to obtaining the overall probability of a different combination with the same number of correct outputs) $U_{cj}(z, l_c)$ takes the form

$$U_{cj}(z, l_c) = \sum_{k=0}^{j} \pi_{jk} z^k, \quad (28)$$

where π_{jk} is the probability that the group of first j software versions produces k correct outputs.

Note that $U_{cj}(z, l_c)$ can be obtained by using the recurrent expression:

$$U_{cj}(z, l_c) = U_{cj-1}(z, l_c) \bigotimes_{+} \left[r_{cm_j} z^1 + \left(1 - r_{cm_j}\right) z^0\right]. \quad (29)$$

According to its definition, $p_{cj}(l_c)$ is the probability that the group of first j software versions produces k_c correct outputs and the group of first $j-1$ software versions produces $k_c - 1$ correct outputs given that l_c software versions can be executed simultaneously. The coefficient π_{jk_c} in polynomial $U_{cj}(z, l_c)$ is equal to the conditional probability that the group of first j software versions produces k_c correct outputs given that l_c software versions can be executed simultaneously.

In order to let the coefficient π_{jk_c} in polynomial $U_{cj}(z, l_c)$ be equal to $p_{cj}(l_c)$, the term with the exponent equal to k_c should be removed from $U_{cj-1}(z, l_c)$ before applying (29) (excluding the combination in which $j - 1$ first software versions produce k_c correct outputs while the m_jth software version fails).

If after the execution of j first software versions the number of correct outputs produced is k and $k + n_c - j < k_c$, then the required number of correct outputs k_c cannot be obtained even if all the $n_c - j$ subsequent software versions produce correct outputs. Therefore, the terms $\pi_{jk} z^k$ with $k < k_c - n_c + j$ can be removed from $U_{cj}(z, l_c)$.

The above considerations lie at the base of the following algorithm for determining all of the probabilities $p_{cj}(l_c)$ ($k_c \le j \le n_c$).

(1) For the given l_c, determine the order of software version termination $m_1, m_2, \ldots, m_{n_c}$ using the algorithm from Section 4.1.2.

(2) Determine the u-function of each software version of cloud service c according to (26).

(3) Define $U_{c0}(z, l_c) = 1$. For $j = 1, 2, \ldots, n_c$,

  (a) obtain $U_{cj}(z, l_c)$ using (28) and, after collecting like terms, represent it in the form (29),

  (b) remove from $U_{cj}(z, l_c)$ all the terms $\pi_{jk} z^k$ for which $k < k_c - n_c + j$,

  (c) If $j \ge k_c$, assign $p_{cj}(l_c) = \pi_{jk_c}$ and remove term $\pi_{jk_c} z^{k_c}$ from $U_{cj}(z, l_c)$.

5.2. Evaluating Response Time Distribution of the Cloud Services Composition Application. Having the pairs $p_{cj}(l_c(x))$, $t_{cm_j}(l_c(x))$ for each possible realization $l_c(x)$ of L_c ($1 \le x \le h_c$) and probabilities $\Pr\{L_c = l_c(x)\} = Q_c(x)$, one can obtain the p.m.f. of random response times T_c for each cloud service

by applying (22). If the conditional p.m.f. $p_{cj}(l_c(x))$, $t_{cm_j}(l_c(x))$ are represented by the u-function

$$\widetilde{u}_c\left(z, l_c\left(x\right)\right) = \sum_{j=k_c}^{n_c} p_{cj}\left(l_c\left(x\right)\right) z^{t_{cm_j}(l_c(x))} \tag{30}$$

then the u-function representing the p.m.f. of the random value T_c takes the form

$$\widetilde{U}_c\left(z\right) = \sum_{x=1}^{h_c} Q_c\left(x\right) \widetilde{u}_c\left(z, l_c\left(x\right)\right). \tag{31}$$

In accordance with the four kinds of execution patterns of cloud services in cloud services composition application, we present four kinds of composition operators for u-function operation corresponding to formula (25):

(1) The composition operator $\otimes_{\text{sequ}}$ for sequence execution pattern:

$$\begin{aligned}\widetilde{U}\left(z\right) &= \widetilde{U}_1\left(z\right) \bigotimes_{\text{sequ}} \widetilde{U}_2\left(z\right) \\ &= \sum_{l_1=1}^{L_1} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) z^{\widetilde{T}_{1l_1}} \bigotimes_{\text{sequ}} \sum_{l_2=1}^{L_2} \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\widetilde{T}_{2l_2}} \\ &= \sum_{l_1=1}^{L_1} \sum_{l_2=1}^{L_2} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\widetilde{T}_{1l_1} + \widetilde{T}_{2l_2}}.\end{aligned} \tag{32}$$

(2) The composition operator $\otimes_{\text{para}}$ for parallel execution pattern:

$$\begin{aligned}&\widetilde{U}\left(z\right) \\ &= \widetilde{U}_1\left(z\right) \bigotimes_{\text{para}} \widetilde{U}_2\left(z\right) \\ &= \sum_{l_1=1}^{L_1} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) z^{\widetilde{T}_{1l_1}} \bigotimes_{\text{para}} \sum_{l_2=1}^{L_2} \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\widetilde{T}_{2l_2}} \\ &= \sum_{l_1=1}^{L_1} \sum_{l_2=1}^{L_2} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\max(\widetilde{T}_{1l_1}, \widetilde{T}_{2l_2})}.\end{aligned} \tag{33}$$

(3) The composition operator $\otimes_{\text{split}}$ for split execution pattern:

$$\begin{aligned}&\widetilde{U}\left(z\right) \\ &= \widetilde{U}_1\left(z\right) \bigotimes_{\text{split}} \widetilde{U}_2\left(z\right) \\ &= \sum_{l_1=1}^{L_1} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) z^{\widetilde{T}_{1l_1}} \bigotimes_{\text{split}} \sum_{l_2=1}^{L_2} \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\widetilde{T}_{2l_2}} \\ &= p_1 \sum_{l_1=1}^{L_1} \Pr\left(T_{1l_1} = \widetilde{T}_{1l_1}\right) z^{\widetilde{T}_{1l_1}} + p_2 \sum_{l_2=1}^{L_2} \Pr\left(T_{2l_2} = \widetilde{T}_{2l_2}\right) z^{\widetilde{T}_{2l_2}},\end{aligned} \tag{34}$$

where the p_1 and p_2 are the probabilities that the splits, corresponding to $\widetilde{U}_1(z)$ and $\widetilde{U}_2(z)$, are chosen to execute.

(4) The composition operator $\otimes_{\text{loop}}$ for loop execution pattern.

The composition operator $\otimes_{\text{loop}}$ can be expressed by multiple composition operators $\otimes_{\text{sequ}}$, because the loop execution pattern can be transformed to an accumulation of multiple sequence execution patterns. The number of composition operators $\otimes_{\text{sequ}}$ transformed is equal to the number of times of repeated execution in loop pattern.

Hence, one can obtain the u-function $\widetilde{U}(z)$ representing the p.m.f. of the random entire application response time T as

$$\begin{aligned}\widetilde{U}\left(z\right) &= \bigotimes_{f} \left(\widetilde{U}_1\left(z\right), \ldots, \widetilde{U}_C\left(z\right)\right) \\ &= \bigotimes_{f} \left(\sum_{x=1}^{h_c} Q_c\left(x\right) \widetilde{u}_c\left(z, l_c\left(x\right)\right)\right),\end{aligned} \tag{35}$$

where the composition operator $\otimes_f$ is an abstract composition operator that it can be one of the composition operators $\otimes_{\text{sequ}}$, $\otimes_{\text{para}}$, $\otimes_{\text{split}}$, and $\otimes_{\text{loop}}$.

In accordance with the composition patterns in cloud services composition application, the concrete u-function $\widetilde{U}(z)$ representing the p.m.f. of T can be obtained by replacing $\otimes_f$ by one of the composition operators $\otimes_{\text{sequ}}$, $\otimes_{\text{para}}$, $\otimes_{\text{split}}$, and $\otimes_{\text{loop}}$.

5.3. Evaluating Response Time Distribution of Different Cloud Services Executed on the Same Hardware. Now consider the case where all of the software cloud services are consecutively executed on the same hardware consisting of h parallel identical modules with the availability a. The number of available parallel hardware modules H is random with p.m.f. $Q(x) = \Pr\{H = x\}$, $1 \le x \le h$, defined in the same way as in (17).

When $H = x$, the number of software versions that can be executed simultaneously in each cloud service c is $l_c(x)$. The u-functions representing the p.m.f. of the corresponding cloud service response time T_c are $\widetilde{u}_c(z, l_c(x))$ defined by (30). The u-function $\widehat{U}(z, x)$ representing the conditional p.m.f. of the entire application response time T (given that the number of available hardware modules is x) can be obtained for any x $(1 \le x \le h)$ as

$$\begin{aligned}\widehat{U}\left(z, x\right) &= \bigotimes_{+} \left(\widetilde{u}_1\left(z, l_1\left(x\right)\right), \ldots, \widetilde{u}_C\left(z, l_C\left(x\right)\right)\right) \\ &= \prod_{c=1}^{C} \widetilde{u}_c\left(z, l_c\left(x\right)\right).\end{aligned} \tag{36}$$

Having the p.m.f. of the random value H we obtain the u-function $\widetilde{U}(z)$ representing the p.m.f. of T as

$$\widetilde{U}\left(z\right) = \sum_{x=1}^{H} Q\left(x\right) \widehat{U}\left(z, x\right). \tag{37}$$

Table 1: Parameters of fault-tolerant cloud services and software versions.

No. of cloud services	L_c	k_c		Software versions							
				1	2	3	4	5	6	7	8
1	1	1	c	5	15	7	8	12	6	—	—
			τ	17	10	20	32	30	75	—	—
			r	0.71	0.85	0.85	0.89	0.95	0.98	—	—
2	2	2	c	5	15	7	8	12	—	—	—
			τ	28	55	35	55	58	—	—	—
			r	0.71	0.85	0.85	0.89	0.95	—	—	—
3	4	3	c	4	3	4	6	5	4	9	6
			τ	17	20	38	38	48	50	41	63
			r	0.80	0.80	0.86	0.90	0.90	0.94	0.98	0.98
4	1	2	c	12	16	17	17	—	—	—	—
			τ	17	10	20	32	—	—	—	—
			r	0.75	0.85	0.93	0.97	—	—	—	—
5	3	1	c	5	9	11	7	12	—	—	—
			τ	30	54	40	65	70	—	—	—
			r	0.70	0.80	0.80	0.80	0.89	—	—	—

5.4. Optimizing the Structure of Cloud Service Composition Application with Fault-Tolerant Mechanism Based on UGF and GA. When a fault-tolerant cloud service application is designed, one has to choose software versions for each cloud service and find the sequence of their execution in order to achieve the entire application's greatest reliability subject to cost constraints. The software versions are chosen from a list of the available products. Each software version is characterized by its reliability, response time, and cost. The total cost of the entire application is defined according to the cost of its software versions. The cost for each software version can be the purchase cost if the software versions are commercial and the off-the-shelf cost, or it can be an estimate based upon the software version's size, complexity, and performance.

Assume that B_c functionally equivalent software versions are available for each cloud service c and that the number k_c of the software versions that should agree in each cloud service is predetermined. The choice of the software versions and the sequence of their execution in each cloud service determine the entire application's reliability and performance.

The permutation $\mathbf{x}_c^*$ of B_c different integer numbers ranging from 1 to B_c determines the order of the software version that can be used in cloud service c. Let $y_{cb} = 1$ if the software version b is chosen to be included in cloud service c and $y_{cb} = 0$ otherwise. The binary vector $y_c = \{y_{c1}, \ldots, y_{cB_c}\}$ determines the subset of software versions chosen for cloud service c. Having the vectors $\mathbf{x}_c^*$ and $\mathbf{y}_c$ one can determine the execution order x_c of the software versions chosen by removing from $\mathbf{x}_c^*$ any number b for which $\mathbf{y}_{cb} = 0$. The total number of software versions in cloud service c (equal to the length of vector $\mathbf{y}_c$ after removing the unchosen software versions) is determined as

$$n_c = \sum_{b=1}^{B_c} y_{cb}. \tag{38}$$

The application structure optimization problem can now be formulated by finding vectors x_c for $1 \le c \le C$ that maximize $R(w)$ subject to cost constraint

$$\Omega = \sum_{c=1}^{C} \sum_{b \in x_c} \omega_{cb} \le \Omega^*, \tag{39}$$

where ω_{cb} is the cost of software version b used in cloud service c, Ω is the entire application cost and Ω^* is the maximal allowable application cost. Note that the length of vectors $\mathbf{x}_c$ can vary depending on the number of software versions chosen.

In order to encode the variable-length vectors $\mathbf{x}_c$ in the GA using the constant length integer strings one can use $(B_c + 1)$-length strings containing permutations of numbers $1, \ldots, B_c, B_c + 1$. The numbers that appear before $B_c + 1$ determine the vector $\mathbf{x}_c$. For example, for $B_c = 5$ the permutations $(2, 3, 6, 5, 1, 4)$ and $(3, 1, 5, 4, 2, 6)$ correspond to $\mathbf{x}_c = (2, 3)$ and $\mathbf{x}_c = (3, 1, 5, 4, 2)$ respectively. Any possible vector $\mathbf{x}_c$ can be represented by the corresponding integer substring containing the permutation of $B_c + 1$ numbers. By combining C substrings corresponding to different cloud services one obtains the integer string a, that encodes the entire application structure.

The encoding method is used in which the single permutation defines the sequences of the software versions chosen in each of the C cloud services. The solution encoding string is a permutation of $n = \sum_{c=1}^{C}(B_c + 1)$ integer numbers ranging from 1 to n. Each number j belonging to the interval $\sum_{c=1}^{m-1}(B_c + 1) + 1 \le j \le \sum_{c=1}^{m}(B_c + 1)$ corresponds to software version $j - \sum_{c=1}^{m-1}(B_c + 1)$ of cloud service. The relative order in which the numbers corresponding to the software versions of the same cloud service appear in the string determines the structure of this cloud service.

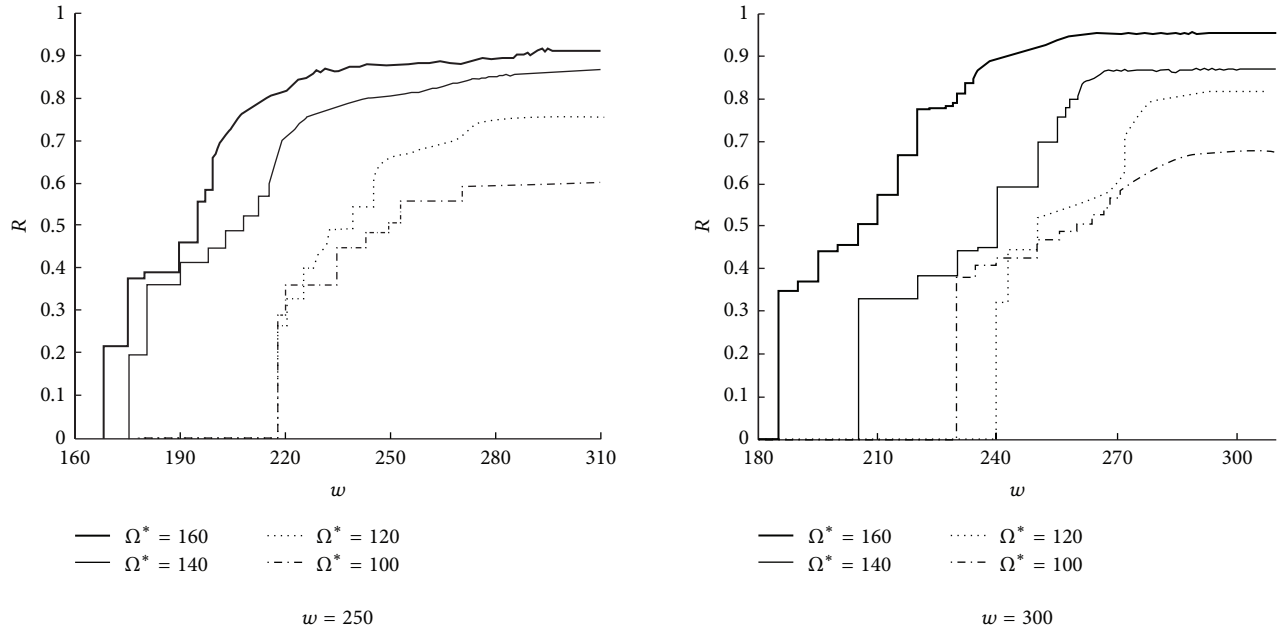

Figure 1: $R(w)$ functions for the solutions obtained.

Table 2: Parameters of solutions obtained for $w = 250$.

Ω^*	Sequence of software versions	$T_{\min}$	$T_{\max}$	Ω	$R(250)$	$\widetilde{\varepsilon}(\infty)$
160	231 | 541 | 37162 | 324 | 214	166	307	159	0.913	188.34
140	34 | 241 | 64231 | 234 | 123	173	301	140	0.868	194.43
120	5 | 431 | 31562 | 43 | 21	205	249	119	0.752	217.07
100	3 | 241 | 4562 | 43 | 41	205	270	100	0.598	220.52

6. Illustrative Examples

Consider a fault-tolerant cloud services composition application consisting of five cloud services in serial running on fully available hardware. The parameters of the software versions that can be used in these cloud services are presented in Table 1. This table contains the values of k_c and L_c for each cloud service and the cost, reliability, and response time for each software version.

Two sets of solutions were obtained for the maximal allowable application response times $w = 250$ and $w = 300$. For each value of w, four different solutions were obtained for different cost constraints. These solutions are presented in Tables 2 and 3. The tables contain the application corresponding cost and reliability for each optimal solution, the expected conditional response time, minimal and maximal possible application response times, and the corresponding optimal execution sequences of the software versions chosen.

Comparing the entire application cost and the reliability of the optimal solutions corresponding to $w = 250$ and $w = 300$ in Tables 2 and 3, it can be seen that the entire application cost and the reliability of the optimal solution corresponding to $w = 300$ are always equal or greater than ones corresponding to $w = 250$ in the case of the same value of Ω^*.

Comparing the entire application cost and the reliability of the optimal solutions corresponding to the different 4 maximal allowable application costs in Tables 2 and 3, it can be seen that the entire application cost and the reliability of the optimal solution corresponding to larger Ω^* are always equal or greater than ones corresponding to smaller Ω^* in the case of the same value of w.

From Tables 2 and 3, it can also be seen that the software versions executed in practice gradually become more and more along with the growth of the value of Ω^*.

These phenomenon above indicates that the selection of suitable Ω^* and w is helpful to improve the reliability of the cloud services composition application and cut down the cost.

To help the designers of the cloud services composition application to select the suitable Ω^* and w, the values of the functions $R(w)$ of all of solutions obtained are drawn in Figure 1. At first, the designers can intuitively find out which curves cross over the value of reliability demand. On this basis, the designers can easily find which points (solution) can meet the maximal allowable response time. The approach abovementioned can be easily realized by software. Thus, it can be applied in online prediction and optimization situation.

Table 3: Parameters of solutions obtained for $w = 300$.

Ω^*	Sequence of software versions	$T_{\min}$	$T_{\max}$	Ω	$R(300)$	$\widetilde{\varepsilon}(\infty)$
160	341 | 4521 | 85632 | 324 | 41	188	369	160	0.951	210.82
140	53 | 541 | 28361 | 431 | 51	173	301	140	0.868	194.43
120	6 | 241 | 61372 | 241 | 31	240	307	120	0.813	252.87
100	4 | 142 | 2386 | 43 | 41	219	295	100	0.672	238.05

7. Conclusions

The traditional reliability and performance prediction and optimization techniques, for example, Markov model and state space analysis, have some defects such as being too time consuming and easy to cause state space explosion and unsatisfied the assumptions of component execution independence by Markov model. Aiming at the defects of Markov model, an optimization model of reliability and performance based on MSS for cloud services application is proposed in this paper, which eliminates the limitation for component execution independence, and more fits the actual execution of cloud services composition application. On this basis, aiming at the defects of state space analysis technique, a fast optimization algorithm with very small time consumption based on UGF and GA for the reliability and performance of cloud services composition application is presented in this paper, which eliminates the risk of state space explosion. The model and algorithm presented in this paper can be applied in online prediction and optimization for reliability and performance of cloud services composition application.

Acknowledgment

This research was supported by the National Natural Science Funds Fund of China (61172084); Science and Technology Support Program of Hubei Province of China (2013BHE022); Natural Science Foundation of Hubei Province of China (2013CFC026); Key new product research and development of Hubei Province of China (2012BBA25002, 2012IHA015).

References

[1] W. Venters and E. A. Whitley, "A critical review of cloud computing: researching desires and realities," *Journal of Information Technology*, vol. 27, no. 3, pp. 179–197, 2012.

[2] L. Qi, W. Dou, X. Zhang, and J. Chen, "A QoS-aware composition method supporting cross-platform service invocation in cloud environment," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1316–1329, 2012.

[3] J. Leukel, S. Kirn, and T. Schlegel, "Supply chain as a service: a cloud perspective on supply chain systems," *IEEE Systems Journal*, vol. 5, no. 1, pp. 16–27, 2011.

[4] C. Y. Ming and P. Y. Jen, "A QoS aware services mashup model for cloud computing applications," *Journal of Industrial Engineering and Management*, vol. 5, no. 2, pp. 457–472, 2012.

[5] L. Zhao, Y. Ren, M. Li, and K. Sakurai, "Flexible service selection with user-specific QoS support in service-oriented architecture," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 962–973, 2012.

[6] W. Dou, L. Qi, X. Zhang, and J. Chen, "An evaluation method of outsourcing services for developing an elastic cloud platform," *Journal of Supercomputing*, vol. 63, no. 1, pp. 1–23, 2010.

[7] Q. Wu, Z. B. Li, Y. Y. Yin, and H. Zeng, "Adaptive service selection method in mobile cloud computing," *China Communications*, vol. 9, no. 12, pp. 46–55, 2012.

[8] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, "Dynamic web service selection for reliable web service composition," *IEEE Transactions on Services Computing*, vol. 1, no. 2, pp. 104–116, 2008.

[9] Y.-K. Lin and P.-C. Chang, "Reliability evaluation of a computer network in cloud computing environment subject to maintenance budget," *Applied Mathematics and Computation*, vol. 219, no. 8, pp. 3893–3902, 2012.

[10] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A self-adaptive hierarchical monitoring mechanism for clouds," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1029–1041, 2012.

[11] M. L. Chiang, "Efficient diagnosis protocol to enhance the reliability of a cloud computing environment," *Journal of Network and Systems Management*, vol. 20, no. 4, pp. 579–600, 2012.

[12] X. Yang and H. Zhang, "Cloud computing and SOA convergence research," in *Proceedings of the 5th International Symposium on Computational Intelligence and Design (ISCID '12)*, vol. 1, pp. 330–335, Hangzhou, China, 2012.

[13] R. Sharma, M. Sood, and D. Sharma, "Modeling Cloud SaaS with SOA and MDA," in *Proceedings of the 1st International Conference on Advances in Computing and Communications*, vol. 190, pp. 511–518, Kochi, India, July 2011.

[14] R. K. Das, S. Patnaik, and A. K. Misro, "Adoption of cloud computing in e-governance," in *Proceedings of the 1st International Conference on Computer Science and Information Technology*, vol. 133 of *Communications in Computer and Information Science*, pp. 161–172, January 2011.

[15] L. Qi, W. Dou, X. Zhang, and J. Chen, "A QoS-aware composition method supporting cross-platform service invocation in cloud environment," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1316–1329, 2012.

[16] X. Chen, X. Liu, F. Fang, X. Zhang, and G. Huang, "Management as a service: an empirical case study in the internetware cloud," in *Proceedings of the IEEE International Conference on E-Business Engineering (ICEBE '10)*, pp. 470–473, Shanghai, China, November 2010.

[17] P. Kumar, V. K. Sehgal, D. S. Chauhan, P. K. Gupta, and M. Diwakar, "Effective ways of secure, private and trusted cloud computing," *International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 412–420, 2011.

[18] X. Xu, C. Cheng, and J. Xiong, "Reliable integrated model of cloud & client computing based on multi-agent," *Journal of Computational Information Systems*, vol. 6, no. 14, pp. 4767–4774, 2010.

[19] J. Liu, J. Zhou, J. Wang, X. Chen, and H. Zhou, "Cloud service: automatic construction and evolution of software process

problem-solving resource space," *Journal of Supercomputing*, pp. 1–25, 2010.

[20] J. Murhland, "Fundamental concepts and relations for reliability analysis of multistate system," in *Reliability and Fault Tree Analysis. Theoretical and Applied Aspects of System Reliability*, pp. 581–618, SIAM, 1975.

[21] E. El-Neweihi, F. Proschan, and J. Sethuraman, "Multistate coherent systems," *Journal of Applied Probability*, vol. 15, no. 4, pp. 675–688, 1978.

[22] R. E. Barlow and A. S. Wu, "Coherent systems with multistate components," *Mathematics of Operations Research*, vol. 3, no. 4, pp. 275–281, 1978.

[23] S. Ross, "Mulivalued state element system," *Annals of Probability*, no. 7, pp. 379–383, 1979.

[24] A. Lisnianski and G. Levitin, *Multi-State System Reliability: Assessment, Optimization and Applications*, vol. 6 of *Series on Quality, Reliability & Engineering Statistics*, World Scientific, River Edge, NJ, USA, 2003.

[25] I. Ushakov, "Optimal standby problems and a universal generating function," *Soviet Journal of Computer Systems Science*, vol. 25, pp. 79–82, 1987.