Int J Software Informatics, Vol.3, No.4, December 2009, pp. 465–482 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics ISSN 1673-7288 http://www.ijsi.org
©2009 by Institute of Software, Chinese Academy of Sciences. All rights reserved. Tel: +86-10-62661040

QoS-Aware Replication in Service Composition*

Kun You, Zhuzhong Qian, Bin Tang, Sanglu Lu, and Daoxu Chen

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China)

Abstract Service composition is a useful technique to assemble light, independent services to meet the complicated and dynamic requirements. Previous research has addressed the quality-of-service (QoS) aware service composition path selection problem, i.e. choose the right service replicas to construct a composite service, so that the overall QoS could meet the request. However, as the requests grows, the existing service replicas could not support the incoming requests even when we choose the best service composition path. That means the existing services may not satisfy the QoS requirements no matter how we choose the composition path. The reason for it is the scale of the existed service component replicas is small compared with the increasing users requirements. In this case, more service replicas should be deployed on suitable nodes to improve the QoS. But which service component should be selected and where these service replicas should be deployed is a challenge. In this paper, we make a deep study on this service replication problem. We give a detailed description of service replication triggering time. And then, we propose LDCS (**L**ongest **D**elay Service **C**omponent **S**election) to select the bottle-neck service component by evaluating the real-time performance of all these components. Finally, we employ MACP (**M**aximum **A**vailable **C**apacity **P**ath) algorithm to select a suitable node to deploy this service replica. Simulation results approve that our approach is effective and efficient.

Key words: service composition; quality of service; replication

You K, Qian ZZ, Tang B, Lu SL, Chen DX. QoS-Aware replication in service composition. *Int J Software Informatics*, **2009, 3(4): 465–482. http://www.ijsi.org/ 1673-7288/3/465.htm**

1 Introduction

In service-oriented architecture, the function is relatively simple, therefore it usually cannot meet the users' requirements. Service composition is to combine individual and reusable services to a more powerful, complicated application, which is considered as an effective approach for quick service provision. Several research projects (e.g. the SAHARA project[12], the CANS project[13]) have been proposed to solve some crucial problems for service composition. The composite services are described as a sequence of service components which point out the function of the service. Each service component may have multi-instances. In this paper, we call a service instance as a service replica. Each replica of one service component has the same function, while

* This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2009CB320705; the National Natural Science Foundation of China under Grant No.90718031; Jiangsu Natural Science Foundation of China under Grant No. BK2008264.
Corresponding author: Kun You, Email: ouk@dislab.nju.edu.cn
Manuscript received 2009-07-31; revised 2009-11-21; accepted 2009-11-30; published online 2009-12-30.

it may be deployed in different physical nodes and may have different performance. One of the key issues in the service composition is how to select a set of proper service replicas for one service composition task which guarantees the QoS of the composite service.

Many researchers have addressed this issue[2−5,15−18]. However, they mainly focused on how to generate an optimal service composition path based on available replicas. They didn't consider that even the best path may not meet the QoS requirements. It takes place when the service requests keep growing and some of the existing replicas trend to be overloaded. In this case, service replication may be the only way to meet the QoS requirements, i.e., to properly deploy more service replicas.

The service replication technology is useful to improve the service reliability, which has been considered in some research work[6,7]. For example, when a service replica fails, we can finish the requirements with another backup replica. However, the replication technique is applied for single service but not in service composition. And, they mainly focused on the consistency and management of the replicas and hardly considered the real-time status of the service component replicas to investigate which component should be replicated and where to deploy the replica, so as to maximize the performance for the service request. It is also worth mentioning that there is research work about the location of replicas in some other scenarios and systems such as in mirror web servers[21], file systems[22] and distributed operators[23]. However, service replication is different from file replication and operator replication. Compared with file replicas, service replicas do not only consume the storage but also other system resources, e.g. CPU and memory, since they are all executable objects. Besides, the cost of replicating a service component is much higher than replicating a file and should be considered. Operators are also executable objects, however, their functions are less complex than services. Usually, the consideration for operator replication is the input and output data amounts. We should consider more aspects such as the computing time and the cost of service nodes for service replication.

In this paper, we present an in-depth study of the service replication problem for improving the QoS of composite services. We first find out the bottle-neck service component, and then, deploy one more service replica for this component in a proper position to improve the performance to meet the QoS requirements.

The service replication problem presented in this paper is based on an overlay network extracted from the substrate networks as shown in Fig.1. Each service component has several replicas which have been deployed at different nodes. The user's request is represented by a sequence of service components. System selects a set of service replicas to achieve the composite service at runtime. However, as we mentioned before, with the load growing, the QoS would be getting worse. Our work is to find out the bottle-necked service component, i.e., which kind of replica we need to deploy, and then choose a right node to deploy the replica, so that it can get the best QoS improvement. For simplicity, this paper only considers one single QoS attribute, the response time, which is actually one of the most important QoS metric. The main contributions of our work are as follows:

- We propose an approach to trigger service replication during composite service execution based on the QoS attributes obtained using a monitoring mechanism;

- We propose LDCS (**L**ongest **D**elay Service **C**omponent **S**election), a simple but efficient method to find out the bottleneck service component;
- We propose MACP (**M**aximum **A**vailable **C**apacity **P**ath) to select a proper node for replica deployment from candidate nodes.

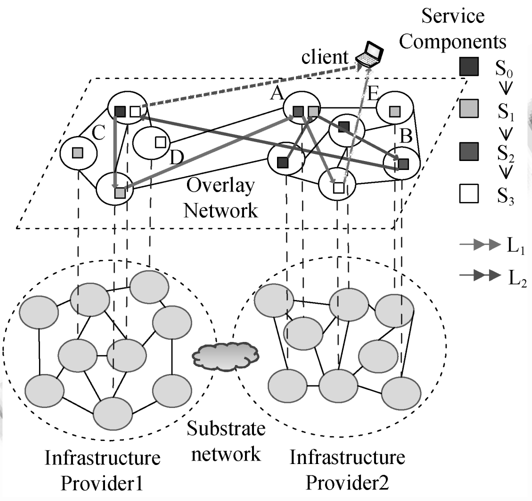


Figure 1. Scenario of service composition overlay

Simulation results show that this approach can achieve both QoS improvement and good load balancing for service composition.

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 introduces some basic concepts and describes the issues to be solved about the service replication problem. In Section 4, we introduce the replication algorithms in details. Section 5 presents the experiment results and the performance evaluation. Finally, conclusions are presented in Section 6.

2 Related Work

Here we broadly classify the existing related work into four categories: (i) service overlay network; (ii) QoS-aware composition path selection; (iii) service reconfiguration; and (iv) service replication.

Service overlay network Various overlay networks have been proposed recently. D. Andersen et al. have proposed a resilient overlay network architecture that allows distributed Internet applications to detect and recover from path failure and performance degradation[24]. Z. Duan et al. advocated the notion of service overlay network as an effective means to guarantee the end-to-end QoS[25]. Z. Li et al. studied the impact of topology on the overlay routing service in terms of routing performance and routing overhead[1]. S. Kalasapur et al. mentioned that the locality of service is an important factor when the tasks including services interact directly with users[19]. All involved service need to be located as close to each other as possible, and the locality of service component replicas can significantly affect the QoS of the service composition path. The work above studies the performance of the physical networks and establishes overlay networks for service composition to satisfy the QoS requirements. Our work is based on service overlay networks. We deploy new service replicas on the service overlay network. According to the relationships between the overlay networks and physical networks, the replicas can be deployed for practical systems.

QoS-aware composition path selection A lot of research work has addressed the QoS-aware composition path selection problem. T. Yu et.al investigated the services selection with end-to-end QoS constraints[3,17]. They proposed a combination-based approach and a graph-based approach to select service composition path with QoS Guarantees. They also proved that all the different composition flow models can be reduced to the sequential service composition model in the service selection[3]. X. Gu et al. proposed SpiderNet, a middleware for QoS-assured service composition[4,16]. Multiple QoS constraints are considered to select the service composition path. B. Raman and R. H. Katz proposed a LIAC algorithm considering the load balancing issue[5]. The basic idea of LIAC is converting the available load balancing parameter into the interlayer path cost. Then the service component selection problem is reduced to the shortest path problem which can be solved with the famous Dijkstra's shortest path algorithm. Concerning the dynamic of P2P system, L. Xiao et al. introduced a concept of interference which describes the changing level of the executing composition path when some service providing nodes fail or exit in the system[20]. Then they proposed an approach of selecting more available and easily recovered composition paths. However, none of them discussed the situation that no composition path can meet the QoS requirements caused by system overloaded. In this paper, we utilize redundant resources in the networks for deploying more service replicas at some nodes to deal with system overload.

Service reconfiguration To resolve the heterogeneity and dynamic of distributed service oriented networks, and satisfy the static or dynamic QoS requirements of the high-assurance systems, dynamic system reconfiguration via service composition has been considered. W.T. Tsai et al. proposed a service-oriented dynamic reconfiguration framework for dependable distributed computing[27]. They introduced the framework from service specification, dynamic reconfiguration issues for SOA and the design and implementation of dynamic reconfiguration service. The model and the architecture of the dynamic reconfiguration service direct the system getting reconfiguration. They also worked at the dynamic system reconfiguration via service composition for dependable computing[28]. The research work gave the threshold for probing cycles, processed reconfiguration and applied the collaborative reconfiguration algorithm among the DRS agents when the active service failed and none of the standby services could replace the failed service. I-Ling Yen et al. studied the QoS-reconfigurable web services and compositions for high-assurance systems[29]. They proposed the configurable service transformation framework and QoS analysis for service composition with reconfigurable services. The QoS-reconfiguration problem is a key issue in service composition. It is related to all layers of the service-oriented system from the network layer to application layer. When the services provided to the users have the QoS deviations or even failures beyond a certain proportion, the reconfiguration is needed. Our work proposes a detailed approach to solve the QoS deviation of the service composition system. It can make the system have more configurations.

Service replication Some research work has addressed the service replication problem in order to improve the performance of distributed service-oriented systems. X. Ye et al. proposed an active replication-based middleware which is used for replicating single-thread web service and maintaining the status of the replicas[6]. They kept the consistency of the replicas and handled the errors though proxy service sites. J. Salas et al. proposed a WS-Replication infrastructure in wide area network (WAN) to provide high levels of service availability[7]. These work mainly addressed the availability and consistency of the replicas, and other QoS benefits of service replications were not utilized. Besides, their approaches can only be used for single service, while we investigate the service replication problem for improving the QoS of composite services.

3 Problem Description

3.1 Scenario

As shown in Fig.1, the scenario in our work can be described as follows: The substrate network which contains multiple network architectures and technologies is organized by different infrastructure providers. The overlay network for providing service composition leases nodes and links resources from the infrastructure providers. The infrastructure providers grantee the maximum resources performance for the overlay network. Then, we can monitor the execution states of the composite services responding to the user requests. The load of the system and QoS of the composition services can be studied on the overlay networks. We describe the problem with the example of video providing service. This service contains four steps: 1 video splitting, 2 video transcoding, 3 video joining, and 4 captioning, which are denoted as S_0, S_1, S_2, S_3 respectively. The video service provider leases nodes from infrastructure providers. The nodes are the ones which have the sufficient resources (CPU, memory and so on) to satisfy one or more demands of the service component replicas. The links are also leased in the similar way to grantee the transmission bandwidth and availability. The overlay network is established and can be managed by the video service provider. When the clients' requests come in, the service provider selects proper service replicas to construct the service composition paths to satisfy the QoS demands of the clients. Then our consideration is that: With the increasing of the clients requests, the load of the nodes and links increases accordingly, result in the dramatic decrease of QoS. If no matter how we choose the service composition paths, the QoS demands of the clients cannot be satisfied, we should deploy more service replicas to improve the system performance and meet the QoS requirements. In the rest of this paper, we present a detailed study of the service replication problem in this situation.

3.2 Basic concepts

Our work is based on the service overlay network as shown in Fig.1. This section gives some related concepts.

- **Overlay Network**: For more details about how to construct the overlay network, please refer to[1,30]. We model the overlay network as a directed graph$G =$

(V, E) , where V is the set of nodes, and E is the set of directed links. Each node in V is extracted from the substrate network, if it meets the resource requirements of one or more service components and some replicas of service components can be deployed on it. (Because the load of the host is considerable in our work, in this paper the overlay network nodes are the ones which have the ability to deploy one or more service component replicas. That is a little different from the common SON[24,25].) For a specific service component S_i, each overlay node is in one of the three status as follows:

– The node has deployed a service replica of service component S_i ;

– The node has the ability of deploying a replica of S_i, while currently this replica hasn't been deployed on the node;

– The node cannot deploy a replica of S_i

Each direct link in E is composed by physical links. The link between two overlay nodes denotes that they are neighbors in physical networks ignoring the routing nodes which are not able to deploy service replicas.

- **Service Component Replica**: For each stand-alone service component, it has several replicas running on different nodes. These replicas have the same functions while their locations and QoS are different, and they are managed by the different service component providers.

- **Replica Node Set (RNS)**: For a specific service component S_i, the replica node set is made up of the nodes that have the replica of S_i. For example, in Fig.1, the node set $\{A, B\}$ is the replica node set ofS_2 (i.e. $RNS(S_2)$).

- **Candidate Node Set (CNS)**: For a specific service component, the candidate node set is made up of the nodes that has the ability of deploying a replica of S_i. For example, in Fig.1, the node set $\{C, D, E\}$ is the candidate node set of S_2 (i.e. $CNS(S_2)$).

- **Service Component Replication**: We call the process of deploying a replica of service component S_i on a specific overlay node in the candidate node set of S_i as service replication.

- **Service Composition Flow Model**: In this paper, the users' requests are described as the sequential composition flow models. It has been proved that all the other description models can be converted to this flow model[3]. The flow form can be expressed simply as $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_i \rightarrow \cdots \rightarrow S_n$ (S_i is some stand-alone service component), which gives the logic execution order of the service components.

- **Service Composition Path (CP)**: It is the path (including the service component processing nodes and transmission links passing through) for a composite service execution instance. (e.g. L_1 and L_2 are both service composition paths in Fig.1).

- **QoS Metrics**: QoS is a broad concept that encompasses a number of non-functional properties such as price, availability, reliability, and reputation[2]. In this paper, we consider one of the key QoS metrics: The Response Time which includes service processing time and transmission time[3]. We measure the end-to-end response time rather than that of the stand-alone service component. And the end-to-end response time is the summation of every stand-alone service component execution time and the transmission time of every link passing through.

3.3 Problem description

Generally speaking, four issues should be addressed for service replication:

- **Replication Triggering Time**: When should we trigger a service replication procedure? Service replication is different from data or file replication since it is high-cost, so we shall do the replication only if we cannot avoid it. We need to decide when we should trigger the service replication.
- **Replication Quantity**: When it is the time to trigger the service replication, we should consider the number of replicas to be deployed. Firstly, the QoS requirements should be satisfied after the multiple replicas being deployed; secondly, the cost caused by replication should be minimized. To achieve these requirements, two techniques are involved to simplify the problem of deploying multiple replicas in the overlay network with large scale nodes:
  - Step-by-step greedy service replication: Once the replication is triggered, a single replica is deployed based on the algorithm proposed in Subsections 4.2 and 4.3 first. If the QoS requirement is not satisfied, the replication would be triggered again, another replica will be deployed. This process will stop until the QoS requirement is satisfied.
  - Nodes Partition: If the number of the nodes is large, we can cluster them into different groups according to their geographic regions. For each group, we do the replication in a similar way.

  Based on these two techniques, this paper focuses on how to deploy a single replica in an overlay network with limited number of nodes.
- **Service Component Selection**: Which service components should be selected to deploy more replicas? It will be discussed in Subsection 4.2.
- **Overlay Node Selection**: Where should the replica of the selected service component be located. We will discuss it in details in Subsection 4.3.

4 QoS-Aware Service Replication

QoS-aware service replication includes: (1) service component selection and (2) replication node selection. In the first phase, we need to find out the bottle-necked component in the composite service denoted as $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_i \rightarrow \cdots \rightarrow S_n$, which primarily effects the QoS of the composite service. In the second phase, we should select a node to deploy the related service replica among the candidates that can deploy such replica. And because the service replication is costly, we also should consider the time to trigger the replication procedure.

In this section, we first propose the policy of triggering replication, i.e., determine the time when we should replicate services. And then, we give the component selection algorithm LDCS to choose the bottle-neck service component. Finally, we propose MACP to find the suitable node to deploy the related service replica.

4.1 Replication triggering time

QoS-aware service composition path selection relies on estimated QoS values, however, Canfora et al.[9] pointed out that, when the service is running, the QoS will commonly deviate from the estimate. In the worst case, the actual QoS of a composite service would violate the users' QoS requests. For the service provider, replication should be triggered if and only if the actual QoS deviate from the requirement. To characterize the deviation, we use a similar way as proposed in[9] for service re-planning. The QoS deviation occurs when the actual QoS deviates from expectation for more than a given percentage. That is, given the estimated response timeRT_{est} of a service and the actual response time (the time between the task being submitted to the first service component and being done at the last service component) measured by a monitoring mechanism. Formally, for a service, if the actual response time RT_{act} and the estimated response time RT_{est}satisfy $(RT_{act}-RT_{est})/RT_{est} > Ratio(threshold)(e.g.10\%)$, we say that there is a QoS deviation. For a fixed time interval t, define $N_c$as the number of service requests completed during the interval$t$ and N_d as the number of service deviation, then the replication of service component should be triggered if $N_d/N_c > DeviationRatioThreshold$ (e.g. 10).

4.2 Service component selection

In this section, we aim to find out the bottle-neck service component in composite service denoted as $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_i \rightarrow \cdots \rightarrow S_n$. In Section 4.1, we triggered the service replication when the QoS deviations are above a certain threshold. It is obviously that the deviation of the service composition is made up of one or more service components' deviation. We expect to find the bottle-neck service component by estimating the characteristics effecting the delay of every service component. We call it the Longest Delay Service Composition Selection (LDCS).

LDCS(**L**ongest **D**elay Service **C**omponent **S**election) is described as follows:

In a given time period T_r, user requests arrive at the system. We trigger the replication when the number of the QoS-deviation of composite services reaches the threshold. We assume there are m_d service instances which are not able to satisfy the QoS demands of the users. We mark these m_d instances as a set of m_d elements $IS = \{IS_1, IS_2, \ldots, IS_{m_d}\}$. For a certain element IS_k, the i^{th} service component of it noted as IS_k^i, then we can estimate the service components by $Delay(IS_k^i)$.

The delay related to a stand-alone service component component is composed of three parts:

- The transmission time from IS_k^{i-1} to IS_k^i, denoted as DIn_k^i;

- The processing time of IS_k^i, denoted as DP_k^i ;
- The transmission time from IS_k^i to IS_k^{i+1}, denoted as $DOut_k^i$.

Then, the delay related to IS_k^i is $Delay_k^i = DIn_k^i + DP_k^i + DOut_k^i$.

For each service component i $(i\epsilon\{1, 2, \ldots, n\})$, we use $Delay^i$ as an estimation value comprehensively considering the deployment of the i^{th} service component and the instances execution delay $Delay_k^i(k = 1, 2, \ldots, m_d)$. LDCS compares

$Delay(1), Delay(2), \ldots, Delay(n)$, then finds $c = argmax_{1\leqslant i\leqslant n} Delay(i)$.

First, we introduce some notations used to estimate the delay of each component as follows:

D_0: The amount of original data

DA^i: The average amount of data to be executed by IS_k^i

Bw_{in}^i: The average transmission bandwidth from IS_k^{i-1} to IS_k^i

DTR^i: The conversion ratio of data amount of S_i, i.e. $DTR(i) = DA(i + 1)/DA(i)$, it is a constant determined just by the functions of S_i . (Easily, we can get $DA(i) = D_0 * \prod_{k=1}^{i-1} DTR(k)$.)

Bw_{out}^i: The average transmission bandwidth from IS_k^i to IS_k^{i+1}

$MinCT^i$: The average minimum processing time of IS^i when the processing nodes have the maximum available resources (e.g. CPU and Memory)

$MaxLoad^i$: The total maximum load of all the nodes having a replica of S_i

$CurrentLoad^i$: The total current running load of all the node having a replica of S_i

The processing time DP^i is determined by $MinCT^i$ and the current idle load (i.e.$MaxLoad^i - CurrentLoad^i$)

We estimate DIn^i,DP^i and $DOut^i$ as follows:

- The average transmission time DIn^i of data coming from the previous component S_{i-1} is:
  $DIn^i = DA^i/Bw_{in}^i$
- The processing time DP^i is determined by $MinCT^i$ and the current idle load (i.e. $MaxLoad^i - CurrentLoad^i$):

$$DP^i = DA^i * MinCT^i * MaxLoad^i/(MaxLoad^i - CurrentLoad^i)$$

- The average transmission time $DOut^i$ of data existing from S_i, and entering the next component S_{i+1} is
  $DOut^i = DA^{i+1}/Bw_{out}^i$
  So,

$$\begin{aligned} Delay(i) &= DIn^i + DP^i + DOut^i \\ &= D_0 * [\prod_{k=1}^{i-1} DTR(k)/Bw_{in}^i + \prod_{k=1}^{i-1} DTR(k)/Bw_{out}^i + \prod_{k=1}^{i-1} DTR(k) \\ &\quad * MinCT^i * \frac{MaxLoad^i}{(MaxLoad^i - CurrentLoad^i)}] \end{aligned}$$

If we define $CurrentLoad(0)$ as the clients' request load entering the system, then we have

$$\frac{CurrentLoad(i)}{CurrentLoad(i-1)} = DTR(i), for \quad i = 1, \cdots, n$$

And

$$\frac{CurrentLoad(i)}{\prod_{k=1}^{i-1} DTR(k)} = CurrentLoad(0)$$

Hence,

$$Delay(i) = D_0 * \{\prod_{k=1}^{i-1} DTR(k)/Bw^i_{in} + \prod_{k=1}^{i-1} DTR(k)/Bw^i_{out} + \prod_{k=1}^{i-1} DTR(k) * MinCT^i * \frac{MaxLoad^i/\prod_{k=1}^{i-1} DTR(k)}{MaxLoad^i/\prod_{k=1}^{i-1} DTR(k) - CurrentLoad^i/\prod_{k=1}^{i-1} DTR(k)}$$
$$= D_0 * (a_i + b_i * \frac{c_i}{(c_i - x)})$$

where

$$a_i = \prod_{k=1}^{i-1} DTR(k)/Bw^i_{in} + \quad \prod_{k=1}^{i-1} DTR(k)/Bw^i_{out}$$

$$b_i = \prod_{k=1}^{i-1} DTR(k) * MinCT^i$$

$$c_i = \frac{MaxLoad^i}{\prod_{k=1}^{i-1} DTR(k)}$$

and

$$x = CurrentLoad(0)$$

Notice that $(Bw_{in})^i_k$ and $(Bw_{out})^i_k$ can be obtained using a monitoring mechanism, then Bw^i_{in} and Bw^i_{out} can be calculated by

$Bw^i_{in} = \sum_{k=1}^{m_d} (Bw_{in})^i_k / m_d$

$Bw^i_{out} = \sum_{k=1}^{m_d} (Bw_{out})^i_k / m_d$

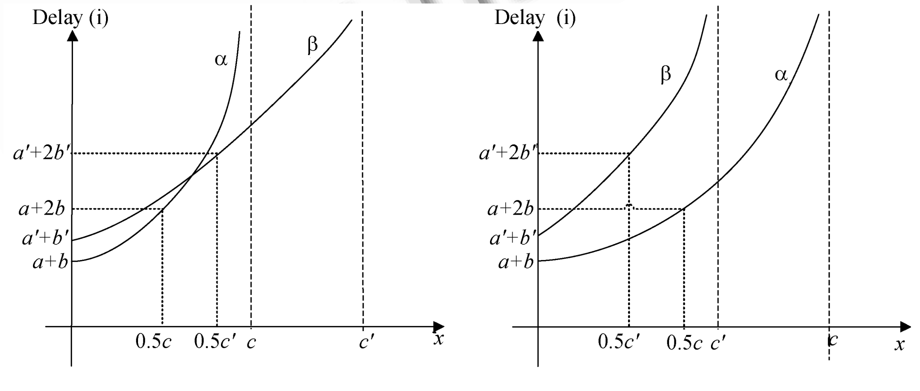


Figure 2. Delay of each service component

Then, we can get a_i, b_i, c_i. For m_d user requests, x is variable. We will depict it in Fig.2.

Let α denote the curve of $Delay(i)$ where $a_i = a, b_i = b, c_i = c$ and β denote the curve of $Delay(j)$ where $a_j = a', b_j = b', c_j = c'$ as shown in Fig.2. Without loss of generality, we assume that $a' + b' > a + b$. Then when $c' < c, Delay(i) < Delay(j)$. Otherwise, when $c' > c$, the relation between $Delay(i)$ and $Delay(j)$ is determined by x, the system realtime load. Since the LDCS is only triggered when the system load is high, we think of $Delay(i) > Delay(j)$ because x must be high under such a situation. Based on the simplified comparison rule, we can get $c = argmax_{1 \leqslant i \leqslant n} Delay(i)$.

4.3 Replication node selection

After a service component (marked as S_i) is selected to deploy more replica, we expect to provide the location of the replica, i.e. the node on which the new replica would be deployed. We select the node from $CNS(S_i)$ .We design our algorithm for the replication node selection based on making full use of the idle transmission and processing capacity of the links and nodes in overlay networks. To achieve end-to-end QoS improvement, we should consider the entire service composition path (including the nodes and the links passing through) rather than single node or single link. Therefore, we expect to find the right one in $CNS(S_i)$, which is on the service composition path that has the maximum probability to have the most available capacity. We call it MACP (Maximum Available Capacity Path). We first introduce some notations.

- **Link Available Capacity**: It is the available transmission bandwidth of the link. From service component, according to the data amount of this step, to the response time required, we are given a threshold B of the available bandwidth by $DataAmount/RT_{req}$, where $DataAmount$ is the normalization amount of data after the i^{th} service component (e.g. 1MB), and RT_{req} is the required time for the outgoing transmission time of the i^{th} service component. We periodically monitor the current available bandwidth, compare it with B, and compute the probability that the current available bandwidth is greater than B.

- **Node Available Capacity**: It is the available processing capacity of the node. The processing time of a node is expressed by $DataAmount * MinCompTime * MaxLoad/AvaiLoad$, where $DataAmount$ is the amount of data to be processed; $MinCompTime$ is the minimum computing time unit data amount; $MaxLoad$ is the maximum load of the node. $AvaiLoad$ is the current available load of the node, i.e. $(MaxLoad - CurrentLoad)$. We also give the threshold of the response time in this step by client requirements (denoted as RT_{req}). Since

$$RT_{req} = DataAmount * MinCompTime * MaxLoad/AvaiLoad_{req}$$

where $AvaiLoad_{req}$ is the required available load of the node, $AvaiLoad_{req}$ can be calculated as

$$AvaiLoad_{req} = DataAmount * MinCompTime * MaxLoad/RT_{req}$$

We denote $AvaiLoad_{req}$ as the threshold C for convenience. We periodically monitor and compare the current available load with C, and compute the probability that the current available load being greater than C.

Probability of Available Capacity : The available bandwidth of the links ($AvlbleBw_{li}$) and the available load of the nodes($AvlbleLoad_{Nk}$) are the variable value when executing. We measure and calculate the ($AvlbleBw_{li}$) and ($AvlbleLoad_{Nk}$) periodically. Given the available link bandwidth threshold as B, and the available node load threshold as C, we define that:

$$\rho_{li} = Pr[AvlbleBandwidth_{li} \geqslant B]\rho_{Nk} = Pr[AvlbleLoad_{Nk} \geqslant C] \tag{4.1}$$

For a certain service composition path *cp*, the probability of the available resource of the path elements (links and nodes passing through) satisfying the demand threshold is defined as $\pi_{cp}(S)$, and we have

$$\pi_{cp}(S) = \prod_{li\epsilon cp} \rho_{li} * \prod_{Nk\epsilon cp} \rho_{Nk} \tag{4.2}$$

MACP Using LDCS in Subsection 4.2, we can find the service component (marked as S_i) to be deployed a replica. For the selected component S_i, candidate nodes set $CNS(S_i)$ contains the nodes that satisfy the resource demand of S_i. For the other service component $S_k(k \neq i)$, replica node set $RNS(S_k)$ contains the nodes that have the replica of S_k. The algorithm is to find a node N_s in $CNS(S_i)$, to make the path *cp* maximize $\pi_{cp}(S)$.

From equation(4.2), we have that

$$-log\pi_{cp}(S) = \sum_{li\epsilon cp}(-log\rho_{li}) * \sum_{Nk\epsilon cp}(-log\rho_{Nk}) \tag{4.3}$$

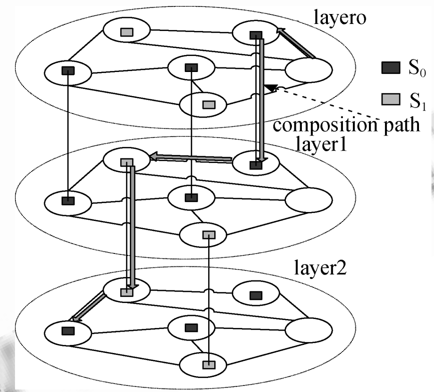


Figure 3. Transformed graph for service composition

So to find a composition path that maximizes $\pi_{cp}(S)$ is to find a composition path that minimizes $-log\pi_{cp}(S)$. We can assign the weight $-log(\rho_{li})$ to each link l_i and $-log(\rho_{Nk})$ to each node N_k.The probability ρ_{li}, ρ_{Nk} is respectively in (0, 1), so $-log(\rho_{li}) > 0\&\&-log(\rho_{Nk}) > 0$.Then we expect to find the shortest composition path for $-log\pi_{cp}(S)$.We noted that the shortest path cannot be found just by Dijkstra's shortest path algorithm, because the cost of the nodes lying on the composition path should be considered. Here, we leverage the multi-layer approach presented in[10]. Given a network graph G, service components are located on the graph nodes. When a service composition flow has n service components, the original graph is copied for $n+1$ times. If there is a i^{th} service component instance on node N, a vertical edge is added between the and the layer.The starting node and terminal node of the vertical edge are respectively the node N in i^{th} and $(i+1)^{th}$ layer. Horizontal edges in every layer are the links in G. A simple example is shown in Fig.3. The composition service has two service components:s_0,s_1. Between the first two layers, if a node has located "s_0 ", then the vertical edge is added. Between the bottom two layers, if a node has located "s_1", then the vertical edge is added. Any path from the nodes in top layer to the nodes in bottom layer by the edges in the multiple-layer graph will pass through "s_0 " and "s_1" orderly.

By the method described above, we can find a service composition path *cp* that minimizes $-log\pi_{cp}(S)$, then maximize$\pi_{cp}(S)$.That is, we find a service composition path *cp* that has the maximum probability to have the sufficient available link bandwidth and node capacity. Therefore this service composition path *cp* can satisfy the QoS requirements. Then the candidate node on *cp* in $CNS(S_i)$ is the node to be selected. The algorithm is described in Algorithm 1 in the next page. The computational complexity of our MACP algorithm is $O(n^2 * |V|^2)$, where n is the number of service components and $|V|$ is the number of nodes in G.

5 Performance Evaluation

5.1 Simulation setup

In this section, we setup simulation experiments to estimate the performance of our service replication algorithms. As explained in Subsection 3.1, we can use nodes partition and step-by-step greedy method for deploying multiple replicas in large-scale overlay networks, so we only evaluated the performance of developing single replica in medium scale networks. We use the topology generator GT-ITM[26] to generate the overlay networks. In any overlay network, the maximum available bandwidth of each link is generated by GT-ITM. The maximum load of each node N_i (denoted as $MaxLoad_{N_i}$) is uniformly distributed in Refs.[4, 24]. This range is based on the measured values in Ref.[11]. We generate 10 different random topologies with 100 nodes and another 10 topologies with 50 nodes.

5.2 Service component selection

The purpose of service component selection is to find out the bottleneck service component S_i from $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_i \rightarrow \cdots \rightarrow S_n$.The parameters of the experiment are given as follows:

- The number of service components (marked as n): n is a random integer value in range [3, 10].

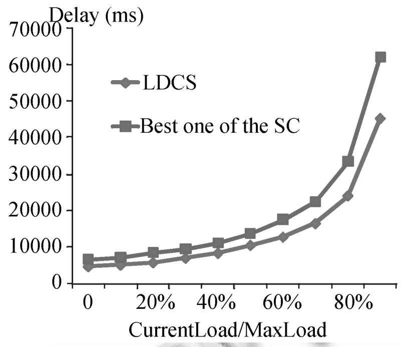
- The number of replicas of each service component S_j (marked as r_j): for each S_j in $\{S_1, S_2, \ldots, S_n\}$,r_j is randomly selected from the range [1, 10].

```
Algorithm 1:   MACP
  Input:
  $G = (V, E)$ : The overlay network, where $V$ is the set of nodes,
  and $E$ is the set of links
  $S = [S_1, S_2, ..., S_i, ..., S_n]$ : The vector of service components.
  $CNS(S_k)$and $RNS(S_k)$ for each$S_k$
  $S_i$ : The service component to be replicated.
  Output: Node $N_i$ belonging to $CNS(S_i)$
  for each $S_j$ in $S$ do
      for each $N_k$ in $[RNS(S_j) + CNS(S_j)]$ do
          Monitor the current load status of periodically
          $AvlbleLoad_{N_k} = MaxLoad_{N_k} - CurrentLoad_{N_k}$
          $\rho_{N_k} = Pr[AvlbleLoad_{N_k} \geq C]$
  for each $l_q$ in $E$ do
      Monitor the current bandwidth status of periodically
      $AvlbleBandwidth_{l_p} =$
      $MaxBandwidth_{l_p} - OccupiedBandwidth_{l_p}$
      $\rho_{l_q} = Pr[AvlbleBandwidth_{l_p} \geq B]$
  begin
      Build layered model of graph $G'$ as follows:
      i.  $G'$ contains $(n + 1)$ layers that each layer is a copy of $G$
      ii.   Set edges in $G'$ and set cost for edges
      begin
          a)Between $Layer_i$ and $Layer_{i+1}$, edge $< j^i, j^{i+1} >$ span
          from $i^{th}$ layer to $(i + 1)^{th}$ layer if and only if
          $N_j \epsilon CNS(S_i)$. Set the cost of the edge as $-log(\rho_{N_j})$ .
          b)Between $Layer_k$ and $Layer_{k+1}$ , edge $< j^k, j^{k+1} >$ span
          from $k^{th}$ layer to $(k + 1)^{th}$ layer if and only if
          $N_j \epsilon RNS(S_k)$. Set the cost of the edge as $-log(\rho_{N_j})$.
          c)In each $Layer_p$, edge $< j^p, k^p >$ is the edge in $E$. Set
          the cost of the edge as $-log(\rho_{l_q})$.
      end
  end
  Find the shortest path $cp$ in $G'$ from $s$ to $d$ using Dijkstra's
  shortest path algorithm.
  Result: the $i^{th}$ node on the path $cp$ .
```

Algorithm 1. MACP

- Data amount (marked as DA): for every client request, we will set up a session to provide composition service. DA is a random value in [1MB, 100MB]. As mentioned before the end-to-end delay is proportional to the data amount, so we normalize the delay into $delay/DA$ (i.e. the delay we compare is the value when DA is 1MB) for fair comparison.

To evaluate the proposed algorithm LDCS, we deploy one more replica R_j for each service component $S_j (j = 1, \ldots, n)$ respectively, and deploy the replica based on both MACP and *random* (select the node randomly) algorithms. And then, we calculate the *delay* of all the new composite services based on the above replication strategies. Among all these *delays*, one of them matches the result of LDCS, i.e. the replica deployed is just the one LDCS get. And we select the minimum one in the rest of the *delays*, which is called BestOneofOtherSC (Best One of Other Service Component).



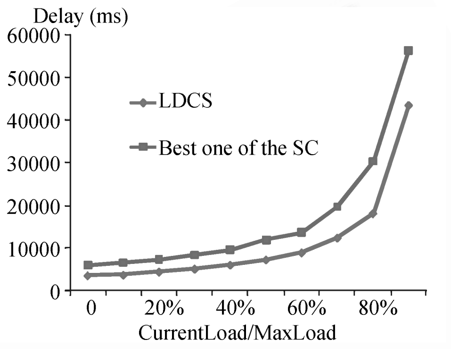


Figure 4. Service component selection +*random* Figure 5. Service component selection+MACP

We run 20 sets of experiment with different network topologies. For each topology, the experiment is run 10 times with randomly given parameters. The simulation results are shown in Fig.4 and Fig.5 when random and MACP are used respectively. It is shown that no matter what replication node selection algorithm is employed, the service component selection algorithm LDCS can get the bottle-neck service component, and maximize the reducing of the end-to-end delay.

5.3 Replication node selection

In this step, based on the selected service component S_i with LDCS, we expect to select a proper overlay node V_i among the candidates to deploy a replica of S_i. Since there is few research works about service replication, we compare the performance of MACP with *random* replication. Furthermore, we also give the performance of no replication is deployed ($NoRep$) to show the benefits of service replication.

The experiment is set up as follows: First, we choose the 100-nodes overlay topology. The number of replicas of each service component is less than 10. For each client request, we set an active session for the composite service task. The delay of the session is the time between the request transferred to the entry node and response acquired at the exit node. The service composition path we selected is the shortest delay path in the current situation. The data amount of each session is the randomly value in [1MB, 10MB] in the experiment. For comparison, we will divide one session into n sessions if the data amount of it is nMB. We measure the delay of a new session just when there are x active sessions running in the system and the result is shown in Fig.6. We randomly generated 10 topologies with 50 sets of comparison experiments on each topology. Fig.6 shows the average value.

In a similar way, we ran the experiment on 50-nodes overlay topology, and the replica scale of each service component is not beyond 5. The result is shown in Fig.7.

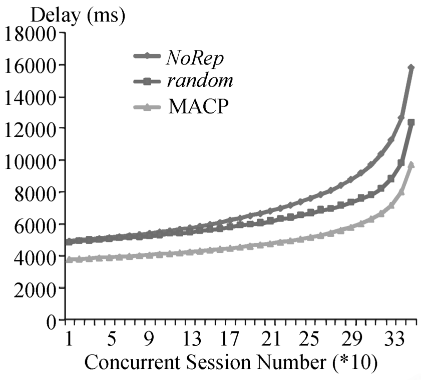


Figure 6. 100-Overlay-Nodes, 10-Replicas

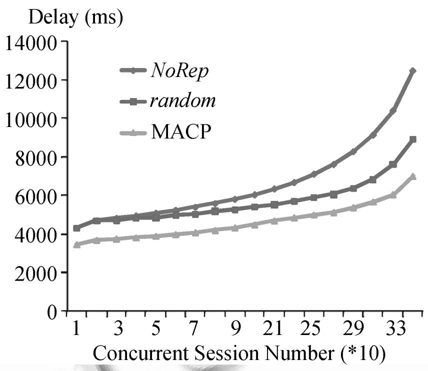


Figure 7. 50-Overlay-Nodes, 5-Replicas

Fig.6 and Fig.7 show that when we select a candidate node to deploy a replica of the bottle-neck service component, the delay of the task will decrease. And the QoS improvement after MACP replication is better than that after *random* replication which randomly chooses a node in $CNS(S_i)$.

By comparing the trend lines of Fig.6 and Fig.7, we can see that when the replica scale is smaller (i.e. shown in Fig.7), the decrease of the response time by our approach is more remarkable. It is because when the node scale and replica scale are small, the participation of a new replica has more effect. Therefore, with large node scale and large replica scale, we will add new replicas continuously by greedy algorithm.

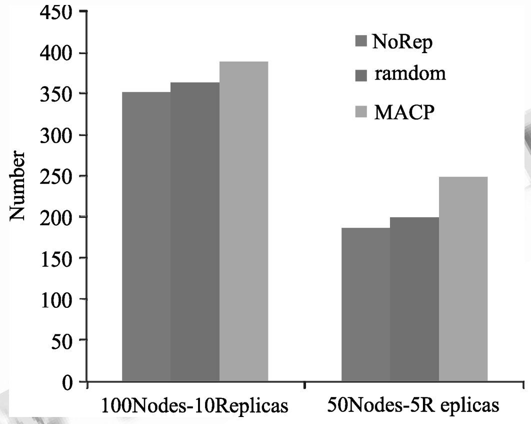


Figure 8. Concurrent session number

Meanwhile, if we deploy a replica, the system can achieve better load balancing. We cumulate the session (each of which has 1MB data) number until there is at least one service component is overloading ($\sum_{N_i \epsilon RNS(S_i)} RequestLoad \geqslant \sum_{N_i \epsilon RNS(S_i)} MaxLoad$).The number of concurrent active sessions under different nodes scales and replica scales is shown in Fig.8. We can see from the result that when selecting a node by MACP to deploy a replica, the system can handle more concurrent requests.

6 Conclusion

This paper investigates service component replication problem based on overlay network. To the best of our knowledge, this is the first time to improve the QoS of composite service by deploying more service replicas on idle nodes. As the service replication procedure is costly, we first present the replication triggering time. And then, we propose LDCS algorithm to find out the bottle-neck service component which need to deploy more service replicas. Finally, we give MACP to select a proper node to deploy the service replica. The simulation results show that our approach decrease the response time of composite service as well as balance the load. In the future work, we will consider multiple QoS metrics, and discuss the deployment of multiple replicas in details.

References

[1] Li Z, Mohapalra P. The Impact of Topology on Overlay Routing Service. Proc. of IEEE INFOCOM, 2004. 408–418.

[2] Zeng L, Benatallah B, Ngu AHH, Dumas M, Kalagnanam J, Chang H. QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering, 2004, 30(5): 311–327.

[3] Yu T, Lin KJ. Service selection algorithms for Web services with End-to-end QoS constraints. Journal of Information Systems and E-Business Management, 2005, 3(2): 103–126.

[4] Gu X, Nahrstedt K, Yu B. Spidernet: An integrated peer-to-peer service composition framework. Proc. of IEEE International Symposium on High-Performance Distributed Computing (HPDC-13), 2004: 110–121.

[5] Raman B, Katz RH. Load balancing and stability issues in algorithms for service composition. Proc. of IEEE INFOCOM, 2003. 1477–1487.

[6] Ye X, Shen Y. A Middleware for Replicated Web Services. Proc. of IEEE ICWS, 2005. 631–638.

[7] Salas J, Perez-Sorrosal F, Patino-Martinez M, Jimenez-Peris R. WS-replication: A Framework for Highly Available Web Services. Proc. of ACM WWW, 2006. 357–366.

[8] Yu YT, Lau MF. A Comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. Journal of Systems and Software, 2006, 29(5): 14–21.

[9] Canfora G, Penta MD, Esposito R, Villani ML. QoS-Aware Replanning of Composite Web Services. Proc. of IEEE ICWS, 2005. 121–129.

[10] Choi S, Turner J, Wolf T. Configuring Sessions in Programmable Networks. Proc. of IEEE INFOCOM, 2001. 60–66.

[11] Dinda PA. The Statistical Properties of Host Load. Scientific Programming, 1999, 7(3-4): 211–229.

[12] Raman B, et al. The SAHARA Model for Service Composition Across Multiple Providers. Proc. of International Conference on Pervasive Computing (Pervasive 2002), 2002. 585–597.

[13] Fu X, Shi W, Akkerman A, Karamcheti V. CANS: Composable, Adaptive Network Services Infrastructure. Proc. of 3rd USENIX Symposium on Internet Technologies and Systems, 2001.

[14] Xu D, Nahrstedt K. Finding Service Paths in a Media Service Proxy Network. Proc. of SPIE/ACM Multimedia Computing and Networking Conference, 2002. 171–185.

[15] Xue G, Zhang W. Multiconstrained QoS routing: greedy is good. Proc. of IEEE GLOBECOM, 2007. 1866–1871.

[16] Gu X, Nahrstedt K. Distributed Multimedia Service Composition with Statistical QoS Assurances. IEEE Transactions on Multimedia, 2006, 8(1): 141–151.

[17] Yu T, Zhang Y, Lin KJ. Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Transactions on the Web, 2007, 1(1).

[18] Alrifai M, Risse T. Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. Proc. of ACM WWW, 2009. 881–890.

[19] Kalasapur S, Kumar M, Shirazi BA. Dynamic service composition in pervasive computing. IEEE

Transactions on Parallel and Distributed Systems, 2007, 18: 907–918.

[20] Spector AZ. Achieving Application Requirements. Distributed Systems. Ed.S.Mullender, ACM Press, 1989. 19–33.

[21] Jamin S, Jin C, Kurc A, Raz D, Shavitt Y. Constrained Mirror Placement on the Internet. Proc. of IEEE INFOCOM, 2001. 31–40.

[22] Harvesf C, Blough DM. The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables. Proc. of IEEE P2P'06, 2006.

[23] Abrams Z, Liu J. Greedy is Good: On Service Tree Placement for In-Network Stream Processing. Proc. of IEEE ICDCS, 2006.

[24] Anderson D, Balakrishnan H, Kaashoek F, Morris R. Resilient Overlay Networks. Proc. of ACM SOSP, 2001. 131–145.

[25] Duan Z, Zhang ZL, Hou T. Service Overlay networks: SLAs, QoS and Bandwidth Provisioning. Proc. of 10th IEEE ICNP, 2002.

[26] GT-ITM. Georgia Tech Internetwork Topology Models. http://www.cc.gatech.edu/projects/gtitm/

[27] Tsai WT, Song W, Paul R, Cao Z, Huang H. Services-Oriented Dynamic Reconfiguration Framework for Dependable Distributed Computing. Proc. of IEEE COMPSAC, 2004. 554–559.

[28] Tsai WT, Song W, Chen Y, Paul R. Dynamic System Reconfiguration Via Service Composition for Dependable Computing. Reliable Systems on Unreliable Networked Platforms, Springer Berlin, Heidelberg, 2007, 4322: 203-224.

[29] Yen IL, Ma H, Bastani FB, Mei H. QoS-Reconfigurable Web Services and Composition for High-Assurance Systems. IEEE Computer Society, 2008, 41: 48–55.

[30] Chowdhury NMK, Boutaba R. A Survey of Network Virtualization, Technical Report. David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Tech. Rep. CS-2008-25, Oct 2008.