Journal of Information Processing Vol. 19 52–61 (Feb. 2011)

*Regular Paper*

# Designing Dynamic Control Mechanisms for Service Invocation

Donghui Lin,[†1] Yohei Murakami[†1] and Masahiro Tanaka[†1]

In service composition environments, users and service entity hosts are always geometrically distributed. Therefore the performance of the service response might be poor when users invoke services that are physically far from them. Such issues are difficult to be solved with traditional caching technologies in the areas of contents delivery network because service providers do not always allow their service entities to be copied to all service entity hosts. In this paper, we deal with the service invocation control problem considering the above issues. First, we formally model the service invocation problem in service composition environments. Then we design several dynamic service invocation control mechanisms to improve the response performance of atomic services and composite services. The evaluation results show that (1) the mechanism for atomic services that considers both potential users for most service invocation requests and potential users for continuous requests can best improve the response performance; (2) the mechanism for composite services that considers the group characteristics of atomic services can improve the response performance more than other mechanisms; and (3) our proposed dynamic mechanisms can bring a stable response performance from the perspective of users.

## 1. Introduction

Service composition environments enable people to create, manage services, and share their services with each other, while users can get additional value of services by composing them based on various requirements. In such environments, service users and service entity hosts are always geometrically distributed, while service providers have different policies to provide their services. The Language Grid[1] is a typical example of such kind of service composition environment in the domain of language services. It uses the collective intelligence approach to gather service users and providers together, and coordinate their incentives.

The features of the service composition environments also bring several problems. First, since users and service entity hosts are distributed in different areas, the invocation response of services might be very slow if users invoke services that are physically far from them. Moreover, when a user invokes a composite service that consists of several atomic services, the invocation response is always much slower if all those atomic services are located in service entity hosts that are far from each other. Second, service providers always have their own policies to provide services, among which the most typical one is about the service license issues. Therefore, it is difficult to apply some previous approaches like caching technologies in the areas of contents delivery network (CDN)[2] to the service composition environments since service entities are not always allowed to be copied to all service entity hosts due to license constraints.

This paper aims to deal with the service invocation problem to improve the service response performance in the service composition environments. To achieve this goal, we consider the following research issues in this paper.

**(1) Modeling the service invocation problem**

To clarify the problems, we first propose a formal model of the service invocation problem in the service composition environment considering different stakeholders, different types and status of services, and so on. We further define several response performance indexes for evaluating the average invocation response time for services and the service response stability.

**(2) Designing dynamic service invocation control mechanisms**

We aim to design dynamic service invocation control mechanisms to improve the response performance of services while considering service license constraints. We propose the approach of dynamically controlling (switching) service entity hosts for executing services based on the invocation request from users. Then, we further design a series of mechanisms for the dynamic service invocation control of both atomic services and composite services.

The rest of the paper is organized as follows: Section 2 provides a formal model of the service invocation control problem based on an example in the Language Grid. In Section 3, dynamic service invocation control mechanisms are proposed for both atomic services and composite services. Section 4 introduces the evaluation and analysis with a series of experiments. Section 5 introduces some related work, followed by the conclusion in the final section.

†1 National Institute of Information and Communications Technology (NICT), Japan

## 2. Service Invocation Problem

### 2.1 An Example of Service Invocation Problem

We first use an example of the Language Grid [1)] to show the service invocation problem. The Language Grid provides a service composition environment for users to share, create and combine Web services in language domains (language services). One reason why we use the Language Grid as an example is that it is an existing environment from which we can get real data and conduct experiments for our proposed approaches. What is more important is that the Language Grid satisfies our research target very well: multiple service users and multiple service entity hosts are geometrically distributed in the Language Grid; service providers have their own policies to provide services especially the policies on service licenses. Therefore, although using the example of the Language Grid, this research is suitable for all other similar service composition environments.

To investigate the response performance, experiments have been conducted by a Language Grid user from Thailand. The user invokes an atomic service and a composite service which is composed of three atomic services on the Language Grid. Each service is invoked for 10 times from both the service entity host in Thailand that is close to the user and the service entity host in Japan that is far from the user. The result shows that it takes 2.54 times and 4.87 times of response time for invoking an atomic service and a composite service respectively on the host in Japan than on the host in Thailand for the Thai user. Similar experiments have been conducted between Language Grid users in Japan and in Denmark. When invoking an atomic service on the service entity host in Japan, the user from Denmark receives the response 5.39 times slower than the user from Japan. Therefore, it is necessary to consider how to improve the performance of the service invocation response in service composition environments.

### 2.2 Definitions of Service Invocation Problem

To clarify the problem, we describe several definitions in this section.

**Definition 1** *Service composition environment* is defined by a tuple $SC = (U, S, H, H^*, W, P, PC, T)$ where
(1) $U = \{u_1, u_2, \ldots, u_n\}$ is a finite set of users that invoke services;
(2) $S = \{s_1, s_2, \ldots, s_m\}$ is a finite set of atomic services;
(3) $H = \{h_1, h_2, \ldots, h_k\}$ is a finite set of service entity hosts where services are executed;
(4) $H^* = \{< u_1, h^*(u_1) >, < u_2, h^*(u_2) >, \ldots, < u_n, h^*(u_n) >\}$ is a finite set of the nearest service entity hosts to each user with the fastest response;
(5) $W = \{w_1, w_2, \ldots, w_j\}$ is a finite set of composite services, where each composite service $w_i$ is composed of a set of atomic services $\{s_1, s_2, \ldots, s_l\}$;
(6) $P = \{p_1, p_2, \ldots, p_q\}$ is a finite set of service providers;
(7) $PC = \{PC_{p_1}, PC_{p_2}, \ldots, PC_{p_q}\}$ is the set of policies from all service providers, where the policy set of each service provider $p_i$ consists of policies for providing its atomic services $PC_{p_i} = \{< s_1, policy_1 >, < s_2, policy_2 >, \ldots, < s_v, policy_v >\}$;
(8) $T = \{t_1, t_2, \ldots, t_o\}$ is a set of invocation time periods.

As for policies of service providers, we mainly focus on the policy of providing services with limited service licenses.

**Definition 2** *Current service entity hosts.* Let $n$ be the maximum license number available for an atomic service $s$ which is described as the policy of its provider, the set of current service entity hosts for $s$ is defined as $H^c(s) = \{h^c_1(s), h^c_2(s), \ldots, h^c_n(s)\}$ where the service entity of $s$ is currently deployed and is invocable by users in the service composition environment.

**Definition 3** *Invocation request sequence.* During a certain invocation time period $t$, the invocation request sequence to an atomic service $s$ is represented by a set of sequential vectors $R^t_s$=$\{< r^t_s(1), u_{a_1} >; < r^t_s(2), u_{a_2} >; \ldots; < r^t_s(n), u_{a_n} >\}$ where $u_{a_i}$ and $u_{a_j}$ can be the same user or different users, and each invocation request is represented by $r^t_s(i)$ $(i = 1, 2, \ldots, n)$.

**Definition 4** *Invocation response time.* We use $c(u, s, h)$ to denote the invocation response time for an atomic service $s$ from a user $u$ when $s$ is executed in the service entity host $h$. Let $R^t_s$=$\{< r^t_s(1), u_{a_1} >; < r^t_s(2), u_{a_2} >; \ldots; < r^t_s(n), u_{a_n} >\}$ be the invocation request sequence to an atomic service $s$ in the invocation time period $t$ and let $h^c_i(s)$ be the nearest service entity host that $s$ is available to be executed in the $i^{th}$ invocation for user $u_{a_i}$, we use $c(r^t_s(i))$ $(i = 1, 2, \ldots, n)$ to define the response time of each invocation, and we have $c(r^t_s(i)) = c(u_{a_i}, s, h^c_i(s))$.

Based on **Definition 3**, the invocation request sequence to a composite service $w$ in the invocation time period $t$ can be represented by $R^t_w$ in the same form as the atomic service. Each composite service $w$ is composed of a group of atomic services $\{s_1, s_2, \ldots, s_k\}$ with some control flows between them such as the sequential relation, the parallel relation, the selective relation and so on. Therefore, the invocation request to an atomic service $s$ might be a request of $s$ itself or a request of the composite service $w$ where $s$ is among the atomic services that compose $w$. The invocation response time of a composite service $w$ can be gained from the aggregation of the invocation response time of the atomic services that compose it. In this research, we use the aggregation approach based on workflow patterns[3),4)]. For example, we can calculate the response time of each invocation to the composite service $w$ as $c(r^t_w(i)) = \sum_{j=1}^{k} c(u_{a_i}, s_j, h^c_i(s_j))$ if the composite service is composed by a sequential pattern. If the pattern is a parallel one (AND-AND), we can calculate the response time as $c(r^t_w(i)) = max\{c(u_{a_i}, s_1, h^c_i(s_1)), c(u_{a_i}, s_2, h^c_i(s_2)), \ldots, c(u_{a_i}, s_k, h^c_i(s_k))\}$.

We further define some performance indexes for service invocation.

**Definition 5** *Average invocation response time for service.* Let $R^t_s = \{< r^t_s(1), u_{a_1} >; < r^t_s(2), u_{a_2} >; \ldots; < r^t_s(n), u_{a_n} >\}$ be the invocation request sequence to an atomic service $s$ in the invocation time period $t$, we use $c^t_s$ to define the average invocation response time for the atomic service $s$, and we have $c^t_s = \frac{1}{n}\sum_{i=1}^{n} c(r^t_s(i))$. Similarly, we can define the average invocation response time for composite service $w$ as $c^t_w = \frac{1}{n}\sum_{i=1}^{n} c(r^t_w(i))$.

**Definition 6** *Average invocation response time for a user.* Let $S$ be the finite set of atomic services ($S = \{s_1, s_2, \ldots, s_m\}$) and $R^t_{s_i}$ be the invocation request sequence to an atomic service $s_i$ in the invocation time period $t$, we use $c^t_u$ to define the average invocation response time for a user $u$, and we have $c^t_u = \frac{1}{z}\sum_{i=1}^{m}\sum_{u_{a_i}=u} c(r^t_{s_i}(i))$ ($z$ is the total number of invocation requests for all services in the time period $t$ from $u$).

**Definition 7** *Average invocation response difference rate for all users.* Let $U$ be the finite set of users ($U = \{u_1, u_2, \ldots, u_m\}$) and $c^t_{u_i}$ be the average invocation response time for the user $u_i$, we define the average invocation response difference rate for all users in the invocation time period $t$ as $d^t$, and we have $d^t = \frac{\sum_{i=1}^{m} |c^t_{u_i} - \frac{1}{m}\sum_{i=1}^{m} c^t_{u_i}|}{\sum_{i=1}^{m} c^t_{u_i}}$.

The smaller the value of the average invocation response difference rate is, the more stable service composition environment is for all users, i.e., the average response time for all services is less different between users.

## 3. Dynamic Service Invocation Control Mechanisms

To improve the response performance of the service invocation, it is important to make a user invoke service entities on nearer service entity hosts. However, since service entities cannot be deployed in all service entity hosts due to the license policies from the service providers, it is necessary to dynamically control the service invocation on service entity hosts, more specifically, to switch the hosts for requested service entities based on each request or each user. We propose several dynamic service invocation control mechanisms in this section for an atomic service and a composite service. We assume that the following information can be acquired in service composition environments: (1) the nearest service host to a user $u$ with the fastest response; (2) the current nearest service host to a user $u$ where the service $s$ can be invoked; (3) atomic services that compose a composite service $w$; (4) the history request count for invoking an atomic service $s$ or a composite service $w$ by a user $u$ at the time period $t$; and (5) the request reservation information for invoking an atomic service $s$ or a composite service $w$ by a user $u$ at the time period $t$.

### 3.1 Invocation Control for Atomic Service

#### 3.1.1 Request-based Invocation Control

Invoking service entities on nearer service entity hosts can directly improve the response performance. Therefore, the most straight approach for the invocation control of an atomic service $s$ is to dynamically make the service entity of $s$ available for invocation on the nearest service host to the user $u$ for each request $r^t_s(i)$, which is called request-based invocation control. With the limitation of the maximum possible license number of service entities, it is necessary to control the service entity hosts for executing the service entity. This can be realized by the service entity migration[5),6)] among service hosts or the license control using a framework like the floating license[7)]. Every time a request for service invocation comes, the requested service entity is always controlled to be available on the nearest service entity host to the user as shown in **Algorithm 1**.

However, the biggest challenge of the request-based invocation control is that there might be frequent changes of service host of the service $s$ by the service migration or license control and therefore it always costs much time to control the availability of the service entity among service hosts.

**Algorithm 1** Request-based invocation control mechanism for atomic service

```
procedure AtomicServiceInvocation1 $(s, t)$
1: for all service request $r_s^t(i)$ do
2:     $h^*(u_{a_i}) \leftarrow nearestServiceHost(u_{a_i})$
3:     if $h^*(u_{a_i}) \notin H^c(s)$ then
4:         $h_i^c(s) \leftarrow currentNearestServiceHost(u_{a_i}, s)$
5:         Switch availability of $s$ from service host $h_i^c(s)$ to $h^*(u_{a_i})$
6:         $H(s) \leftarrow (H(s) \setminus \{h_i^c(s)\}) \cup \{h^*(u_{a_i})\}$
7:     end if
8: end for
```

### 3.1.2 User-based Invocation Control

Since invocation control of switching service entity hosts for executing services always costs time, a more effective control mechanism is expected. Therefore, we propose the user-based invocation control mechanism to reduce the switching frequency among service hosts, where the availability of service entities is controlled periodically based on the service invocation trends from users. Users are always from different areas in service composition environments such as the Language Grid. We have observed that the invocation requests from users vary during different time periods. Based on the historical invocation request information and the invocation request reservation information from users, we can anticipate the service invocation request trend from users in different time periods. Therefore, the user-based invocation control mechanism is designed to minimize the frequency of switching service entity hosts for executing service entities. As is described in **Algorithm 2**, based on the information of a possible invocation request $ut[u_i][s]$ of users for each time period, service entities of a service $s$ are always dynamically controlled to be available on the service hosts which are nearest to the users $U^*(s)$ with the most possible invocation requests of $s$. Switching service entity hosts for executing service entities always occur at the beginning of each time period, while there is no switching of the service entity availability during the time period.

**Algorithm 2** User-based invocation control mechanism for atomic service

```
procedure AtomicServiceInvocation2 $(s, t)$
1: for all user $u_i$ do
2:     $ut[u_i][s] \leftarrow historyRequest(u_i, s, t)$
3:     if $(hasReservedRequest(u_i, s, t) = 1)$ then
4:         $ut[u_i][s] \leftarrow reservedRequest(u_i, s, t)$
5:     end if
6: end for
7: Sort $ut[u_i][s]$ for all users
8: $U^*(s) \leftarrow \phi$
9: Add users with the $|H^c(s)|$ maximum $ut[u_i][s]$ to $U^*(s)$ ($|H^c(s)|$ is service
   entity number of $s$)
10: for all $u_i \in U^*(s)$ do
11:     $h^*(u_i) \leftarrow nearestServiceHost(u_i)$
12:     if $(h^*(u_i) \notin H^c(s))$ then
13:         $h_i^c(s) \leftarrow currentNearestServiceHost(u_i, s)$
14:         Switch availability of $s$ from service host $h_i^c(s)$ to $h^*(u_i)$
15:         $H^c(s) \leftarrow (H^c(s) \setminus \{h_i^c(s)\}) \cup \{h^*(u_i)\}$
16:     end if
17: end for
18: for all service request $r_s^t(i)$ do
19:     Update $historyRequest(u_{a_i}, s, t)$
20: end for
```

However, although a minimum cost of switching of a service entity is required in the user-based invocation control, it might lack some flexibility to deal with the cases where real service invocation request situation is not the same as what is anticipated.

#### 3.1.3 Hybrid Invocation Control

To consider both the switching cost of service hosts and the flexibility of handling different service invocation request situations in real cases, we propose the hybrid invocation control mechanism by combining the above two approaches for different cases, which is shown in **Algorithm 3**.

**Algorithm 3** Hybrid invocation control mechanism for atomic service

```
procedure AtomicServiceInvocation3 ($s$, $t$)
1:  AtomicServiceInvocation2 ($s$, $t$)
2:  $H^0(s) \leftarrow H^c(s)$
3:  $cr^* \leftarrow 1$
4:  Set $cr(s)$ (control bound of a continuous request for $s$ from the same user)
5:  for all service request $r_s^t(i)$ do
6:      if $(i > 1)$ then
7:          if $(u_{a_i} = u_{a_{i-1}})$ then
8:              $cr^* \leftarrow cr^* + 1$
9:          else
10:             $cr^* \leftarrow 1$
11:         end if
12:     end if
13:     if $cr^* > cr(s)$ then
14:         $h^*(u_{a_i}) \leftarrow nearestServiceHost(u_{a_i})$
15:         if $h^*(u_{a_i}) \notin H^c(s)$ then
16:             $h_i^c(s) \leftarrow currentNearestServiceHost(u_{a_i}, s)$
17:             Switch availability of $s$ from service host $h_i^c(s)$ to $h^*(u_{a_i})$
18:             $H^c(s) \leftarrow H^c(s) \cup \{h^*(u_{a_i})\} - \{h_i^c(s)\}$
19:         end if
20:     else if $u_{a_i} \in U^*(s)$ then
21:         Switch availability of $s$ so that the current set of service entity host
            returns to $H^0(s)$
22:     end if
23: end for
```

For each time period, the service entity of a service $s$ is initially controlled based on the user-based invocation control mechanism where service entities are switched to be available on the service hosts that are nearest to the users with the most potential requests. During the time period, service invocation requests are monitored in real time. If situations become different from what is anticipated, the available service entities will be dynamically switched to the nearest service host to the user that has continuous requests of the service when the request number exceeds the control bound $cr(s)$.

### 3.2 Invocation Control for Composite Services

#### 3.2.1 Individual Invocation Control

Service composition environments provide users with additional values of various new services by composing atomic services. Composite services are always used more frequently than atomic services. A simple invocation control mechanism for a composite service $w$ can be realized by applying invocation control mechanisms for all individual atomic services $S(w) = \{s_1, s_2, \ldots, s_n\}$ that compose the composite service $w$. The average response time for invoking a composite service $w$ can be expected to be reduced by applying an invocation control mechanism for each individual atomic service $s_i (i = 1, 2, \ldots, n)$. **Algorithm 4** shows the individual control using the hybrid invocation mechanism for each atomic service.

**Algorithm 4** Individual invocation control mechanism for composite service

```
procedure IndividualInvocation ($w$, $t$)
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: for all service $s_i \in S(w)$ do
3:     AtomicServiceInvocation3 ($s_i$, $t$)
4: end for
```

However, since the individual invocation control mechanism does not consider the group characteristics of atomic services, applying optimal invocation control mechanisms for atomic services individually does not guarantee that it is also optimal for the whole composite service.

### 3.2.2 Group Invocation Control

Since each composite service consists of a group of atomic services, we propose the group invocation control mechanism for a composite service based on the group characteristics of atomic services. That is to say, the group invocation control considers the group of atomic services that compose a composite service as an integral part instead of individuals. For example, if an atomic service $s_a$, an atomic service $s_b$ and an atomic service $s_c$ are used together in a composite service $w$ by users, then these three services can be considered as a service group when the invocation control policies are decided in the service composition environment. If we only consider the composite service $w$, we can create the same three algorithms as is described in Section 3.1 that are applied in $s_a$, $s_b$ and $s_c$ simultaneously (we call CompositeServiceInvocation1 $(w, t)$, CompositeServiceInvocation2 $(w, t)$ and CompositeServiceInvocation3 $(w, t)$). For example, CompositeServiceInvocation1 $(w, t)$ can be realized by **Algorithm 5**, which is modified from AtomicServiceInvocation1 $(s, t)$ (**Algorithm 1**). Similarly, we can realize CompositeServiceInvocation2 $(w, t)$ and CompositeServiceInvocation3 $(w, t)$ based on the modification of AtomicServiceInvocation2 $(s, t)$ (**Algorithm 2**) and AtomicServiceInvocation3 $(s, t)$ (**Algorithm 3**).

**Algorithm 5** Request-based invocation control mechanism for composite service

```
procedure CompositeServiceInvocation1 $(w, t)$
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: for all workflow request $r_w^t(i)$ do
3:     $h^*(u_{a_i}) \leftarrow nearestServiceHost(u_{a_i})$
4:     for all service $s_k \in S(w)$ do
5:         if $h^*(u_{a_i}) \notin H^c(s_k)$ then
6:             $h_i^c(s_k) \leftarrow currentNearestServiceHost(u_{a_i}, s_k)$
7:             Switch availability of $s_k$ from the service host $h_i^c(s_k)$ to $h^*(u_{a_i})$
8:             $H(s_k) \leftarrow (H(s_k) \setminus \{h_i^c(s_k)\}) \cup \{h^*(u_{a_i})\}$
9:         end if
10:    end for
11: end for
```

For the atomic services ($s_a$, $s_b$ and $s_c$) that compose the composite service, they may be invoked by users as either atomic services separately or a composite service $w$. Therefore, it is necessary to divide all the invocations to an atomic service into two parts: one part is for the atomic service invocation and the other is for the composite service invocation. For the composite service invocation part, we use the composite service invocation that is applied to all composed atomic services together. For the atomic service invocation part, we use the atomic service invocation that is applied to all consisted atomic services separately. **Algorithm 6** shows the group control based on the hybrid invocation mechanism for atomic services and composite services.

**Algorithm 6** Group invocation control mechanism for composite service

```
procedure GroupInvocation $(w, t)$
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: for all service $s_k \in S(w)$ do
3:     for all service request $r_{s_k}^t(i)$ do
4:         if ($r_{s_k}^t(i)$ is an invocation request to the composite service $w$) then
5:             CompositeServiceInvocation3 $(w, t)$
6:         end if
7:         if ($r_{s_k}^t(i)$ is an invocation request to the atomic service $s_k$) then
8:             AtomicServiceInvocation3 $(s_i, t)$
9:         end if
10:    end for
11: end for
```

Simply put, the difference between an individual invocation control mechanism and a group invocation control mechanism lies in that the former applies a dynamic invocation control of atomic services separately while the latter considers the group characteristics of atomic services and always executes a dynamic invocation control for the atomic services simultaneously when the atomic services are invoked as a composite service. That is to say, in the individual invocation control mechanism, all the atomic services only apply AtomicServiceInvocation

(AtomicServiceInvocation1, AtomicServiceInvocation2 or AtomicServiceInvocation3) whether they are invoked as atomic services or composite services. However, in the group invocation control mechanism, all the atomic services apply AtomicServiceInvocation when they are invoked as atomic services whereas they apply CompositeServiceInvocation when they are invoked as composite services.

## 4. Experiments and Analysis

### 4.1 Experiment Settings

To simulate the target service composition environments, it is necessary to satisfy the basic conditions: multiple service users and multiple service entity hosts that are geometrically distributed, service license constraints based on the policies of the service providers. Besides the above basic conditions, we also refer to the real data from existing service composition environments, the Language Grid. Therefore, we simulate the service composition environments by setting the user number $n = 100$ and the service entity host number $m = 20$. For each user $u_i$, the invocation response time of an atomic service $s$ on any service entity host $c(u_i, s, h_j)$ (ms) is randomly simulated with different probabilities as follows which is similar with the response time distribution on the Language Grid: (1) $p(100 \leq c(u_i, s, h_j) < 200) = 0.1$; (2) $p(200 \leq c(u_i, s, h_j) < 500) = 0.2$; (3) $p(500 \leq c(u_i, s, h_j) < 1000) = 0.2$; (4) $p(1000 \leq c(u_i, s, h_j) < 2000) = 0.4$; (5) $p(2000 \leq c(u_i, s, h_j) < 5000) = 0.1$. For each atomic service, the time cost for switching service entity hosts for executing service entities is randomly decided between 1,000 ms and 5,000 ms since the time cost varies according to the approaches (migration of service entity or license control) of switching service entity hosts. During the experiments, we simulate 10 time periods for each service and invoke the service for 5,000 times in each time period of simulation. Moreover, for each invocation request to an atomic service or a composite service, we set 50% as the probability that the current request is a continuous request from the same user. For each atomic service $s_i$, the sequence of users for invocation requests is totally random. Each composite service is set to be randomly composed by three to five atomic services with the sequential patterns and parallel patterns. The response time for a composite service $w$ from $u_i$ can be aggregated by the invocation response time of atomic services that constitute $w$.

### 4.2 Analysis

Based on the experiment settings, we conduct three measurements: the response performance of an atomic service, the response performance of an composite service, and the stability of response performance. The response performance of a service is evaluated by computing the average service invocation response time from all users for ten time periods (10*5000 invocations), which is described in **Definition 5**.

#### 4.2.1 Response Performance of Atomic Services

For the response performance of atomic services, we conducted four types of simulations. In each type, we have a different configuration for the percentage of requests that are made by users who make the most amounts of requests (Max Request User: 25%, 50%, 75%, random). For example, *Max Request User: 25%* means that in this type of simulation, 25% of the total requests are from the users with most amounts of requests in the time periods. In that case, the other 75% requests are randomly distributed among the rest of users.

From the results shown in **Fig. 1**, we have the following observations. First, we find that the hybrid invocation control mechanism that considers both potential users for most service invocation requests and potential users for continuous requests always improves the response performance and performs the best. The reason lies in that the hybrid invocation control mechanism costs very little for switching service entity hosts while it provides flexibility to consider the dynamic changes of user requests as well. Second, the request-based invocation control mechanism and the user-based invocation control mechanism decrease the response performance when the requests from users are totally random because the cost of switching service entity hosts is too much for the request-based invocation control mechanism and there are no dominant users for the user-based invocation control mechanism. However, when there exist dominant users (Max Request User: 25%, 50%, 75%), the user-based invocation control mechanism can improve the response performance in most cases, while the request-based invocation control mechanism can improve the response performance only when the original response time is large. Moreover, the response performance of user-based invocation control mechanism approaches that of the hybrid invocation control mechanism if the users who have the most amounts of requests for service invo-

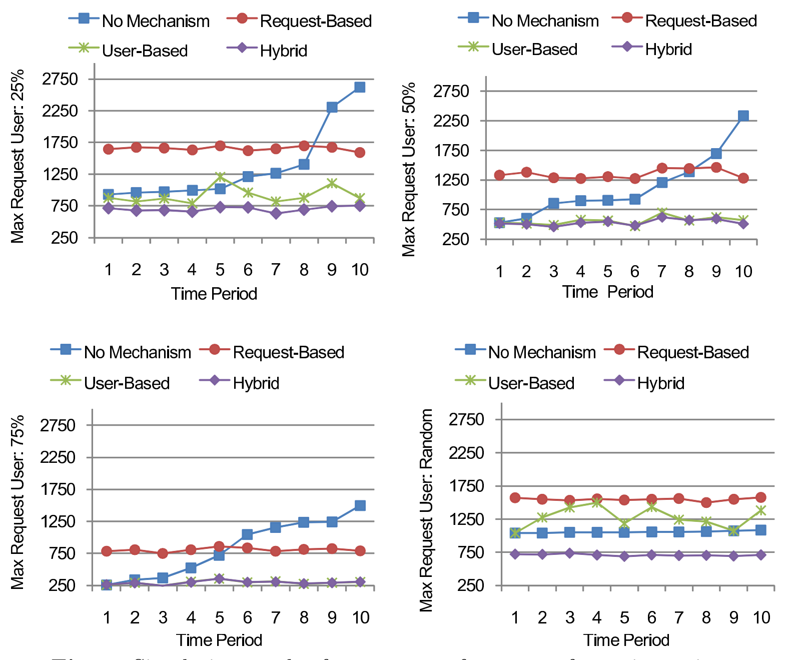

**Fig. 1** Simulation result of response performance of atomic services.

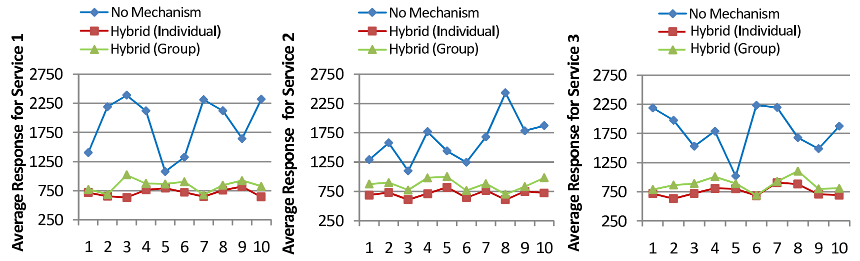

**Fig. 2** Influences of response performance of composite services on response performance of atomic services.

**Table 1** Simulation result of response performance of composite services.

| Simulation | No Mechanism | Individual Control | Group Control |
|---|---|---|---|
| 1–5,000 | 4,068 ms | 2,890 ms | 2,209 ms |
| 5,001–10,000 | 6,490 ms | 3,432 ms | 1,788 ms |
| 10,001–15,000 | 8,643 ms | 3,098 ms | 1,877 ms |
| 15,001–20,000 | 3,897 ms | 2,678 ms | 2,216 ms |
| 20,001–25,000 | 4,654 ms | 2,766 ms | 2,358 ms |
| 25,001–30,000 | 4,357 ms | 2,780 ms | 2,197 ms |
| 30,001–35,000 | 9,402 ms | 4,222 ms | 1,922 ms |
| 35,001–40,000 | 6,543 ms | 3,754 ms | 2,302 ms |
| 40,001–45,000 | 4,789 ms | 3,243 ms | 2,425 ms |
| 45,001–50,000 | 6,778 ms | 3,544 ms | 1,876 ms |
| Average | 5,962 ms | 3,241 ms | 2,117 ms |

cation are significantly dominant (Max Request User: 50%, 75%).

#### 4.2.2 Response Performance of Composite Services

Since the hybrid invocation control mechanism for atomic service can bring the best response performance of atomic services, we use it in the individual invocation control mechanism and the group invocation control mechanism for composite services. In **Fig. 2**, Service 1, Service 2 and Service 3 compose the composite service we use for the experiments with sequential patterns.

The result in **Table 1** shows that using the group invocation control for composite service can get a better response performance comparing to the individual invocation control. Second, the results in Fig. 2 show that using the group invocation control for composite service may insignificantly decrease the response performance for each atomic service. The reasons lie in that the individual invocation control mechanism only considers the optimal invocation mechanism of individual atomic services and therefore it is not always optimal for the response performance of a composite service, but optimal for each atomic service. However, the group invocation mechanism always considers the optimal response performance for the invocation of a composite service, and it is not always optimal for each individual atomic service. Therefore, when using the group invocation control mechanism, the response performance of each atomic service decreases as shown in Fig. 2.

#### 4.2.3 Stability of Response Performance

**Table 2** shows the results by using the performance index of the average invocation response difference rate (defined in **Definition 7**) for 100 users. The results are based on 10*5000 simulations by comparing the invocation without dynamic control and the hybrid invocation control mechanism. From the results,

**Table 2** Average invocation response difference rate for 100 users.

| Simulation | No Mechanism | Hybrid Invocation Control |
|---|---|---|
| 1–5,000 | 52.70% | 23.09% |
| 5,001–10,000 | 58.91% | 19.82% |
| 10,001–15,000 | 49.89% | 21.78% |
| 15,001–20,000 | 48.12% | 25.24% |
| 20,001–25,000 | 53.45% | 20.12% |
| 25,001–30,000 | 51.17% | 19.21% |
| 30,001–35,000 | 48.97% | 21.89% |
| 35,001–40,000 | 57.72% | 24.51% |
| 40,001–45,000 | 52.33% | 21.23% |
| 45,001–50,000 | 56.71% | 17.93% |
| Average | 53.00% | 21.48% |

we can see that the average invocation response difference rate for each 5,000 simulations ranges from 48.12% to 58.91% when the dynamic invocation control mechanisms are not applied, which reflects the response time distribution we have set for the simulation. However, the average invocation response difference rate decreased to range from 17.93% to 25.24% when applying the hybrid invocation control mechanisms, which means that the response performance becomes much more stable.

## 5. Related Work

QoS-aware service composition has become an important topic in the area of service composition. Zeng, et al.[8] propose a multidimensional QoS model for Web service composition and several optimization approaches for service selection in both static environments and dynamic environments. Although we do not deal with all the QoS dimensions, we consider the real problems and constraints in the service composition environments.

Dynamic service deployment and invocation has been discussed in the area of grid computing, where the issue of efficient and reliable resources sharing is always the focus[5),9)]. However, services are always wrapped by resources provided by various service providers for different purposes in service composition environments. Therefore, policies of service providers might bring the constraints of service license problems that we deal with in this research.

Research efforts on the dynamic service execution control can also be found in some previous work. Liu, et al.[10] addresses the issue of improving the performance of service-based applications. They propose a dynamic service execution mechanism to explore implicit parallelism at runtime, which is different from our research focus. One of our significant contributions is that we propose the service invocation problems that consider the distribution of service users and service entity hosts, and the policies of service providers.

## 6. Conclusion

This paper deals with the issues of the service invocation control in service composition environments considering the service license constraints by service providers. First, we proposed a formal model for the service invocation problem. Then, we design dynamic service invocation control mechanisms for both atomic services and composite service to improve the response performance. Furthermore, several performance indexes are defined to evaluate the response performance of atomic services and composite services, and the stability of the response performance. The experimental results shows that (1) For atomic services, the hybrid invocation control mechanism that considers both potential users for most service invocation requests and potential users for continuous requests can best improve the response performance; (2) For composite services, the group invocation control can improve the response performance more but may insignificantly decrease the response performance for each atomic service; and (3) Dynamic invocation control mechanisms can bring a relatively stable response performance from the perspective of users.

There are also several issues that should be further considered in our future work. First, more experiments should be conducted by varying simulation configurations to observe the important factors that might affect the performances. Second, more complicated situations for composite services should be considered, such as how to deal with the problem that one atomic service might be simultaneously requested by multiple composite services and so on.

**Acknowledgments** This work was supported by Strategic Information and Communications R&D Promotion Programme (SCOPE) from the Ministry of Internal Affairs and Communications of Japan.

## References

1) Ishida, T.: Language grid: An infrastructure for intercultural collaboration, *IEEE/IPSJ Symposium on Applications and the Internet* (*SAINT-06*), pp.96–100 (2006).
2) Pathan, M. and Buyya, R.: A taxonomy of CDNs, *Content Delivery Networks*, pp.33–78 (2008).
3) Jaeger, M., Rojec-Goldmann, G. and Muhl, G.: QoS aggregation for Web service composition using workflow patterns, *Proc. 8th IEEE International Conference on Enterprise Distributed Object Computing*, IEEE Computer Society Washington, DC, USA, pp.149–159 (2004).
4) Van Der Aalst, W., Ter Hofstede, A., Kiepuszewski, B. and Barros, A.: Workflow patterns, *Distributed and Parallel Databases*, Vol.14, No.1, pp.5–51 (2003).
5) Byun, E. and Kim, J.: DynaGrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications, *Parallel Computing*, Vol.33, No.4-5, pp.328–338 (2007).
6) Casati, F. and Shan, M.: Dynamic and adaptive composition of e-services, *Inf. Syst.*, Vol.26, No.3, pp.143–163 (2001).
7) Bontis, N. and Chung, H.: The evolution of software pricing: From box licenses to application service provider models, *Internet Research: Electronic Networking Applications and Policy*, Vol.10, No.3, pp.246–255 (2000).
8) Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J. and Chang, H.: QoS-aware middleware for web services composition, *IEEE Trans. Softw. Eng.*, Vol.30, No.5, pp.311–327 (2004).
9) Friese, T., Smith, M. and Freisleben, B.: Hot service deployment in an ad hoc grid environment, *Proc. 2nd International Conference on Service Oriented Computing*, ACM, pp.75–83 (2004).
10) Liu, H., Wang, X., Luo, T., Song Li, X. and Li, W.: Improving the performance of service-based applications by dynamic service execution, *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp.174–182 (2008).

(Received May 31, 2010)
(Accepted November 5, 2010)
(Released February 9, 2011)

**Donghui Lin** was born in 1982. He received his M.E. degree in computer science and engineering at Shanghai Jiao Tong University in 2005, and Ph.D. degree in social informatics at Kyoto University in 2008. He is a researcher of National Institute of Information and Communications Technology, Japan. His research interests include services computing, multiagent systems and intercultural collaboration. His current work focuses on the QoS issues in service composition environments, which has been presented in ICSOC and SCC, the major international conferences in the area of services computing.

**Yohei Murakami** was born in 1978. He received his Ph.D. in informatics degree from Kyoto University in 2006. He is a researcher of National Institute of Information and Communications Technology, Japan. He currently leads the research and development of Language Grid project, the purpose of which is to share language resources as Web services and enable users to create new services. His research interests lie in services computing and multiagent systems. He founded the Technical Committee on Services Computing in the IEICE in 2009.

**Masahiro Tanaka** was born in 1981. He earned his Ph.D. in informatics from Kyoto University in 2009. He is a researcher of National Institute of Information and Communications Technology, Japan. He is currently working on services computing, focusing on runtime execution management of Web services. His achievements about runtime execution framework for composite Web services have been published at major international conferences in services computing. He also has experiences on development of infrastructures for Web service composition and service-based applications.