Strategy-Proof Pricing for Cloud Service Composition

Masahiro Tanaka and Yohei Murakami

Abstract—The on-demand provisions of cloud services create a service market, where users can dynamically select services based on such attractive criteria as price and quality. An intuitive model of a service market is a reverse auction. In the first price auction, however, a service that is cheaper and provides better quality is not necessarily selected. This causes undesirable outcomes both for users and service providers. A possible solution is the Vickrey-Clarke-Groves (VCG) mechanism, where the dominant strategy for a service provider is to report the true cost of his service. In spite of this desirable property, implementing the VCG mechanism for service composition suffers from computational cost. The calculation of payments to service providers based on the VCG mechanism requires iterative service selection, even though each service selection can be NP-hard. Approximation algorithms cannot be applied because approximate solutions do not assure the desirable property of the VCG mechanism. Thus, we model VCG payments for service markets and propose a dynamic programming (DP)-based algorithm for service selection and VCG payment calculation. Our proposed algorithm solves service selection in quasi-polynomial time and gives an exact solution. Moreover, we extend it and focus on the iterative service selection process for VCG payment calculation to improve its performance. Our series of experiments show that our proposed algorithm solves practical scale service composition.

Index Terms—Service composition, service selection, dynamic programming, VCG mechanism

1 Introduction

In the cloud computing era, the elasticity of cloud environments has introduced significant benefits to both SaaS service providers and users who build service-oriented architecture (SOA) systems by combining services. SaaS service providers are freed from managing computational resources by operating their services on IaaS platforms. Since we focus on SaaS services in this paper, we hereinafter refer to them as a "*service*." Users can immediately start using them based on their needs. This creates a *service market*, where users can dynamically select services from a number of functionally equivalent candidates based on their non-functional properties including price and quality. On the other hand, in such a service market, the elasticity of cloud environments allows service providers to dynamically set the prices and the quality of their services, both of which are expected to be determined by their autonomous agents. The users also ask their autonomous agents to purchase services because there are too many functionally equivalent services to manually manage.

An SOA system designer often defines an abstract business logic that combines services based on their functional properties. Once the service interfaces are standardized in the service market, executable services can be bound for invocation at runtime based on the non-functional properties of the services. The user's autonomous agent is expected to find a combination of services that satisfy the overall requirements for the composite service based on the price and the quality of the services.

The most intuitive service market model is a reverse auction, where service providers offer their services at a certain price and quality and users select a combination of them and pay the proposed price. However, how service providers price their services is not trivial. If the model is a (sealed) first price auction, a service provider has no dominant strategy unless it knows the prices of its competitors' services. The service provider earns profit only if he successfully sets his service's price cheaper than the others and his service is selected. If he fails to do so, he loses profit even when the actual cost of his service is cheaper and its quality is better than his competitors. Although the best price for the service provider is one that is slightly lower than the others, such information is impossible to know. This leads to undesirable outcomes for both users and service providers.

A possible approach to the above problem is the Vickrey-Clarke-Groves (VCG) mechanism [1], [2], [3]. Payments based on it guarantee that desirable properties include *strategy-proofness*. If a mechanism is strategy-proof, the dominant strategy for service providers is to report the true value of their service. In the context of service composition, the "true value" corresponds to the service's cost: the minimum value they need to receive.

On the other side of the coin, the VCG-based mechanism suffers from computational complexity. Given a set of candidate services, the computational cost to find a combination of services that minimizes the overall price of the composite services under a certain constraint on quality. Moreover, the VCG mechanism runs service selection $N + 1$

- *M. Tanaka is with the National Institute of Information and Communications Technology, 3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0289, Japan. E-mail: mtnk@nict.go.jp.*
- *Y. Murakami is with the Unit of Design, Center for the Promotion of Interdisciplinary Education and Research, Kyoto University, Yoshida-honmachi, Sakyo-ku, 606-8501, Kyoto, Japan. E-mail: yohei@i.kyoto-u.ac.jp.*

Manuscript received 16 Sept. 2013; revised 22 Apr. 2014; accepted 16 May 2014. Date of publication 11 July 2014; date of current version 7 Sept. 2016. Recommended for acceptance by I. Bojanova and C.-H. Hsu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2338310

times when a user selects N services for a composite service. Although there are polynomial algorithms that find approximate solutions, they do not assure strategy-proofness. This makes implementing a strategy-proof mechanism more computationally expensive. A previous work applied VCG-based payments to cloud service selection as a combinatorial reverse auction. Mihailescu and Teo showed strategy-proof selection designed for dynamic pricing [4]. But they focus on maximizing the utility of users and service providers. The computational complexity for service selection (winner determination) based on their model is polynomial due to its simplicity.

In this paper, we first show strategy-proof pricing based on the idea of the VCG mechanism. Then we propose a dynamic programming (DP)-based approach that implements service selection and calculates the payments. This approach's computational complexity is quasi-polynomial when it gives an exact solution. We also propose an extended approach to reduce the computational complexity. Our DP-based approach records the process of the service selection. Since calculation of the VCG payments requires that service selection be iteratively performed excluding one service provider, the extended approach uses the record of the process as a cache and greatly reduces the computational complexity in the iterative service selection.

The rest of this paper is organized as follows. In Section 2, we explain the assumption regarding stakeholders in this paper, the fundamental model of service composition, and a VCG-based mechanism based on the model. Next we describe our DP-based algorithm for the VCG payment calculation in Section 3. In Section 4, we show some experimental results about computational cost, the success ratio of service composition, and utility of users and service providers. After introducing related works in Section 5, we conclude this paper in Section 6.

2 Model

In this section, we explain our assumption regarding stakeholders and the fundamental model of service composition. Then we apply the VCG mechanism idea to our model by formalizing VCG payments and the procedure to calculate them following our model.

2.1 Stakeholders

In this paper, we focus on the composition of various functions that are available as services and on "SaaS" services that work as software components, not on PaaS or IaaS. This is why we refer to SaaS services and their providers as "*services*" and "*service providers*," respectively. In addition, we assume services are operated on scalable IaaS platforms. Thus, services can manage as many requests as received because the dynamic changes of the requirements for computational resources are handled by the IaaS platforms. Since the service providers are different from the IaaS platform providers, the former are not concerned with the capacity of the computational resources.

Fig. 1 illustrates the relationships among the stakeholders. On each IaaS platform, many service providers offer their services. The user chooses services that may be operated on several different IaaS platforms. Our scope in this paper includes only the service providers, the user, and their deals. Therefore, we do not discuss the operation of the IaaS platform, which aims for computational resource management based on the amount of requests.

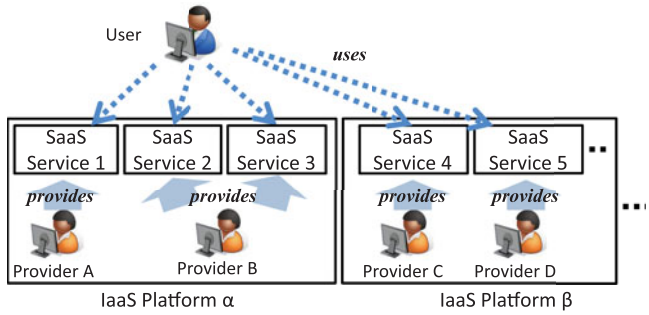

Fig. 1. Stakeholders of service composition.

Our assumption is natural in the cloud computing era. One well-known service in the real world is Instagram,[1] which is operated on Amazon EC2, a scalable IaaS platform. Various functions of Instagram are provided as APIs. Application developers can easily combine APIs with those of other services as a composite service.

2.2 Service Composition

We assume a composite service is designed based on *abstract services*, which only define the interface. An executable service is called a *concrete service*, which is bound to the abstract service at the runtime. We assume that many concrete services are available in a service market and can be bound to abstract services. A service provider offers a concrete service at a particular price and quality. A user of a composite service sets the maximum price and the minimum quality of the overall composite service as constraints and selects a combination that satisfies them.

Unfortunately, no big, open service market exists in the real world. However, more and more services are becoming available on the web. For example, the statistics of ProgrammableWeb[2] show that the number of web services is still increasing. Approximately 10,000 services are registered on ProgrammableWeb. Another example is the Language Grid [5], which is a platform for language services, such as machine translators and dictionaries. Since some standardized interfaces are defined for it, service providers can make their services composable by wrapping language processing programs based on the interfaces. On the other hand, the trend to monetize services continues to increase and a wide variety of business models have emerged. For example, the GoogleTranslate API used to be free, but now users need to pay based on the amount to be translated. Following that trend, such other translation APIs as J-Server and WEB-Transer have introduced similar business models. As shown by these examples, service markets are becoming real.

Fig. 2 shows an example of service composition. This is a typical example of a combination of SaaS services. A composite service is represented as a workflow that contains three abstract services. An abstract service corresponds to a

1. http://instagram.com/.
2. http://www.programmableweb.com/.

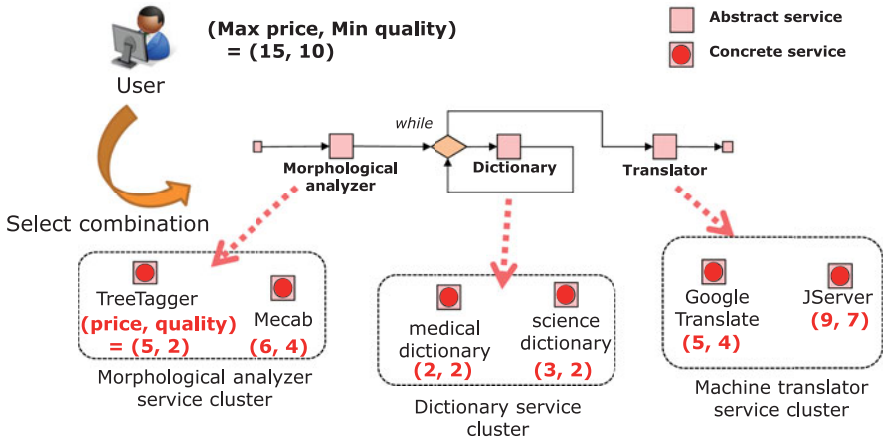

Fig. 2. Example of service composition.

service cluster, which is a collection of bindable concrete services. In this example, two concrete services are available in each service cluster. Concrete services in the same service cluster are functionally equivalent, but they have different non-functional properties. In a machine translation cluster, for instance, the cost of GoogleTranslate is much lower than JServer but the latter's translation quality surpasses the former's. In this paper, we focus on price and overall quality representing such non-functional properties as throughput and availability. In Fig. 2, a pair of values around a concrete service represents its price and quality.

2.3 Formalization

We denote the jth service in the ith service cluster as $s_{i,j}$. We also denote the price and the quality of $s_{i,j}$ as $p_{i,j}$ and $q_{i,j}$, respectively. Given N service clusters, the price of the overall composite service is defined as the sum of the prices of selected services $\sum_i^N \alpha_{i,j} p_{i,j_i}$, where $\alpha_{i,j}$ represents the service selection. In this paper, we define $\alpha_{i,j}$ as $\alpha_{i,j} \in \{0, 1\}$ because we assume a user selects only one concrete service for an abstract service. We also assume a user has aggregate function $f(\{\alpha_{i,j}\}, \{q_{i,j}\})$, which takes the selected services and quality values and returns the composite service's overall quality. The aggregate function's objective is to represent the structure of the user's preference and criteria about quality, which can be much more complex than price. The user determines maximum price P and minimum quality Q as constraints to minimize the total price under the given constraints. If any combination of concrete services fails to satisfy the constraints, the service composition fails.

The goal and constraints are formally defined as follows. Since $\alpha_{i,j}$ is 1 or 0, this problem is 1-0 integer programming:

$$min. \sum_i^N \alpha_{i,j} p_{i,j_i} \tag{1}$$

$$\sum_i^N \alpha_{i,j} p_{i,j_i} \leq P \tag{2}$$

$$f(q_{1,j_1}, \ldots, q_{N,j_N}) \geq Q \tag{3}$$

$$\forall i, \sum_j \alpha_{i,j} \geq 0 \tag{4}$$

$$\forall i, \sum_j \alpha_{ij} = 1. \tag{5}$$

The applicability of the formalization depends on the application. If an input to a service can be divided into any fragment size that can be processed by different services to return the same results, we define $\alpha_{i,j}$ as a real number between 0 and 1. In this case, this problem is defined by linear programming and the computational complexity is much less than the 1-0 integer programming. In general, unfortunately, the services provided by different service providers return different results. In the above example, different translation services return different translation results. If we divide a document into sentences and translate them using different translation services, the overall results may be inconsistent. This is why we formalize service composition as 1-0 integer programming.

The symbols for formalization including those used in the latter sections are summarized in Table 1.

2.4 VCG-Based Mechanism

If a user selects services and determines payment as a first price reverse auction, results of the service selection can be unstable. We show two example scenarios to describe this problem.

In the first scenario, assume two services, s_A and s_B. A pair of $(cost, quality)$ of s_A is $(10, 5)$ and that of s_B is $(12, 5)$. The service providers of these services need to determine a price that exceeds the cost of the service to make a profit. However, if they set a higher price than the others, they lose any chance to make a profit. Assume the provider of s_A sets its price to 14, and the provider of s_B sets its price to 13. Under this result, the user selects s_B, although the cost of s_A is cheaper. In other words, the provider of s_A loses any possible profit due to a pricing failure.

In the second scenario, assume again two services, s_A and s_B. The pair of $(cost, quality)$ of s_A is $(10, 6)$ and s_B is $(12, 5)$. Even though the provider of s_A sets its price to 14 and the provider of s_B sets its price to 13, the budget for the service cluster is 13.5 due to the selection's result at another service cluster. Here, the user also selects s_B, although s_A provides both better cost and quality.

Such undesirable results reflect that service providers must determine the price of their services by only relying

TABLE 1
Formalization Symbols

Symbol	Description
P	maximum price
Q	minimum quality
S	set of available services
$s_{i,j}$	jth service in ith service cluster
$p_{i,j}$	price of s_{ij}
$q_{i,j}$	quality of s_{ij}
N	number of service clusters
m	number of services in a service cluster
d	discretization parameter
$\alpha_{i,j}$	flag that shows whether $s_{i,j}$ is selected. If selected $\alpha_{i,j}$ is 1, otherwise 0.
$f(\{\alpha_{i,j}\},\{q_{i,j}\})$	quality aggregate function
W	set of services selected from S satisfying constraints P and Q.
$W_{-i,j}$	set of selected services except for $s_{i,j}$
$W'_{-i,j}$	set of services that would be selected if $s_{i,j}$ were not available.
$u_{i,j}$	utlity of s_{ij}'s provider
U	user's utility
M	matrix used for DP, whose row corresponds to a selection at a service cluster and whose column corresponds to a certain quality value.
$e_{c,q}$	element of M that corresponds to a subsolution of service selection
$bs_{c,q}$	subsolution of service selection, which contains services from clusters 1 to c and achieves the cheapest price at quality q.
$bp_{c,q}$	price realized by $bs_{c,q}$
$AS_{c,q}$	set of all subsolutions expanded from all $bs_{c-1,q}$ and $s_{c,j}$, all services in service cluster c.
$AP_{c,q}$	set of price values given by $AS_{c,q}$.

on the expectation of the pricing of other service providers. However, the prices of services change frequently based on requests from users and the cost of computational resources. This makes it difficult for service providers to manually determine prices based on the prices of competitor services.

To solve this problem, we apply the idea of the VCG mechanism. In combinatorial auctions, for instance, it ensures strategy-proofness, where a buyer's dominant strategy is to report her true value. In the context of service composition, the following differences exist with typical VCG auctions. First, our model is a reverse auction. Service providers report the cost of their services as true values. Second, unlimited capacity of services can be provided because they are deployed on scalable IaaS platforms. Third, services are selected from a particular number of service clusters. A service in a service cluster cannot be an alternative of other services in a different service cluster. As a result, excluding service s_{ij} from service cluster i in Algorithm 1 may completely change the service selection in different service clusters.

One of our goals is to propose a strategy-proof mechanism for our service composition regardless of the above differences. We define the payments to service provider $s_{i,j}$ as follows:

$$payment_{i,j} = \sum_{s_{i',j'} \in W'_{-i,j}} p_{i',j'} - \sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'} \tag{6}$$

W, $W_{-i,j}$, and $W'_{-i,j}$ are defined as follows:

- W. Set of selected services
- $W_{-i,j}$. Set of selected service except for $s_{i,j}$
- $W'_{-i,j}$. Set of services that would be selected if $s_{i,j}$ were not available.

Below we show that the payment defined by formula (6) is strategy-proof.

Theorem. *The payment defined in (6) is strategy-proof.*

Proof. When a service provider of $s_{i,j}$ sets a price that exceeds its true cost, there are two possible cases. One, the service is not selected because the price and the profit of the service are zero. Another is the case where the service is selected. But neither $\sum_{s_{i',j'} \in W'_{-i,j}} p_{i',j'}$ nor $\sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'}$ are affected by the price of $s_{i,j}$. Therefore, $s_{i,j}$ does not bring a profit. Note that the service providers are rational and do not provide their services if their payment is lower than the cost. Therefore, they do not set a price that is lower than their true cost. As a result, a false report of a service's cost does not increase the profits of service providers. □

Algorithm 1. RunVCG(S, P, Q)

```
S: set of available services {s_{i,j}}
P: max price of composite service
Q: min quality of composite service
W ← Select(S, P, Q)
if W is not found then
  return Failure of service composition
end if
for all s_{i,j} ∈ W do
  W'_{-i,j} ← Select(S_{-i,j}, P, Q)
  if W'_{-i,j} is not found then
    return Failure of service composition
  end if
payment_{i,j} ←
    Σ_{s_{i'j'} ∈ W'_{-i,j}} p_{i'j'} − Σ_{s_{i',j'} ∈ W_{-i,j}} p_{i',j'}
end for
U ← P − Σ_i^N α_{i,j} p_{i,j_i}
if U < 0 then
  return Failure of service composition
end if
return payments_{i,j} s.t. ∀i, j s_{i,j} ∈ W
```

Since we assume the proposed price set by the service providers equals their cost, we define the utility of the service providers as follows:

$$u_{i,j} = payment_{i,j} - p_{i,j} \tag{7}$$

P is the maximum total price determined by the user. Considering this to be the user's budget for the service composition, we define the user's utility as follows:

$$U = P - \sum_{i}^{N} \alpha_{i,j} p_{i,j_i}, \tag{8}$$

$u_{i,j} > 0$ always holds as shown below

$$\begin{aligned} u_{i,j} &= payment_{i,j} - p_{i,j} \\ &= \sum_{s_{i',j'} \in W'_{-i,j}} p_{i',j'} - \sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'} - p_{i,j} \\ &> \sum_{s_{i',j'} \in W} p_{i',j'} - \sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'} - p_{i,j} \\ &= \left(\sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'} + p_{i,j} \right) \\ &\quad - \sum_{s_{i',j'} \in W_{-i,j}} p_{i',j'} - p_{i,j} \\ &= 0. \end{aligned}$$

Note that $\sum_{s_{i',j'} \in W} p_{i',j'} < \sum_{s_{i',j'} \in W'_{-i,j}} p_{i',j'}$ because W is cheaper combination of services than $W_{-i,j}$.

On the other hand, U can be a negative value based on the above definition. Because a rational user does not accept negative utility, service composition fails in such a case. This shows that additional payments may cause failure of the service composition in exchange for strategy-proofness. We discuss this issue in Section 4.

The procedure, which includes service selection and VCG payment calculation, is shown in Algorithm 1:

The **Select** procedure returns a combination of services that satisfy the constraints on price P and quality Q. If no combination is found, **Select** returns a failure of the service composition. Given service clusters N, **Select** is performed N times to exclude each service $s_{i,j}$ from S to calculate the VCG payments. Once **Select** fails, the service composition fails. $S_{-i,j}$, which is used by **Select** (line 9), is a set of services given by excluding $s_{i,j}$ from S. After determining the payments to all service providers, the procedure calculates U to check if the user's utility is positive. As mentioned above, the composition fails if the user's utility is negative.

It is well-known that the computational complexity of the winner determination in auctions is NP-hard when an exact solution is needed. One of the most intuitive approaches is backtrack and branch-and-bound. Suppose the computational complexity of **Select** in line 4 is $O(m^N)$, given N service clusters that have m services. Similarly, the computational complexity of **Select** in line 9 is $O((m-1) \cdot m^{N-1})$. Since lines 8-14 are iterated N times, the computational complexity of the whole process is $O(m^N + N((m-1) \cdot m^{N-1})) \simeq O(Nm^N)$. This is serious when we apply Algorithm 1 to a large-scale problem.

Another problem is that the desirable properties of VCG mechanisms can be negated by such malicious behaviors as shill biding. Such behaviors usually appear under the settings of combinatorial auctions. In combinatorial auction, for example, a buyer can submit a bid on a bundle of multiple goods. However, in Internet auction, the buyer can split a bid as if different buyers submitted the split bids. This problem is called *false-name bids*, which is one of the most well-known problems of the VCG mechanism in Internet auction. In [6], the authors showed that the buyer can gain utility by the false-name bids. Most of the malicious behaviors including false-name bids appear only in combinatorial auction, where the sum of values of separated bids and the value of a bid on the bundle can be different. In our model, however, we do not introduce bids on a bundle of services. The price of a composite service is the sum of prices of selected services. In that sense, our model is not a combinatorial auction although the user selects a set of services. Therefore, such malicious behaviors do not work in our model. The design of our model is based on the assumption that there is no limitation on the capacity of services that are operated on IaaS platforms.

3 DP-Based Algorithm

Although the 1-0 integer programming problem shown in the previous section is NP-hard, the strategy-proofness of VCG payments requires an exact solution; approximation algorithms are not appropriate. In this paper, we apply a dynamic programming-based algorithm, which is a quasi-polynomial time algorithm.

In addition to the applicability to NP-hard problems, the DP-based approach has another advantage. It can record the process's progress in quasi-polynomial sized spaces. We can reduce the calculation by reusing this record when we iteratively run **Select**, excluding a service selected during the first selection.

In the following, we first introduce a DP-based algorithm for service selection and then propose an extension of the algorithm to improve the performance of VCG payment calculation.

3.1 Service Selection

First, we define matrix M, whose row corresponds to a selection at a service cluster and whose column corresponds to a certain quality value. Element $e_{c,q}$ of M corresponds to a subsolution of the service selection and is formally defined as follows:

$$e_{c,q} = (bs_{c,q}, bp_{c,q}),$$

where $bs_{c,q}$ is a subsolution of the service selection, which contains services from clusters 1 to c and achieves the cheapest price at quality q. q is a discretized value of the aggregated quality of the services in subsolution $bs_{c,q}$. An aggregated quality is a real value, but we discretize it as $round(quality * d)$ to place it in M, where d is the scale for discretization. $bp_{c,q}$ is the price achieved by $bs_{c,q}$.

Based on these definitions, Algorithm 2 gives the best feasible selection and quality.

First, the algorithm initializes the first row of M according to the services in service cluster 1 (lines 4-8). The subsolutions that only contain services in service cluster 1 are put by the quality. f is the quality aggregate function, which takes a set of services and returns the aggregated quality values. Then it iteratively sets values to the following rows that expand the subsolutions. A subsolution is expanded by combining a subsolution with a possible service in the next service cluster. If the same quality value is given by different subsolutions, the cheaper subsolution overwrites the existing one. If an expanded subsolution does not satisfy the maximum price constraint, it is discarded. After the

calculation finishes all the rows, the algorithm searches for the last row for an element with the cheapest price.

Algorithm 2. Select(S, P, Q)

```
1: S: set of available services {s_{i,j}}
2: P: max price of composite service
3: Q: min quality of composite service
4: for all service s_{1,j} ∈ S do
5:   if p_{1,j} < P and (e_{1,f({s_{1,j}})} is undefined
       or p_{1,j} < bp_{1,f({s_{1,j}})}) then
6:     Update e_{1,f({s_{1,j}})} to ({s_{1,j}}, p_{1,j})
7:   end if
8: end for
9: for c = 1 to N − 1 do
10:  for all e_{c,q} which is defined do
11:    for all s_{c+1,j} in service cluster c + 1 do
12:      sol' ← combine bs_{c,q} and s_{c+1,j}
13:      p' ← price of sol', q' ← f(sol')
14:      if p' < P and (e_{c+1,q'} is undefined
         or p' < bp_{c+1,q'}) then
15:        Update e_{c+1,q'} to (sol', p')
16:      end if
17:    end for
18:  end for
19: end for
20: return e_{N,q} s.t. ∀q' ≠ q, bp_{N,q} ≤ bp_{N,q'}, q > Q
```

There are some requirements for quality aggregate function to implement Algorithm 2. First, the aggregated quality values must also be defined for the subsolutions without being limited to complete solutions that contain selections at all the service clusters. Second, the quality aggregate function must be a monotonic increasing function with respect to quality. Finally, the price of the services must be in a finite range because the quality is discretized to be mapped in M.

Not all applications satisfy the above requirements. The calculation of aggregated quality is often much more complex than the calculation of price. Time and availability are typical QoS metrics to which we can apply our method. Many previous papers such as [7] have proposed the quality aggregation of such QoS metrics for composite services based on the control constructs. For example, the response time of a composite service that executes component services in sequence is defined as the sum of the response time of component services. When a composite service executes component services in parallel, the overall response time is the max values of those of component services. Most of such proposed aggregation functions satisfy the above requirements.

The effect of price discretization is less significant than the aggregation of quality regarding applicability. Since price lists for real services require little precision, we consider discretization scale d to be at most 100. We discuss the effect of the discretization scale on the computational complexity in Section 4.

We consider the algorithm's computational complexity. For the nth service cluster, it requires the calculation of price and updating the value $O(nmQ/N)$ times, because the average service prices are approximately Q/N. Therefore, each time the process in an iteration of a loop for a service cluster runs, the maximum number of filled elements in a row is expected to increase by approximately Q/N until a row is filled with an element. At the nth row, the maximum price is expected to be nQ/N. For each price value, the calculation of the quality and its update are performed m times. Therefore, the computation cost for a row in M is $O(nmQ/N)$. Since each service composition using N rows is repeated N times for VCG payment calculation, the overall execution time is $\Sigma_{n=1}^{N} O(nmQ/N) \cdot N \simeq O(mQN^2)$.

3.2 Extension for VCG Payment

As shown in Algorithm 1, the VCG payment calculation iteratively performs service selection as many times as the number of service clusters. Since each selection is computationally expensive, VCG payment calculation may take a long time. Thus, we extend Algorithm 2 to reduce the computational cost.

The idea of our extended algorithm is to reuse matrix M. Not all of its elements need to be updated when we iterate the selection that excludes $s_{i,j}$, which was selected during the first selection. To reuse M, selections that contain $s_{i,j}$ must be removed first. Elements whose best selection relies on $s_{i,j}$ also need to be updated.

Algorithm 3. SelectForVCG(S, P, Q)

```
1: S: set of available services {s_{i,j}}
2: P: max price of composite service
3: Q: min quality of composite service
4: for all service s_{1,j} ∈ S do
5:   if p_{1,j} < P then
6:     if e_{1,f({s_{1,j}})} is undefined
       or p_{1,j} < bp_{1,f({s_{1,j}})} then
7:       Update e_{1,q} to ({s_{1,j}}, f({p_{1,j}}))
8:     end if
9:     Add {s_{1,j}} to AS_{c,q_{i,j}}
10:    Add {p_{1,j}} to AP_{c,q_{i,j}}
11:  end if
12: end for
13: for c = 1 to N − 1 do
14:   for all p s.t. e_{c,q} is defined do
15:     for all s_{c+1,j} in service cluster c + 1 do
16:       sol' ← combine bs_{c,q} and s_{c+1,j}
17:       p' ← price of sol', q' ← f(sol')
18:       if p' < P then
19:         if e_{1,f({s_{1,j}})} is undefined
            or p' < bp_{c+1,q'} then
20:           Update e_{c+1,q'} to (sol', p')
21:         end if
22:         Add sol' to AS_{c,q'}
23:         Add p' to AP_{c,q'}
24:       end if
25:     end for
26:   end for
27: end for
28: return e_{N,q} s.t. ∀q' ≠ q, bp_{N,q} ≥ bq_{N,q'}, q > Q
```

To achieve the above idea, we extend an element in matrix M as follows:

$$e_{c,q} = (bs_{c,q}, bp_{c,q}, AS_{c,q}, AP_{c,q}),$$

$AS_{c,q}$ is a set of all the subsolutions that are expanded from all $bs_{c-1,q}$ and $s_{c,j}$, which are all the services in service cluster c. $AP_{c,q}$ is a set of price values given by $AS_{c,q}$. From the definition, $bs_{c,q} \in AS_{c,q}$ and $bp_{c,q} \in AP_{c,q}$ hold. Excluding a service may make the current best subsolution recorded at $e_{c,q}$ infeasible. Thus, the extension records more possible subsolutions that correspond to a certain price without being limited to the best one.

Algorithm 2 is revised and renamed Algorithm 3 for the extension. This algorithm is performed for the first selection in Algorithm 1 (line 4).

The significant difference is adding a selection and its quality value to $AS_{c,q}$ and $AP_{c,q}$ (lines 9-10, 22-23). This is performed when the price of the selection satisfies the price constraint. Matrix M is preserved after the first service selection is finished.

Once the user determines the services, Algorithm 4 efficiently selects services, excluding services $s_{i,j}$ which are selected at the first service selection.

Algorithm 4. SelectForVCGExcluding($s_{i,j}$, S, P, Q)

```
 1: s_{i,j}: service to be excluded
 2: S: set of available services {s_{i,j}}
 3: P: max price of composite service
 4: Q: min quality of composite service
 5: for all e_{c,q} in M do
 6:   if sol_{c,q} ∈ AS_{c,q} contains s_{i,j} then
 7:     Remove sol_{c,q} and p_{c,q} ∈ AP_{c,q}
 8:     Add e_{c,q} to set D
 9:   end if
10: end for
11: for c = i to N − 1 do
12:   for all e_{c,q} in D do
13:     for all s_{c+1,j} in cluster c + 1 do
14:       sol' ← combine bs_{c,q} and s_{c+1,j}
15:       p' ← price of sol', q' ← f(sol')
16:       if p' < bp_{c+1,q'} and p' < P then
17:         Update e_{c+1,q'} to (sol', p')
18:         Add e_{c+1,q'} to set D
19:       end if
20:     end for
21:     Remove e_{c,q} from D
22:   end for
23: end for
24: return e_{N,q} s.t. ∀q' ≠ q, bq_{N,p} ≤ bp_{N,q'}
```

The solutions that contain excluded service $s_{i,j}$ are removed first (line 7). The element in M from which $s_{i,j}$ is excluded is added to set D because it needs to be updated later (line 8). The update of the elements starts with the row that corresponds to the cluster of the excluded service. This algorithm iteratively updates each row by propagating changes. After updating the last row, it finds the best solution from the last row.

The computational cost of **SelectForVCGExcluding** given $s_{c,j}$ is $\sum_{n=c}^{N} nmQ/N$. This is because the nQ/N elements can be updated in the nth row. The m subsolutions are generated for each element, and the updates are iterated from the cth row to the last row. Since SelectForVCGExcluding is iterated for services selected in each service cluster, the computational complexity for the entire update is $\sum_{c=1}^{N}\sum_{n=c}^{N} nmQ/N$, which is less than Algorithm 2, $\sum_{n=1}^{N} O(nmQ/N) \cdot N$.

4 Evaluation

The algorithms shown in the previous section effectively calculate the VCG payments. We implemented a prototype and investigated the performance of our algorithms.

The following is the experimental framework. We performed service selection and payment calculation based on Algorithm 1. The evaluation criteria are the execution time to finish the process, the success ratio of the service composition, and the utility values of a user and the service providers. The initial conditions are maximum price P and minimum quality Q given by a user and price $p_{i,j}$ and quality $q_{i,j}$ proposed by the service providers. We repeat the process a certain number of times by setting random values to the initial conditions to investigate the general properties of the proposed algorithms. We defined constant values $P_0 = 100$, $Q_0 = 50$ and generated random values $0.5 \leq r_p, r_q \leq 1.5$. Then we set P and Q to $r_p P_0$ and $r_q Q_0$, respectively. Similarly, we also generated random values $0.5 \leq r_{p_{i,j}}, r_{q_{i,j}} \leq 1.5$ and set $p_{i,j}$ and $q_{i,j}$ to $r_{p_{i,j}} P/N$ and $r_{q_{i,j}} P/N$, respectively. For a comparison of multiple settings, we used the same sequence of random values.

We also defined the following experimental parameters: number of service clusters N, number of concrete services in a service cluster m, and scale for discretization d. In the following sections, we show some properties including the performance of our algorithms with respect to the experimental parameters.

All the experiments were conducted on a machine that was composed of Intel(R) Xeon(R) X5675 and 72-GB RAM. We implemented our prototype system for our experiments in Java (Oracle JDK 7 Update 51). The CPU had 12 cores, but the prototype was implemented as a single-threaded program. Thus, only one core is used. The maximum amount of memory used was around 6 GB.

4.1 Number of Service Clusters

We investigated the execution times as the number of service clusters increased from 5 to 100. For each service cluster, we performed service composition ten times and plotted the execution times on a chart. The number of concrete services in a service cluster, m, was set to 20. The scale for price discretization d was 10.

In Fig. 3, the horizontal axis represents the number of service clusters, and the vertical axis represents the execution times. There are three series in the figure. The red crosses show the execution times based on the basic DP-based algorithm (Algorithm 2; shown as DP basic with VCG payments in Fig. 3. The green x's show the execution times based on the extended algorithm (Algorithms 3 and 4; shown as DP ext with VCG payments). The blue triangles show the

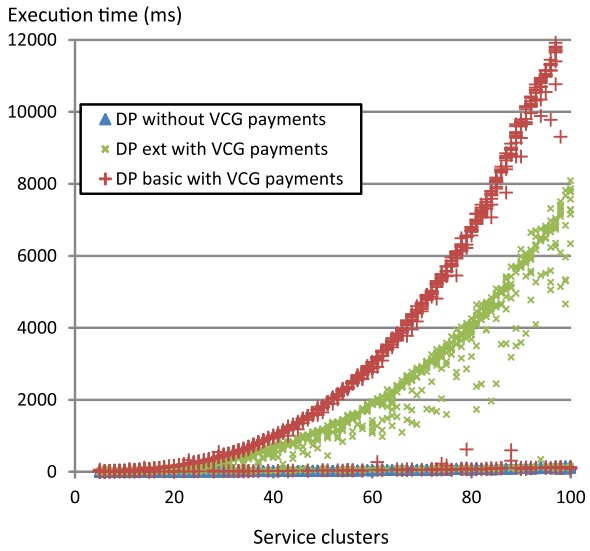

Fig. 3. Execution time of DP-based algorithm: service clusters.

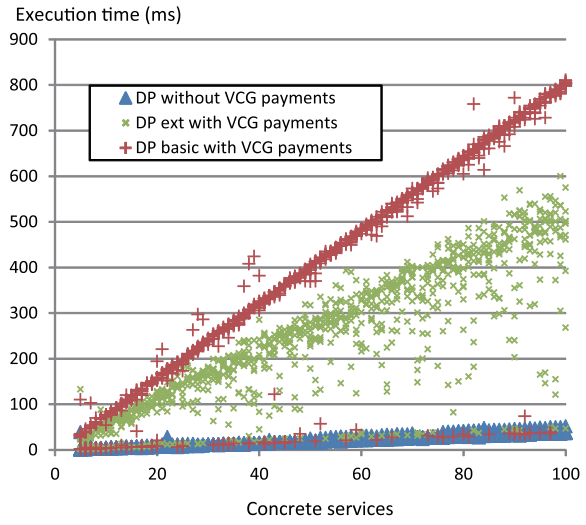

Fig. 4. Execution time of DP-based algorithm: concrete services.

execution times to find a combination of concrete services without calculating the VCG payments (shown as DP without VCG payments). This corresponds to the execution of the first **Select** in Algorithm 1 (line 4). We investigated this execution time to determine how much the computational cost increases for the strategy-proofness of VCG payments.

Fig. 3 shows that the execution times of the basic DP algorithm exponentially increase as the number of service clusters increases. The execution time of the extended DP algorithm is also exponential, but more moderate. Based on the average execution times for all the trials shown in Fig. 3, the extended DP algorithm is 1.7 times faster than the basic one.

For both algorithms, the execution times are categorized into two groups. The group near the bottom line shows the times when the first service selection fails. If a combination of services is not found at the first service selection, the following selection for calculating the VCG payments does not run. This is why the algorithm finishes soon. The extended algorithm records many more subsolutions and corresponding prices, but the other only records the best ones. This makes the extended algorithm slightly more time-consuming than the basic one when service composition fails.

The execution time without calculating the VCG payments is much shorter than the other settings. We used the basic DP algorithm for this setting. As discussed in Section 2, this runs **Select** once while the other runs it $N+1$ times. Therefore, the difference with the other cases increases as the number of service clusters N increases. The experimental results of the basic and extended DP-based algorithm in Fig. 3 roughly follow the estimation of the computational complexity shown in Sections 3.1 and 3.2 respectively. In the extended algorithm, the computational cost of the worst case is almost the same as the basic algorithm. However, the algorithm checks existing elements in M and stops searching when the price of the new subsolution exceeds the existing one. This works as pruning and improves the performance.

4.2 Number of Concrete Services

We also investigated the execution times as the number of concrete services in a service cluster, m, increases from 5 to 100. The number of service clusters, N, was set to 20. The scale for price discretization d was 10.

In Fig. 4, the horizontal axis represents the number of concrete services in a service cluster. The vertical axis represents the execution times. There are also three series in Fig. 3. The red crosses show the execution times based on the basic DP-based algorithm. The green x's show the execution times based on the extended algorithm. The blue triangles show the execution times to find a combination of concrete services without calculating the VCG payments.

The extended algorithm also has a significant advantage over the basic one in this experiment, although both show linear times. For the basic algorithm, there are clearly two groups where the service composition succeeded or failed. For the extended algorithm, there are still two groups but the difference between them is much smaller. When service composition succeeds, the execution time of the extended algorithm is shorter, but it takes longer when the service composition fails. Based on the average execution times for all the trials shown in Fig. 4, the extended DP algorithm is 1.8 times faster than the basic one.

According to the above discussion, the computational complexity is $O(mQN^2)$. Therefore, the execution time with respect to the number of concrete services in a service cluster is expected to be linear. The result shown in Fig. 4 also roughly follows this estimation. Similar to the previous experiment, the computational cost of the extended algorithm in the worst case is almost the same as the basic algorithm. Therefore, the extended algorithm's performance is much better because of the pruning based on the existing values in M.

4.3 Scale of Discretization

The scale of discretization d greatly impacts both the applicability to the real problem and the computational complexity. If the scale is small, the possible range of prices is limited while the computational complexity is small. For example, the offered price can go to one decimal place when d is set to 10.

According to the definition of discretized price $round(quality \cdot d)$, on the other hand, the computational complexity is expected to linearly increase when the scale

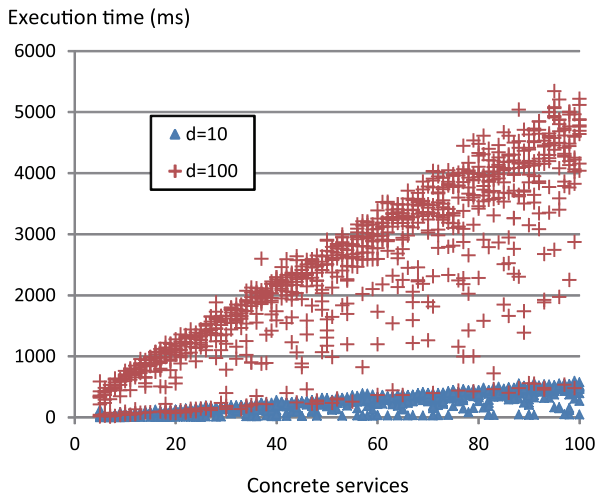

Fig. 5. Execution time and discretization scale: concrete services.

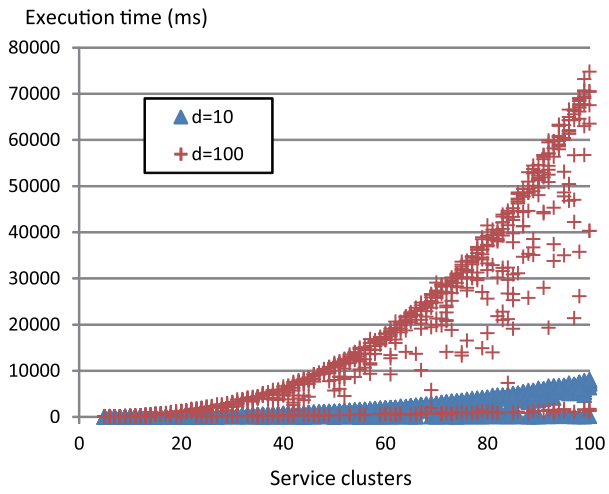

Fig. 6. Execution time and discretization scale: service clusters.

of discretization increases. Since the required price "resolution" depends on the application, d should be appropriately set based on a tradeoff between performance and precision.

We compared execution times, $t_{d=10}$ and $t_{d=100}$, in which two settings, $d = 10$ and $d = 100$ resulted, respectively. In Fig. 5, the horizontal axis represents the number of concrete services in a service cluster, and the vertical axis represents the execution time. Both series represent execution times where two values of d were the results.

Our result shows that the execution times of both settings are linear. The execution time in which $d = 100$ resulted is around ten times bigger than the execution time where $d = 10$ was the result. Based on the discussion in Section 3.1, the overall computational complexity is estimated to be $O(mQN^2)$. After discretization, the maximum number of columns in M was dQ. Thus, the computational complexity was revised to $O(mdQN^2)$. The result shown in Fig. 5 follows this estimation. Fig. 6 also shows the execution times of the two settings regarding the number of service clusters. The result in Fig. 5 also follows this estimation.

4.4 Success Ratio of Composition

If our model fails to find a combination of services and calculate the payments, there are three possible reasons, shown in Algorithm 1. First, a feasible combination of services might not exist that satisfies the maximum price and the minimum quality based on the prices proposed by providers (line 6). Second, as a result of excluding a service selected during the first selection, a combination of services might not exist that satisfies the constraints (line 11). Third, the sum of the VCG payments might exceed the maximum price given by the user (line 17). Service composition fails in this case because we assume that the user does not accept negative utility. This failure never happens in a service market that follows the 1st price reverse auction, where service providers propose prices by concealing their true cost and the user pays the proposed price. Therefore, we consider this failure a sort of tradeoff between strategy-proofness and the success ratio of service composition.

To determine how much the success ratio suffers by introducing VCG payments, we compared the success ratio of service composition in two cases: when we introduced VCG payments and when the user pays the price proposed by service providers. The latter is an extreme case because it maximizes the success ratio while the utility of service providers is always zero. Thus, the success ratio certainly becomes smaller when we introduce VCG payments.

Fig. 7 shows the experimental results regarding the success ratios. The horizontal axis is m and shows the number of concrete services in a service cluster. The number of service clusters N is 20. We repeated the procedure in Algorithm 1 10 times and plotted the successes of the service composition. The vertical axis is the resulting success ratio of the service compositions. The blue triangles represent the success ratios when the user pays the price proposed by the service providers without VCG payments. The red crosses represent the success ratios when the user pays the VCG payments. The success ratios increase, and the number of concrete service increases. Since the number of possible combinations of services increases, it becomes more likely to find feasible combinations. Although the success ratio becomes lower when we introduce VCG payments, we don't believe the difference is fatal.

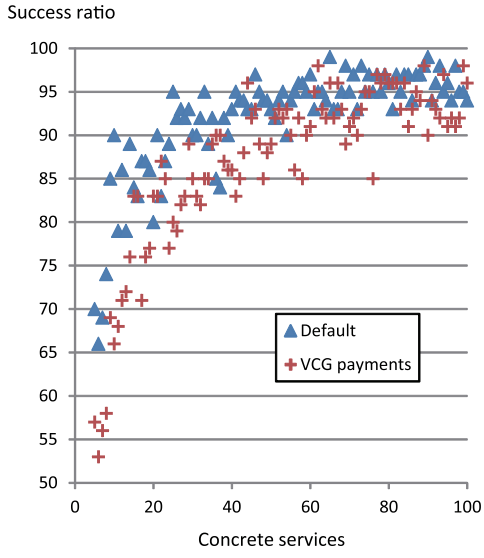

Fig. 7. Success ratio and number of concrete services.

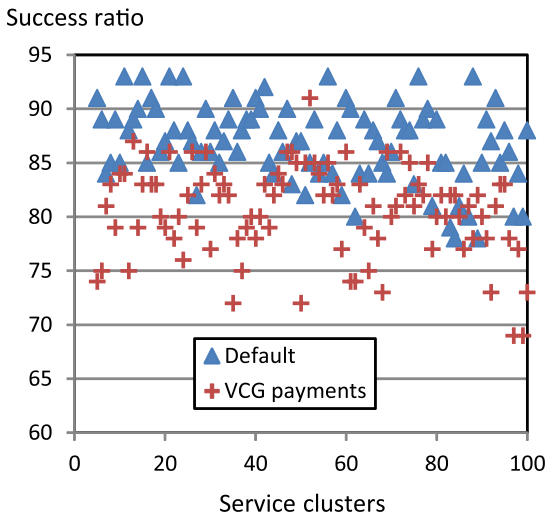

Fig. 8. Success ratio and number of service clusters.

We also investigated the success ratio by increasing the number of service clusters. Fig. 8 shows the results; the number of service clusters does not affect the success ratio.

4.5 Service Supply and Utility

In a service market, the number of alternative services can increase or decrease. This change affects the utility of the user and the service providers. For example, an increase in the size of service cluster m can be interpreted as an increase of the supply and may reduce the utility of the service providers. We assume that a supply of services can satisfy as much demand of computational resources as requested thanks to an IaaS platform, but an increase of service providers may give a chance to find cheaper services. To experimentally clarify this, we investigated the utility of the service providers.

Fig. 9 shows the sum of the service providers' utility ($\sum_{i,j} u_{i,j}$). The horizontal axis is the number of concrete services in service cluster m. The number of service clusters N is 20. We repeated the procedure in Algorithm 1 10 times. The vertical axis represents the average of the service providers' utility. According to formula (8), the user's utility is determined by the service providers' utility. This is why we show only the service providers' utility.

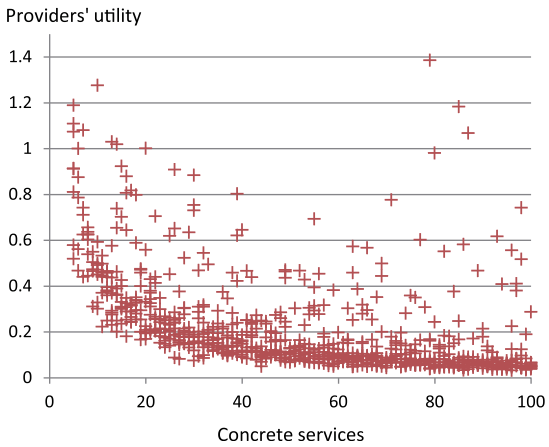

Fig. 9. Utility and number of concrete services.

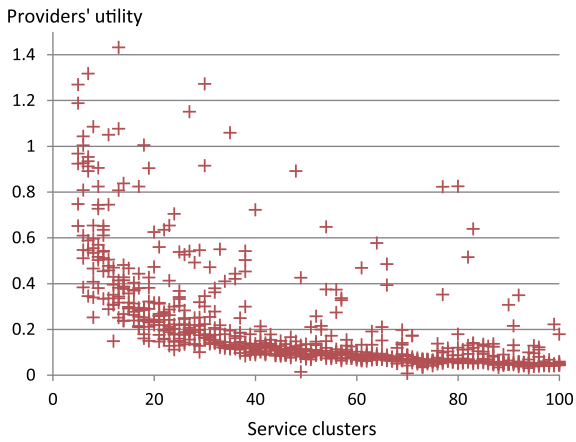

Fig. 10. Utility and number of service clusters.

As shown in Fig. 9, the providers' utility decreases as the number of concrete service increases. Following the definition, the provider's utility is determined by the difference between the prices of the cheapest and the second cheapest combinations. The difference becomes small when the number of possible combinations increases. Thus, the providers' utility decreases as the number of concrete services increases. On the other hand, the user's utility increases because the amount of the payments decreases.

When the number of concrete services increases, the success ratio also increases but the average utility decreases. The decrease of the providers' utility gets slimmer as the number of concrete services increases. From the viewpoint of each service provider, there is a tradeoff between the success ratio of the service composition and the utility.

We also investigated the utility by increasing the number of service clusters. Fig. 10 shows that the service providers' utility decreases when the number of service clusters increase, because the average of the service providers' utility is given as $(P - U)/N$.

4.6 Comparison with Fixed Pricing

The utility received by users and service providers is often lower than more intuitive models, such as fixed pricing. One possible reason is the service composition failure, caused by the additional payments of the VCG mechanism that exceed user's budget P, as discussed in Section 4.4. From the service providers' viewpoint, the payment given by the VCG mechanism can be too small because the user pays less than or equal to his budget P. Since it is difficult to theoretically judge the advantages/disadvantages of our model regarding utility, we experimentally compared our VCG mechanism with fixed pricing.

For fixed pricing, we assume service providers have a parameter $margin$ and propose the price $(1.0 + margin) \cdot p_{i,j}$. The model is a simple reverse auction where the user pays the sum of the prices of the selected services. We set the number of clusters N to 20 and the number of concrete services in a service cluster to 20. The scale of discretization d was set to ten. We iterated the service composition 100 times and recorded the utility. The results are shown in Figs. 11 and 12.

Fig. 11 shows the sum of the utility of the user and service providers. We refer to it as the *"system utility."* The horizontal axis is the step of the iteration of the service

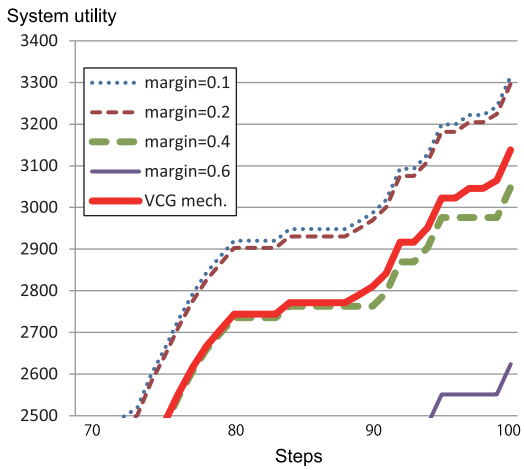

Fig. 11. Comparison with fixed pricing: system utility.

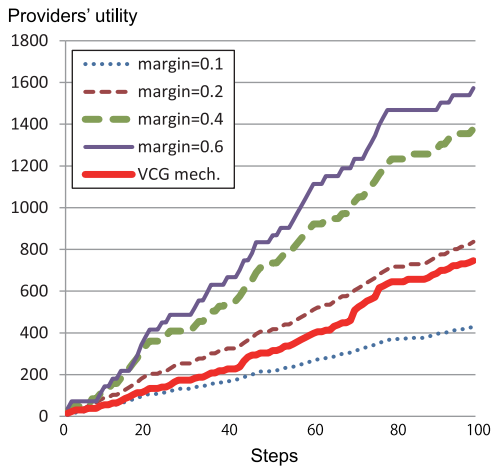

Fig. 12. Comparison with fixed pricing: providers' utility.

composition. The vertical axis is the accumulated system utility. For fixed pricing, four series correspond to different values of $margins$, 0.1, 0.2, 0.4, and 0.6. Another series corresponds to the system utility given by the VCG mechanism.

As shown in Fig. 11, $margins$ 0.1 and 0.2 resulted in almost the same system utility, which outperforms the others. This is a natural outcome because the low profit ratio leads to a high success ratio of service composition. The next is the VCG mechanism, and the difference of the values is not significant. The system utility is better than those given by $margins$ 0.4 and 0.6. In this experiment, the VCG mechanism is not the best, but we can interpret the disadvantage of the system utility as a tradeoff of strategy-proofness.

Fig. 12 shows the service providers' utility in the vertical axis. Although 0.1 is the best $margin$ in terms of the system utility, the value resulted in low providers' utility. $margin$ values 0.4 and 0.6 resulted in better providers' utility than the VCG mechanism, but the system utility given by the margin values are smaller than the VCG mechanism. Only $margin$ 0.2 resulted in better results both for the system and the provider's utility than the VCG mechanism. This experiment suggests that service providers can earn acceptable utility without worrying about the failure of pricing. Note that the utility of the users decreases when the utility of the service providers increases and vice versa because of formula (8). Therefore, any providers' utility can be accepted based on the system design policy if the system utility is high enough.

5 Related Works

In the area of services computing, many previous works have proposed various approaches, including integer programming [8], constraint satisfaction [9], and GA [10]. There is no significant difference between our service composition model and the fundamental models assumed in the previous works. They assume that abstract composite services are given and the user or the system selects concrete services. One real service platform that follows the same style of service composition is the Language Grid [5]. On it, approximately 120 concrete services are available, based on such service types as machine translators and dictionaries. Since all the interfaces of the service types are standardized, users can transparently bind concrete services by specifying them in the extended header of a request, such as in SOAP.

Pricing services is another growing topic in this area, and even the discussion of goal concepts is active. We focused on strategy-proofness in this paper, but other solution concepts can be applied to profit sharing in composite services. Matsubara introduced the Shapley value [11] for the profit allocation of a composite service and cited some properties of profit sharing based on it and compared it with a simpler approach [12]. But they did not show any experimental or theoretical results. From more practical aspects, it is reasonable to price services based on the detailed cost and performance. The elasticity of cloud computing also forces users to directly face the tradeoff between price and performance. Assume an IaaS such as Amazon EC2. The prices of the services differ based on the specifications or the performance. But the relationship between the price and the actual performance is not clear. Wang et al. discussed the tradeoffs between cost and performance for various applications [13]. A tradeoff is obvious when the user can improve the performance of his application by paying more for more high-performance machines. But they also show a non-trivial tradeoff in which a high-performance machine can reduce the execution time, and the cost is also reduced compared to cheaper, lower-performance machines. Li et al. proposed a framework to estimate the cost considering the performance and application types. Their framework helps users select cloud services based on criteria including typical benchmarks, the latency of the starting services, communication overhead, and access to storage [14]. In their approach, the costs of cloud services from the viewpoint of providers are estimated from power consumption and amortization. Another work by Sharma et al. [15] applied option theory to estimate the value of cloud resources to determine prices.

In addition to the fundamental achievements by the above work, more application oriented designs for pricing have also been proposed. Upadhyaya et al. proposed a mechanism for cost sharing when multiple users benefit from investments on a common platform [16]. They assume that users can invest in such a shared cache of a database on the cloud. But the investment demand dynamically changes and depends on the users. If those who need the investment first pay all the cost, users may falsely report demands as

delays. The proposed mechanism assures strategy-proofness about the declaration of the demands. Dash et al. also focused on whether the user invests in the improvement of performance [17]. Based on regret theory [18], their method determines whether users should invest by calculating the difference between the result of their actual choice and the expected profit of another choice.

As mentioned in Section 2, an approximate solution does not assure strategy-proofness. In mechanism design, various approaches have been proposed mainly for combinatorial auctions, whose computational complexity is NP-hard. Zurel and Nisan approach improves the initial solution until the solution assures strategy-proofness [19]. Assuming some limitations, Lehmann proposed an approximation algorithm whose solution assures strategy-proofness without exploiting VCG payments [20]. On the other hand, Kfir-Dahav showed the axioms required so that approximate algorithms assure strategy-proofness [21]. But these works are for general settings and are not specialized for service selection.

The popular goal of researches that propose pricing cloud services is maximizing utility. Various approaches have been proposed, including the genetic algorithm [22], stochastic dynamic programming [23], and optimal control problem [24]. Other works have adopted auction models for their formalizations [4], [25], [26], [27], [28], [29]. Some aimed for strategy-proofness by extending typical auctions, as we did in this paper. Mihailescu and Teo introduced a reverse auction model for the selection of services (cloud platforms in their context) [4]. They also introduced VCG payments to exploit dynamic pricing by assuring strategy-proofness. Users declare their required types of items and counts. In their model, the computational cost of service selection (winner determination) is not as serious as in our model. Another work [28] achieved strategy-proofness using MDP to maximize utility. The major interest of the above works is to maximize the utility of users and/or service providers, but we focus more on the computational efficiency of VCG payment calculation.

6 Conclusion

In this paper, we modeled VCG payments for service composition and proposed an efficient algorithm for service selection and VCG payment calculation. Even though VCG payments have such desirable properties as strategy-proofness for pricing services, they require iterative service selection, each of which is NP-hard. To investigate how our algorithm works on the problem, we implemented it and conducted a series of experiments. The following are the contributions of this paper:

- Our proposed DP-based algorithm solves service selection in quasi-polynomial time and largely improves the performance of VCG payment calculation compared to the basic DP.
- Our experiments clarified how VCG payments affect some important properties of service composition, including the success ratio and utility as well as the computational cost.

Our experimental results show that our extended algorithm improves the performance of VCG payment calculation by caching the progress of the first service selection and reusing it when the algorithm iteratively solves the service selection. In our experiment, given 20 concrete services in approximately 100 service clusters, the proposed algorithm solved service composition and calculated VCG payments in a few seconds. The scale of this problem cannot be solved by the intuitive backtrack and branch-and-bound approach. Although our implementation for the experiment is a prototype that was not fully optimized, it is promising to apply the algorithm to real scale problems.

We also investigated some properties of our model and algorithms. By introducing VCG payments, the failure of service composition might happen even after a feasible combination of concrete services is found. Thus we confirmed that the decline of the success ratio of the service composition is not significant. Moreover, experiments regarding utility values show the relationship with the amount of service supply and difference with fixed pricing, which is a more intuitive model.

The experimental results in this paper are based on services that have random values as their price and quality. In the future, we will extend our model and algorithms for real services and applications.

References

[1] W. Vickrey, "Counterspeculation, auction, and competitive sealed tenders," *J. Finance*, vol. 16, no. 1, pp. 8–37, 1961.

[2] E. H. Clarke, "Multipart pricing of public goods," *Public Choice*, vol. 11, no. 1, pp. 17–33, 1971.

[3] T. Groves, "Incentives in teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, 1973.

[4] M. Mihailescu and Y. Teo, "Strategy-proof dynamic resource pricing of multiple resource types on federated clouds," in *Proc. 10th Int. Conf. Algorithms Archit. Parallel Process.*, 2010, pp. 337–350.

[5] T. Ishida, *The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability*. New York, NY, USA: Springer, 2011.

[6] S. M. Makoto Yokoo and Y. Sakurai, "The effect of false-name bids in combinatorial auctions: New fraud in internet auctions," *Games Econ. Behav.*, vol. 46, no. 1, pp. 174–188, 2004.

[7] D. Menasce, "QoS issues in web services," *IEEE Internet Comput.*, vol. 6, no. 6, pp. 72–75, 2002.

[8] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 311–327, May 2004.

[9] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *Proc. 5th Int. Semantic Web Conf.*, 2006, vol. 4273, pp. 130–143.

[10] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proc. 7th Annu. Conf. Genetic Evol. Comput.*, 2005, pp. 1069–1075.

[11] L. S. Shapley, "A value for n-person Games," in *Contributions to the Theory of Games*, H. Kuhn and A. Tucker, Eds. Princeton, NJ, USA: Princeton Univ. Press, 1953, pp. 307–317.

[12] S. Matsubara, "Profit sharing in service composition," in *Proc. 9th Int. Conf. Serv.-Oriented Comput.*, 2011, pp. 645–652.

[13] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: Pricing in the cloud," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, pp. 6–12.

[14] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Shopping for a cloud made easy," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, p. 5.

[15] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya, "Pricing cloud compute commodities: A novel financial economic model," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2012, pp. 451–457.

[16] P. Upadhyaya, M. Balazinska, and D. Suciu, "How to price shared optimizations in the cloud," *Proc. VLDB Endowment*, vol. 5, no. 6, pp. 562–573, 2012.
[17] D. Dash, V. Kantere, and A. Ailamaki, "An economic model for self-tuned cloud caching," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 1687–1693.
[18] G. Loomes and R. Sugden, "Regret theory: An alternative theory of rational choice under uncertainty," *Economic J.*, vol. 92, no. 4, pp. 805–824, 1982.
[19] E. Zurel and N. Nisan, "An efficient approximate allocation algorithm for combinatorial auctions," in *Proc. 3rd ACM Conf. Electron. Commerce*, 2001, pp. 125–136.
[20] D. Lehmann, L. I. Oćallaghan, and Y. Shoham, "Truth revelation in approximately efficient combinatorial auctions," *J. ACM*, vol. 49, no. 5, pp. 577–602, 2002.
[21] N. E. Kfir-Dahav, "Mechanism design for resource bounded agents," *Proc. 4th Int. Conf. MultiAgent Syst.*, 2000, pp. 309–315.
[22] M. Macías and J. Guitart, "A genetic model for pricing in cloud computing markets," in *Proc. ACM Symp. Appl. Comput.*, 2011, pp. 113–118.
[23] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 158–171, Jul.–Dec. 2013.
[24] V. Kantere, D. Dash, G. Francois, S. Kyriakopoulou, and A. Ailamaki, "Optimal service pricing for a cloud cache," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1345–1358, Sep. 2011.
[25] S. Zaman and D. Grosu, "A combinatorial auction-based mechanism for dynamic VM provisioning and allocation in clouds," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 129–141, Jul.–Dec. 2013.
[26] I. Fujiwara, K. Aida, and I. Ono, "Applying double-sided combinational auctions to resource allocation in cloud computing," in *Proc. 10th IEEE/IPSJ Int. Symp. Appl. Internet*, 2010, pp. 7–14.
[27] S. Shang, J. Jiang, Y. Wu, Z. Huang, G. Yang, and W. Zheng, "DABGPM: A double auction Bayesian game-based pricing model in cloud market," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.*, 2010, pp. 155–164.
[28] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proc. IEEE/ACM 21st Int. Symp. Quality Serv.*, 2013, pp. 1–6.
[29] W.-Y. Lin, G.-Y. Lin, and H.-Y. Wei, "Dynamic auction mechanism for cloud resource allocation," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 591–592.

Masahiro Tanaka received the master's and PhD degrees from the Department of Social Informatics, Kyoto University, in 2005 and 2009, respectively. He is currently a researcher at the National Institute of Information and Communications Technology (NICT). His research interests include services computing, with specific interests in large-scale information analysis and the runtime control of service composition.

Yohei Murakami received the PhD degree in informatics from Kyoto University in 2006. He is an associate professor of Unit of Design, Center for the Promotion of Interdisciplinary Education and Research, Kyoto University. He currently leads the research and development of the Language Grid, an infrastructure for language services. He received the Achievement Award of the Institute of Electronics, Information and Communication Engineers for this work in 2013. His research interests include services computing and multiagent systems. He founded the Technical Committee on Services Computing in the Institute of Electronics, information and Communication Engineers (IEICE) in 2012.