An Optimization Approach for Service Deployment in Service Oriented Clouds

Tiejiang Liu
School of Computer Science
Fudan University
Shanghai, P.R.China
liutiejiang@fudan.edu.cn

Tun Lu
School of Computer Science
Fudan University
Shanghai, P.R.China
lutun@fudan.edu.cn

Zhenyu Liu
Shanghai Key Laboratory of
Computer Software Evaluating and Testing
Shanghai, P.R.China
lzy@ssc.stn.sh.cn

***Abstract*—In service oriented cloud, in order to meet the different users' service requirements, the cloud venders need to manage the service applications' deployment effectively first. During the deployment process, the deployment costs and the influence of service deployment to the cloud are very important issues to be paid attention to, which lacks in current research. In this paper, we design a novel optimization approach for service deployment to address above issues, which works in a service deployment framework. The approach can improve the deployment efficiency and reduce the deployment costs. *Atom-service*s, the basic units of service application, are partitioned to different *service families* with its *compatibility* and *installation policy*. We present three algorithms to automatically standardize the *installation expression*s, simplify and optimize the *installation expression series* during the service deployment process respectively. An analysis about the service *deployment result* is given in the paper and the experiment data demonstrates the approach's efficiency.**

***Keywords*—cloud computing, service deployment, deployment optimization**

I. Introduction

Recent years, cloud computing has become one of the most important research subjects in the field of scientific research and information technology[4], [5]. At a word, cloud computing is a kind of development pattern which is based on Internet and the kernel technique is computer science and its related fields. Typically, the vender of the cloud delivers common service applications online which can be accessed from any web browser. These service applications are deployed into the cloud by application owners. The end user can access these service applications in a pay-for-use mode[6].

In service oriented cloud, for the venders of the cloud, service deployment is one of the most important tasks to implement. There are many researches on the service deployment[2], [7], [8], [9], [10], [11]. Most of them emphasize particularly on the service deployment process itself and design different deployment approaches. [2] introduces some service deployment approaches, such as script-based, language-based and model-based deployment approaches, which are exampled by Nixes, SmartFrog and Radia respectively. [7] presents a language-based approach called Abacus, which can make the service deployed automatically without any configuration information. [8] puts forward a dynamic service deployment approach based on FlexiNet IST – an European Union's web project. But the costs, such as time costs and resource costs, during the service deployment are neglected in these researches.

[10] considers that one of the most challenges of service deployment is to take functional and non-functional factors into account, such as capability, reliability, operation cost and QoS. A successfully deployed service application maybe need to re-deploy with the change of the factors. [11] points out that current approaches usually make service deployment as low, expensive, and error-prone process. [9] optimizes the service deployment according to QoS of different services and provides service applications with different SLA to end-users. From the angle of end-users, it can present service products with different QoS, which benefits end-users' service selection. But there is few research on the deployment cost of the cloud vender. And after the services are deployed into the cloud, whether the new service applications or the old ones can be successfully invoked is a serious problem to be solved which lacks in recent research.

In this paper, we emphasize the deployment cost of the service oriented cloud venders and design an approach to optimize the service deployment process which can guarantee the *deployment result state* is *usable* and *safe*. Firstly, we class all *atom-service*s with different *service families* according to their distinguishing function interfaces. Each *atom-service* has its own *compatibility* and *installation policy*, which is one of the most important research bases. Three kinds of *compatibilities* and two kinds of *installation policies* determine that some *atom-service*s could be simplified from *installation expression series* whose functions can be matched by some other *atom-service*s in the same *service family*. We divide the optimization process into two phases. One is the simplification to *installation expression series* itself, and the other is the optimization to *installation expression series* and *installed atom-services* during the service deployment process. A system framework to deploy service with optimization approach is built up and simulated experiment results show that our approach can get a satisfying optimization effect during service deployment process.

The rest of this paper is organized as follows. In next section, we introduce some conceptions and definitions used in this paper. Installation expression series standardization algorithm and simplified algorithm are presented in section 3. Section 4 describes the optimization process during service deployment. The simulated experiment and its analysis are shown in section

978-1-4244-6763-1/10/$26.00 ©2010 IEEE

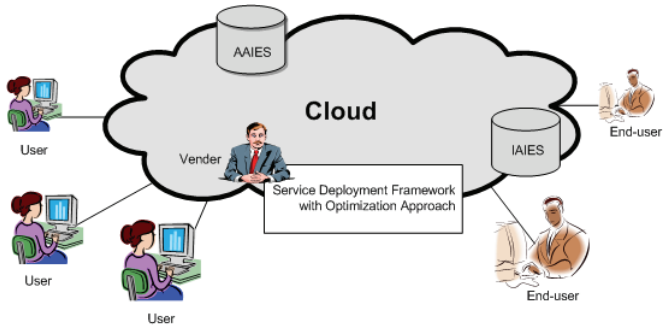

Fig. 1. A Cloud Computing Environment

5. And we summarize the paper and bring forward the future work in section 6.

II. Conceptions and Definitions

Here we introduce some conceptions. Fig.1 shows a common service oriented cloud computing environment. The end-user needn't to know the detail of the cloud. With the service applications interfaces, the end-user can get what he wants in a pay-for-use mode. The user is the owner of the service applications deployed in the cloud. He ought to give a service application deployment requirement to the vender of the cloud. And the vender should control and manage the cloud. For the received deployment requirements, he should implement them with high efficiency and guarantee the usability and safety.

In cloud environment, cloud application is generally composed of some basic units (atom-service, *as*)[9] which belong to different service families. Relying on mutual interaction and collaboration, these atom-services with a logical sequence can implement the application's function. This logical sequence is named as atom-service sequence, denoted by π_{as}. A service family is composed of a set of services which have the similar functions. $AS_{fn(i)}$ denotes the service family who has a family number *i*. For any atom-service as_{fn}^{vn} in a service family, *fn* donates the service family's number, *vn* the atom-service's version number. Atom-services belonging to the same service family own the different version numbers because of the different development phases. Compatibility (*comp*) is introduced into this paper. Three kinds of compatibilities – *BC*, *FC*, *BC*∧*FC* – can depict the relationship between atom-services in the same service family who have the different version numbers. If an atom-service as_{fn}^{p} can fulfill the functions of as_{fn}^{r} ($p>r$), we say as_{fn}^{p} has the backward compatibility (*BC*). If as_{fn}^{p} can fulfill as_{fn}^{r}'s functions ($p<r$), then as_{fn}^{p} has the forward compatibility (*FC*). If the conditions above are both satisfied, the compatibility is *BC*∧*FC*. If some *as* exists in a service family whose compatibility is neither *BC* nor *FC*, we can take it out from the original service family to form a new one.

Definition 1 Atom-service installation expression (*ie*). We use a 4-tuple $ie(as_{fn}^{vn})=(as_{fn}^{vn}, comp, qos, IP)$ to donate as_{fn}^{vn}'s installation expression, which means that as_{fn}^{vn}'s compatibility is *comp* and its quality of service is *qos*. Many researches have been done on QoS, so for simplification we use one dimension value for QoS computation, where smaller qos value means better QoS and the operation relation among them is addition. If necessary, other methods used to analyze and evaluate QoS can be adopted when implementing the approach introduced in this paper. *IP* is installation policy. When *IP* is *true* (IP_T), as_{fn}^{vn} should be installed into Cloud and could not be ignored during our optimizing process. If *IP* equals *false* (IP_F), as_{fn}^{vn} would be concerned and could be got rid of from the installation series during the process. $IE(\pi_{as})$ donates the *ie(as)*s series.

Considering the depiction convenience, $(as_{fn}^{vn}, \_, \_, \_)$ is used to show that as_{fn}^{vn}'s compatibility, QoS and installation policy can be random values.

The vender of the cloud manages a large available atom-service set, whose installation expressions is denoted by *AAIES* (available atom-service installation expression set). When a cloud user brings forward a service deployment requirement, he can provide the atom-services himself. He also can trust in our approach for selecting appropriate atom-services from *AAIES* automatically. *IAIES* (installed atom-service installation expression set) is used to donate atom-services that have been installed into cloud environment. During the process of service deployment optimization, we need refer to *IAIES* to simplify $IE(\pi_{as})$.

Definition 2 Deployment requirement (*DR*). A service deployment requirement contains the service's atom-services installation expression series and the total QoS expectation, which can be denoted by $DR=(IE(\pi_{as}), QoS_E)$. The atom-service installation expression $ie(as)=(as_i, \_, \_, \_)$, such as $(as_1^2, \_, \_, F)$, $(as_3, BC, \_, F)$, $(as_3^6, FC, 0.3, T)$. That is to say, only the family number of an atom-service is necessary. Other parameters can be selected according to *AAIES* automatically. Only when *ie(as)* is integrated, can *IP* be possibly set as true. We consider that this atom-service be provided by user and should be installed into the cloud.

Definition 3 Deployment result state (*DRS*). [3] brings forward a conceptual framework about the component system deployment with success and safety state, which is further studied in [1]. Inspired by this, we introduce the conceptions of usability and safety into the service deployment process which is usually neglected in the research field of service deployment in cloud computing. A service deployment result has four states. After a new service been deployed into the cloud environment, if the new service application and the old ones can all be successfully invoked, we name this state *Usable & Safe (US)*. If the new service application can be successfully invoked but whether the old ones' function be influenced is uncertain, we name this state *Usable (U)*. Otherwise, the state is *Safe (S)*. If we don't be sure about any one of the services' invoked result, the state is named *Indeterminable (I)*. The relationship between the service invoked result and *DRS* is shown in table 1.

During simplifying process, if simplified *IE* can implement original service deployment requirement, we say this simplification satisfies *usability-condition*. If after simplified *IE* been installed into the cloud environment, the old service applications can be successfully invoked, we say this simplification satisfies safety-condition. If both usability-condition and safety-

Table 1. Deployment Result State Table

condition		state
the new service application can be successfully invoked	the old service application can be successfully invoked	
√	√	*US*
√	?	*U*
?	√	*S*
?	?	*I*

condition are satisfied, we say this is a *perfect simplification*, which is our expected simplification result.

As mentioned above, we will know that only the interaction between the atom-services belonging to the same service family can influence the deployment result state, which is one of the most important research bases that we design the service deployment optimization approach. All the conceptions and definitions are shown in table 2.

Table 2. Conceptions and Definitions Summary Table

as	the basic unit of service application
π_{as}	the logical sequence of atom-services
$AS_{fn(i)}$	the service family who has a family number *i*
comp	the abbreviation of compatibility. Three kinds of compatibilities are introduced in this paper – *BC*, *FC*, *BC∧FC*
ie	atom-service installation expression, $ie(as^{vn}_{fn})=(as^{vn}_{fn}$, *comp, qos, IP)*
IP_T (IP_F)	installation policy is true (false)
AAIES	available atom-service installation expression set, which denotes the large available atom-service set
IAIES	installed atom-service installation expression set, which denotes atom-services that have been installed into the cloud environment
DR	deployment requirement, *DR=(IE(π_{as}), QoS_E)*
DRS	deployment result state. There are four *DRS* in this paper – *Usable & Safe(US), Usable(U), Safe(S) and Indeterminable(I)*

III. IE Simplified Algorithm

When user brings forward a service deployment requirement, the approach will standardize the *DR* first. Any *ie(as)* in *IE* will be standardized to an integrated form, which is implemented by algorithm 1.

For example, there exists an *ie(as)* in *DR*, where *ie(as)=(as_3, BC, _, F)*. There are five *ie(as)* in *AAIES* whose family number is 3. They are *(as^1_3, BC, 0.4, F), (as^4_3, FC, 0.2, F), (as^2_3, BC∧FC, 0.1, F), (as^2_3, BC, 0.6, F), (as^3_3, BC, 0.5, F)*. According to algorithm 1, *ts(as)* can be figured out: *ts(as)={(as^1_3, BC, 0.4, F), (as^2_3, BC, 0.6, F), (as^3_3, BC, 0.5, F)}*. And at last, we get *$ie(as)\_{Stan}$=(as^1_3, BC, 0.4, F)*.

After *DR* has been standardized, we begin to simplify atom-service installation expression series. Two steps are introduced to carry out this aim. Firstly, simplification process is limited

Algorithm 1: Standardize Atom-service Installation Expression Series

Input: Deployment Requirement, DR=(IE(π_{as}),QoS_E)

Output: Standardized Atom-service Installation Expression Series, $IE(\pi_{as})\_{Stan}$

Step:

1. *for each $ie(as_i)$ in IE(π_{as})*

   a. if $ie(as_i)$ is integrated, then continue

   b. select all ie(as) into $ts(as_i)$① from AAIES, where Match_IE(ie(as), $ie(as_i)$) is true

   c. if $ts(as_i)=\emptyset$, quit the procedure and return error message "Standardize IE unsuccessfully; no matched ie(as) can be found"

   d. select MinQoS($ts(as_i)$)② to replace $ie(as_i)$

2. *if SumQoS($IE(\pi_{as})\_{Stan}$)③>QoS_E, quit the procedure and return error message "Standardize IE unsuccessfully; no matched IE(π_{as}) can be found "*

3. *return $IE(\pi_{as})\_{Stan}$*

Sub-function: *Match_IE($ie(as_1)$, $ie(as_2)$)*

Match_IE($ie(as_1)$, $ie(as_2)$) is true, while

- *$fn(as_1)$=$fn(as_2)$*

  and

- *if[($vn(as_2)$ is not null) and $vn(as_1)=vn(as_2)$] or ($vn(as_2)$ is null)*

  and

- *if [($comp(as_2)$ is not null) and $comp(as_1)$= $comp(as_2)$] or ($comp(as_2)$ is null)*

  and

- *if [($qos(as_2)$ is not null) and $qos(as_1)\leq qos(as_2)$] or ($qos(as_2)$ is null))*

① *$ts(a_i)$, a temporary set for $ie(as_i)$*

② *MinQoS($ts(as_i)$) will return the ie(as) who owns the minimum qos in the set*

③ *SumQoS($IE(\pi_{as})\_{Stan}$) will return the sum of all $ie(as_i)$s' qos*

in *IE* according to the relationship between *ie(as)*s. Secondly, before atom-service has been installed into cloud, *IE* can be simplified further and *IAIES* be reduced to a certain extent, after analyzing the relationship between *IAIES* and *IE*.

In order to simplify *IE*, we need to know the information of each atom-service family. The projection of *IE(π_{as})* on family *i* is also an atom-service installation expression series, denoted by $Proj^{IE(\pi_{as})}_{F(i)}$. It is composed by the *ie(as)*s in *IE(π_{as})* whose family number is *i*, according to its former order in *IE(π_{as})*. Firstly, we get the family number series *FN(π_{as})* through traversing $IE(\pi_{as})\_{Stan}$. Secondly, we pick up all $ie(as_i)$s to compose the projection on family *i* – $Proj^{IE(\pi_{as})\_{Stan}}_{F(i)}$ (*i*∈*FN(π_{as})*).

For a service deployment, the more complex the function is, the more atom-services the service contains. At the same time, the scale of the deployment and the cost of the cloud vender are much larger. So, optimization of service deployment is one of the most important processes to improve the ability of cloud popularization. The service deployment result should keep the same state after optimization. Algorithm 2 can implement this simplification.

For example, $Proj^{IE(\pi_{as})}_{F(2)}$={*($as^3_2$, BC, 0.2, F), ($as^2_2$, BC, 0.1, T), ($as^2_2$,BC∧FC, 0.4, F), ($as^1_2$, FC, 0.9, F)*}. According to

Algorithm 2: Simplify $IE(\pi_{as})$'s projection $Proj_{F(i)}^{IE(\pi_{as})}$ on family i

Input: $Proj_{F(i)}^{IE(\pi_{as})}$

Output: $Proj_{F(i)}^{IE(\pi_{as})}\_Simp$

Step:

1. *let* IE_t=$Proj_{F(i)}^{IE(\pi_{as})}$
2. *Select all ie(as)s into* IE_{new} *from* IE_t *where IP(as)=true, and then delete these ie(as)s from* IE_t
3. *for each ie(as) in* IE_{new}*, do Simplified(ie(as),* IE_t*)*
4. *while* $IE_t \neq \emptyset$*, do*
   a. *let* ie_t =*MinQoS(*IE_t*), add* ie_t *into* IE_{new}
   b. *do Simplified(*ie_t*,* IE_t*)*
5. *adjust the ie(as) order in* IE_{new} *according to* $Proj_{F(i)}^{IE(\pi_{as})}$
6. *let* $Proj_{F(i)}^{IE(\pi_{as})}\_Simp$= IE_{new}
7. *return* $Proj_{F(i)}^{IE(\pi_{as})}\_Simp$

Sub-function: *Simplified(*ie_t*,* IE_t*)*

a. if comp(ie_t*)=BC, then delete all ie(as)s from* IE_t *where vn(ie(as))*$\leq$*vn(*ie_t*)*

b. if comp(ie_t*)=FC, then delete all ie(as)s from* IE_t *where vn(ie(as))*$\geq$*vn(*ie_t*)*

c. if comp(ie_t*)=BC*$\wedge$*FC, then delete all ie(as)s from* IE_t

algorithm 2, we can get that $Proj_{F(2)}^{IE(\pi_{as})}\_Simp$={($as_2^3$, *BC, 0.2, F*), ($as_2^2$, *BC, 0.1, T*)}.

Notice that, the simplified *IE* may not have the best QoS, but it is the best optimized process which has a better QoS with the deployment result state *US*. When any one wants to carry out the approach, with the precondition that keeping the function *Simplified(*ie_t*,*IE_t*)* fixed, he can choose other strategies to deal with QoS and adjust algorithm 2. For example, we can enumerate all possible simplified results and select the one with the best QoS (We don't pursue best QoS in this paper. The approach can be adjusted by other QoS analysis methods without influence our research achievement).

Algorithm analysis: Algorithm 2 is used to simplify the atom-service installation expression series *IE*. When an *as* is deleted from *IE*, there must be other *as'* which can complete the function of *as* ensured by sub-function *Simplified*. So IE_Simp can implement the original *IE's* functions. That is to say, this simplification satisfies *usability-condition*. And at the same time, it satisfies *safety-condition*. So, algorithm 2 can carry out a *perfect simplification*.

IV. Service deployment optimization and analysis

Based on the relationship between the atom-services which belonging to the same service family, we simplify *IE* itself according to algorithm 2. Next, taking *IAIES* into account, we will do more optimizations during service deployment process relying on algorithm 3.

For example,$Proj_{F(2)}^{IE(\pi_{as})}\_Simp$={($as_2^3$, *BC, 0.2, F*), ($as_2^2$, *BC, 0.1, T*)}. In *IAIES*, there are three *ie(*$as_{\overline{2}}$*)*s – (as_2^1, *FC, 0.4, T*), (as_2^2, *FC, 0.5, F*), (as_2^5, *FC, 0.2, F*). (as_2^2, *FC, 0.5, F*) has been uninstalled from the cloud and deleted from *IAIES*, whose function replaced by (as_2^2, *BC, 0.1, T*) according to algorithm

Algorithm 3: Service Deployment Optimization

Input: $Proj_{F(i)}^{IE(\pi_{as})}\_Simp$

Step:

1. *let* IE_t=$Proj_{F(i)}^{IE(\pi_{as})}\_Simp$
2. *for each ie(as) in* $Proj_{F(i)}^{IE(\pi_{as})}\_Simp$
   a. *if IP(ie(as))=true*
   • *if comp(ie(as))=BC, then uninstall all ie(as')s and delete them from IAIES, where IP(as')=false and vn(as')*$\leq$*vn(ie(as)) and fn(as')=fn(ie(as))*
   • *if comp(ie(as))=FC, then uninstall all ie(as')s and delete them from IAIES, where IP(as')=false and vn(as')*$\geq$*vn(ie(as)) and fn(as')=fn(ie(as))*
   • *if comp(ie(as))=BC*$\wedge$*FC, then uninstall all ie(as')s and delete them from IAIES, where IP(as')=false and fn(as')=fn(ie(as))*
   b. *else*
   • *if exist one ie(as') in IAIES where Satisfy(ie(as'),ie(as))=true, then continue*
   c. *install ie(as) into the cloud and add it to IAIES*
3. *return*

sub-function: *Satisfy(*ie_1*,*ie_2*)*

a. if fn(ie_1*)*$\neq$*fn(*ie_2*) then return false*

b. if comp(ie_1*)=BC*

- *if vn(*ie_1*)*$\geq$*vn(*ie_2*), then return true*
- *else return false*

c. if comp(ie_1*)=FC*

- *if vn(*ie_1*)*$\leq$*vn(*ie_2*), then return true*
- *else return false*

d. if comp(ie_1*)=BC*$\wedge$*FC, then return true*

2. (as_2^3, *BC, 0.2, F*) has been canceled from the installation series because of (as_2^1, *FC, 0.4, T*). After service deployment, there are three *ie(*$as_{\overline{2}}$*)*s existed in *IAIES* – (as_2^1, *FC, 0.4, T*), (as_2^5, *FC, 0.2, F*), (as_2^2, *BC, 0.1, T*).

Algorithm analysis: Algorithm 3 optimizes the service deployment by analyzing the relationship between *ie(as)s* in *IE* and *IAIES* during atom-services series been installed into the cloud. Firstly, atom-services with the IP_T must be installed into the cloud. And at the same time, the functions of atom-services uninstalled have been implemented by the new installed atom-services. So this operation satisfies both *usability-condition* and *safety-condition*, which is a *perfect simplification*. Secondly, whether the atom-services with the IP_F is installed into the cloud, it lies on the *compatibility* and *installation policy* of atom-services in *IE*. And *IAIES* is also considered. If an atom-service is canceled from *IE*, there must be other atom-services in *IAIES* which can implement its function. So this simplification satisfies *usability-condition*. And whether installed or not, it satisfies *safety-condition*, which shows that it is *perfect simplification* too. That is to say, the simplified operation using algorithm 3 is also a *perfect simplification*.

We install the *IE* projections on each service family into the cloud using algorithm 3, and we will realize the service optimization deployment.

According to above research, we design the system framework, as shown in Fig.2. The service deployment framework

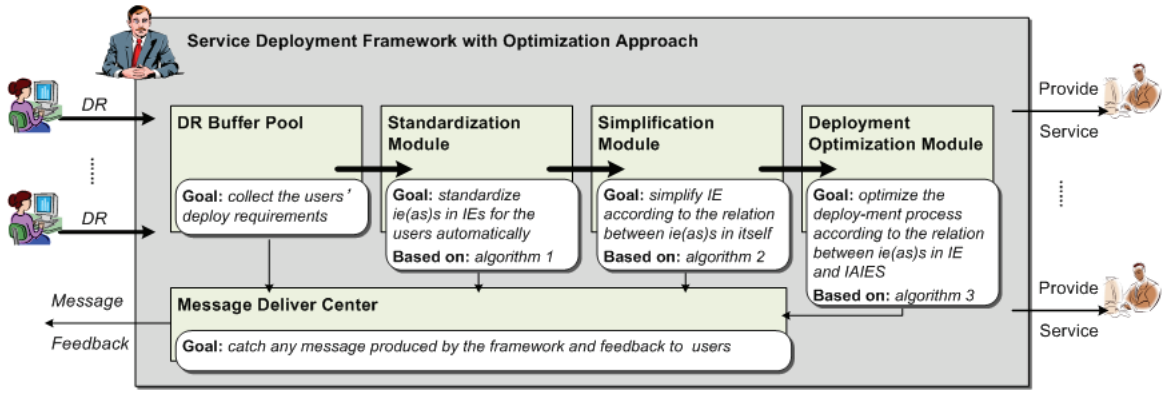

Fig. 2. Service Deployment Framework with Optimization Approach

with optimization approach contains five main modules. *DR Buffer Pool* collects all the users' deployment requirements. When the buffer is full, it calls *Standardization Module* to standardize the atom-services installation expressions series. Then *Simplification Module* does the first step of simplification according to algorithm 2. During atom-services being installed into the cloud, *Deployment Optimization Module* can do the further simplification. *Message Deliver Center* feeds back the catched messages to the users.

V. Experiment Analysis

According to the approach, there are eight important parameters which influence the optimization effect most – the number of *AAIES* (p_1), the number of *IAIES* (p_2), the number of *ie(as)*s to be installed (p_3), the number of service families (p_4), the number of versions in one family (p_5), the ratio of *ie(as)s* installed with IP_T (p_6), the ratio of *ie(as)s* which does not belong to an existing service family (p_7), the ratio of *ie(as)s* to be installed which belongs to an existing service family with IP_T (p_8). The optimization in time cost is obvious. For the time complexity of our algorithms is $O(mn)$ in the worst case, which is millisecond-level in our simulated experiment. In real deployment environment, the time cost of any service installation is at least second-level. If any one service is simplified from *IE*, the vender of the cloud can save considerable time.

Now let us analyze the resource cost. There are two parameters that can show the effect of optimization of our approach – the ratio of the reduced number of *ie(as)s* in *IE* through algorithm 2 and 3 (R_{IE}), the ration of the uninstalled number of *ie(as)s* in *IAIES* through algorithm 3 (R_{IAIES}). $EO(R_{IE}, R_{IAIES})$ is used to depict it.

The simulated experiment is implemented in Visual C# 2008, and executed on a PC with a 2.0 GHz Intel CPU and 2G DDR RAM. The operating system is Windows 7. We simulate a vast atom-service set which is *AAIES* on a server with a random process and assign a installation expression to each atom-service. A cloud environment is built up, where many service applications are simulated to be deployed. These service applications' atom-services compose *IAIES*. A deployment requirement is also simulated out. We optimize the service deployment process using algorithm 1-2-3, and validate our approach by analyzing the experiment results.

Fig.3 shows the experiment results. Firstly, let $Par(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$ to be *(10000, 2000, 0.2, 100, 0.5, 0.05, 50,10)*. Randomly repeating 20 times can get the result 3-a with the mean value of $EO(R_{IE}, R_{IAIES})$=(11.9%, 2.49%). Through Fig.3-a, we can see that the optimized effect is acceptable to a certain extent. Secondly, with seven parameters fixed, we analyze the influence made by one changed. So that we can know the influence trend made by the parameters. Here we give three simulated results.

Fig.3-b shows that the more *ie(as)*s to be installed, the more optimized extent the approach got.

In Fig.3-c, we adjust the ratio of *ie(as)*s to be installed that does not belong to any existing service family. The experiment shows that, along with the rising of the ratio, the optimized effect goes down. Because those *ie(as)*s does not belong to any existing service family, which should be installed to the cloud and could not be simplified. The parameter's influence is opposite to that of Fig.3-b.

At last, we rise the ratio of *ie(as)*s to be installed which belongs to an exist service family and has a true *IP*. The simulated result shows in Fig.3-d. We can conclude that, high ratio of this kind of *ie(as)* can make the ratio of the reduced number of *ie(as)*s in *IE* go down for the same reason of Fig.3-c. But it can make the ratio of the uninstalled number of *ie(as)*s in *IAIES* rise. Algorithm 3 requests these kind of *ie(as)* be installed in to the cloud which can make some atom-services with the same function be uninstalled.

The simulated experiment results show that, the change of the eight parameters can influence the optimization result of our approach. If a cloud owns a larger *AAIES*, *IAIES* and *DR buffer pool*, it can get a better optimization effect.

VI. Conclusions and Future Work

The ability to control and manage the service applications' configurations and deployment is one of the most important tasks to the venders of a service oriented cloud. In order to reduce the cost during the services deployment process and guarantee the deployment state, we design a novel optimization

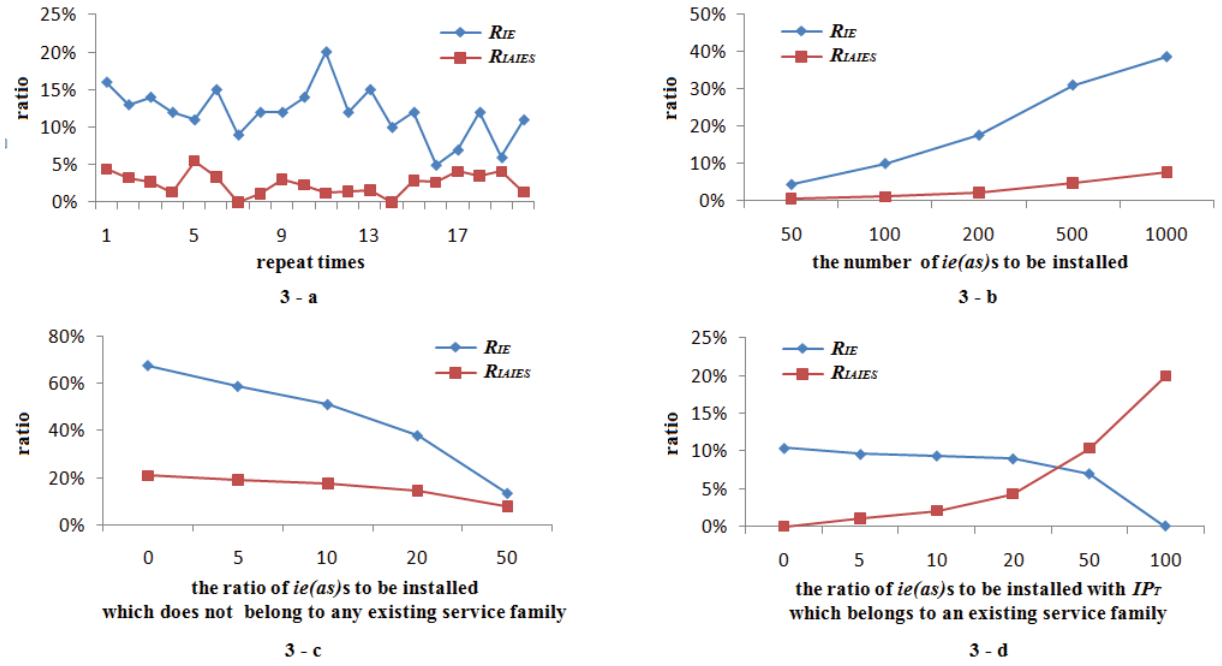

Fig. 3. Simplified Ratio for *IE* and *IAIES*

approach for service deployment. We make the following contributions in this paper:

(1) A service resource deployment framework, which composed by five main modules. Users' deployment requirements are collected by *DR Buffer Pool*. After that, *Standardization Module*, *Simplification Module* and *Deployment Optimization Module* complete the phases of standardization, simplification and optimization in turn. At last, optimized service application can be deployed into the cloud. During this process, any catched message can be fed back to users.

(2) An optimization approach. Before optimization operation, we introduce atom-service installation expression in this paper, where conceptions such as family number, version number, compatibility and installation policy of atom-services are presented as the bases of the optimization algorithms. We design three algorithms to standardize the *ie(as)*s automatically, simplify and optimize the *IE* respectively during service deployment process.

(3) A series of simulated experiments evaluating the approach. Through experiment results, we can conclude that, the eight main parameters influence the optimization effect of our approach. Larger *AAIES*, *IAIES* and *DR buffer pool* are recommended to get a better optimization effect.

In future, we intend to design a multi-dimension QoS-based service deployment system, which can get a better optimization effect and at the same time a satisfying QoS. Further more, because we have collaborated with a world-famous communication lab for a long time, we will realize the framework in communication field and evaluate its efficiency in real world.

VII. Acknowledgements

The work is supported by National High-Tech Research & Development Program (863 Program) under Grant No. 2009AA02Z304.

References

[1] LU Tun, *Modeling and Reasoning of the Software Component Based System Recovery Based on Survivability Specification*. Journal of Software, Vol.18, No.12, pp.3031-3047, December 2007.
[2] Vanish Talwar, et al., *Approaches for Service Deployment*. IEEE Internet Computing, Vol.9, No.2, pp.70-80, Mar./Apr. 2005.
[3] Parrish A, Dixon B, Cordes D, *A conceptual foundation for component-based software deployment*. Journal of Systems and Software, Vol.57, No.3, pp.193-200, 2001.
[4] Chen.Kang, Zheng.Wei-Min, *Cloud computing: System instances and current research*. Journal of Software, Vol.20, No.5, pp.1337-1348, May 2009.
[5] Marios D.Dikaiakos, et al., *Cloud computing Distributed Internet Computing for IT and Scientific Research*. IEEE Internet Computing, Vol.13, No.5, pp. 10-13, 2009.
[6] http://en.wikipedia.org/wiki/Cloud_computing.
[7] Xiaoning Wang, Wei Li, Hong Liu, Zhiwei Xu, *A Language-based Approach to Service Deployment*. IEEE International Conference on Services Computing, pp.69-76, 2006.
[8] Christos Chrysoulas, et al., *Applying a Web-Service-Based Model to Dynamic Service-Deployment*. CIMCA-IAWTIC, pp.128-133, 2005.
[9] Hiroshi Wada and Junichi Suzuki, Katsuya Oba, *Queuing Theoretic and Evolutionary Deployment Optimization with Probabilistic SLAs for Service Oriented Clouds*. Proceedings of the 2009 Congress on Services, pp.661-669, 2009.
[10] G. Huang, et al., *Towards architecture model based deployment for dynamic grid services*. IEEE International Conference on E-Commerce Technology for Dynamic E-Business, pp.14-21, 2004.
[11] William Arnold, et al., *Pattern Based SOA Deployment*. ICSOC 2007, LNCS 4749, pp.1-12. 2007.