SAPF: An On-demand and Trustworthy Middleware Provisioning Framework for A Service Cloud Platform

He Huang, Xudong Liu, Hailong Sun, Dalu Guo, Xu Wang,

The Institute of Advanced Computing Technology, Beihang University, Beijing
{huanghe, liuxd, sunhl, guodl, wangxu}@act.buaa.edu.cn

Abstract. The development of Service oriented software may involve different kinds and layers of service middleware, which configuration is complex and brings extra burden to developers. Under the support of Cloud computing technology, this paper designs and implements a trusted middleware provisioning framework based on the demand of developers, dynamically configuring the environment of service middleware, ensuring the stabilization and reliability of middleware in order to reduce the burden and promote the efficiency of developers.

Keywords: service-oriented middleware; cloud computing; provisioning

1 Introduction

Service Oriented Architecture (SOA) advocates the loose coupling between modules, emphasizing the reuse and integration of existing resources. The SOA has improved the efficiency of software development, but the service-oriented development involves different kinds and layers of service middleware, which may bring extra burden to developers in the complex process of setting up and managing the service middleware environment. It will be meaningful for developers to obtain a service middleware provisioning framework, which will dynamically provide and maintain the service middleware for developers who should focus on software logic.

In recent years, cloud computing [1] has been widely studying. The cloud computing principally focuses on transparently delivering services to the users of cloud platform. Normally, the services are divided into three layers: SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service).

The PaaS layer is response for setting up software environment. With the technical issues of PaaS, we can make full use of existing infrastructure resources to establish suitable software environment which meets different developers' or super stratum applications' various requests. In this paper, we pay attention to leverage the cloud computing technologies to dynamically provide on-demand Service-Oriented Middleware (SOM) environment which can relieve the extra burden, and greatly facilitate the service-oriented software development for SOA developers.

In the complex environment of cloud platform, the reliability of software environment is essential to be concerned about, especially for service middleware

which work on different cloud infrastructures. We have come up with a set of fault tolerance and recovery mechanism in order to ensure the trustworthiness of our service middleware provisioning framework in cloud environment.

In this paper, our contribution includes: (1) a dynamic service middleware provisioning framework (SAPF) in cloud environment; (2) a service middleware and service optimization deployment algorithm (SASOD); (3) a service middleware fault recovery mechanism (SARM); (4) A group of experiments to evaluate the function of the provisioning framework and the performance of the fault recovery mechanism.

2 Related Work

To design and implement an on-demand trustworthy middleware provisioning framework in cloud environment involving three technical fields: middleware providing, middleware fault tolerance and middleware fault recovery.

With the study of SaaS, Guo, C. J. et al. [2] put forward software managing framework based on the view of multi-tenants. In cloud platform, the PaaS layer delivers the providing service of software environment, A. Azeez et al. [3] discusses a multi-tenant SOA middleware for cloud computing.

The primary way for fault tolerance is the replication. In the SOA, replication mechanism consists of the service middleware replication and the service replication. J. Osrael et al. [4] design a service middleware replication framework based on Axis2, moreover, J. Salas et al. [5] propose an approach with service replication.

The roll-back recovery mechanism is widely used for fault recovery. Log and checkpoint are two kinds of technique to deal with fault state. Former research focuses on decreasing the time consumption of analysis process to improve the recovery performance. But in the field of service-oriented middleware, we cannot avoid the time consumption in the process of software reconfiguration and restarting.

3 On-demand and Trustworthy Service Middleware Framework

As we all know, the reliability of service middleware directly determines the efficiency and stability of the upper platform or application. In service oriented software development, developers should focus on the logic and integrating, rather than building and maintaining complex service middleware environment. For this reason, the developers desiderate a framework which should be responsible for providing and managing a set of suitable service middleware to support and facilitate the development. With the technical issues of PaaS, we can make on-demand and instant middleware providing, compared with many traditional Master/Worker style distributed middleware management frameworks, and we design a three-layer framework to promote the efficiency of middleware provisioning.

In order to deliver better providing and managing services of the SOM, we need to add some new functions to the original service middleware. We define *Service Software Appliance* to distinguish the service middleware added management functions from the original one.

- *Define 1 (Service Software Appliance)***:** The *Service Software Appliance*(SA for short) is a special type of service middleware, which adds new managing-related functions (e.g. dynamic deployment, undeployment, restart, multi-runtime instance in a single VM, etc.) based on original service middleware.

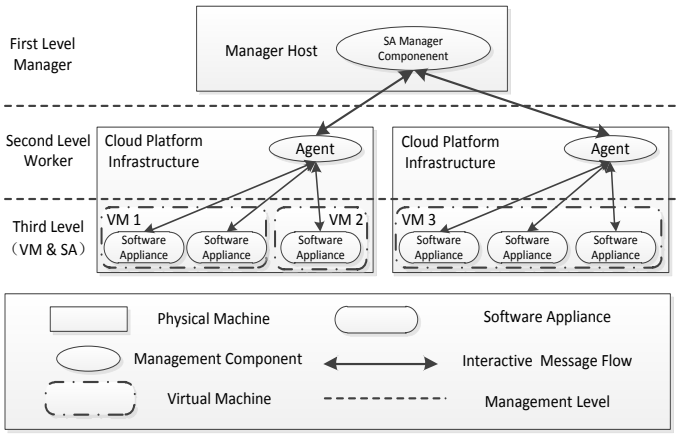

Fig.1. SAPF: a three-layer service middleware provisioning framework

As shown in Fig.1, on top of the framework is Manager Layer, which includes a physical machine (usually a server) with an essential middleware management component, named *Service Appliance Manager Component* which not only deals with the requests from the upper platform or applications, but also globally takes charge of management and state monitor of the lower layer. On the second layer is the collection of *Service Appliance Management Agent* which is pre-installed on each infrastructure of the cloud platform, and provides all dynamic management interfaces of local SA. All of the SAs running in different virtual machines form the third layer of this framework, and each of them is managed by upper agent or component.

3.1 Software Appliance and Service Optimization Deployment Algorithm

Elastic deployment is an important feature of cloud computing. The SAPF is in the environment of cloud computing, so it should support on-demand deployment to meet the dynamic requests from different developers. We define the *Software Appliance Request* to describe the requests from upper layer applications or users.

- *Define 2 (Software Appliance Request)*: a request R to the SAPF is a tuple (*O, C, G, Ws, D*) where O is the type of request, *C* is the SA involved, *G* is the full set of global services (including web applications, web service, composite service, etc.), $Ws \in W$ is the service identification, and $D \subseteq W$ is the dependent set between *Ws* and the other services, *D* can be $\emptyset$ means that *Ws* has no dependent relationship with other services.

With the *Define 2*, we need to define the full set of infrastructures, $= \{H_1, H_2, H_3 \cdots, H_m | m \in N^+\}$. For each infrastructure H_i , $H_i = Tomcat_i \cup Axis2_i \cup BpmnEngine_i$, among which $Tomcat_i = \cup_{j=1}^{kt_i} Tomcat_{i,j}$is the deployment situation of web application SA where i is the global sequence number of infrastructure, and $kt_i$is the local sequence number of SA, as well as $Axis2_i, BpmnEngine_i$ are the deployment situations of service SA and composite SA.

The main idea of SASOD (Software Appliance and Service Optimization Deployment) algorithm is to leverage cloud computing technologies to make an optimization deployment which makes rational utilization of existing resources to meet the developers' requests. SASOD Algorithm will set a threshold $N_{i\_threshold}$ of minimum SA deployment number, according to the actual performance of each infrastructure. When a software appliance request is coming, the *Service Appliance Manager Component* on the top layer will dynamic decide whether needs to deploy a new SA of right type or use the existing ones from *H* with lower load. After execution of the SASOD algorithm, the SAPF can accomplish the on-demand of service, SA deployment and fill the dependence requirement between services.

3.2 Software Appliance Recovery Mechanism

The SASOD algorithm guarantees the on-demand deployment of SA. After deploying SAs and services in the SAPF, the *Service Appliance Management Agents* begin to monitor the running state of them by heartbeat polling. Once the SA running fault occurs, the upper agent captures the fault message and reports it to the SA component, and trigger the SARM (Software Appliance Recovery Mechanism).

When fault occurs, as other middleware, the SA will likely lose all services deployed in it and become unreliable, even crash. The primary idea of the SARM is to globally search all the SAs to find out a set of candidate SAs which have similar deployment state of service with the fault one, and then pre-calculate the recovery time consumption of each candidate with the purpose of determining the best one used for recovery.

In order to guarantee the efficiency of fault-recovery, we prepare three different recovery strategies for dynamic choosing and executing. The first strategy is to restart the fault SA locally and re-deploy all services deployed in the SA before the occurring of fault. The second strategy is to choose a local SA candidate which has less difference of services with the fault one, and then move the different services to the local SA. The last one is to choose a remote SA candidate which has less difference of services with the fault one, and then transmit the different services to the remote SA. Obviously, the last two strategies can efficiently save recovery time by avoiding configuration and restarting of SA, but which one to be chosen is dynamic according to the amount of each pre-circulation time: $T_{restart\_Strategy}$, $T_{local\_Strategy}$ and $T_{remote\_Strategy}$.

We assume a fault recovery process to make the SARM clear. Using a service SA as the recovery example ($Axis2_{i,j}$), we need to make a more detailed define that $Axis2_{i,j} = \{Ws_{i,j,1}, Ws_{i,j,2}, \cdots, Ws_{i,j,n_{ij}}\}$ to express the deployment situation of services where i is the global sequence number of infrastructure, j is the local sequence

number of SA and n_{ij} is the amount of services in $Axis2_{i,j}$.According to the definition above, we have $H_i^{'} = \cup_{j=1}^{k} Axis2_{i,j}$ to express the global situation in the fault recovery assumption, omitting $Tomcat_i$ and $BpmnEngine_i$. Here are eight variables:

1. $S_{Ws_{i,j,k}}$ is the size of the $Ws_{i,j,k}$ service file.
2. $RTT_{i,j}$ is the round trip time between $H_i^{'}$ and $H_j^{'}$.
3. $T_{Axis2_{i,j}\_start}$ is the actual measurement of time that $Axis2_{i,j}$ starts on $H_i^{'}$.
4. $T_{H_{i,j}\_transmit}$ is the actual measurement of time that deploy $Ws_{i,j,k}$ service on $Axis2_{i,j}$.
5. $RS_{H_{i,j}\_transmit}$ is the actual measurement of transmission rate which equals $\frac{2\times\sum S_{Ws_{i,j,k}}}{RTT_{i,j}}$.
6. $T_{esti\_H_i\_restart}$ is the estimated time that a certain SA start on H_i where

$$T_{esti\_H_i\_restart} = \alpha_{base} \times T_{esti\_H_i\_restart} + (1-\alpha_{base}) \times T_{Axis2_{i,j}\_start} \quad (1)$$

7. $T_{esti\_axis2_{i,j}\_deploy}$ is the estimated time that a certain service deploy on $Axis2_{i,j}$where

$$T_{esti\_axis2_{i,j}\_deploy} = \beta_{base} \times T_{esti\_axis2_{i,j}\_deploy} + (1-\beta_{base}) \times T_{Ws_{i,j,k}\_deploy} \quad (2)$$

8. $T_{esti\_H_{i,j}\_transmit}$ is the estimated time that transmit services between $H_i^{'}$ and $H_j^{'}$ where

$$T_{esti\_H_{i,j}\_transmit} = \gamma_{base} \times T_{esti\_H_{i,j}\_transmit} + (1-\gamma_{base}) \times \frac{\sum S_{Ws_{i,j,k}}}{T_{H_{i,j}\_transmit}} \quad (3)$$

With eight key variables above, we can clearly state the time pre-calculation of each strategy as follows (assuming $Axis2_{i_0,j_0}$is in fault):

- Local Restart Strategy execution time pre-calculation ($T_{restart\_Strategy}$):

$$T_{restart\_Strategy} = T_{esti\_H_{i_0,j_0}\_restart} + \left|Axis2_{i_0,j_0}\right| \times T_{esti\_axis2_{i_0,j_0}\_deploy} \quad (4)$$

- Local Immigration Strategy execution time pre-calculation ($T_{local\_Strategy}$):

We use D_j to express the difference of services between fault one and candidates.

$$D_j = Axis2_{i_0,j_0} - Axis2_{i_0,j}, j \neq j_0 \quad (5)$$

$$T_{j\_migrate} = |D_j| \times T_{esti\_axis2_{i,j}\_deploy} \quad (6)$$

$$T_{local\_Strategy} = min\{T_{j\_migrate}\} \quad (7)$$

- Remote Transmission Strategy execution time pre-calculation ($T_{remote\_Strategy}$):

$$D_{i\_copy} = Axis2_{i_0,j_0} - H_i^{'}, i \neq i_0 \quad (8)$$

$$S_{D_{i\_copy}} = \sum S_{Ws_l}, Ws_l \in D_{i_{copy}} \quad (9)$$

$$T_{i\_copy} = \frac{S_{D_{i\_copy}}}{T_{esti\_H_{i,j}\_transmit}} \quad (10)$$

Considering the large number of global SA, the remote strategy will use pruning to narrow the scope of comparison. After removing the infrastructures which have much cost than restart strategy and local immigration strategy, we begin to calculate:

$$D_{i,j} = Axis2_{i_0,j_0} - Axis2_{i,j}, i \neq i_0 \quad (11)$$

$$T_{i,j\_emigrate} = T_{i\_copy} + \left|D_{i,j}\right| \times T_{esti\_axis2_{i,j\_deploy}} \tag{12}$$

$$T_{remote\_Strategy} = min\{T_{i,j\_emigrate}\} \tag{13}$$

After getting the three pre-calculation time, the *Service Appliance Manager Component* will determine which recovery strategy to choose, and execute related recovery process so that the SARM can decrease the fault recovery time consumption.

4 Implement of SAPF

The SAPF delivers the trustworthy service of SA provisioning with on-demand feature that can instantly meet the requests from different developers in the cloud platform. *Service Appliance Manager Component* cooperates with *Service Appliance Management Agent* to support the framework.

- The Implement of *Service Appliance Manager Component*

The component is designed base on Apache ServiceMix which is an open-source enterprise service bus project. The component has 3 main modules: (1) a request cache and scheduling module to schedule the request to the proper agent; (2) a global SA state storage module to fetch and update the running state of each SA from the agents; (3) a fault detection and recovery module to deal with the fault recovery tasks.

- The Implement of *Service Appliance Management Agent*

The agent consists of an executable program and a function service. We implement 4 main modules: (1) a local service repository to store the service resources to facilitate the fault recovery; (2) a request analysis and execution module to analyze and execute the request dispatched from the component; (3) a state monitor module to keep rolling and reporting the running state of local SAs; (4) a management service to provide the dynamic management interfaces of SA.

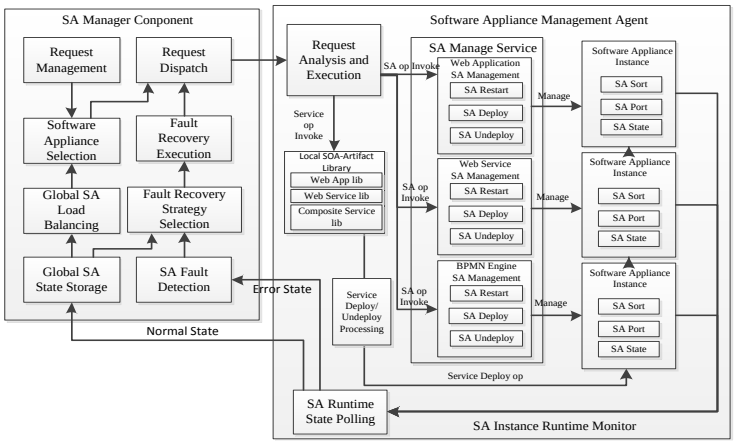

Fig.2. The implement of SA Manager Component and SA Management Agent

5 Performance Evaluation

Focusing on the on-demand provisioning and trustworthy runtime management features discussed above, we designs three experiments to validate the function and efficiency, the environment of experiment contains ten physical machine each of which contains processor of Inter Core 2 Duo E8400,4GB RAM,100Mbps bandwidth network, with the windows7 OS, 1.6.0 version of JVM, in this experiment we use Tomcat6.0, Axis2 1.5.1 and BPMN Engine[6] as three kinds of original middleware, the SAPF will automatically configure them and supplement some functions to them which make them dynamically provide and to be easily monitored.

Experiment 1 is based on the service SA, three groups of experiment simulate the request to 100 web services, the third group has the replication mechanism which the group 1 and 2 does not have and the number of replications is three which means every web service has three replications. 25 out of the 100 services rely on other web services when they are deployed which means the related web services have to be deployed on the same VM and the service SA.

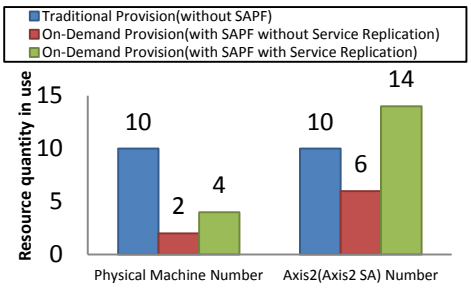

Fig.3. The comparison of resource usage in three provision ways

The Fig.3 implies that under the traditional way of SOM, it needs ten physical machines and ten service containers because of a physical machine only having one SOM, but in our way, it has saved a lot of physical machines for that a physical machine can have many instances of SA. In order to promote the reliability of development process, our platform adopts replication mechanism, so the number of replication is 3 in the third group which actually deployed 300 web services. The deployment of replications need more infrastructure and SA, but compared to traditional way, it still saves the resource of physical machines.

Experiment 2 stimulates a random request list from users to invoke the web services that have already deployed in our system. Under the same condition of error rate of 10%, two groups of experiments respectively use restart recovery mechanism and SARM and statistics the time of accomplishment of the list to validate the efficiency of recovery.

Experiment 2 has 5 physical machines each of them contains 6 web service SA, the number of web services is 60 each of them has 5 replications.

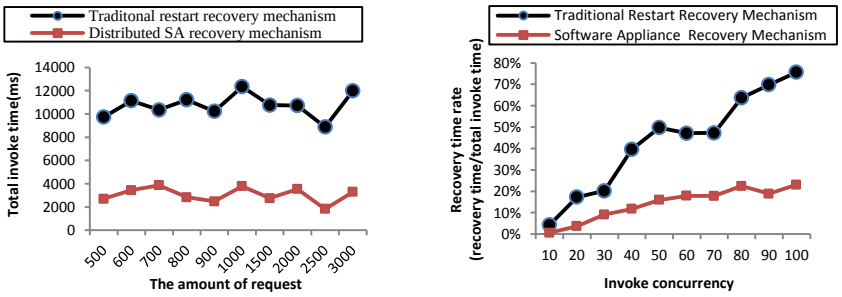

Fig.4 (a) (b). The performance comparison of two recovery mechanism

From Fig. 4(a), we can conclude that under the same number of web service invoking request under same ratio of error, The recovery time of SA is far below the traditional restart recovery mechanism, it is because when the fault happens, the SA recovery mechanism recover from error by using the SAs which are in the right state in our system, avoiding the time cost of restarting middleware.

Under the same web service deployment environment and same error ratio, but different number of concurrency, the rate of error recovery time (recovery time/total time of executing) is different which reflects the rate of recovery time in the system running time. From Fig. 4(b), we can see that with the increment of concurrency, the rate of traditional restart recovery mechanism grows rapidly and brings great influence to our system, when the number of concurrency reaches to 100, the rate of traditional recovery mechanism grows up to 75%. With the SARM, the number of concurrency has slight effect on our system which implies that the new recovery mechanism brings little influence to our system.

Acknowledgments

This work was supported partly by the State Key Lab for Software Development Environment under Grant SKLSDE-2010-ZX-03.

References

1. M. P. Papazoglou, P. Traverso, S. Dustdar, et al.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer. vol. 40 (11), pp. 64-71(2007).
2. Guo, C.J., W. Sun, et al.: A framework for native multi-tenancy application development and management. In: 9th IEEE International Conference on E-Commerce Technology (2007).
3. A.Azeez, S.Perera, D.Gamage, et al.: Multi-Tenant SOA Middleware for Cloud Computing. In: IEEE 3rd International Conference on Cloud Computing (2010).
4. J. Osrael, L. Froihofer, M. Weghofer, et al.: Axis2-based replication middleware for web services. In: Proceedings of the IEEE International Conference on Web Services 07(2007).
5. J. Salas, F. Pérez-Sorrosal, M. Patiño-Martínez, et al.: WS-Replication: a framework for highly available web services. In: Proceedings of 15th Int. Conf. on World Wide Web, pp. 357–366, ACM Press(2006).
6. Y. Ji, H. Sun, X. Liu, et al.: A decentralized framework for executing composite services based on BPMN. In: Service computation (2009).