Realization of the High-density SaaS Infrastructure with a Fine-grained Multitenant Framework

SHIMAMURA Hisashi, SOEJIMA Kenji, KURODA Takayuki, NISHIMURA Shoji

Abstract

In achieving a SaaS-type cloud computing that will contain applications for multiple tenants, the density of containing tenants per infrastructure greatly affects the costs for providing services.

This paper introduces the fine-grained application flow control technology that maximizes the density of containing tenants through letting the tenants share applications and then separating the tenants from each other for individual processes.

Keywords

SaaS, multitenancy, aspect oriented programming

1. Introduction

In recent years, services, so-called SaaS (Software as a Services), are expanding that furnish the features of software on servers via networks. One of the characteristics of the SaaS is allowing users to use the features of software provided as a service based on the rate according to the utilization volume without owning hardware and software, so it can be considered as one of the cloud computing forms.

Although there are a variety of use cases considered for the SaaS, if a SaaS provider is providing services for enterprises and organizations, the users of these services are called the tenants. The reason why the tenants are distinguished from simple service users is that the tenants themselves may have end users. For example, if a SaaS provider is providing for a customer management service, enterprises that use this service to manage their own customer information are the tenants.

Typically, a SaaS business for small- and medium-size companies is required to provide for services at costs lower than those of services for large companies. To reduce the costs for providing for services, the multitenancy technology is essential that allows the tenants to share the same hardware and middleware. This paper introduces the fine-grained application flow control technology that maximizes the density of containing tenants per hardware through letting the tenants share application processes and then separating the tenants from each other for individual processes.

2. Multitenant System

2.1 Multi Instance Type

There are several types of multitenancy approaches so as to operate applications in the SaaS form. The simplest way is to prepare server hardware or a virtual machine for each tenant and run applications on respective operation systems. Thus, a multitenant environment can be easily built (a and b in **Fig. 1**).

To enhance the infrastructure utilization efficiency, it is necessary for the tenants to share up to higher layers. For example, a method of providing each tenant with processes by making the operating system common to the tenants, or a method of deploying application programs for each tenant by making up to the application server common to the tenants can be considered (c and d in Fig. 1).

Benefits from these methods are that the service level can be easily maintained and changes in applications programs are not needed for multitenancy because the operating status of each tenant can be monitored and controlled relatively easily. On the other hand, since application programs corresponding to the number of tenants are provided, deployment alone consumes the volumes of hardware and operating system resources in proportion to the number of tenants contained.

Such multitenancy is a method of deploying and running multiple application programs, so this research refers to it as the multi instance type multitenancy.

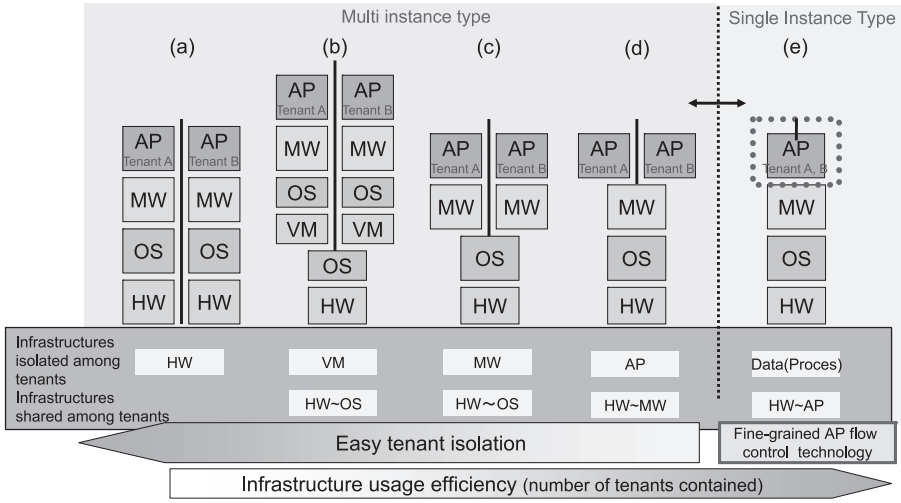

Fig. 1 Type of multitenancy.

2.2 Single Instance Type

To further improve the efficiency of containing tenants per hardware, there is a method that separates tenants for each process by making up to application programs common to the tenants (e in Fig. 1). With this method, there are such drawbacks that application programs must be changed for multitenancy and processing for a tenant is likely to affect that for another. In contrast, there is such a benefit that the efficiency of containing tenants per hardware can be maximized because only one application program is deployed regardless the number of tenants contained.

In this research, this multitenant system is referred to the single instance type multitenancy. Unless otherwise noted, multitenancy refers to the single instance type in this paper.

3. TIP–Through Model

This section explains the TIP-Through model that achieves the single instance type multitenancy.

3.1 Outline

This research designed the Tenant ID Pass Through (TIP-Through) model as a core of the fine-grained application flow control technology. The TIP-Through model defines the scheme of propagating the identifiers of tenants as the processing targets (tenant IDs) and the scheme of allocating the tenants to appropriate processes and resources when different processes are required for the tenants, independently of the application flow, then applies them to the application on demand.

Separating the necessary features for application processing and those necessary for multitenancy from each other in this way results in the following effects:

- Higher efficiency of multitenancy for existing applications
- Standardized code base for single tenant applications used by respective system integrators and multi tenant applications for SaaS

3.2 Configuration of the TIP–Through Model

The TIP-Through model consists of three elements; (1) Identification, (2) Tunneling, and (3) Allocation. **Fig. 2** illustrates the schematic diagram of the TIP-Through model. Respective features and operation examples are explained below.

(1) Identification

"Identification" is a feature that interrupts the application process at the beginning and identifies which tenant is targeted by the process.

For example, assume that a string for identifying the tenant is being inserted at the beginning of the path for the request URL via HTTP. For access to "http://testservice.example.com/tenant1/", which tenant is intended by the process is identified using string "tenant1" as the key and a pair of this string and the identifier for identifying the process for the corresponding access is recorded in the tunneling feature.

(2) Tunneling

Tunneling is a mechanism that links identification to allocation. This feature stores the application process associated with the tenant identifier by the identification feature, propagates the tenant identifier to the allocation location, and enables referencing from that location.

For example, if an application runs on the Java EE server, a pair of the thread identifier that identifies the application process and the tenant identifier is stored in the tunneling feature. As long as normal method calls continue, the tenant identifier corresponding to the process can be acquired by inquiring with the thread identifier as a key. However, cases in which remote calls or thread-to-thread communications via RMI or SOAP spread over threads separately require a scheme that creates tunneling features individually for the calling party and called party and links the tenant identifier between the tunnels.

(3) Allocation

The allocation feature interrupts the application process and allocates the process for the tenant's unique resource.

This feature duplicates or divides the shared resources of the entire application and allocates to the tenants or switches the tenant's unique data such as the database properly for each tenant. This part controls the application process flow depending on the tenant.

For example, many Web applications search for the necessary data for processing and store it in the database. To achieve multitenancy, proper storage is required so that data of the tenants will not be mixed together. Prior to such processing of the tenant's unique resource, the tenant identifier being processed is queried to the tunneling feature and the process flow is controlled so that it will access the database for the tenant corresponding to the tenant identifier in order to achieve multitenancy.

4. Applying the TIP–Through Model to Applications

There are many ways considered about how to apply the TIP-Through model to applications. Major application methods are: (a) Embedding directly into the source code, (b) Using the plug-in mechanism of the application server or application framework, and (c) Using Aspect Oriented Programming (AOP). In this research, the infrastructure for applying the TIP-Through model using the technology of AOP in (c) has been produced experimentally as the multitenant library for the J2EE application. This section explains the overview of AOP and implementation of the multitenant library below.

4.1 Aspect Oriented Programming (AOP)

AOP is a programming approach that separately describes common features which are hard to handle in object-oriented programming and spreading over many classes. For example, common features such as logging and exception handling, spreading over classes, can be described separately from the application's original process then they can be embedded into the application process. Such a description is referred to as the

	Definitions	Features
Identification	A feature that identifies which tenant is targeted by the service requirement	- Identifying the target tenant according to the service requests - Detection of the tenant identification
Tunneling	A mechanism that links identification to allocation	- Storing the tenant identifier after being associated with service requests - Propagating the tenant identifier for allocation
Allocation	Allocating the process for the tenant's unique resource according to the tenant identifier	- Duplicating or dividing the shared resources of the entire application - Allocating the tenant unique data to the appropriate tenants

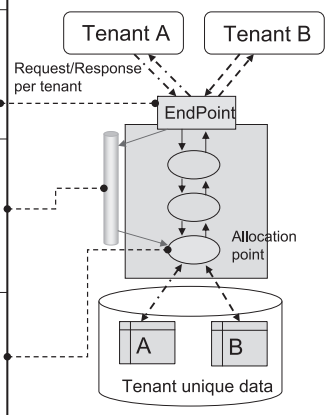

Fig. 2 Schematic diagram of the TIP-Through model.

aspect. The compiler or loader provided by AOP implementation inserts the aspect at the specified position in the application code.

In this research, characteristics for multitenancy are described as aspects and they are applied to the application by the feature provided by AOP. Specifically, TIP-Through model identification and allocation features are described as aspects and inserted into the application process by AOP. Benefits of applying using AOP are:

- Application side code correction and recompiling are not necessarily required.
- Since the additional code for multitenancy can be applied or reset via setting alone, the multi tenant version and single tenant version can share the most part of the code and switching can be easily attained.

4.2 Multitenant Library

In this research, a multitenant library was produced experimentally so that AOP can easily apply TIP-Through to applications. Major components of the multitenant library are explained below according to **Fig. 3** .

(1) Feature Plug-in Library

The identification, tunneling, and allocation features of TIP-Through can be divided into several patterns depending on the structure of the application as the target and on the specifications of the desired multitenant application. Providing parts matching these patterns and reducing the implementation man-hours for the identification and allocation features can greatly save the costs for multitenancy.

(2) Multitenant Container Feature

This feature is an entrance for respective parts of identification, tunneling, and allocation accessing each management feature described below. The multitenant container feature plays the role of a container that integrates and links various management features constituting the multitenant library together.

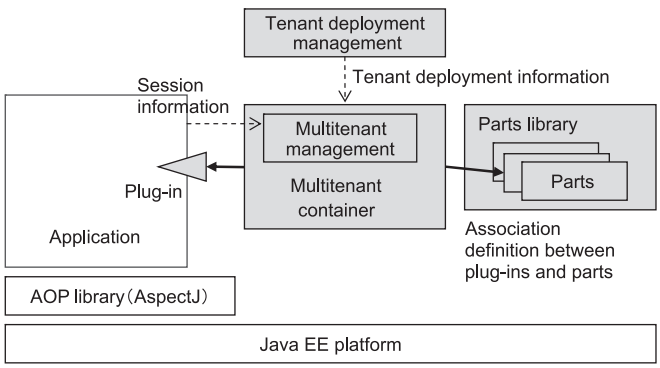

Fig. 3 Overview of multitenant library configuration.

(3) Tenant Deployment Management Feature

This feature adds new tenants to the services. The tenant deployment management feature registers the tenant's unique resource information in the multitenant management feature for the multitenancy application and creates and registers resources on the application server and database server.

(4) Multitenant Management Feature

This feature is the core of the multitenant library and mainly consists of the following features:

- **Session management feature**

The session management feature realizes linkage of the tenant identifier between tunnels if propagation of the tenant identifier cannot be achieved in a tunnel alone. For example, this feature is used to identify multiple application processes for a series of HTTP requests as a set of processes for a particular tenant.

- **Plug-in management feature**

This feature applies the tenant's unique information to the functional parts of identification and allocation defined in the parts library then executes the functional parts. For the tenant's unique information, the tenant identifier is acquired from the tunnel and the unique information of the corresponding tenant from the tenant management feature described below.

- **Tenant management feature**

This feature manages the tenant's unique information such as the access URL, database name, and resource file path per tenant that is set by the tenant deployment management feature.

4.3 Multitenancy Procedure

Not all of multitenancy can be automatically achieved. You must apply each feature of the TIP-Through model while looking at the application code. If multitenancy is automatically applied ad hoc to existing applications, serious degrades may be caused.

In this research, we therefore created a guide for the multitenancy procedures of existing applications, assuming that the multitenant library was to be used. This guide is organized as described below and explains necessary procedures.

- Design: Identification of the location for access to the resource as the allocation target and decision of the identifica-

tion method
- Implementation: Implementation of the codes for the allocation and identification features
- Deployment: Setup for applying the above implementation to the application and middle-ware
- Test: Testing degrades and multitenancy

5. Discussion

We applied multitenancy to a simple inventory control application using the fine-grained application flow control technology in this research. Up to approximately 50 tenants were successfully deployed on a server without any specific problems in operation. Containing tenants at a much higher density can also be achieved, although it depends on the frequency of access to each tenant. For comparison, we performed deployment with the multi instance type multitenancy model shown as d in Fig. 1. Approximately 20 tenants fully consumed the file handles of the operating system and the operation was in trouble. Therefore, it can be said that the fine-grained application flow control technology utilizes system resources efficiently.

We also consider that isolating multitenancy requirements from the application logic using this technology is effective for reduction in man-hours for multitenancy of existing applications.

On the other hand, the prerequisite for the current multitenant library is that many tenants share the same application as one of the issues. Therefore, the requirement of customization for each tenant is not supported. However, it has been demonstrated that the customization requirement can be supported to some degree by switching particular processes for each tenant using the TIP-Through model. We therefore consider that integration into the multitenant library is feasible.

Additionally, if a very high load is applied to a particular tenant, the processing performance of other tenants may be affected. This problem cannot be currently avoided for multitenancy of the single instance type. Technical development relating to performance monitoring and resource control for each tenant is required.

Further, this technology is the model that shares many resources among tenants. Considerations in security are required e.g. to check to see if resources of a tenant may be mistreated as those of another.

6. Conclusion

This paper explained the single instance type multitenancy that allows for maximization of the number of tenants contained per hardware and then explained the fine-grained application flow control technology for achieving such multitenancy. It is expected that multitenancy of existing applications can be efficiently realized through multitenancy using the TIP-Through model as the core of this fine-grained application flow control technology.

*Oracle and Java are registered trademarks of Oracle Corporation and its subsidiaries and affiliates in the United States and other countries.

Authors' Profiles

SHIMAMURA Hisashi
Principal Researcher
Service Platforms Research Laboratories

SOEJIMA Kenji
Assistant Manager
Service Platforms Research Laboratories

KURODA Takayuki
Researcher
Service Platforms Research Laboratories

NISHIMURA Shoji
Assistant Manager
Service Platforms Research Laboratories