Cloud Services for Deploying Client-Server Applications to SaaS

Jianbo Zheng and Weichang Du

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
{jianbo.zheng,wdu}@unb.ca

Abstract. The Software as a Service (SaaS) model of cloud computing is becoming the trend of the new generation of software development due to its low investment, flexibility, and accessibility. Nowadays there are many well used conventional software applications, especially client-server applications. Reuse these application in cloud platforms will benefit both enterprises and the customers. This paper proposes a service framework for easily deploying conventional client-server applications to cloud running as SaaS. The service framework consists of four services: tenant awareness services, tenant management service, application auto-deployment service, and cloud resources management service. The proposed service framework has been implemented and verified on the Amazon AWS cloud engine.

1 Introduction

Software as a Service (SaaS) is an emerging business model in the software industry due to its advantages of flexibility, quick deployment, and scalability. Recently more enterprises have been attracted to build or upgrade their applications or services from local infrastructure to cloud.

In the past decades, the client-server computing model was commonly used in enterprise applications. In this model, software development companies license their software application packages to customers and assist the customers to deploy the software on their own IT infrastructures.

Meanwhile, in SaaS model, software applications usually are adopt by using a multi-tenant architecture, that is, a single application can serve multiple customers or tenants at same time. SaaS based applications can also make enterprises support rapidly onboarding new customers, which is essential to grow user bases of applications. In SaaS model, software developers or venders become service providers which deploy their software applications on cloud and provide deployed applications as services via internet. The customers become tenants of SaaS applications, which no longer need to purchase or install the applications. Instead, customer or tenants subscribe and pay for services on demand via internet.

There have been active research project to investigate migrate conventional client-server applications to cloud environments. Gartner [1] summarizes five general ways to do such migration: re-host on IaaS, refactor for PaaS, revise for IaaS or PaaS, rebuild on PaaS, and replace with SaaS, and gives advice on how to choose the method. Amazon [2] [3] also gives its migration instruction about deploying or moving an existing system to Amazon cloud.

Generally speaking, Enterprises have three options to move their software applications from their local IT infrastructure to the cloud. The first option is to use an existing SaaS application with similar functionality to replace the current local application, such as using the Salesforce CRM to replace the current CRM system. However, it is hard to find out an application which is exactly suited to an enterprise's business. Moreover, an enterprise may have already had their own software which is fit to their current business.

The second option is to re-design and redevelop a current local application based on a PaaS, like Google App Engine, or Windows Azure. However, re-designing and re-developing the application on a new PaaS cloud platform with new technologies will result in high cost and risks to an enterprise.

The third option is to re-deploy or re-host the current local applications to IaaS, like Amazon EC2, or Open Stack. This solution is much more intuitive. With various types of virtual machines, IaaS platform allows enterprises easily to move their current local applications to cloud without many modifications, as shown in Fig. 1. However, this IaaS based solution supports neither resource sharing among tenants nor flexible management of dynamic tenants' subscriptions and usages. As shown in the figure, each tenant keeps an application instance with a running virtual machine on the IaaS cloud. This solution may have a significant resource waste for the software service provider who has more than one tenant to serve.

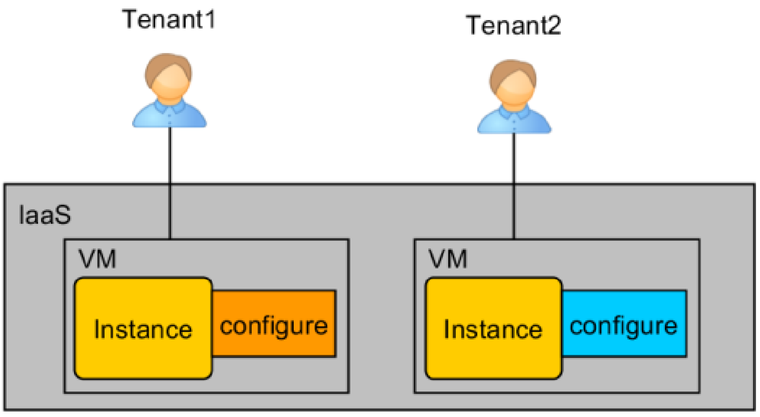

Fig. 1. Simple deployment solution of the application management

To overcome the weak points of the pure IaaS solution, this paper proposes a service framework, named Application to SaaS framework (A2SF), to provide an improved solution and helps software developers deploy their original client-server applications to Cloud, running as SaaS applications. Fig. 2 shows the improved solution for application instance management in A2SF framework. In this improved solution, the original application is divided into two parts, private components and utility components. The private components usually are the customers' data related components, such as the configuration component. For the private components, A2SF wraps them in the multi-tenant awareness layers (service proxy layer in the figure). The utility components usually are the application programs, which will never be changed during application execution. In A2SF, the utility components are considered as shared recourses among tenants. A2SF runs as a service load balancer to share and reuse utility components among tenants.

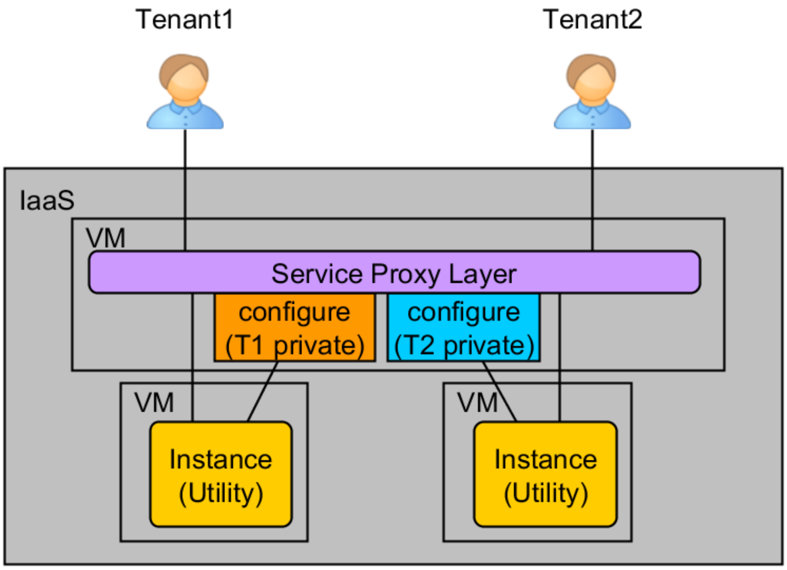

Fig. 2. Improved solution of the application management

The rest of paper is organized as following: Section 2 describes the overview and runtime architecture of A2SF, as well as its components. Section 3 gives a case study on deploying a real web application SugarCRM with A2SF to Amazon AWS cloud, as well as the related experiments' results and analysis. Section 4 briefly describe the related works and compared with our work. Section 5 concludes the paper.

2 A2SF Services

2.1 Overview

A2SF provides multi-tenant awareness services and application instance management services, running as an application container as well as a service load balancer. There are three essential requirements to A2SF services: tenant management and identification, tenant isolation and data security, and automatic service scalability.

2.1.1 Tenant Management and Identification Services

For each tenant, A2SF assigns a unique token to it, which will be carried in each service request from clients. Using the token, the tenant identification service is able to identify which tenant the service request comes from. This tenant identification service, working together with the original application's authentication module, provides identification and authentication functionalities for the deployed application, which is running as a SaaS application in the cloud.

2.1.2 Tenant Isolation and Data Security Services

In A2SF, two approaches of data isolation are applied. One approach is virtualization-based isolation. By this approach, in A2SF, the different tenants' runtime data and configurations of the deployed application will be stored in different virtual machines. The other approach is application-based isolation. A2SF provides two types of multi-tenant awareness services, which execute the access control on services (application) and resources (data) in the runtime A2SF. In the application level, tenants are only allowed to access their own services and resources.

2.1.3 Application Instance Management Services

A2SF provides application instance management services to support automatic application scalability. These services not only implement the virtualization-based isolation but also support the automatically elastic application scalability which is based on the tenants' statuses.

A2SF implements an application instance manager service which is responsible for generating and recycling application instances based on the tenant application management rules. Based on the pre-customized scripts, A2SF supports dynamically to assign an online tenant a virtual machine and deploy the tenant-customized application instance on that virtual machine. The cloud resources management services of A2SF allocate and recycle the cloud resources in the unit of application instance.

2.2 Runtime Services Architecture

A2SF runtime architecture consists of tenant management services, multi-tenant awareness services, and application instance management services, in addition to components or services from the original client-server application.

Fig. 3 shows the runtime architecture of A2SF. The purple parts in the figure are the main services, which includes the service proxy service, tenant manager (including tenant manager interface) service, application instance manager service, data access service, and database service.

Multi-tenant awareness services are implemented by service proxy service and data access service. The tenant manager service provides the tenant information management service and tenant identification service. The application instance manager service is responsible to automatically generate and recycle application instances based on statuses of real-time tenant access. The database service stores the application configuration and tenants' information, such as tenant proxy rules, tenant statuses, and logs.

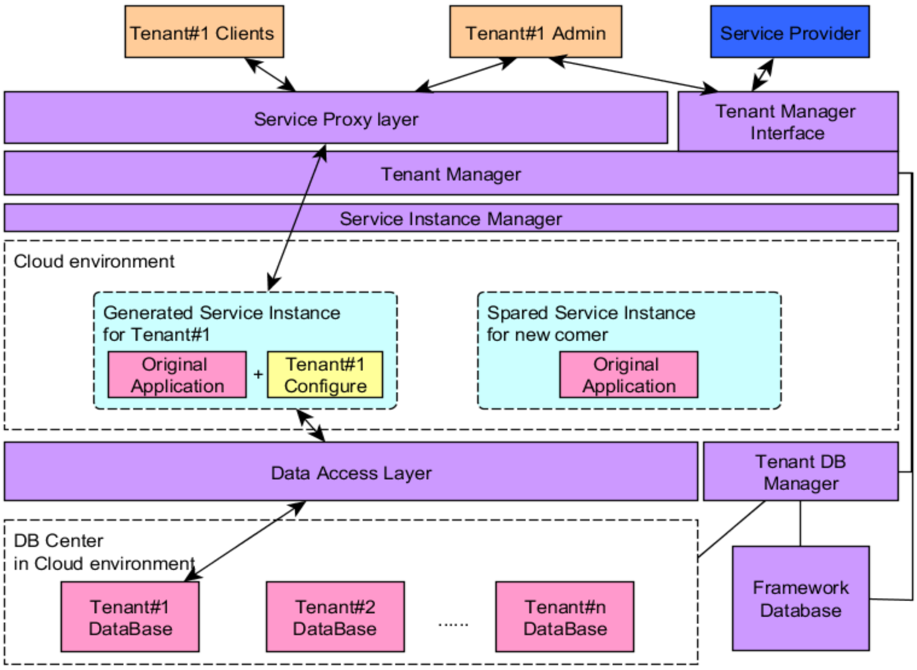

Fig. 3. A2SF runtime architecture

2.3 Services Descriptions

2.3.1 Service Proxy Service

The service proxy service is responsible for service access control. In A2SF, tenants are only allowed to access their own application instances. As shown in Fig. 3, the service proxy service is the single entrance of the deployed application, and all clients or end users' service requests are sent to it, instead of their original local servers.

When the service proxy service starts up, it will load all the tenants' proxy rules. After receiving a client request, the service proxy service first acquires the token of the request and identifies the request's source by invoking the tenant identification service. Once the request is identified, the service proxy service handles the request based on the tenant's proxy rules and forwards the request to the proper application instance. If the request cannot be identified, the service proxy service will deny the service request.

As the original client-server application will be wrapped and deployed to the cloud as a whole package without modification, the original communication protocols as well as the business logic processes between clients and server do not need to be changed between the application deployed in cloud and the original client-side application.

2.3.2 Data Access Service

The data access service is responsible for the data access control between application instances and databases, to assure that tenants are only allowed to access their own data.

In A2SF, an application is considered to have two essential parts, programming code and user data. The programming code is the common part and can be shared with all tenants, while the user data belongs to tenants' private data which should be protected under the access control. Furthermore, the user data can be categorized into local data and remote data. The local data is stored in local files. For instance, the configuration file is a typical example of local data. The remote data is usually stored in a database.

The data access service stores the local data in the database, initializes the local data before the application instance runs, and stores and clears it after the application instance stops.

The data access service provides two ways to deal with the remote data. One way is to implement a data proxy server, which means all data accesses should go through the data proxy server first. This method is for those applications, in which the data connection configurations are hard coded. The other way is to treat the data connection configuration file as tenant's private local data. This way is only suitable for those data connection configurations are separated from the code and easily replaced.

2.3.3 Tenant Manager Service

The tenant manager service consists of the following sub-services: tenant information management service, tenant identification service, and tenant status service.

Through the tenant manager service interface, tenant administrators can manage their information, customize their subscribed services, and review their usage reports. Also, the service provider, namely the vendor of the deployed application, can manage the tenants' subscriptions, set the customization rules for the tenants, and setup the configuration of A2SF runtime.

The tenant identification service is responsible to verify the token, identify the tenant, and return the tenant information including the tenant proxy rules. And the tenant status service is used to obtain the tenants' current statuses. The tenant manager service keeps all the tenants' statuses and updates them regularly, based on the tenants' service requests. The tenant status information includes the tenant application instance status, tenant connection number, tenant live time, and so on. Based on these statuses, the tenant manager service sends the application instance management request to the application instance manager, such as application instance generation request and application instance recycling request.

2.3.4 Application Instance Manager Service

The application instance manager service manages the running application instances in the cloud, including application instance generation and recycle services. This service implements the allocation and recycle of the cloud resources.

The service instance is a server side application, which is composed of a running virtual machine instance and the customized application deployed in it. Each online or live tenant who has clients or end-users to access the deployed application is assigned a customized application instance. Whether or not to generate an application instance is controlled by the tenant manager service. The tenant manager service keeps the tenants' statuses and updates them regularly. When a tenant status matches its rule of generating or recycling the application instance, the tenant manager service will invoke the services, provided by the application instance manager service, to generate or recycle the service instance for the tenant.

When the application instance manager service receives an application instance generation request for a returning tenant, firstly, it obtains an available virtual machine from the IaaS cloud. Secondly the application instance manager service deploys the original application on the virtual machine. Thirdly, the application instance manager service initializes and customizes the original application by loading the tenant's private data to the virtual machine from the data center. Finally, the application instance manager service sends the result and generated application instance information to the tenant manager to update the tenant's statuses.

When the application instance manager service receives a service instance recycling request for an off-line tenant, firstly, the application instance manager service stops the tenant's application instance. Secondly, the application instance manager service collects the tenant's local private data on the virtual machine and saves it to the data center. Finally, the service instance manager service clears the tenant's local private data, returns the virtual machine, and sends the result to the tenant manager to updates the tenant's statuses.

In order to perform the above steps, the application instance manager service wraps the standard operations of the IaaS services (launch, start, stop, and terminate a virtual machine), as well as the remote controlling operations or commands (copy, move, service start, service stop and so on) on the virtual machine, which are usually provided by underlying cloud engine.

Moreover, the virtual instance launching time maybe too long to an application that needs quick responding. So in order to shorten the generation time of application instance, A2SF also implements a virtual pool, managed by the application instance manager service, to always keep a number of spare virtual machines that are waiting to host generated application instances.

3 Experiment

A real-world client-server application SugarCRM has been deployed to Amazon EC2 based on an implemented A2SF prototype. SugarCRM is one of the most popular customer relationship management applications currently, which is implemented in the PHP programming language and supports multiple types of databases including MySQL.

3.1 Deploying SugarCRM on Amazon EC2 with A2SF Services

We classify the components of SugarCRM can be into three types: utility components, local private components, and remote private components, as shown in Table 1.

Table 1. The classification of SugarCRM components

Type	Components
Local private components	Folders: *custom, upload, cache, data, modules;* Files: *.htaccess, sugarcrm.log, config.php, config_override.php*
Remote private components	The database "*sugarcrm*"
Utility components	The rest of components of SugarCRM

The utility components are deployed in the deployable package once for all tenants. The deployable package is a virtual machine image (in the Amazon cloud, it is also called as AMI). The local private components are saved in the bucket of Amazon S3 named by tenant id and tenant name. The remote private components are stored in a MySQL instance in the Amazon RDS.

To integrate the A2SF service and the SugarCRM software together we first configured the Amazon Security Credentials with the list of private components. We then created a customized AMI (Amazon Machine Image) for generating SugarCRM application instances. The AMI ID will be updated later by the tenant management service for dynamic tenant management. Thus when a new or returned tenant comes, the service proxy service can easily and quickly launch a new SugarCRM application instance and customized it for the new tenant based on the AMI. Table 2 shows the contents of the created deployable package for SugarCRM SaaS application.

Table 2. The contents of the deployable package

Component	Description
Service proxy server	This is a runnable jar file, which runs on the portal server as the service entrance.
A2SF management center	This is a war file, which is deployed on a Tomcat server in the portal server.
Data proxy server	This is a runnable jar file. In this migration, it is deployed in the AMI.
A2SF scripts	These scripts are runnable bash files, which can be executed to customize the application instance and save the tenant private local data.
A2SF DB	This is a MySQL database for A2SF framework.
SugarCRM	This is the original application, which is the target of this migration.

Fig. 4 shows the deployed of SugarCRM with A2SF on Amazon AWS Cloud. Firstly, before deploying SugarCRM, A2SF service has already been deployed on Amazon EC2 cloud, and the portal service is the A2SF management service. It will be the entrance of the deployed SugarCRM as well in the cloud. The EC2 AMI is the deployable package of SugarCRM. Secondly, after the portal service is running, we accessed the A2SF management service and registered the SugarCRM deployable package as well as the application information, so that SugarCRM application instances can be generated correctly. Finally, we added several tenants with different IP ranges. We then use several client browsers to access the entrance/portal service on different computers with different IP addresses to simulate multiple tenants' accesses.

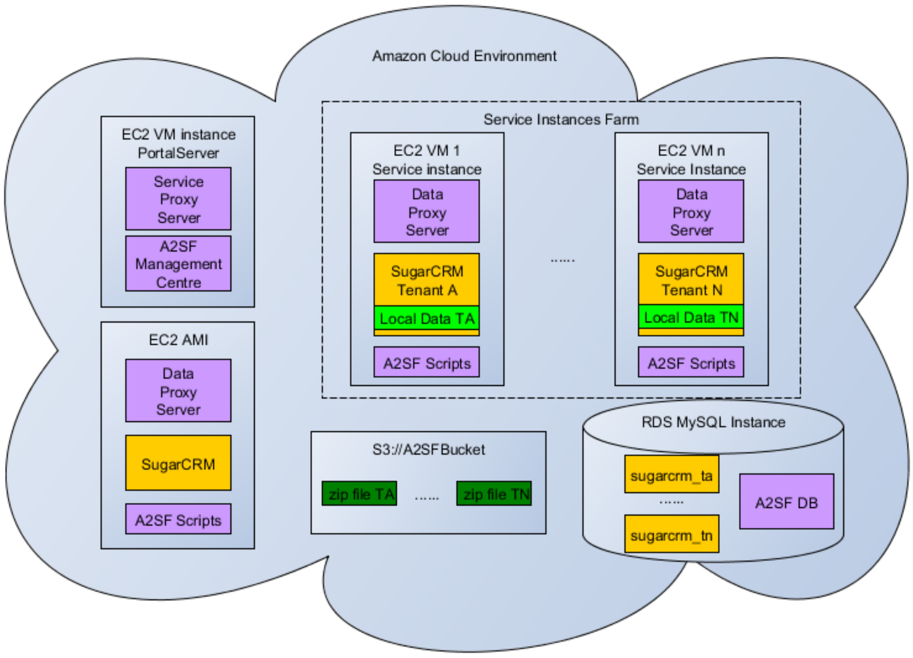

Fig. 4. Deployed of SugarCRM with A2SF on AWS Cloud

3.2 Performance Experiments

For performance evaluation, we also conducted a serial of experiments on the average time latency of A2SF service. In the experiments, 10 computers with different IP address were chosen to simulate 10 tenants. We measured the following four time segments: the first response time, the average time of the normal response, the processing time of the service proxy, and the processing time of the data proxy.

We first recorded the response time of the SugarCRM application deployed on Amazon AWS in IaaS mode without using A2SF. Table 3 shows the performance measurement for a single tenant of the deployed SugarCRM application.

Table 3. Performance of single tenant IaaS SugarCRM

Measure Item	Value (millisecond)
First response time (FRT)	3428
Average stable response time (ASRT)	169.05
Average service proxy time (ASPT)	62.39
The average data proxy time (ADPT)	27.04

We then we used the 10 computers and ran the testing scripts on each computer. Table 4 shows the result measurement.

Table 4. Performance of multi-tenants SugarCRM with A2SF

	FRT	ASRT	ASPT	ADPT
Tenant 1	20438	172.38	70.32	25.92
Tenant 2	52014	180.42	62.83	24.68
Tenant 3	24863	177.50	70.10	30.30
Tenant 4	3509	161.30	62.03	23.43
Tenant 5	60271	172.47	69.19	29.59
Tenant 6	52924	166.03	61.02	21.94
Tenant 7	59248	174.30	69.89	30.59
Tenant 8	54631	175.94	61.08	25.15
Tenant 9	3417	163.46	61.81	22.90
Tenant 10	65741	173.40	70.37	31.05
Average	39705.6	171.72	65.86	26.56

The time latency of using A2SF is around 70 ms.

The first response time of the migrated SaaS system in the best case was 3417 ms and in the worst case it was 65741 ms. These two values also can be taken as the times of generating an application instance on a pre-prepared virtual machine instance and generating an application instance from launching a new virtual machine instance.

The average time of application instance generation is related to the size of the virtual machine pool and the size of target application. Adding a virtual machine pool would improve the efficiency of application instance generation. The experiments show that when the size of virtual machine pool in A2SF was set to 2, the average time of the first response became 39705.6 ms which is significantly lower than the first response time of the worst case.

The experiments also show that the average service proxy processing time is around 65ms, and the average data proxy processing time is around 27 ms. Under the same computing capability, the service proxy processing time and the data proxy processing time are stable. In case of supporting more tenants, we can choose to increase the number of CPU allocate to the service proxy service or add a load balancer service with multiple service proxy service to process service requests.

4 Related Works

Tsai et al. [4] proposes a two-tier SaaS scaling and scheduling architecture that duplicates at both service and application levels along with a resource allocation algorithm that takes different computing power of server nodes into consideration.

Chong et al. [5] proposes a SaaS maturity model, which takes configurability, multi-tenancy efficiency, and scalability as the key attributes of SaaS and classifies SaaS software as four maturity levels. In the model, each level is distinguished from the previous level by addition of one of the three attributes.

Guo et al. [6] proposes a multi-tenancy enabling programming model and framework, consisting of a set of approaches and common services, to support and speed up multi-tenant SaaS application development.

Cai et al. [7] [8] proposes an end-to-end methodology and toolkit for transforming existing web applications into multi-tenant SaaS applications. However, using this migration methodology, developers have to modify the original application software as well as server configurations.

Song et al [9] defines a SaaSify Flow Language (SFL) and proposes a SFL tool which would help convert Java web applications to SaaS applications. However, the SFL tool also has its limitations. First, the SFL tool seems only support Java web applications with JDBC access to databases. Second, since the SFL tool uses memory in threads to keep and share tenant information among layers, it is reasonable to believe that the tool could not support applications whose components deploy on different computers.

The above and other similar research projects have made significant progress in the area of multi-tenant SaaS applications. In our research we more focus on re-deployment of existing client-server applications to cloud, running as SaaS applications, without modification the original code.

5 Concluding Remarks

In this paper, we proposed a cloud services, named A2SF services, for helping deploying conventional client-server applications to cloud, running as multi-tenant SaaS applications. Re-deploying conventional client-server applications, integrated with A2SF services, are relatively easy. The runtime architecture of A2SF services for multi-tenant SaaS are simple and effective, though they are relatively conservative compared to other proposed multi-tenant SaaS systems. The proposed A2SF services have been implemented on Amazon EC2 cloud computing engine with a deployed real-world CRM application. Future work includes performance improvement of A2SF services and implementations of A2SF services on other cloud platforms.

References

1. Gartner. Gartner Identifies Five Ways to Migrate Applications to the Cloud (May 16, 2011), http://www.gartner.com/it/page.jsp?id=1684114
2. Amazon, The total cost of (non) ownership of web applications in the Cloud, Technical report, Amazon Web Services (October 2012)
3. Varia, J.: Migrating your existing applications to the AWS cloud. A Phase-driven Approach to Cloud Migration (2010)
4. Tsai, W.T., Sun, X., Shao, Q., Qi, G.: Two-tier multi-tenancy scaling and load balancing. In: 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE), pp. 484–489. IEEE (November 2010)
5. Chong, F., Carraro, G.: Architecture strategies for catching the long tail. MSDN Library, Microsoft Corporation, pp. 9–10 (2006)
6. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A framework for native multi-tenancy application development and management. In: The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, CEC/EEE 2007, pp. 551–558. IEEE (2007)
7. Cai, H., Wang, N., Zhou, M.J.: A transparent approach of enabling SaaS multi-tenancy in the cloud. In: 6th World Congress on Services (services-1), pp. 40–47. IEEE (July 2010)
8. Cai, H., Zhang, K., Zhou, M.J., Gong, W., Cai, J.J., Mao, X.: An end-to-end methodology and toolkit for fine granularity SaaS-ization. In: IEEE International Conference on Cloud Computing, CLOUD 2009, pp. 101–108. IEEE (September 2009)
9. Song, J., Han, F., Yan, Z., Liu, G., Zhu, Z.: A SaaSify tool for converting traditional web-based applications to SaaS application. In: 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 396–403. IEEE (July 2011)