Towards Effective and Intelligent

Multi-tenancy SaaS

by

Qihong Shao

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved Jan 2011 by the
Graduate Supervisory Committee:

Wei-Tek Tsai, Chair
Ronald Askin
Jieping Ye
Milind Naphade

ARIZONA STATE UNIVERSITY

May 2011

ABSTRACT

Cloud computing has received significant attention recently as it is a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and visualized manner. SaaS (Software-as-a-Service) provide a now paradigm in cloud computing, which goal is to provide an effective and intelligent way to support end users' on-demand requirements to computing resources, including maturity levels of customizable, multi-tenancy and scalability. To meet requirements of on-demand, my thesis discusses several critical research problems and proposed solutions using real application scenarios:

Service providers receive multiple requests from customers, how to prioritize those service requests to maximize the business values is one of the most important issues in cloud. An innovative prioritization model is proposed, which uses different types of information, including customer, service, environment and workflow information to optimize the performance of the system. To provide "on-demand" services, an accurate demand prediction and provision become critical for the successful of the cloud computing. An effective demand prediction model is proposed, and applied to a real mortgage application.

To support SaaS customization and fulfill the various functional and quality requirements of individual tenants, a unified and innovative multi-layered customization framework is proposed to support and manage the variability of SaaS applications. To support scalable SaaS, a hybrid database design to support SaaS customization with two-layer database partitioning is proposed. To support secure SaaS, O-RBAC, an ontology based RBAC (Role based Access Control) model is

used for Multi-Tenancy Architecture in clouds. To support a significant number of tenants, an easy to use SaaS construction framework is proposed.

As a summary, this thesis discusses the most important research problems in cloud computing, towards effective and intelligent SaaS. The research in this thesis is critical to the development of cloud computing and provides fundamental solutions to those problems.

To my parents and Xuanhui

ACKNOWLEDGMENTS

I wish to express my great thanks to all the people who gave me tremendous support and help during my Ph.D. study.

First and foremost, I am heartily thankful to my advisor, Prof. Wei-Tek Tsai. This dissertation would not have been possible without his guidance. From him, I learned how to find high-impact problems, conduct rigorous research, and make effective presentations. Dr.Tsai's brilliance makes my research much more enjoyable. His persistence and dedication encourage me to go through many obstacles. For me, Dr.Tsai is not just an advisor in research, but also whole life advisor. He given me many advises in multiple aspects, especially on how to move beyond my comfort zone to reach my full potential. I would benefit from all these for my whole career.

I also thank my other dissertation committee members, Dr.Jieping Ye, Dr. Ronald Askin and Dr.Milind Naphade, for their constructive suggestions for my dissertation and invaluable help for my career. Their suggestions make my dissertation more complete and accurate. Their recommendations open up many opportunities for my career.

Furthermore, I am proud to be a member of ASU-SRLab. I benefit a lot from the discussions with all of the SRLabers, especially, Dr.Yinong Chen, Xin Sun, Jay Elston, Yu Huang, Wu Li, and Guanqiu Qi. Specially, I appreciate all support from my best friends, Fang Chen, Ye Jiang, Yang Xiao, Huiping Cao and Hongyu Yu. Their consistent support in the past five years helps me handle all difficulties in my research and life.

Finally, I would like thank my mom for her support and encouragement. With-

out her consistent mentally support, I could not finish my PhD study. Also I would like to thank my husband Xuanhui for his love, faith, and confidence in me. This dissertation is dedicated to all of them.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

1. INTRODUCTION

With the advent of modern technology, cloud computing becomes a most important technique on the Internet. Cloud computing is defined as "a computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction", according to NIST (National Institute of Standards and Technology, Information Technology Laboratory)'s standard[61]. Figure 1 is a sample architecture of Cloud Computing. It builds on decades of research in visualization, distributed computing, utility computing, and more recently networking, web and software services. It implies a SOA (service oriented architecture), reduced overhead for the end-user and the long-term cost of ownership, provides great flexibility, on-demand services and unlimited service supply. It extends today's service-oriented architectures into business platforms for the next-generation economy.

There are some essential characteristics of cloud computing in NIST Definition document:

- On-demand self-service. A consumer can have provision computing capabilities, and use resources (server time, network storage and etc.) in a "pay-as-you-go" way, which is an automatical way without any human interaction with service providers.

- Infinite computing resources. The services are provided to end customers in a

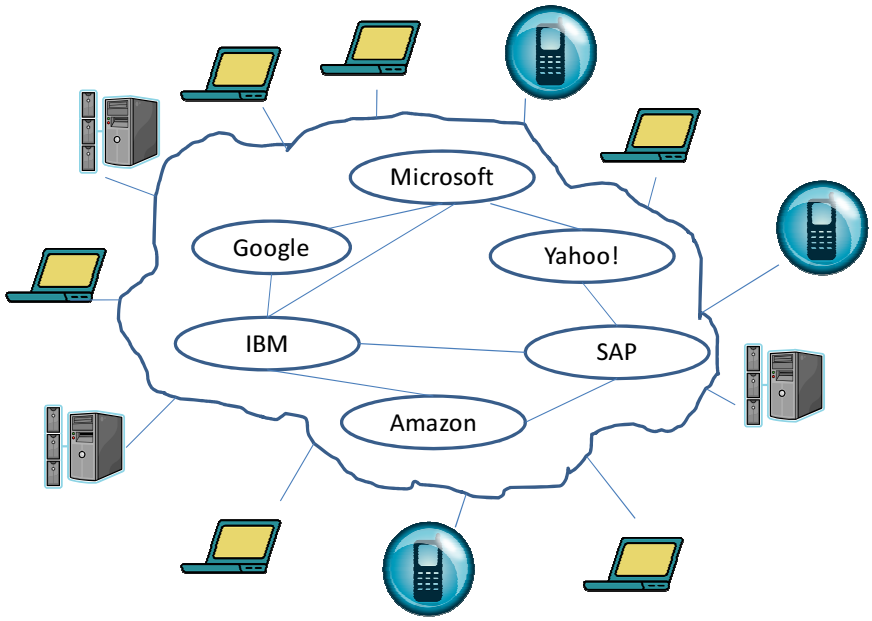


Fig. 1. Cloud Computing

rapid and automatic way. The provisioning can be used to support unlimited resources when consumers can purchase services at any time and with any quantity according to their requirements.

- Broad network access. The whole cloud can have resources form multiple resources, and the network covers many service providers. The services provided by different service providers are available over the whole network. It is easy to access the cloud by heterogeneous platform, from both thin and thick client platforms through multiple devices, such as mobile phones, laptop, PDA and etc.

- Resource pooling. A multi-tenancy model is supported by pooling the service provider's computing resources together. It can assign or reassign different physical and virtual resources dynamically according to consumers' demands.

The customers do not need to know the implementation details and exact locations of the service providers, but simply specify their requests at a very high level. Also the customers do not need to worry about the manage and maintenance of services, but use services and resources when they need.

- Measured Service. To measure the usage of services, a metering capability should be provided at different levels of services, such as storage, processing, bandwidth, and active user accounts. Service usages and resource utilization should be recorded, controlled and monitored for both the providers and consumers.

From a technical point of view, cloud computing is the provision of dynamically scalable and often visualized resources as a service over the Internet on a utility basis. From a conceptual point of view, cloud computing refers to a paradigm shift in computing whereby computing resource and underlying technical infrastructure are abstracted away from the user. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the "cloud" that supports them.

There are four types of deployment models in Cloud Computing, according to wikipedia [33]:

- Private cloud. The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

- Community cloud. The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission,

security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

- Public cloud. The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

- Hybrid cloud. The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

In public cloud, the service sold is named as utility computing. Current examples of public utility computing include Amazon Web Services, Google AppEngine, and Microsoft Azure. Thus, cloud computing is the sum of SaaS and utility computing, but does not normally include private clouds.

Utility Computing(public/hybrid) is preferable to running a private cloud. The reasons are due to the demand and cost consideration. First of all, service demand could be various over time. Provisioning a data center for the peak time workload, it must sustain a few days per month under-utilization while other time over-utilization. Instead, cloud computing lets an organization pay by the hour for computing resources, potentially leading to cost savings even if the hourly rate to rent a machine from a cloud provider is higher than the rate to own one. Secondly, customers' demands are usually unknown in advance. For example, a web startup will need to support a spike in demand when it becomes popular, followed poten-

tially by a reduction once some of the visitors turn away. Finally, we can calculate the "cost associativity" [47] of cloud computing and compare with traditional utility data center way. For example, using 1000 EC2 machines for 1 hour costs the same as using 1 machine for 1000 hours. A web business with varying demand over time and revenue proportional to user hours, we have captured the tradeoff in a patch analytic as follows: the expected profit from using Cloud Computing is the net revenue per user-hour ($revenue - cost_{cloud}$) by the number of user-hours. If we perform the same calculation for a fixed-capacity data center, which has the net revenue as ($revenue - \frac{Cost_{datacenter}}{Utilization}$) by factoring in the average utilization, including non-peak workloads of the data center, by the number of user-hours. It is easy to see, considering the average utilization, cloud computing can have a much better revenue than private data center.

Cloud computing can be classified into three levels of service models according to NIST document:

- "Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings."

- "Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations."

- "Cloud Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls)."

Figure 2 shows the main IT companies with their support in cloud computing, as summarized in [84]. For example, Salesforce.com is a SaaS provider which supports CRM applications. Google AppEngine and Microsoft Windows Azure are PaaS, while Amazon EC2 is sitting at IaaS.

1.1. SaaS Maturity Model

Microsoft has proposed the following SaaS maturity levels [29] with each level adds additional features to the previous level:

- Level 1 - Ad-Hoc/Custom: the simplest level and similar to the traditional

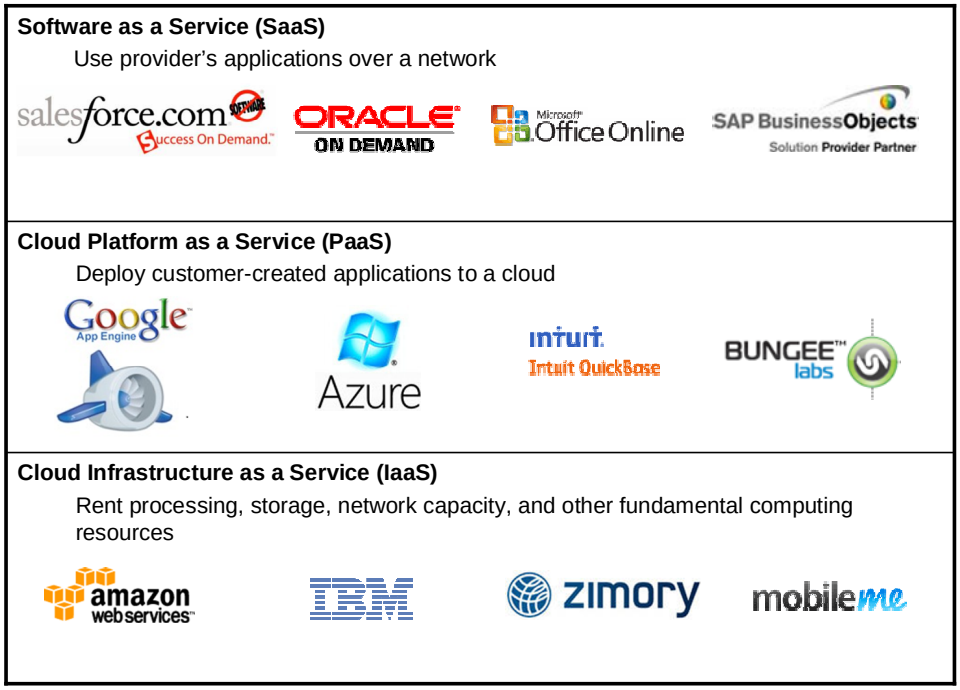


Fig. 2. Cloud Differentials: Service Models[84]

application service provider (ASP) model. At this level, each client has its own customized version of the application and runs its own instance of the application on a server. There is no sharing among tenants and each instance of the software needs to be individually developed. Many existing software programs will satisfy this level by moving the software to a centralized server to provide services for clients. Figure 3(a) illustrates this level.

- Level 2 - Configurable: adds flexibility to the software. Each client has its own customized version of the software; however, at this level a client can specify configuration choices by choosing various configuration options provided by the same software. In the previous level, each version of the software is individually developed for each client, but at this level only one software

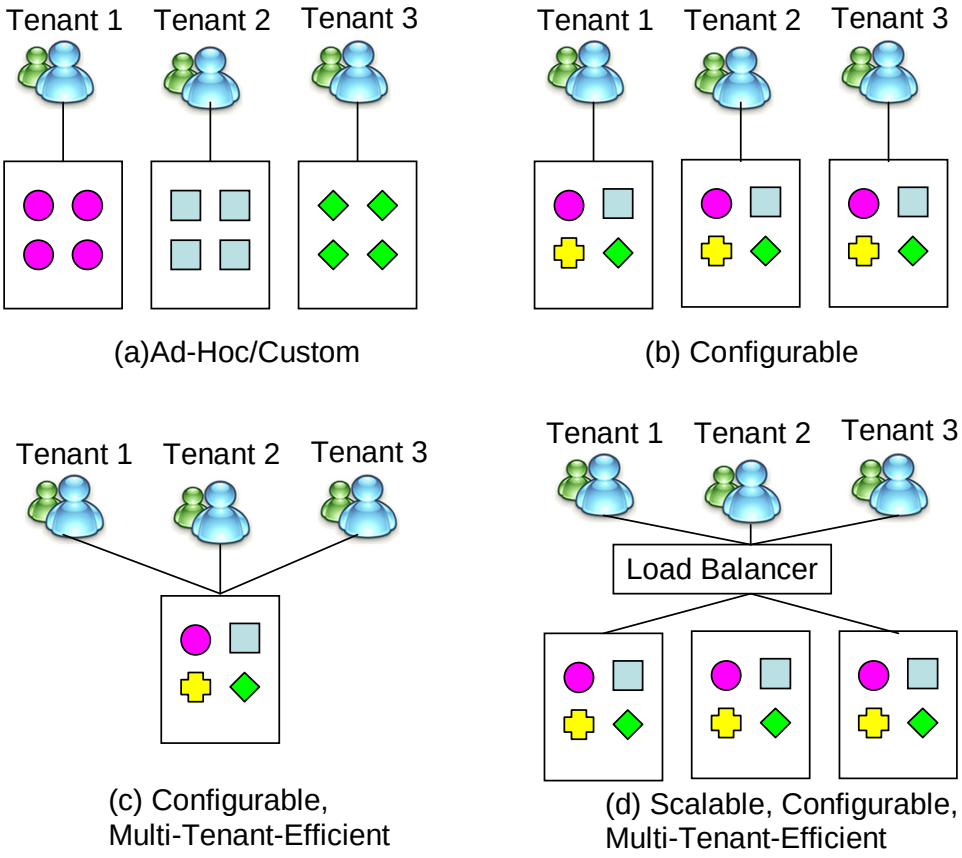


Fig. 3. SaaS Maturity Levels

program is developed with many configuration options to be selected by individual clients. Comparing to the previous level, the software at this level will be more sophisticated and complicated, however, only one version will be developed. The management and maintenance will be eventually easier as the developer no longer need to develop hundreds of thousand versions for each individual customer. Figure 3(b) illustrates this level.

In general, customization may be lightweight or heavyweight:

1. Lightweight variants: These are services with different options or features, and the same services are offered to different customers with different

policies and/or SLA (service-level agreement). For example, a premium service with advanced features such as large storage, better use interface, and allow 24-hours access, and a regular service with standard features such as limited storage, plain user interface, and accessibility limited to certain hours only.

2. Heavyweight variants: These are services that provide different underlying business processes including industry-specific requirements, infrastructure including communication requirements. These services may have similar names but they are rather different if one examines inside these services.

- Level 3 - Configurable, Multi-Tenant-Efficient: adds multi-tenancy architecture to the previous level as shown in Figure 3(c). At this level, all clients will run the same version of the software; however each client can see an initialized configuration of the same software. Note that in the previous level, each client will see a customized version and run that version individually, but at this level, while each client sees a customized version, but in reality each client is sharing the same software with hundreds of thousands of other clients. It is easy to see that the SaaS software at this level is even more complex than the SaaS software at the previous level. In the previous level, while the software need to be customizable by clients, as only one copy will be used for each client, the software does not need to handle the runtime management. But the SaaS software at this level needs to address these new issues. The SaaS

software needs to keep track individual configuration for each client, maintain their databases, and provide customized services at runtime. One can actually view that the SaaS software at this level has a mini-OS behind it that runs a database, and partition the workspace for hundreds of thousand individual customers at runtime.

- Level 4 - Scalable, Configurable, Multi-Tenant-Efficient: The next level of SaaS adds scalability to the previous level as shown in Figure 3(d). One issue of the SaaS software of the previous level is that it may not scale up. As each SaaS software needs to track hundreds of thousands clients, and provides timely services, the workload at the software may be too heavy. One way to solve the problem is to have multiple copies of the same SaaS software of level 3, and each can be called to provide services at runtime. A client will not interact with the SaaS directly, but will interact with a load balancer first, the load balancer will dispatch each request to an appropriate copy of the software for execution. The load balancer constantly monitors the workload of each software copy, and dispatches any new request from a client to an appropriate software copy for execution. The number of copies running at the back end can be increased in case of an increased workload, or decreased in case of a reduced workload. In this way, appropriate number of copies will be maintained at the server to provide optimal performance. The cloud environment for this level will be more complicated than the cloud environment for the previous level as the load balancer will interact with various SaaS copies at runtime. The cloud

environment at this level potentially can provide better services than the cloud environment at the previous level as it can adjust the resources according to the changing environment.

SaaS software may also run on a virtual machine (VM), either with or without multi-tenancy architecture. A VM is an isolated copy of a real machine or system, and multiple VMs can run on the same physical system. A VM can be as large as an OS, for example, a physical machine may run multiple OS platforms to serve different customers. The concept of VM is not new as it has been around for at least forty years. At that time, machines were expensive, and thus multiple OS platforms run on top of a physical machine to save cost. Forty years later, while machines become inexpensive, the VM concept is still heavily used, particularly in cloud computing for a different reason: as applications and data are more valuable than physical machines, but many these applications run on certain platforms only, and thus multiple VMs will be needed for run those applications.

Note that multi-tenancy architecture and VM complement with each other in cloud computing. Visualization allows different platforms to be provided without much additional programming, however, multi-tenancy provides scalability in terms of both software design and programming as only one copy of software will be developed rather than multiple versions of the software to be developed individually. It is possible to combine both VM and multi-tenancy architecture for scalability and flexibility: VMs used for providing a convenient way to establish a platform for program execution, and multi-tenancy architecture is for software sharing.

1.2. Multi-tenancy SaaS

Each SaaS application has a front end and a back end. The front-end stage provides the opportunities for SaaS consumers to customize their requirements, while the back-end stage uses a consistent and scalable approach to support client with a low unit cost. To achieve this goal, multi-tenancy becomes an important feature of SaaS, in which a single instance of SaaS software supports multiple client tenants. In this manner, the service provider can support multiple tenants at the same time, while from a customer point of view; the tenants are isolated and customized for their unique needs.

Multi-tenancy architecture is different from *multi-instance* architecture where a single instance of the software runs on a server serving multiple clients or tenants but multi-instance architecture has multiple (and different) copies of the software serving their clients. Multi-tenancy architecture needs to partition the data internally, and needs to track individual configurations for different clients.

It has been reported that current multi-instance architecture may support dozens of tenants, while multi-tenancy can support a much larger number of tenants. However, this comes with a price, as the scalability level increase, the isolation level decreases. In other words, potentially, multi-tenancy architecture needs to prevent the QoS of one tenant from being affected by other tenants as they share the software and possibly also the database. Note that level 1 and level 2 SaaS mainly uses multi-instance architecture, this section mainly focuses on level 3 and level 4 SaaS applications.

Multi-tenancy architecture needs to address the following aspects:

- **Resource isolation**: It is important for a SaaS application to separate the resources among tenants in a fair manner as all tenants essentially share the same infrastructure and the software. Each tenant may naturally desire to access all the resources needed to achieve the best service performance, however, in case of resource constraints, this may not be feasible for all the tenants. Thus, the system may assign priorities to tenants, and provide differential services for different clients. One simple approach is to assign resources such as CPU and storage statically to SaaS applications if the client requests are regular or constant. However, in a cloud environment, this is unlikely to be true, and thus a dynamic allocation scheme needs to be used. A tenant may specify its resource requirements such as usage patterns ahead of time, so that the SaaS application may schedule the resources accordingly.

- **Customization**: A SaaS application often allows tenants to customize their services including QoS requirements. For example, Google Doc allows different users to specify various features including look-and-feel of the software, but maybe in the future, it may also allow each user to specify the Service-Level-Agreement (SLA) requirements. Note that in multi-tenancy architecture, each will use the same instance of the software, and thus any customization information need to be stored in a database. The information needs to be retrieved and used at runtime to provide a customized service. While this adds flexibility to SaaS, this slows down the processing, as additional computation will be needed at runtime, and adds complexity to the database as individual

customization needs to be stored in addition to various data.

- **Security**: In a multi-tenancy design, software code and data are shared among tenants, and this creates a significant security risk. A tenant, by accident or by design, may actually access data that belong to another tenant. Security issue is indeed one of the most significant issues in cloud computing and SaaS.

- **Scalability**: From maturity level 1 to level 3, an important scalability consideration is software design and programming issues. Level 3 SaaS allows the same software to be used by all tenants, and thus saves significant software design and implementation effort. However, level 3 SaaS applications may have limited scalability if the infrastructure does not have multiple copies of the same software that can be dynamically created to provide services.

Figure 4 is a sample architecture framework to help address those challenges of multi-tenancy architecture mentioned above, which can support transparently multi-tenancy capabilities both at the built-time and at the running time. The tenants will use the same application instance without suffering significant performance downgrading, as well as system security, isolation, and configurability.

There are two types of developers/users in the framework:

- Application-oriented developers (on the top layer): They are responsible for developing or customizing the content of the UIs (user interfaces), business processes and services, will not be aware of the multi-tenancy architecture.

- Infrastructure-oriented developers (at the bottom layer): They are responsible

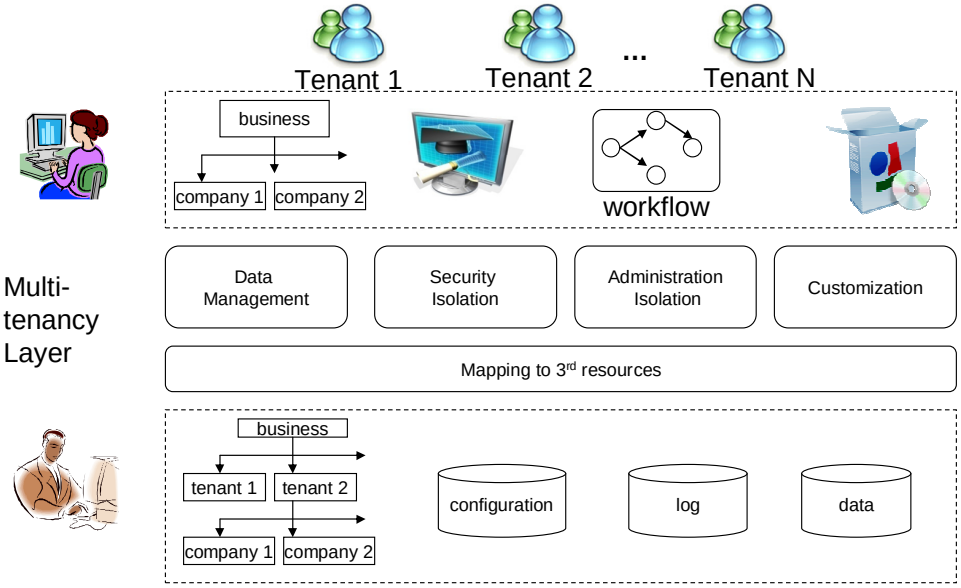


Fig. 4. Multi-tenancy System Architecture Sample

to ensure the effectiveness and reliability of applications with minimum costs.

Multi-tenancy enablement layer is the core of the framework and it provides the separation between the applications and the supporting system resources. The tenants can get the benefits of multi-tenancy without worrying about the complexity of implementing multi-tenancy architecture.

1.3. Real-Time Service-Oriented Cloud Computing

In SaaS, software is maintained and updated on a cloud, and presented to the end users as services on demand, usually in a browser. Many industry service providers start to support SaaS, for example Google's recent announcement of its Chrome OS, an operating system designed to run SaaS applications faster, simpler and more securely further shows that SaaS is viable. With Chrome OS, users will be able to listen to music, play games, watch video, and even store their data online.

As SaaS is gaining popularity, people require more and more real-time features

for a SaaS application. Popular social SaaS applications, such as Facebook, Twitter, made further enhancement to enable real-time communication. Major search engines also jump in the real-time war by providing real-time search results, such as news, sport results, Facebook and Twitter updates. On the enterprise side, start-ups like Arithum [6] provide a real-time cloud computing platform and deliver pioneering real-time consumer and enterprise solutions. Force.com allows real-time integration with other on-premises applications and other third-party cloud services [118].

Cloud computing is inherently real time, and more specifically soft real time, as a user will give up if the cloud does not provide the needed services almost instantaneously. For example, if a popular web-based email system responds slowly, most users will switch to another service provider or use a traditional email server. While the failure of the system to respond quickly may not cause any catastrophic consequences, it might lead to users' dissatisfaction.

Cloud computing uses SOC(Service Oriented Computing) as one of key technologies, however, it has other key aspects that SOC often does not address. Specifically, cloud computing emphasizes significantly about its high-performance server aspects. A cloud environment often has hundreds of thousands of processors with numerous disks interconnected by dedicated high-speed networks, and the infrastructure provides soft real-time computing for Web users.

1.3.1. Client-Site Issues

A cloud client may be a mobile device or a PC that connects to a cloud server. A good example of a client-side system is the recently announced Chrome OS, a network OS developed from Linux, and its open-source version Chromium. They

are essentially lightweight OS designed for web browsing and SaaS applications. Figure 2 shows a comparison between traditional OS boot sequence and Chromium OS fast boot sequence. It takes 14 steps and more than 30 seconds for a traditional OS to open a browser from being turned on, while it takes only 5 steps and 4 seconds for Chromium OS to the same.

Not only are they lightweight, but also adaptive. Specifically, the Chrome software checks the integrity of its code, and if the code is compromised, it will take actions in a pro-active manner to reboot the system. All applications will be web services without installation. The system also verifies the digital signatures of services to ensure that the services are trustworthy. Furthermore, it provides a highly optimized JavaScript engine for efficient processing, and the data are stored in local caches for efficient retrieval. Furthermore, only data needed immediately are retrieved from the Web to avoid unnecessary bandwidth waste. Note that as a consequence of using services, a client system can be viewed as "system + the Web" rather than just the system as it can search services from the Web for execution. All these designs lead to efficient Web service execution with self-adaptive features.

There are several unique challenges in the client site:

- **Efficient client-site execution, caching and pre-fetching**: A lot of feature-rich web applications use JavaScript heavily. The core component of Chrome OS, Chrome browser has a highly optimized JavaScript engine. Moreover, services that are often used can be found at the local cache, and data are also pre-fetched from the Web for better user experience.

- **Paging**: In other words, only retrieve the data that will be used immediately. The obvious reason is that a lot of resources are wasted as most of the results returned will be discarded before being used, and it often results longer lags.

- **Stream Filtering**: Not all requests from the users require computation work. Sometimes they can be safely ignored; sometimes, the according response can be found in server-side cache. But utilizing stream filtering and data comparison tools, the performance can be further improved.

- **Runtime checking, verification, and recovery**: The system keeps on checking the integrity of its code to ensure that nothing is compromised, and if compromised, it takes actions in a pro-active manner to reboot the system. The system also verifies the digital signatures of services to ensure that the services are trustworthy.

- **Environment-aware evolution and adaption**: The environment of a client is the Web, and it depends on search engines to identify the needed services. As the Web is an open and changing environment, the system is thus open and dynamic.

1.3.2. Server Site Issues

Compared to a client, a cloud server may be hosted on a data center with hundreds of thousands of processors and disks interconnected by high-speed networks. One important feature of SaaS is the multi-tenancy architecture. Essentially, a multi-tenancy application is a software program that will be shared by multiple tenants, and the same software can be customized for individual tenants in terms of the

user interfaces as well as functionalities. The cloud server will provide the internal bookkeeping so that data from different clients will be protected. A multi-tenant application can be compared with multi-instance software where individualized software will serve different tenants. The multi-tenancy architecture save significant cost in terms of software application development, however, it also adds significant overhead in bookkeeping to ensure that end users can enjoy satisfactory performance. Some critical issues related to SaaS applications are as follows:

- **Customization**: As a SaaS application serves multiple tenants, it is important that it can be customized for different tenants. This is often done by storing customization data in a backend database in the cloud environment. This leads to significant design issues because this involves database design, concurrent and parallel database processing, and data synchronization. Many cloud companies thus decide to develop special database management systems and distributed file systems to support this operation, such as Google's BigTable and GFS (Google File System). While they provide key features of database management such as indexing and query processing, many traditional database operations are not available for efficient processing. Customization can be done on the level of individual services and the composition of services. The customization can be either tenant-specific or can be multi-tenant aware and reused by other tenants. Most Cloud computing service provides can support customization. For example, Salesforce.com [46] offers a platform (AppExchange platform) that allows tenants to create and deploy their own

extensions to the application

- **Scalability**: Scalability is another feature of multi-tenant SaaS application. Ideally, the cloud environment should provide additional resources proportional to the customer requests while keeping customer tasks at an acceptable level of performance. Two kinds of scalability often need to be addressed including scale-up and scale-out. Scale-up involves making the processor more powerful such as using a more powerful CPU with additional cache space; scale-out involves adding additional processors or networks into the cloud environment . A SaaS application must have a scalable design to meet the real-time requirements.

- **Auto provisioning**: To provide on-demand services to clients, a cloud environment often provides an automated way to supply the needed resources dynamically. The cloud environment is constantly monitoring the progress and status of various applications/services, and if it detects a slowdown of a process due to the lack of resources, it will automatically allocate resources for this task. This is done without any intervention from the end users. In a real-time cloud environment, there are mainly two steps to execute application requests, first step is VM provisioning, which consists of generating VM instances for each application that satisfy its characteristics and requirements. The second step is to map and schedule requests to physical resources. Currently, most existing data centers can only support general VM classes for all types of workload, e.g. Amazon's EC2 supports only the five basic VM

types. In fact, a typical cloud computing can be mixed with heterogeneous applications, such as longtime running computationally intensive jobs, short-time sensitive requests, and data intensive analytic tasks. Due to the diversity and complexity of applications requests, a lack of understanding of application requirement is common nowadays, which is insufficient to diverse types of application requests.

- **Autonomous Operation**: Provision is just one of many autonomous features of a cloud server; it also monitors, verifies, and recovers any tasks from failed processors or storage units. These are done without any intervention of the end users.

- **Scheduling and Prioritization**: A cloud environment needs to schedule numerous tasks, and as a SaaS application software may serve multiple tenants, it also needs a separate scheduler to schedule tasks to various versions of the software for execution, and each version also needs to access the data stored in processor caches, memory, or storage devices.

- **Safety and Security**: The security issues on a cloud focus primarily on data confidentiality, data safety and data privacy. Also the malicious attacks performed in a cloud system, including web service security, Transport Layer Security. The data security can be addressed by some database authority control mechanisms, such as filter-based pattern at the application level, and permission-based pattern at the DBMS level. The system security can be addressed by some metadata on the service implementation modules, at least

for identification purposes.

To meet the requirements of on-demand, several challenges which are not addresses in the existing research, my thesis will discuss several critical research problems and propose solutions using real application scenarios. In details:

1. *Service Requests Prioritization*: service providers receive multiple requests from customers, how to prioritize those service requests to maximize the business values and minimize customers' dissatisfaction is one of the most important issues in cloud. An innovative prioritization model is proposed, which uses different types of information, including customer, service, environment and workflow information to optimize the performance of the system. The model is applied to a real end-to-end mortgage origination process and evaluate the performance of the model.

2. *Service Demand Forecasting*: most services experience seasonal or other periodic demand variation as well as some unexpected demand bursts due to external events. The only way to provide "on-demand" services, is to provision in advance. Accurate demand prediction and provision become critical for the successful of the cloud computing, which reduces the waste of utility purchase and can therefore save money using utility computing. An effective demand prediction model is proposed, and apply it to a real mortgage application.

3. *SaaS Customization*: to support a significant number of tenants, SaaS applications need be customizable to fulfill the various functional and quality

requirements of individual tenants. A unified and innovative multi-layered customization framework is proposed to support and manage the variability of SaaS applications and tenants-specific requirements. Ontology is used to derive customization and deployment information for tenants cross layers. This framework also has an intelligent recommendation engine to support new tenants to deploy using information from existing deployed SaaS applications. A case study in mortgage application is used to demonstrate the proposed model.

4. *Scalable and Robust SaaS*: The multi-tenancy architecture and customization requirements have brought up new issues in software, such as database design, database partition, scalability, recovery, and continuous testing. A hybrid test database design to support SaaS customization with two-layer database partitioning is proposed. Furthermore, constraints in metadata can be used either as test cases or policies to support SaaS continuous testing and policy enforcement.

5. *Secure SaaS*: security is an important issue due to the increase scale of users. Current approaches to access control on clouds do not scale well to multi-tenancy requirements because they are mostly based on individual user IDs at different granularity levels, however, the number of users can be enormous and causing significant overhead in managing security. RBAC (Role-Based Access Control) is attractive because the number of roles is significantly less, and users can be classified according to their roles. A RBAC model is proposed using a role ontology for Multi-Tenancy Architecture (MTA) in clouds. The

ontology is used as to build up the role hierarchy for a specific domain. An ontology transformation operations algorithms are provided to compare the similarity of different ontology. The proposed framework can easy the design of security system in cloud and reduce the complexity of system design and implementation.

6. *EasySaaS*: To support a significant number of tenants, an easy to use SaaS construction framework is highly desirable. An easy SaaS constructing architecture is proposed: an automatic SaaS construction framework. In the architecture, in stead of starting from scratch and customize applications, the tenant can publishing their requirements into the global SaaS platform in the form of application requirement and specification with their unique business requirements, as well as their expectation of the SaaS outcome and test scripts. The SaaS providers proposes their SaaS products, customize their services to meet tenant's requirements. This framework releases the workload of tenants, and provide an easier way to customize tenants' business requirement in a collaborative way. The SaaS providers also get benefits from the shared platform, and fast the development process. A hierarchy global index is used to support the matching and customization process.

The thesis is organized as follows. Chapter 2 discusses the related work, both in research community as well as the industry solutions. Chapter 3 proposes a prioritization model for service request considering different features; Chapter 4 proposes a demand-forecasting model in cloud ; Chapter 5 paper presents a unified and innova-

tive multi-layered customization framework, to support and manage the variability of SaaS applications and tenants-specific requirements; Chapter 6 proposes a hybrid two-layer database partitioning to support scalability in multi-tenancy SaaS; Chapter 7 proposes a novel SaaS testing framework that has considered three dimensions: the SaaS maturity level model, the platform support and the methodology; Chapter 8 proposes a RBAC model for SaaS security using a role ontology for Multi-Tenancy Architecture (MTA) in clouds. Chapter 9 proposes a novel SaaS construction framework. Chapter 10 summarize the thesis.

As a summary, this thesis discusses the most important research problems in cloud computing, towards effective and intelligent SaaS. The research in this thesis is critical to the development of cloud computing and provides fundamental solutions to those problems.

2. RELATED WORK

2.1. Compare with Service Oriented Architecture

SOA is a well know concept and has been interested to the research community for a long time. The first generation is some network-based service-oriented architectures, such as remote procedure calls (RPC), DCOM and Object Request Brokers (ORBs) based on the CORBA specifications. Furthermore, the "Grid Computing" architectures are emerged and many interesting solutions.

SOA is a system architecture in which a collection of loosely coupled services (components) communicate with each other using standard interfaces and message-exchanging protocols [60]. These services are autonomous and are platform independent. They can reside on different computers and use each other's services to achieve the desired goals. A new service can be composed at runtime based on services. Remote services can be searched and discovered through service brokers that publish services for public accesses. Web Services implement a Web-based SOA and a set of enabling standardized protocols such as XML, WSDL, UDDI, and SOAP. SOA has the following unique features, including dynamic composition and discovery, standard-based interoperability and dynamic control and scheduling.

SOA splits the developers into three independent but collaborative entities: the application builders (also called service requestors), the service brokers (or publishers), and the service developers (or providers). The responsibility of the service developers is to develop software services that are loosely coupled. The service brokers publish or market the available services. The application builders find the available services through service brokers and use the services to develop new appli-

cations. The application development is done via discovery and composition rather than traditional design and coding.

SOA can have other variations. Two such variants are Consumer-Centric SOA (CCSOA) [157] and User-Centric SOA (UCSOA)[26]. In addition to publishing service specifications in SOA, CCSOA allow publication of workflows and application collaboration specifications for discovery, matching, and subscription. Conventional SOA supports mainly service producers, i.e., service providers develop and publish their services, and service consumers (application builders) are responsible to discover the right services published as well as use the services in composition. In CCSOA, consumers publish their application requirements together with associated service specifications including the workflow. Once these are published as application templates, service providers can submit their software or services to meet the requirements. This way of computing is consumer-centric because now the service providers will look for application needs from service customers. UCSOA is an extension of CCSOA that allow end users to compose application rapidly.

Workflows are prevalent in diverse applications, which can be scientific experiments, business processes, web services, or recipes. A workflow can be represented by a directed graph of data flows that connect loosely and tightly coupled (and often asynchronous) processing components.

As the number and complexity of workflows are dramatically growing, effective workflow management is attracting more and more attention in the database community Kepler, Triana and Taverna [112] allow users to simulate, and search workflow tasks using keywords and regular expressions, respectively.

Traditional packaged software	Software as a service (SaaS)
Designed for customers to **install, manage and maintain**	Designed for delivery as **Internet-based services**
Architect solutions to be run by an **individual company** in a dedicated instantiation of the software	Designed to run **thousands of different customers** on a single code
Infrequent, major upgrades, sold individually to each installed base customer.	**Frequent small upgrades** to minimize customer disruption and enhance satisfaction.
Version control, **upgrade fee**	Fixing a problem for one customer fixes it for **everyone**

Fig. 5. Comparison of Business Models

When coming to "cloud computing", we have to consider more, e.g. how to use the underlying infrastructure to support the workflow application in the world, the "on-demand" requirement. More specifically, how to access each individual and aggregated resources, group resources from different "clouds" together and etc.

Comparisons of traditional software and SaaS [83] as shown in Table 5:

2.2. Research Community in Multi-tenancy SaaS

2.2.1. Data Tier Design for Multi-Tenancy Architecture

Resource isolation is important for multi-tenancy architecture, and solutions can be from a completely isolated design to a totally shared design. As stated in [5]:

- Separate databases (SD): In this scheme, as shown in Figure 6(a), each tenant has its own database separated from other databases, thus a completely isolated solution. Note that while data are isolated, computing resources such as CPUs, storage, and application code may be still shared among tenants. As

each database is isolated from other databases, it is relatively easy to extend the data model for each client as in conventional systems, and to recover from failures using conventional techniques. This solution is straightforward but expensive, as separate databases need to be maintained including backup. As each tenant needs a separate database, if a SaaS application has many tenants, the number of databases will be large, and this adds significant cost and overhead for the cloud infrastructure. Thus, the scalability of this approach is limited.

- Shared databases but separate schemas (SDSS): In this scheme, multiple tenants store individual data tables in the same database, as shown in Figure 6 (b), but each tenant has its own database schema and data tables. When a new tenant comes, the system creates a discrete set of tables and schemas for it using classic SQL statements. This scheme is easy to implement, easy to extend. If a tenant likes to change its database model, other tenants will not be affected as they have separate tables and schemas.

- Shared database and shared schemas (SDSHS): In this scheme, multiple clients share the same database and they also share their data in the same sets of tables, as shown in Figure 6 (c). This is a flexible approach as only one set of schema will serve all tenants, and the system will maintain only one database. While this is convenient for database design and maintenance, a system fault can corrupt the entire database, or release the data to a different tenant. Considering a mortgage database shared by multiple tenants, the Tenant ID

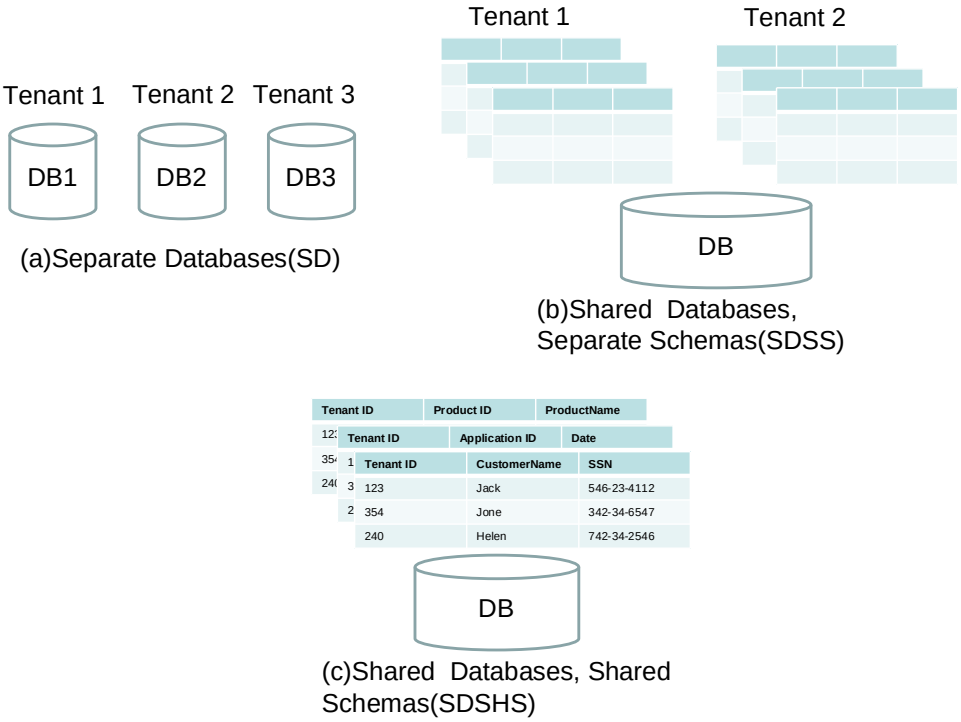


Fig. 6. Design Patterns of Data Tier Multi-Tenancy Architecture

is the key to represent every tenant and used as a foreign key to associated different tables. It has low hardware and backup cost as this allows a large number of tenants to be served by the same database server. However, this approach will be most complex due to security reasons. This scheme is also difficult to restore from failures as the entire database need to be recovered and restored to the previous state, not just the failed parts as the system may not know exactly which parts in data tables have failed.

From the database perspective, a multi-tenant database system needs to offer schemas that are flexible in two respects:

1. It should be possible to extend the base schema to support multiple specialized versions of the application; and

2. It should be possible to change or evolve the base schema dynamically, and the extension should be done while the database is on line. In other words, the evolution should be done without the involvement of the service provider to maintain availability.

Mapping multiple single-tenant logical schemas in the application to one multi-tenant physical schema in the database is not easy because enterprise applications often allow each tenant to extend its base schema. As discussed in [8], seven techniques are available to design flexible schemas for SaaS applications, and they can be further classified into two categories:

- The database "owns" the schema (and it is explicitly defined in DDL or data definition language); and
- The application "owns" the schema (mapped into generic structures in the database).

This section uses an example to illustrate these techniques. A SaaS application with three tenants 1, 2, and 3, each has an Account table with Account ID (Aid) and Name fields. Tenant 1 has extended the Account table with two fields for the hotel industry: Hotel and Rooms. Tenant 3 has extended the Account table with one field for the mortgage industry: Brokers. Tenants share the tables using a tenant ID column (Tenant).

Category 1: Databases-own schemas

User view: Database views can be used to the shared database tables that include

only the data for a specific tenant. When a new tenant comes, one can create a new view for the tenant.

Private tables: Each tenant has its own private instances of the base tables that can be extended as required as shown in Figure 7 (a). In contrast, in all of the other mappings, tenants share tables.

Extension tables: The extensions are vertically partitioned into separate tables that are joined to the base tables along a row ID column. This is shown in Figure 7 (b).

Sparse columns: Every extension field of every tenant is added to its associated base table as a sparse column. This is shown in Figure Figure 7 (c).

These techniques in general perform well, but they provide limited support for schema evolution in the presence of existing data. Moreover they do not scale beyond a certain level.

Category 2: Applications-own schemas

XML: Each base table is augmented by a column that stores all extension fields for a tenant in a flat XML document. As these documents may vary by tenants, they are un-typed. This is shown in Figure 8 (a).

Pivot tables: Each value is stored along with an identifier for its column in a tall narrow table. This is shown in Figure 8 (b).

Application-own-schemas gives the application complete control over schema evolution, however it suffers significantly in terms of performance. Specifically, for

Account1

Aid	Name	Hotel	Rooms
1	Michel	Holiday Inn	135
2	Garry	Super 8	1012

Account2

Aid	Name
1	Bill

Account3

Aid	Name	Brokers
1	Smith	65

(a) Private Tables

Account ext

Tenant	Row	Aid	Name
1	0	1	Michael
1	1	2	Garry
2	0	1	Bill
3	0	1	Smith

Hotel account

Tenant	Row	Hotel	Rooms
1	0	Holiday Inn	135
1	1	Super 8	1012

Broker account

Tenant	Row	Brokers
3	0	65

(b) Extension Tables

Account

Tenant	Aid	Name	Sparse
1	1	Michael	Hotel Holiday Inn Room 135
1	2	Garry	Hotel Super 8 Room 1012
2	1	Bill	
3	1	Smith	Broker 65

(c) Sparse Columns

Fig. 7. Databases-Own Schemas

the XML-based solution, it is necessary to parse the XML documents and then re-assemble the rows to processes those fields in the extension. The performance degradation is proportional to the number of extension fields. For pivot-tables-based solution, the performance degradation may be more than an order of magnitude due to the complexity of processing large tables.

2.2.2. Security

Security mechanisms prevent a tenant from getting the privileges to access data belonging to other tenants. The goal is to have comparable security assurance for multi-tenancy applications as traditional software applications. In general, there are two approaches to realize data security.

Account

Tenant	Aid	Name	Sparse
1	1	Michael	<ext> <hotel> Holiday Inn</hotel> <room> 135</room> </ext>
1	2	Garry	<ext> <hotel> Super 8 </hotel> <room> 1012</room> </ext>
2	1	Bill	
3	1	Smith	<ext> <brokers> 65 </brokers> </ext>

(a) XML

RowKey		Name	Contact
1	Act1	[name: Michael, Hotel: Holiday Inn, room:135]	
1	Act2	[name: Garry, Hotel: Super 8, room:1012]	
1	Ctc1		[……]
1	Ctc2		[……]
2	Act1	[name: Bill]	
2	Ctc1		[…….]
3	Act1	[name: Smith, brokers:65]	

(b) Pivot Table

Fig. 8. Application-Own Schemas

Filter-based approach at the application level: For the SD or SDSS approaches, one can use database name or schema name to control the access of the corresponding tenants. For SDSHS approach, the filter is based on the tenant ID column in every table to access records associated with the corresponding tenants It is easy to implement, but leave some opportunity for malicious access, for example, a hacker can use 'tenantID=X or 1==1' to access data of all tenants. A sample SQL statement is shown as follows:

```
SELECT *
```

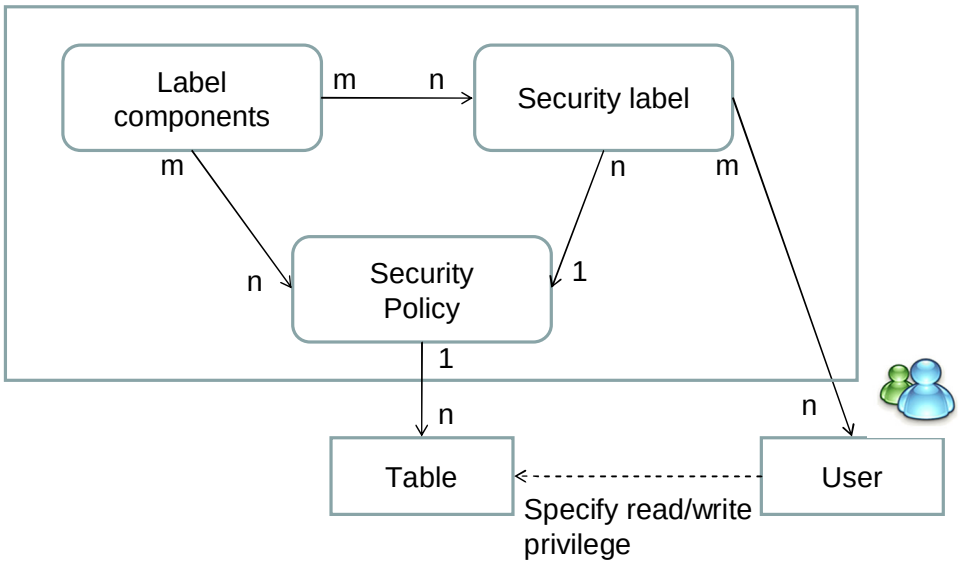


Fig. 9. Entity Relationship graph for LBAC

```
FROM TENANTS_TABLE

WHERE TenantID=2 or 1==1
```

Permission-based approach at the DBMS level: Each tenant is assigned a dedicated DB access account and each has privileges to access its own data only, for example, for SD and SDSS approaches. For SDSHS, one needs to leverage the row-level access control mechanism provided by DBMS, e.g. Label-Based Access Control (LBAC). For example, assume that a tenant has SELECT privilege on an application table for its own data, when the tenant executes a SELECT statement, the Label Security evaluates each row selected and determines whether the tenant can access it based on the privileges and access labels assigned to the tenant by the security administrator. Similarly, Label Security can perform security checks on UPDATE, DELETE, and INSERT statements. In this way, it can prevent potential SQL injection attacks.

Figure 9 shows an ER graph of LBAC. Three types of security labels can be granted to three types of database objects, including row, column and user respectively. Furthermore, a security label can be composed of several security-label components. DB2 V9 is an example system that provides this kind of support, and it uses DB2SECURITYLABEL to hold the security label and attach the security policy to the table.

One can use the LBAC rule set, that is a predefined set of rules, to compare security labels. When comparing values of two security labels, one or more rules in the rule set can be used to determine whether one blocks another. For example, in DB2 V9, there is a single rule set named DB2LBACRULES, there are 16 pre-built security label components in it. In Figure 10, a sample security policy with name "SecurityPolicy_Customer" is generated for the shared table CUSTOMER_ORDER, with a set of security labels inside. There is a column named DB2SECURITYLABEL in the CUSTOMER_ORDER table that can associate each tenant with their own data. Each security label includes one element selected from one of the 16 labels components.

When a new tenant comes in, the operator can simply select one unused label, and grant it to the tenant using the sample SQL statement as follows:

```
GRANT SECURITY LABEL SecurityPolicy_Customer.0001 to USER TenantA for
all access
```

The advantage of LBAC is that it controls cross-tenant data access at the DBMS level instead of the application level. However, it has limitation , for example in DB2, at most 16 security label and 64 elements can be supported, thus the

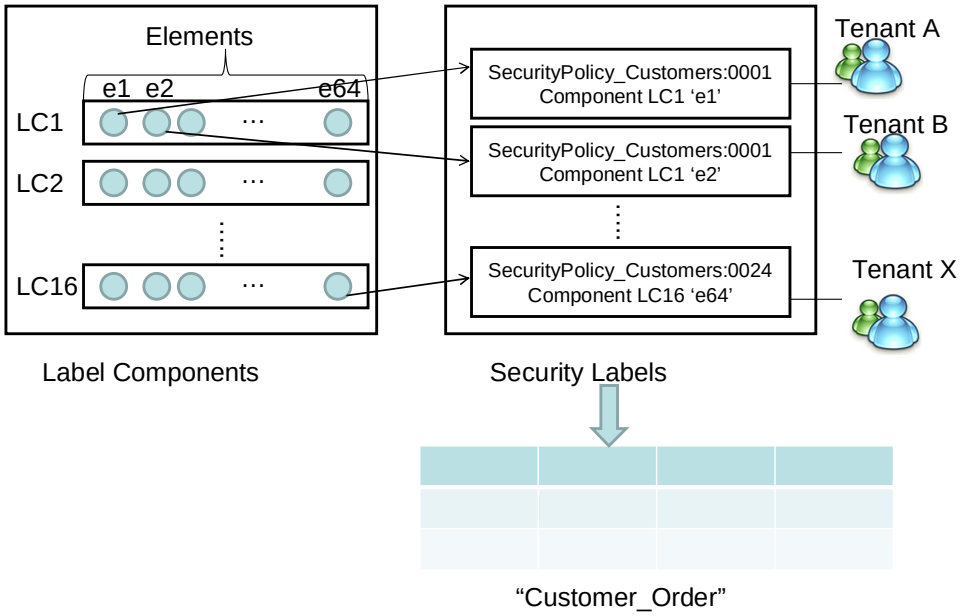


Fig. 10. An Example of LBAC

maximum number of tenants is 1024(16*24). For cloud computing, the number may be too small.

2.2.3. Scalability

Scalability is an important feature of multi-tenancy. With an increased workload, the resources needed should be proportional to the increase in the workload to maintain the system performance. Scalability has two kinds:

- Scale-up or vertical scaling: This is done by adding additional resources, such as CPUs, memory, and disks into a single node in a clustered system. In this way, a node becomes more powerful by having more resources; and
- Scale-out or horizontal scaling: This is done by adding additional nodes (processors) to an existing clustered system. For example, instead of a cluster of thirty nodes, the system may have fifty nodes instead.

The scale-up is easy to use but may not provide linear scalability increase due to the overhead in resource management. The scale-out provides a more cost-effective way, where it can incrementally extend the system by adding more resources to a low-cost hardware set. Furthermore, it can improve the reliability and availability of the system due to the redundancy.

S1: Database Partitioning

In the scale-up scenario, one can create more than one database partition on the same physical machine, while in the scale-out scenario, partitions can be created in multiple physical machines, and each partition has its own memory, CPUs, and storage devices. The data inside a database can be distributed across several partitions. A distribution key is column used to determine the partition in which a particular row is stored.

There are two types of database partitioning approaches: application-based distribution keys and tenant-used distribution keys.

1. Application-based distribution keys: This is a traditional way to partition a database, and this is done by choosing one or more attributes as a distribution key, according to the domain knowledge. For example, "REGION" attribute can be used as a distribution key. However, this scheme needs a good distribution key to balance the load among multiple partitions, and this need data profiling information. This solution also needs to be upgraded to consider the multi-tenancy architecture, and address the related isolation issues.

2. Tenant-based distribution keys: This scheme stores each tenant's data in a

single partition. It can use TenantID as the distribution key. Using this method, one can map the tenant with any specific partition freely by assigning or change the TenantID. As a tenant-aware method, it provides better isolation than the previous one, as well as the availability. Furthermore, it is possible to develop load balancing algorithms to ensure the most partitions have similar loads.

S2: Table partitioning

Table partitioning provides a way to create a table in which ranged of data are stored separately. The advantage of this partition is to improve the query performance and facilitate the table change. For example, one can use ALTER TABLE statement to modify a table. This requires the users have a clear understanding for the table storage information, thus this is for advanced users only.

2.3. Saleforce.com

Salesforce.com [46] identifies itself as the enterprise SaaS company. It has been providing customer relationship management (CRM) using a SaaS model since 2000. As of 2010, they have over 70,000 paying customers.

The Saleforce.com has developed a set of SaaS applications, platform and infrastructure that demonstrate cloud principles of resource scalability and multi-tenancy. The key features of their architecture are a meta-driven database schema and application, a partitioning scheme for their database, and an integrated application development framework. Salesforce.com's architecture is a layered as shown in Figure 11.

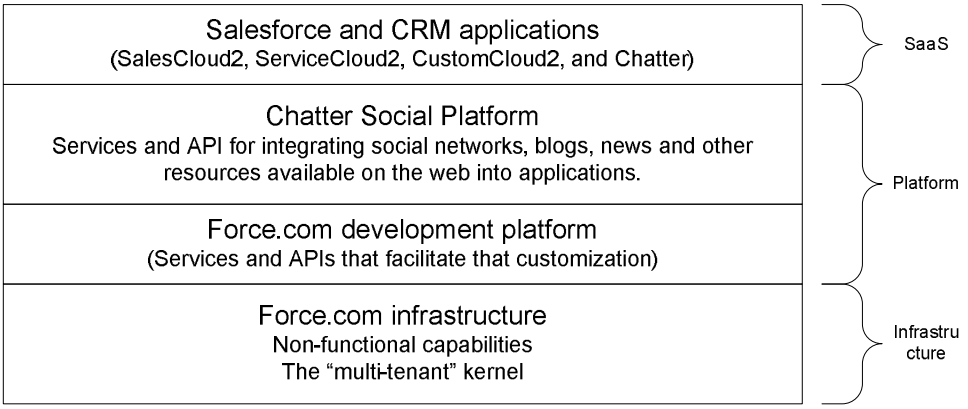


Fig. 11. SalesForce.com Layered Architecture

Salesforce.com architecture layer	**Description**
Salesforce apps	The salesforce.com applications. In Mar 2010, they list 4 apps, SalesCloud2, ServiceCloud2, CustomCloud2, and Chatter.
Chatter Platform	A collection of services and APIs that facilitate the integration of web "chatter" from social networking web sites (such as facebook, LinkedIn.com, and twitter) news outlets and blogs into applications related to sales force and customer relationship management.
Force.com development platform	Tools for creating applications, and a platform for hosting applications.
Force.com infrastructure platform	Services that the platform uses. The infrastructure consists of capabilities that support non-functional aspects of applications, such as performance, optimization and security, and a *multi-tenant kernel*.

Fig. 12. Salesforce.com Architectural Layers

The roles of each of the architectural layers is shown in Table 12. Salesforce.com supports multi-tenancy. Multi-tenancy is the ability for different customers to use the same basic application, while at the same time not affecting each other. This multi-tenancy is manifested at all layers in their architecture.

The SaaS layer of salesforce.com consists of its applications. The applications provide direct benefits to Salesforce.com tenants (customers). For example, the SalesCloud2 application provides capabilities for a sales organization to manage

information about its customers and sales agents. The process that a sales organization would take to use salesforce.com (and become a tenant) involves a setup phase and a usage phase. In the setup phase, the organization comes to an agreement about the level of service they need, establish an account with salesforce.com, configure information about their organization, create accounts for people who will be using the system, and train their users. Once set up, the new tenant can begin using the sales force services.

The platform that Salesforce.com runs on contains two layers, a Chatter social platform and a "development platform" layer. The "chatter" layer is a collection of services and APIs that facilitate application access and usage of information from web sites outside of the salesforce.com cloud. Specifically, the chatter layer facilitates access to social networking web sites (facebook and twitter), google.com, RSS feeds (from news organizations and blogs), and user groups. One motivating factor for this layer is that it supports functions to monitor what people are publishing about an organizations products and service. This information can be used to help a business stay competitive.

The other part of the salesforce.com platform layer is the Force.com development platform. One role of this layer is that of being the host platform for the tenant-customized applications. But, in addition, force.com is a development environment that tenants can use to configure, customize and extend their applications.

The infrastructure layer of the salesforce.com architecture consists of a multi-tenant kernel and a collection of capabilities that support non-functional aspects of the system, such as security, reliability, scalability, and optimization.

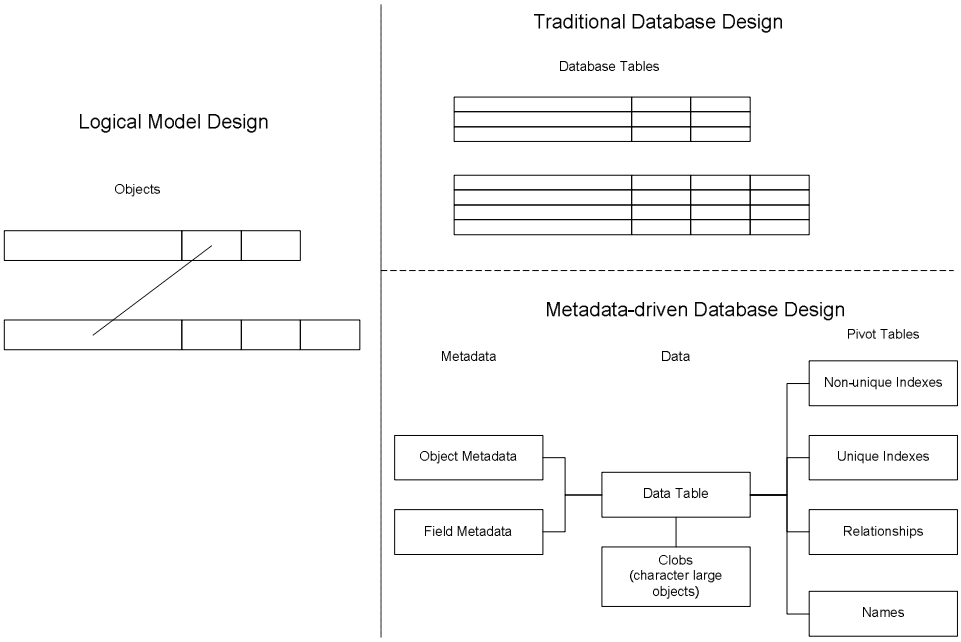


Fig. 13. Comparison between Data-driven and Metadata-driven Databases

One of the key attributes of the salesforce.com architecture is the way it supports multiple tenants. It uses a metadata-driven database. Figure 82 contrasts a traditional database with a metadata-driven database. In traditional database design, objects and fields are defined that represent abstractions of the real-world entities that they represent. Separate database tables are created for each type of object represented. Specific attributes are represented by fields within the tables. Object instances are represented by rows within the tables. Actual data is placed into a database by inserting rows into the database tables. Relationships are represented by fields in one table referring to a key field in another table.

Metadata-driven databases operate somewhat differently. Objects and their fields are mapped to metadata tables. Actual data is stored in either in a single data table, or, for large text objects such as documents, in a separate character large

object storage (Clobs) area. A series of pivot tables is created to make accessing the data within the single data table more efficient.

To support multiple tenants, the object and field metadata contains information about the fields, and also about the tenants. Table 14 describes the schema.

This information is represented as an organization ID of the tenant that defined the objects and fields. Specifically, the object metadata table contains a unique object ID (the key field), the object name, and the organization ID of the tenant. The field metadata table contains a unique field ID (the key field), the field name, the data type, the object ID of the associated object, whether this is an index needs to be created for this field, the sequence number of this field in it associated object, and the organization ID of the tenant that created the field.

The Force.com platform used 4 pivot tables - an Indexes Pivot Table, a Unique Fields Pivot Table, a Relationship Pivot Table, and a Names Pivot Table. The Indexes and Unique Fields pivot tables provide ready access to field values for indexing and uniqueness checks, respectively. The indexes tables identify the fields used as indexes. The unique fields table provides identifies the fields in the database that have uniqueness constraints. The Relationship table identifies objects and their relationships, and is used to implement join operations. The Name table is use to efficiently look up an object name from its ID.

One potential drawback to this type architecture is that all the tenants share the metadata and data tables, so the database can become a bottleneck. Force.com solves this by partitioning up the tables. Each tenant has its own set of tables. This partitioning keeps the database from being a choke point and is what permits their

Table	Field	Description
Object Metadata	#ObjID	A unique identifier, primary key field for object metadata
	OrgID	ID of the tenant that defined this object.
	ObjName	Name of the object.
Field Metadata	#FieldID	A unique identifier, the primary key for metadata table.
	OrgID	The ID of the tenant that defined this field.
	ObjID	The unique ID of the object that contains this field.
	FieldName	The name of the field.
	Datatype	The datatype of the field.
	FieldNum	The relative sequence of this field as compared to other fields in the associated object.
	IsIndexed	A Boolean value representing whether an index needs to be created for this field.
Data	#GUID	A unique data identifier.,the primary key for the datum.
	OrgID	The ID of the tenant organization that created this datum.
	ObjID	The ID of the object this datum is associated with.
	Name	The natural name of the object.
	Value 0 … Value500	Values of the fields. Each value is mapped to a field as specified by the FieldNum value in the Field Metadata table.
Indexes Pivot	#OrgID	The ID of the tenant organization.
	#ObjID	The ID of the object.
	#FieldNum	The field sequence number of this index field.
	GUID	The ID of the datum
	StringValue	If the field type is String, contains the string value
	NumValue	If the field type is numeric, contains the numeric value
	DateValue	If the field type is a date, contains the date value
Relationship Pivot	OrgID	The ID of the tenant organization.
	ObjID	The ID of the object.
	GUID	The ID of the datum.
	RelationID	The ID of the relationship
	TargetObjID	The ID of the target object.

Fig. 14. Description of the schema for a metadata-driven multi-tenant database

solution t scale to the size it has currently grown to.

One last aspect is the development platform. Salesforce.com offers a development platform that allows tenants to customize and design their own applications. One tool they provide is a web-based application that allows tenants to define objects and fields, and applications that use them. The tool -essentially provides an easy to learn and easy to use front end to the metadata tables. The metadata-driven database schema is a key enabler for this development platform. Salesforce.com also provides other means for creating custom extensions. It has Apex, its own scripting language, an Eclipse plug in, and a set of web services that tenants can use to create their own custom applications and extensions based on the Force.com platform.

The Saleforce.com architecture provides an excellent look at issues that confront cloud computer providers. The meta-driven database and the partitioning scheme combine to provide a platform for multi-tenancy that is scalable. The metadata approach also enables the creation of a web-based development that all tenants can use to create their very own variations of the application.

2.4. Oracle's On Demand SaaS Platform

The Oracle SaaS platform consists of four major components, includes virtualization, middleware, database and systems management. Figure 15 shows Oracle SaaS Platform Architecture.

The components of Oracle SaaS Platform work with each other and provide better support for advanced applications. Also each component can be picked up by service provider to meet their specific requirements.

The services in the platform can be classified as follows:

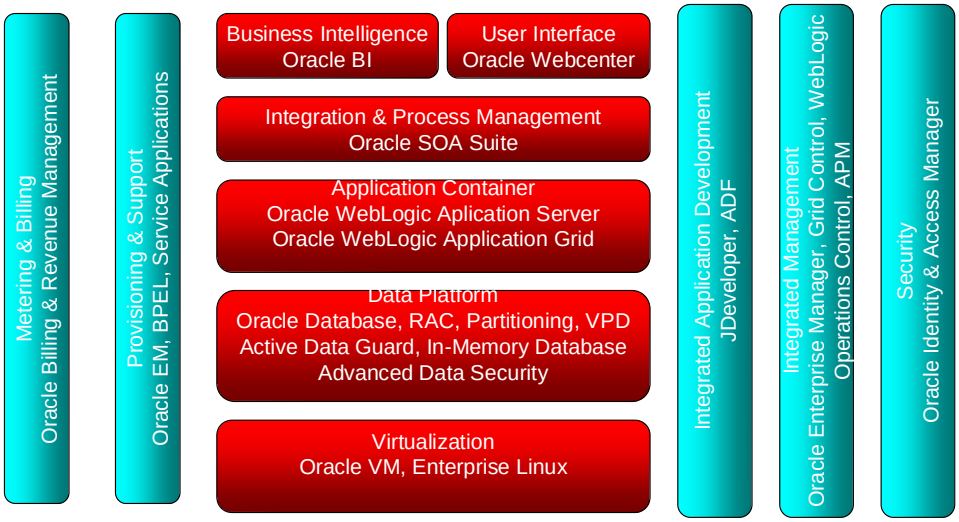


Fig. 15. Oracle SaaS Platform Architecture

- Application services: Application and data architecture, Metadata management and customization for various applications; Application modeling, instantiation, and packing

- Platform services: runtime engine, tenant aware container and VM ; management of resource services, policies, configuration of runtime; monitoring of resource consumption; Security implementation, isolation of tenant data and runtime

- Platform resource services: computer cycle on nodes; Clustering; Data storage, database abstraction, Network resources, IP end points

2.4.1. Database with Scalability

The database layer design is a significant challenge in the whole lifecycle of SaaS development, from design phase, implementation phase and deployment phase. The ISV(Independent Software Vendor)s have to design the database choice: share every-

thing with multi-tenancy data or shored nothing separate database or other choices between these two.

The decision of database multi-tenancy design depends on many factors, such as customization requirement, data privacy and security requirement, business model requirement, scalability requirement and service level agreement.

Oracle Database provides the ideal platform for multi-tenancy due to its support for grid computing: virtualization and provisioning.

- Virtualization: breaking hard-coded connections between providers and consumers of resources, and supplying a resource to consumers without knowing how to accomplished. With virtualization, individual resources (e.g. computers, disks, application components and information sources) are pooled together by type which are available for consumers.

- Provisioning: the system determines how to meet the specific need of the consumer, while optimizing the whole system's performance. With provisioning, when consumers request resources through a virtualization layer, a specific resource is identified to satisfy the request and allocated resources to the consumer at the backend.

The key capabilities of Oracle Database for SaaS include:

- Oracle Real Application Clusters for database grid

- Automated Storage Management for storage grid

- Active DataGuard for disaster recovery

- Partitioning for performance and manageability
- Tablespaces for ease of data management
- In-Memory Database Cache
- Virtual Private Database for data privacy

2.5. Google's App Engine

Google targeted exclusively at traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. AppEngine's impressive automatic scaling and high-availability mechanisms, and the proprietary MegaStore data storage available to AppEngine applications, all rely on these constraints.

2.5.1. Google App Engine

GAE is [43] a platform for developing and hosting web applications in Google managed data centers. It provides a seemingly unlimited computing resource, and visualizes applications across multiple servers and data centers. GAE's infrastructure allows its hosted web applications to scale easily, and frees developers from hardware configuration and many other troublesome system administration tasks. GAE handles deploying code to a cluster, monitoring, failover, and launching application instances as necessary. GAE is designed to be language neutral, however, it currently supports only Python, and Java, and JVM compatible languages.

Figure 17 shows a simplified java version of GAE application architecture[161]. Currently, GAE supports JDK 1.5 and 1.6. Google provides an eclipse plug-in to

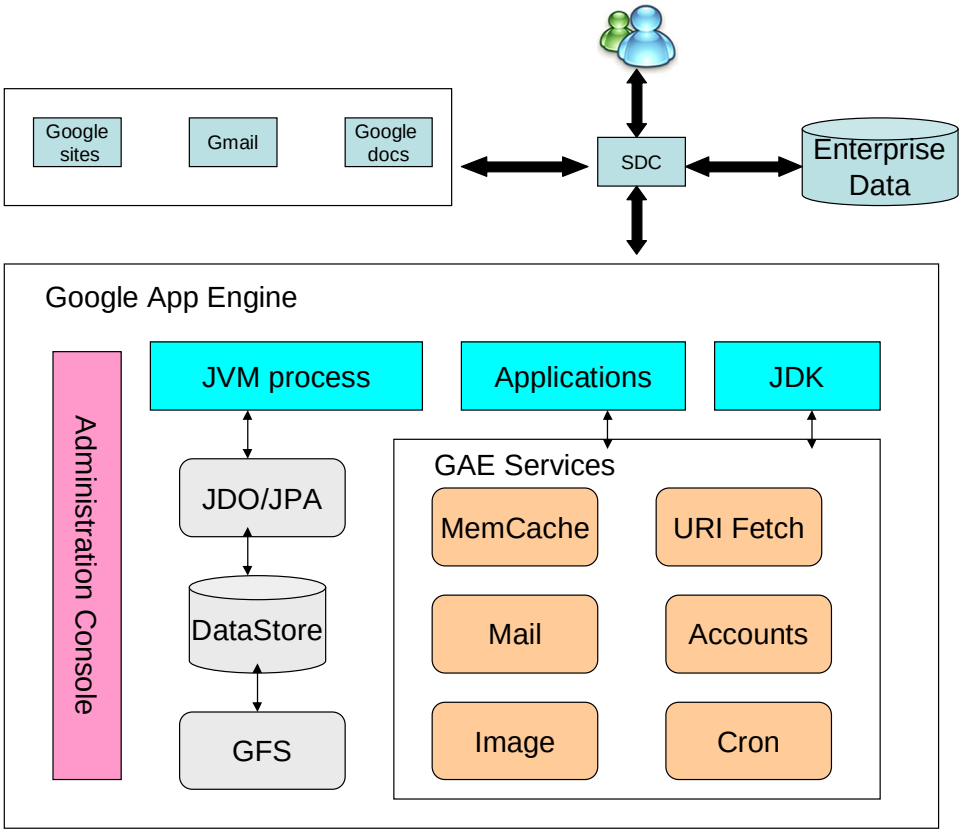


Fig. 16. Simplified Java GAE Application Architecture

assist developers to create, develop and deploy GAE powered applications. The compiled applications are deployed on a read-only file system on a cluster of the Google cloud. Any persistent data need to be written to a database-like service called Datastore via JDO/JPA interfaces. GAE provides an admin console dashboard that allows the application administrators to create GAE applications, monitor their health, and check the usage/quota information. Through Secure Data Connector (SDC), GAE allows its apps to retrieve from and write data to their corporate databases, enabling easy extension from enterprise systems into Google Apps.

2.5.1.1. *Services*

Google App Engine exposes several services via Java API which are listed as below:

- URL Fetch Service: It allows applications to access resources and communicate with other hosts over the internet using HTTP and HTTPS requests.

- Mail Service: Through GAE's mail service, applications can send email messages using Google infrastructure.

- Memcache Service: The memcache service is a distributed in-memory data cache accessible by multiple instances of applications.

- Image Manipulation Service: The service provides API that allows its users to manipulate images, such as resize, rotate, rotate, compress and flip images.

- Task Scheduling (Cron) Service: Task scheduling service, also known as "cron service", enable an application to perform tasks outside of responding to web requests at defined times or regular intervals. In another word, applications can create and run background tasks itself while handing web requests.

- Google Accounts: GAE integrates its applications with Google Accounts for user authentication, which saves developers the effort of implementing an individual user account system, and enables the users to start using the applications faster if they have a google account already.

2.5.1.2. *Data Store*

GAE uses a data storage service, named datastore, to store and query data. Datastore is a schemaless object datastore with a query engine and atomic transactions. The service can be accessed via JDO and JPA implemented by the open source Data Nucleus Access platform.

Datastore is built on top of BigTable, which is built on top of Google Filesystem (GFS), therefore it scales well as the data grow, and has high availability and reliability.

While GAE datastore possesses some similarities to traditional relational database, it is not a traditional relational database. First, datastore is "schemaless." The structure of data entities is enforced by your application code. Second, compared to a traditional relational database, certain limitations exist for datastore, such as no "join" queries, no native many-to-many relationship supports, no grouping, and other aggregate queries.

GAE datastore is a hierarchical database in the sense that entities form distinct strict hierarchies within a datastore [13]. An entity either has a parent or no parent (in this case, the entity is a root entity). A root entity and all of its descendants form a cluster of entities known as an entity group. Every entity in the datastore has a key, an identifier unique to the entity across all entities for an application. A local key of an entity is a combination of its kind and its ID, which is either a name assigned by the application or a numeric ID assigned by the datastore. For the root entity, its key is its local key. The key for a descendant entity is a path consisting of the local keys of its ancestors from the root to the entity itself.

GAE's datastore's underlying storage structure used is BigTable. The operations BigTable supports include: Read, Write, Delete, Single Row Transaction, Prefix Scan and Range Scan. To utilize BigTable, GAE datastore stores entities in a distributed fashion by assigning ranges of entities sorted by the key to different BigTable servers. Therefore, datastore can also be thought as a shared, sorted

array[13]. Members of an entity group stay in the same BigTable server at any given time. A transaction involving entities in the same entity group can be managed by a single server and implemented simply and efficiently. Currently, GAE only supports transactions on one entity group.

Datastore is strong consistent and uses optimistic concurrency control. An update on entity occurs in a transaction, which means it either succeeds or fails. The transaction operates at the root level of the entity group. When there are multiple requests to update entities within the same entity group, a contention occurs. The winner will perform the update, and the other requests will be retried a fixed number of times before a timeout. This yields to a better throughput than a locking concurrency control algorithm according[13].

Datastore by default creates indexes on the kind and all the properties. Composite indexes on multiple properties can be created by configuration files specified by the users. All the indexes are also implemented using BigTable[13].

Compared to traditional relational database, certain limitations exist for datastore, such as no “join” queries, no native many-to-many relationship supports, no grouping and other aggregate queries, etc.

2.5.1.3. *Constraints*

For security reasons, GAE apps run in a restricted ”sandbox” environment, therefore, certain constraints apply:

- GAE apps cannot write to the filesystem. However, reading from the filesystem is allowed. GAE apps must use datastore service to persist data.

- No socket or direct access to another host is supported. HTTP/HTTPS requests should be made through GAE URL fetch service.
- No sub-process or thread is allowed. A web request must be handled in a single process.
- Response time limit. The handling of a web request has to be finished in 30 seconds, otherwise, it will be terminated.
- No system calls.

2.5.2. Google File System (GFS)

GFS is a file system designed to support searching and Web crawling. It is probably the largest file system in the world that is in operation. As Google depends on efficient file access for searching and crawling, the file system needs to be highly efficient and scalable. Google has reported that the number of users used Google has grown significantly over time, and sometimes at a rapid rate. GFS has been a critical technology for Google to provide timely services to their customers, and significant engineering and re-engineering effort has been spent on GFS. According to Quilan[93], a Google engineer, GFS has been under significant design changes to meet the ever changing environment.

An early GFS design is presented in [51], and subsequently many changes have been made. All the files are divided into chunks of 64 MB, and in most cases, files are appended at the end or read by users, and thus the GFS is designed to support these operations. GFS also sits on top of the Google infrastructure with thousands of inexpensive processors.

GFS has made several key design consideration and decisions early:

- The system structure should be simple to save implementation effort;
- The system will use thousands or even hundreds of thousand processors to support scalable operations;
- There will be no caching as the data will be huge and many of data will be streaming, and those streaming data do not need to be cached;
- The inexpensive processors used may fail, but the system should mask these failures by providing at least three replicas. More replicas can be made to increase system availability;
- It should provide familiar interfaces and APIs, and those APIs should be designed to support common Google operations such as snapshot and record append.
- Initial design is high throughput instead of late latency. Note that high throughput and small latency inherently conflict with each other. According to the Queuing theory [81], a system with high throughput often experience high latency, and a system with low latency will have a low throughput.

The first consideration leads to a simple design that allows Google to deploy the software to the market to capture the market share. However, as more people use Google for searching and crawling, user experience became a critical issue, and one key user experience is low latency. Thus, GFS was later changed to address the latency issue.

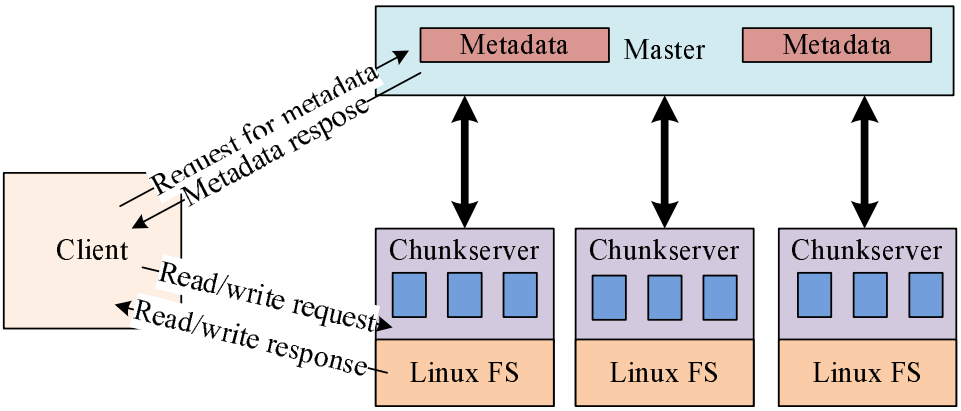


Fig. 17. Google File System (GFS) Architecture

2.5.2.1. *GFS System Structure and Operations*

The system has two kinds of node, a master node that manages metadata, and chunksevers that provide data storage; Data transfers happen directly between clients/chunkservers; Files broken into chunks (typically 64 MB);

The in-memory master server stores and manages the metadata associated with the chunks:

1. Tables that map files to chunk location (64-bit) addresses;
2. Tables that map files to their replica;
3. Table of processes that read and write a specific chunk;
4. Operation logs to support file persistency, information such as those used in file replication in case of chunkserver failures, checkpoints for recovery and so on.

The master server receives periodic updates from chunkservers to update its table entries. It is also responsible to create, replicate and re-balance chunks. Chunks

need to be re-balanced to improve space utilization and access speed. The master is also responsible for garbage collection.

Chunkservers store the data files, with each individual file broken up into fixed size chunks of 64 MB. Each chunk has a unique 64-bit label, and logical mappings of files to constituent chunks are maintained. Each chunk also has a 64-bit checksum to detect system failures. Each chunk is replicated at least three times in the system, but high-demand files may be replicated more often. Chunkservers run Linux and it also maintain data consistency.

If a process needs to access a specific chunk, it needs to obtain a permission from the master server first. After examining if the specific chunk is available, that is, no other processes are using it, the master server will grant a permission for a process to access a specific chunkserver for a period of time by supplying the chuckserver address. During that time period, the process can access the chunkserver, and no other processes can access the same chunkserver to maintain the system consistency. The process will interact with the chunkserver directly without any further interference from the master server. If the chunk is modified by the process, the modification is propagated to replicated chunkservers, and the system will not commit until it hears acknowledgements back from all the replicated chunkservers. This is administrated by the primary chunkserver.

Earlier, GFS used at Google has more than 200 clusters, with more than 5000 machine, 5+ PB filesystems, 40 GB/S read/write load in single cluster in the presence of frequent hardware failures. As more people use Google, the GFS has grown significantly.

Note that GFS is not implemented as a part of the kernel of an OS, and many of its operations are available to users directly. This is to increase system performance as any file system access will not need to go through layers of OS function calls. Traditionally a system call such as a process creation in an OS is an expensive operation.

2.5.2.2. *Lessons Learned for Developing GFS*

One important discussion has been centered on the master node. From the beginning, Google knew that the master node will be the single point of failure for the system, however, Google still decided to implement this way and it was the first decision made. This decision was made so that GFS can be quickly implemented and deployed to the market. With respect to this criterion, Google made the right decision.

However, as more people use Google, many shortcomings became clear. Specifically, the file size increases from a few hundred terabytes to petabytes, and then to tens of petabytes, the amount of metadata increase significantly at the master node and so is the workload of the master node. The use of parallel MapReduce does not help either, as thousands or hundreds of thousand MapReduce operations may request file accesses at the same time, the workload of the master node increases dramatically. Note that as the master node needs to stay in the memory all the time, and the amount of metadata information grows significantly over time.

The single point of failure is also bad for specific applications such as video serving. Also, initially, the GFS has no automated recovery plan for the master node, and thus when it fails, it may take a long time for the master node to be

recovered. Google's solution is to make shadow masters where snapshots of the master node status will be stored for easy recovery.

Furthermore, the initial design goal is high throughput with latency the secondary consideration, and later low latency is also emphasized. From the queuing theory, throughput often conflicts with delay, a high-throughput system favors the server but its customer may experience long delays, while a low-latency system favors the customers as the system needs to supply extra capabilities to server the customers quickly. The way to address the low latency is to replicate many operations to ensure that any failures can be easily masked out, as failures often caused most of the delays. For example, as a write operation may fail, and the failed write operation may cause the system to delay significant. Google will have two write logs, and if one fails, the other one will take over immediately. Also, the mater nodes have their own shadow masters that store checkpointed states of the mater nodes. Also, GFS allows concurrent reads and writes with a fault-tolerant mechanism, and thus its consistency model is more complicated than a typical file system. This is a source of problems, especially if multiple writers are allowed to execute at the same time.

Google also tried a multi-cell approach where multiple cells will be created in a data center, and in this way multiple GFS masters will run on top of a pool of chunkservers. This also requires applications to partition data into different cells.

2.5.2.3. *Other Similar Projects*

Other distributed file system are also available. For example, Hadoop HDFS [57], CloudStore [32]. Both HDFS and Cloudstore were developed around the same time,

and HDFS is written in Java while CloudStore back-end is written in C++ to improve efficiency.

HDFS also stored files as chunks of 64 MB, over a cluster of machines, Similar to GFS, it achieves reliability via replicating data across multiple hosts. Each host can communicate with each other to achieve replication. Like GFS, HDFS also has a unique server, the name node, and if it goes down, the file system becomes unavailable. When it comes back, it needs to replay all the previous commands to ensure that the file system is consistent, and the whole process may take a long time.

CloudStore also has similar architecture as GFS and has three major components: (1) Meta-data server: This provides a global namespace; (2)Block server: Files are stored in chunks like GFS; (3) Client library: This provides the file system API for application to interface CloudStore.

CloudStore is an open-source program, and it replicates many features of the Hadoop project including data integrity, access from C++, Java and Python, replication, and scalability.

One characteristic of HDFS is that a write operation will require writing from the start of the file to the end of the file, and effectively any write operation will be a complete re-write of the whole file. However, KFS allows writing any place where in the file or append something to an existing file. KFS also has a simple automatic load balancing mechanism as the system will move over crowded nodes to less crowded nodes, but HDFS does not have such mechanism.

2.5.3. BigTable

The BigTable was designed to deal with large amount of semi-structured data for Web applications. For example, the URLs (including contents, crawl metadata, links, anchors, and pagerank), the user preference data (including personalized settings, and recent queries/search results), and the geographic locations(including physical entities such as shops and restaurants), roads, satellite image data and user annotations). The volume of these data are monstrous as the world has trillions of URLs, many versions/pages with about 20K bits per page, hundreds of millions of users, thousands of requests per second, and more than 100TB of satellite image data. One may use commercial databases, such as Oracle, DB2 and SQL server, for Web applications, but these database systems are often designed for general business computing, rather than Web applications, and many low-level optimizations are not available. Also, Thus, Google made a decision to develop a database system BigTable internally. The system has many low-level optimizations optimization such as storage optimization and data transfer.

BigTable intends to achieve the following goals:

- It can process requests asynchronously and continuously;
- It can support high read/write rate with millions of operations per second, in some cases;
- It can scan all or interesting subsets of data from the crawled pages to identify the needed information, and perform a join operation over large one-to-one or one-to-many data sets. The latter is a a complex computation with many

network transfers;

- It can examine and track data change over time. This is useful for crawling, if a Web page has not changed, it is not necessary to re-crawl the same page again;

- It can store the historical data of various pages to provide statistics information for data mining and analysis;

- It can scale up in terms of performance as the number of requests and data size increase over time, and the data size is expected to grow in the future;

- It can provide reliable and highly available computing and data services with built-in fault-tolerant mechanisms.

In essence, BigTable is a distributed storage system, rather than a general-purpose database management system, for managing large amount of structured data in the range of petabytes of data across thousands of inexpensive servers. Currently, BigTable handles petabyte data each day with high performance, scalability, and high availability. The initial design of BigTable started in 2004, and it is currently widely used in Google's services including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth.

2.5.3.1. *Building Components Used in BigTable*

BigTable was designed with the following software programs or infrastructure available:

- GFS and this provides scalable Web file system support;

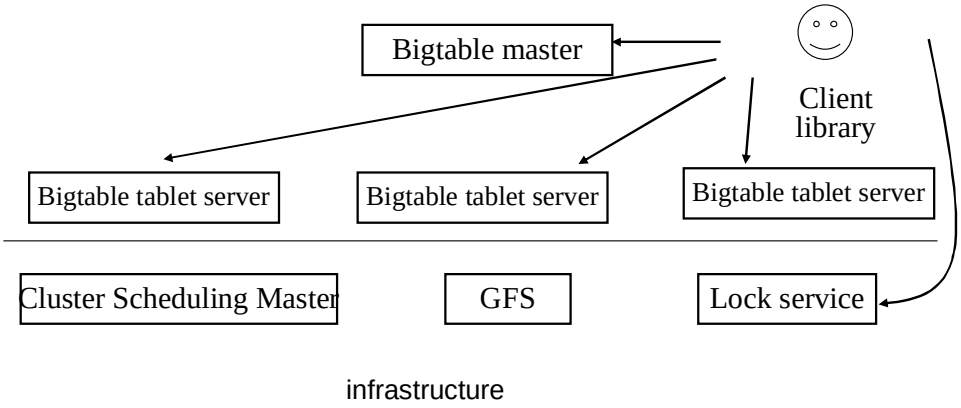


Fig. 18. Google BigTable Architecture

- Scheduler and this can schedule jobs cross a large number of machines, and it also watches any machine failures to reschedule if necessary;
- Lock service and this is a distributed lock manager, and it can reliably hold tiny files;
- MapReduce and this is used for large-scale data processing, and this is used to read/write BigTable data.

The BigTable cluster operates a distributed shared pool of machines that run different types of applications. The cluster management system will schedule jobs to different machines, manage resources on shared machines, monitor machine status, and handle failures at runtime. The system architecture is illustrated in Figure 18.

2.5.4. BigTable Overview

2.5.4.1. *Data Model*

BigTable is essentially a sparse distributed multidimensional sorted map. The index of the map is a string with three components, a row key, column key, and a time-stamp, defined as follows:

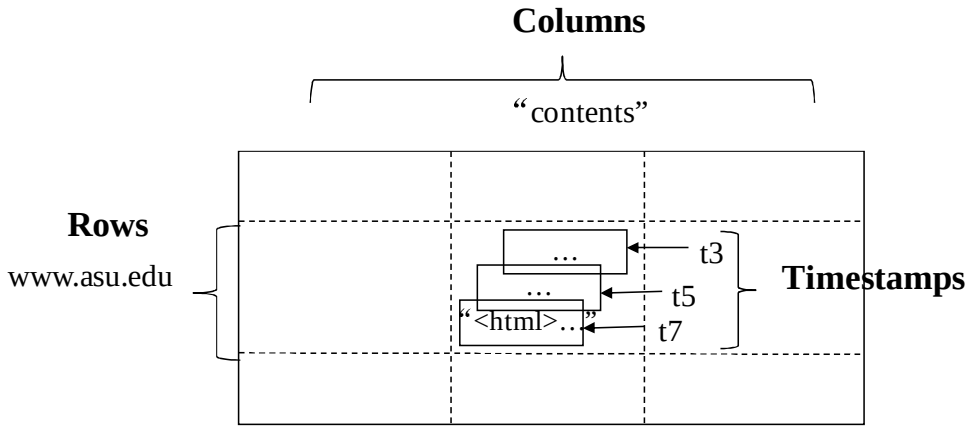


Fig. 19. Data Model of BigTable

(row:string, column:string, time:int64) → string

For example, in Figure 19, a slice of a table that store Web page of “www.asu.edu” is provided.

Rows: the row keys are arbitrary strings (up to 64KB, 10-100 bytes for most users), and the access to data in a row is atomic, supported by transaction processing. The row creation is implicit upon storing data. BigTable sorts the row keys in lexicographic order, and dynamically partition the row range, usually on one or a small number of machines. Each row range is named as a tablet, which is the unit of distribution and load balancing. It is efficient to read and the locality of access is good in this design, for instance, the web pages in the same domain can be grouped together into continuous row range by reversing the host name components of the URLs. One can store data from *www.asu.edu/index.html* under the *my.asu.edu/index.html*. Storing web pages from the same domain close to each other takes the advantage of spatial locality and temporal locality. A user who requests a page is likely to request the related pages (spatial locality) in the near future

(temporal locality).

Columns: web pages are grouped into sets, named column families, which form the basic unit of access control. The column family usually stores data of the same type. A column family can be created only after the column key is generated and used in that table, hence the number of distinct column families is small, and rarely change during operation. The syntax of column key is *family:[optional]qualifier*, where the column family names should be printable, and qualifiers may be arbitrary strings. For example, one possible family name is **language**, which stores each webpage's language ID. Access control and disk/memory accounting are performed at the column-family level, some of them have read base data privilege, while other can view existing data only.

Timestamps: BigTable can store multiple versions of the same data, which are indexed by timestamps. The timestamps are 64-bits integers, which either can be assigned by BigTable, and represent "real time" in microsecond, or be explicitly assigned by client applications. To avoid collision, the applications need to generate unique timestamps. The latest timestamp is stored first, and different versions are stored in a decreasing order. To collect garbage automatically, the client can specify either only last N versions of a cell to be kept, or only new enough versions to be kept.

2.5.4.2. *Implementation Structure*

Tablets: to spread data on different machines, BigTable breaks large tables into tablets at row boundaries, which holds contiguous range of rows and served as a fundamental unit managed by a single machine in a given time. The tablets hold

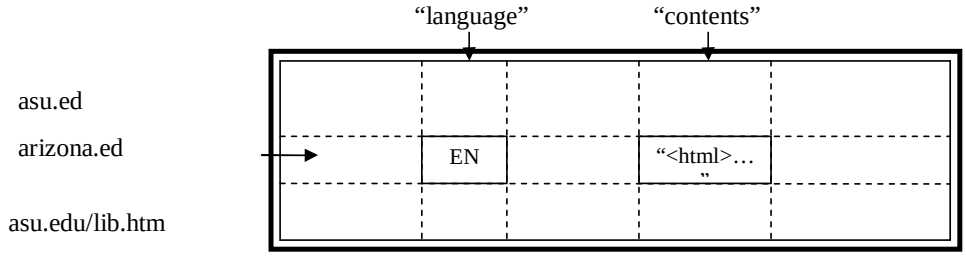


Fig. 20. Sample Tablet

the start and end row key with around 100MB to 200MB of data.

Given a serving machine, it may be responsible for about 100 tablets because at this rate it can perform fast recovery and fined-grained load balancing. For example, if a machine fails, 100 other machines will recover the failure by each picking up one tablet from the failed machine. The master can also decide to move some tablets from one machine to another machine if the former is overloaded. A sample tablet is illustrated in Figure 20.

When a tablet becomes too large, for example, larger than the expected megabytes, the tablet is split into two in Figure

Several issues need to be discussed to use tablets:

a) Locating tablets: As tablets move around from a server to another server, “*given a row, how do clients find the right machine?*” Each tablet has a start row and an end row, so one can identify the tablet by examining the tablet range of each machine. One approach is to use the BigTable master to store the row range for each machine, but the central server will become the bottleneck of the system as the number of tablets is huge.

Instead, one can store special tables that contain tablet location information in BigTable. A “3-level hierarchical lookup scheme” for tablets is proposed in the

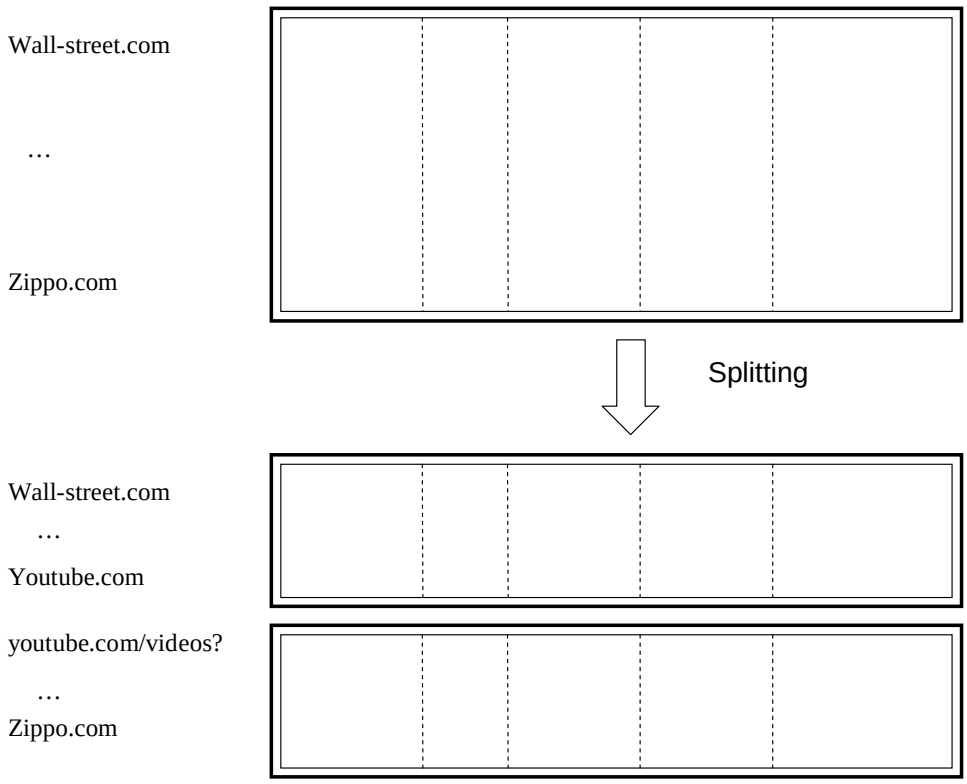


Fig. 21. Split Tablet

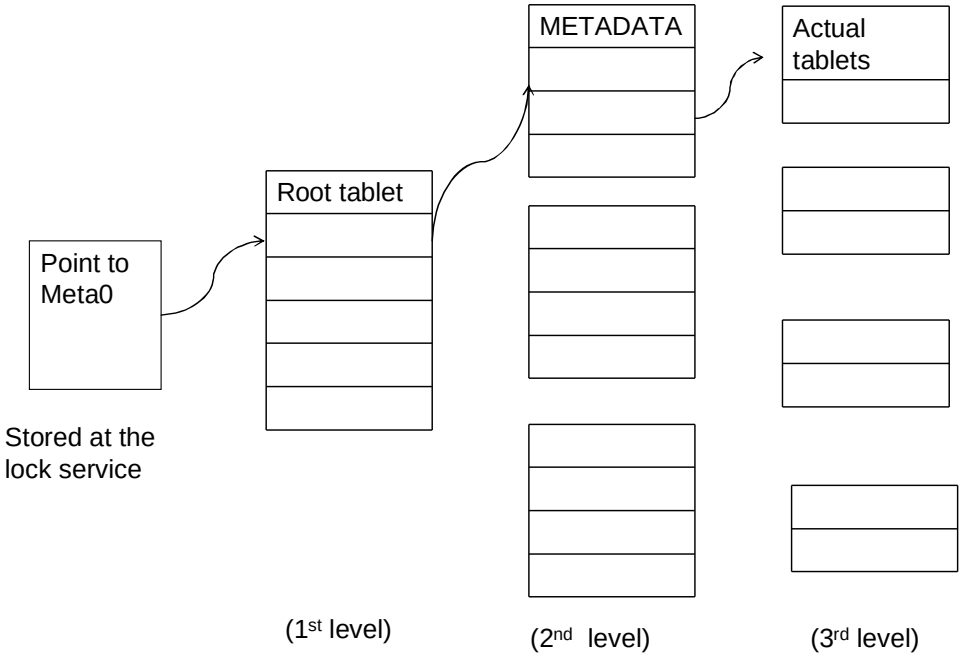


Fig. 22. 3-Level Hierarchical Lookup Scheme

following way (as shown in Figure 22): location is represented as *ip:port* of relevant server. At the 1st level, it bootstrapped from lock server, points to owner of root tablet. At the 2nd level, one can use root data to find owner of appropriate METADATA tablets. At the 3rd level: METADATA table holds locations of tablets of all other user tables.

Specially, to avoid the root tablets becomes the bottleneck, aggressive prefetching and cache technique is applied.

b) Tablet Serving: To store Bigtable data, Google SSTable file format is used. An SSTable provides a persistent, ordered immutable mapping from keys to values, and both keys and values are defined as arbitrary byte strings. The persistent state of tablets are stored in GFS, as shown in Figure 23. To update, the current committed ones are stored in memory in a buffer, the older update are

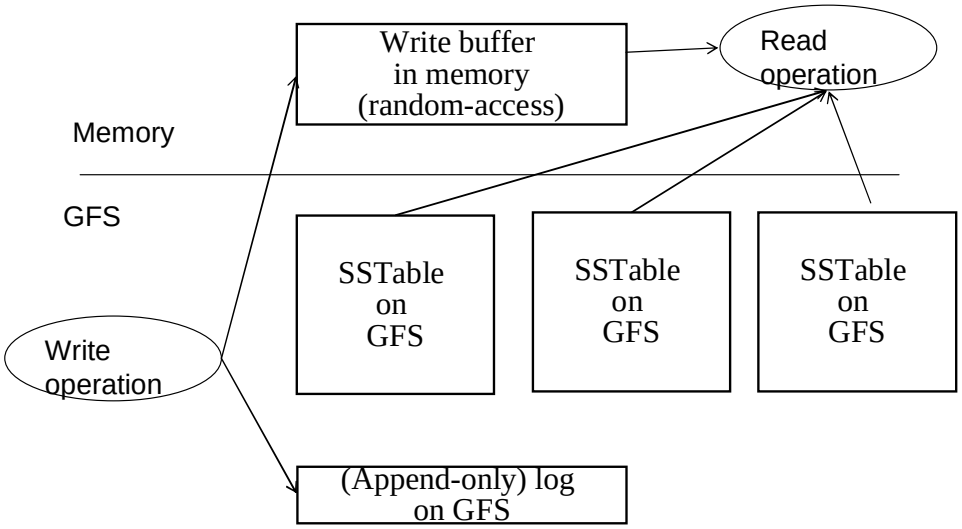


Fig. 23. Table Representation

stored in a set of SSTables, a commit log recorded all the mutations that have not yet been committed. When a write operation arrives at the tablet server, the server checks whether it is well-formed, and the sender has the authority to perform the mutation. After the write has been committed, the contents are inserted into the memory. When a read request comes, the server has also check the well-formedness and the authority. A valid read operation is executed on a merged view of the set of SSTable and memory.

c) Tablet compactions: Each tablet state is represented as a set of immutable compacted SSTable files with the tail of log buffered in the memory. BigTable offers two types of compactions: the minor compaction and major compaction respectively. The first one occurs when in-memory state fills up, the system can pick the tablet with most data and write contents to SSTable in the GFS. The second one can periodically compact all SSTables into new base SSTable on the GFS.

Tablet servers: BigTable is designed to process thousands of tablet servers,

and assuming each server can hold 100 tablets, there will be 1M tablets. All the tablets on one machine share a log; otherwise, one million tablets in a cluster will result in too many files to write, and 1M logs being simultaneously written will perform badly.

By using shared logs, each tabler server has a *write* log for all the tablets stored in it, and updates for multiple tablets coming in same log file. As each chunk contains 64 MB, and thus a new log chunk will be created frequently as the updates keep on coming.

One problem is that during recovery, server needs to read log data to apply mutations for a tablet, and this can cause significant wasted I/O if many machines need to read data for numerous tablets from same log chunk. The recovery mechanism of shared log is as follows: When a machine goes down, the master redistributes its log chunks to other machines to process, and these machines store the processed results locally. The machines that pick up the tablets then query the master for the location of the processed results to update their recently acquired tablet, and then go directly to the machine for their data.

2.5.4.3. *APIs*

The BigTable API supports functions call for creating and deleting tables and column families, changing cluster, table, and column family metadata, such as access control privilege. Also BigTable support querying values from individual rows, or iterate over a subset of the data in a table.

Example 2.5.1 *The following is a piece of C++ code that uses a RowMutation*

abstraction to perform a series of updates.

// Open the table

*Table *T = OpenOrDie("/bigtable/web/webtable");*

// Write a new anchor and delete an old anchor

RowMutation r1(T, "edu.asu.www");

r1.Set("anchor:lib.asu.edu", "ASU");

r1.Delete("anchor:www.arizona.edu");

Operation op; Apply(&op, &r1);

The call performs an atomic mutation to the Webtable: it adds one anchor to www.asu.edu and deletes a different anchor.

There are several features supported in BigTable, and users can manipulate data in diverse and complicated ways as follows:

1. Single-row transactions: This performs atomic read/write/update sequences on data stored under a single row key. Note that, BigTable currently does not support general transactions across row keys, but it provides an interface for batching writes across row keys at the clients.

2. BigTable allows cells: used as integer counters.

3. Execution of client-supplied scripts in the address spaces of the servers, using Sawzall scripts developed at Google.

2.5.4.4. *Refinement*

To achieve better performance, availability and reliability, BigTable has the following refinements and optimizations:

Locality Groups: In BigTable, columns are easy to create and are created implicitly, but column families are heavy to create because one need to specify things like type and attributes. To optimize access, column families can be split into locality groups. This increase performance because small frequently accessed columns can be stored in a different spot than the large infrequent columns.

Compression: there are many opportunities for compression, for example, many similar values in the same row/column at different timestamps, similar values in different columns, similar values across adjacent rows and etc. Within each SSTable for a locality group, encode compressed blocks can keep blocks small for random access (64KB compressed data), exploit fact that many values very similar, and needs to be low CPU cost for encoding/decoding. Two building blocks here:

1. BMDiff: the input is a dictionary with some source text that needs to be compressed, when depressing one can use the dictionary. The output is the sequence of COPY¡x¿ bytes from offset ¡y¿ and LITERAL(¡literal text¿). It stores hash at every 32-byte aligned boundary in the dictionary. For every new source byte, it computes incremental hash of last 32 bytes in a slide window, and lookup in hash table. If it hits, then check and verify the 32 bytes in he dictionary or earlier in the text.

2. Zippy: a LZW-like algorithm works at local level, deal with small size of

redundancy, for example, store hash of last 4 bytes in 16K entry table. It is differences from BMDiff, which has much smaller compression window (local repetitions), and the hash table is not associative.

Other Database Approaches

Note that GFS and BigTable were designed specifically for Web-based cloud computing, and thus their designs were not constrained by conventional database management design or file systems. As Google did not have existing database management systems or file systems, and thus they could start a new design from the scratch. Cattel has survey many other similar database systems designed with similar goals and objective, and he characterized these as datastores rather than databases as these systems do not provide full features of traditional database management systems. These modern datastores have some common features:

- Support call level interface (rather than binding to SQL statements), thus applications may have direct control over database storage;
- Effective index for large data for efficient processing;
- Able to horizontally (by rows) scale throughput over servers: For example, some rows can be assigned to one node, while other rows can be assigned to other nodes for processing;
- Support dynamic data schema: Thus, different tenants can have different data schema requirements but they can still share the same datastore and schema table;

- Indexes are often cached in memory for scalable accesses: This may also distribute and/or replicate indexes among different nodes;

These datastores may store data according to:

- Key-value stores: The system stores data based on a designer-defined key;
- Document stores: The system stores indexed documents rather than specific items in the documents are indexed;
- Extensible record store: The system stores extensible records and they are partitioned across different nodes, and this is sometimes referenced as column-oriented database;

Examples of these modern datastores include SimpleDB and Cassandra.

Another approach is to extend an existing database management system for Web applications. For example, the Oracle Real Application Clusters (RAC). While it was evolved from traditional database management system for Web applications, it has many new features. Specifically, RAC, a clustered database, allows multiple processors to run the database software while accessing a single database. A clustered database is essentially a database that runs on top of a cluster of processors, taking advantages of the scalability and availability of these inexpensive processors.

Note that GFS and BigTable were designed specifically for Web-based cloud computing, and thus their designs were not constrained by conventional database management design or file systems. As Google did not have existing database management systems or file systems, and thus they could start a new design from the

scratch. However, other approaches are also possible, and one such design is Oracle Real Application Clusters (RAC). While it was evolved from traditional database management system for Web applications, it has many new features. Specifically, RAC, a clustered database, allows multiple processors to run the database software while accessing a single database. A clustered database is essentially a database that runs on top of a cluster of processors, taking advantages of the scalability and availability of these inexpensive processors.

Clustered databases can be classified as two categories:

- Share nothing: Each participating node owns a subset of data, and each will operate on its own subset of data, however, the node can communicate with each other to exchange data.

- Share everything: Each processor can access any data in the database.

In both cases, each node is connected with each other via a high-speed network such as infiniBand or Myrinet Having a high-speed network connecting the cluster with storage is critical. Initially, a clustered database uses a storage device such as a disk to transfer data, and this will slow down the operation as the storage device may have the slowest speed. This problem is addressed when system communicate with storage via dedicated communication systems such as Fibre Channel. Fibre Channel is a network technology developed for storage networking that is capable of delivering at the speed of gigabit, and it is a common technology for storage area networks (SAN). In 2008, this technology is able to deliver 21.04 GBaud with a throughput at 5,100 MBps.

Currently, both share-nothing or shared-everything architectures require three different interconnections:

- Ethernet for management;
- InfiniBand or Myrinet for internode communication. InfiniBand is a communication technology mainly used for high-performance computing connecting a computing node with storage devices. Myrinet is high-performance communication technology used in high-performance systems in a cluster; and
- Fibre channel for block storage communication.

Both share-nothing and share-everything clustered databases require careful application partition to achieve optimal performance.

The scalability of this share-nothing architecture depends on the data partitioning scheme. One problem of this approach is that if a node crashes, some portion of data will not be available, and thus often the data are partitioned so that data are actually stored in multiple disks in the background, so that if one of them failed, the other disk can restore the data automatically. This share-nothing clustered database approach is used by many database systems such as IBM DB2, Microsoft SQL Server, MySQL Cluster, and Bizgres MPP.

In the shared-everything or shared-disk architecture, the physical layout usually involves network-attached storage, in which all nodes communicate with a separate shared disk such as a RAID via a high-speed interconnection network such as Fibre Channel. In this architecture, there may be contentions among various

processors when they try to access the same database simultaneously. RAC is one such database. Each processor has its own cache and it is necessary to coordinate various concurrent reads and writes by using the high-speed network to maintain system cache consistency. RAC has a mechanism called Cache Fusion. In general, a database system will face:

- Concurrent readers;
- Concurrent readers and writers on different nodes; and
- Concurrent writers on different nodes.

The first case is not an issue as multiple readers can read data concurrently without any conflict. However, to handle the 2nd and 3rd case, the Cache Fusion keeps a global controller that maintains the status of each cached block at each node. If there is a conflict, the the global controller will maintain the consistency by asking the holding node to transfer the most recent data over the high-speed network to a receiving node without going through the disk system. Actual data writing to disk will happen, when the cache data are being replaced or during checkpointing. Essentially, the concurrency control that was done in traditional database management system is now done at the cache level supported by high-speed network systems that support both processor-to-processor and processor-to-storage communication. A global data controller will maintain the data consistency via a complex protocol that involves cache management, data communication, and concurrent reads and writes. Each writer still need to declare some exclusivity and other writers must yield and wait for the operation to complete, except when the

modified data are available, they will be transmitted via a high-speed network from one cache to another cache if they reside in different nodes. As most of operations done by the global data controller are done either at a computing node or via the high-speed communication systems, the operations are more efficient than the traditional approach via the participating disks. As concurrent writers will still take more time than concurrent readers, this share-everything approach is more suitable for applications with mostly read operations. If the multi-tenancy application have many writers, data partitioning becomes a critical issue as synchronizing multiple writers still take more time than synchronizing readers.

The shared-everything approach is used by IBM DB2 for z/OS, and IBM DB2 pureScale, Sybase Adaptive Server Enterprise, Cluster Edition.

2.6. Microsoft's Azure

Windows Azure [9] is a cloud services operating system that serves as the development, service hosting and service management environment which is written using .NET libraries and compiled to the Common Language Runtime. It is designed for utility computing, and provides facility for developers to write apps, to host apps for compute, to manage apps, and to store data. It provides developers with on-demand compute and storage to host, scale, and manage web applications on the internet through Microsoft data centers. It can support multiple languages and integrates with existing on-premises environment. Developers can use existing Microsoft Visual Studio, as well as integrate popular standards, protocols and languages including SOAP, REST, XML, Java, PHP and Ruby. Windows Azure is intermediate between application frameworks like Google AppEngine and hardware virtual machines like

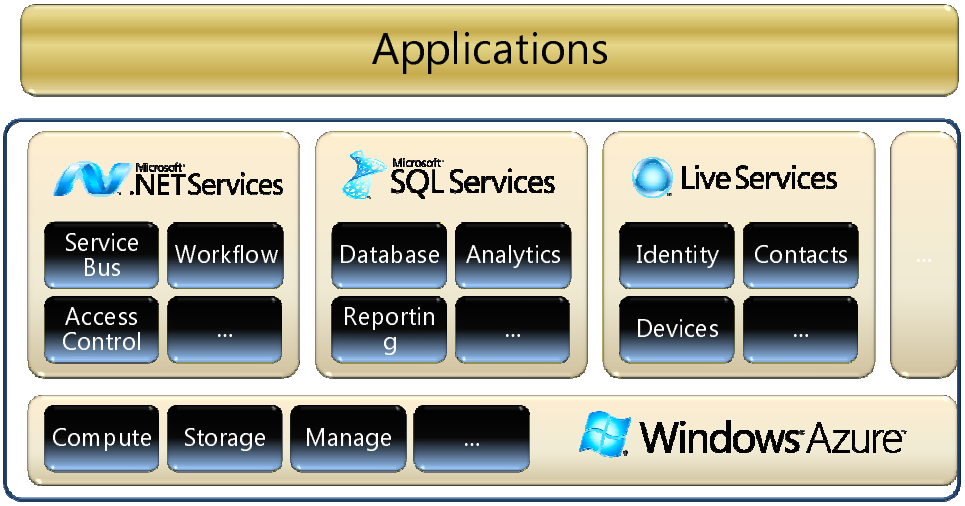


Fig. 24. Overview of Windows Azure

Amazon EC2.

Windows Azure serves as the development, run-time, and control environment for the Windows Azure Platform. It can handle load balancing, resource management and life cycle management of a cloud service based on requirements that the owner of the service established. When one wants to deploy an application in Windows Azure, he can specify the service topology, including the number of instances to deploy and any configuration settings, then Windows Azure would deploy the service and manage upgrades and failures to maintain availability.

The overview of main components in Windows Azure is shown in Figure 24. Several components are contained in the framework, for example, .NET services support service bus, access control, workflow management and etc. The SQL services support database, analytic, reporting and etc.

2.6.1. Architecture of Azure

Microsoft's Windows Azure platform is a group of cloud technologies, each providing a specific set of services to application developers. As Figure 25 shows, it can be used both by applications running in the cloud and by on-premises applications. Three key components of Azure are:

- Windows Azure: a host platform for cloud-based windows applications, which provides a Windows-based environment for running applications and storing data on servers in Microsoft data centers.

- SQL Azure: a set of services that provide access to a cloud-based SQL server. It supports a range of data oriented functions such as reporting, analytic and synchronization. SQL Azure consists of SQL Azure Database, which allows cloud and on-premises applications to store relational and other data.

- Azure AppFabric: provides a way for non-cloud applications (such as on premises applications) and cloud applications (running on Azure) to communicate. AppFabric also provides services that local, on premises applications can use to access cloud storage. It provides two functionalities, messaging and access control, in which messaging is provided by a service bus and the access control part handles authentication of clients, both cloud and on-premises.

There are two types of applications in Azure: cloud applications and on-premises applications (applications that run inside an organization). Cloud applications are created specifically for the cloud, while on-premises applications are not required to be scalable, but has to interact with cloud applications or resources.

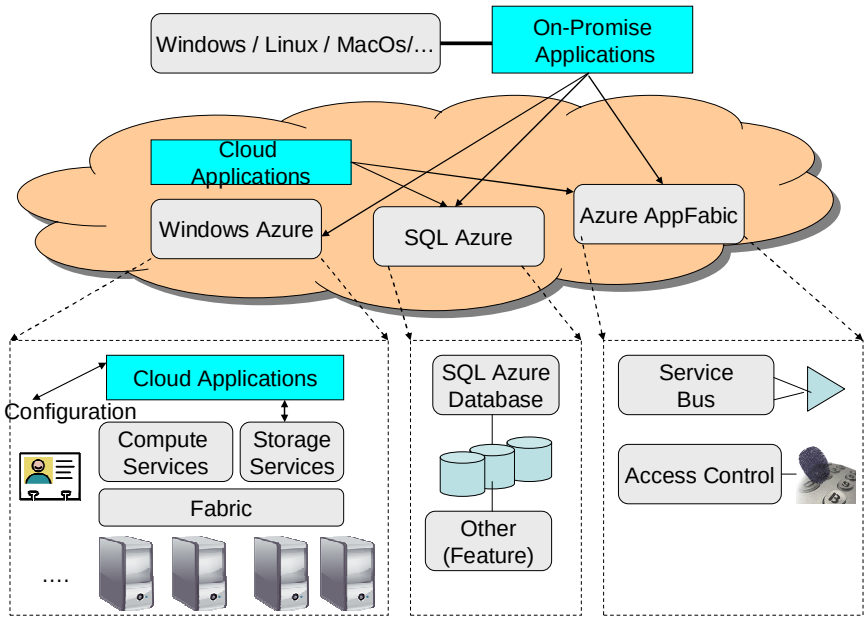


Fig. 25. Windows Azure Platform Support

Both of them have data and control relationships with the key Azure components as shown in Figure 25.

2.6.1.1. *Windows Azure*

The zoom-in view of Windows Azure is shown in the left bottom dashed rectangle of Figure 25. Windows Azure runs on a large number of machines, all located at the Microsoft data centers and accessible through internet. It has two types of services, compute and storage:

The compute services are based on Windows and provide a virtual windows operating environment for cloud applications to run on. Developers can build their own applications using the .NET Framework, or other approaches. It supports multiple programming languages (such as C#, Visual Basic, C++ and Java) and different development tool(e.g. Visual Studio). Applications can run as independent background processes, or combine the two.

The storage services provide access to cloud storage, both Windows Azure applications and on-promised applications can access the Windows Azure storage service in the same way by using RESTful approach. A Window Azure fabric layer monitors cloud applications and processing power into a unified whole. Each application has a configuration file, which specifies various aspects of an application's behavior, e.g. the number of application instances. The configuration file can be manually or programmatically modified. Application behavior is monitored by the fabric layer.

To allow customers generate, configure, and monitor applications, Windows Azure provides a browser-accessible portal. A customer can provide a Windows Live ID, then chooses whether to create a hosting account for running applications, a storage account for storing data, or both.

2.6.1.2. *SQL Azure*

The goal of SQL Azure is to offer cloud-based services for storing and working with data. SQL Azure will eventually include a range of data-oriented capabilities, including data synchronization, reporting, data analytic, and others, the first SQL Azure component to appear is SQL Azure Database, as shown in the middle bottom of Figure 25.

SQL Azure Database provides a cloud-based database management system (DBMS), which allows both on-premises and cloud applications store relational and other types of data in Microsoft data centers. A tenant can pay only for what it uses, while increases and decreases usage (and cost) as their needs change. Using a cloud database also allows converting what would be capital expenses (such as

investments in disks and DBMS software) into operating expenses.

SQL Azure Database is built on Microsoft SQL Server, which offers a SQL Server environment with indexes, views, stored procedures, triggers, and etc. The data can be accessed using ADO.NET and other Windows data access interfaces. Customers can also use on-premises software (such as SQL Server Reporting Services) to work with their cloud-based data.

Tenants can use SQL Azure Database as a local DBMS, but the management requirements are significantly reduced. Rather than monitoring disk usage and servicing log files, a SQL Azure Database customer can focus on the data only. To use SQL Azure Database, one can go to a Web portal and provide the necessary information.

2.6.1.3. *Windows Azure Platform AppFabric*

Windows Azure platform AppFabric is proposed to provide cloud-based infrastructure services.It is composed of two parts:

- Service Bus: To expose an application's services on the Internet simpler, one can make an application expose endpoints that can be accessed by other applications, no matter on-premises or in the cloud. Each exposed endpoint is assigned a URI, which clients can use to locate and access the service. Service Bus also handles the network address translation and getting through firewalls without opening new ports for exposed applications.

- Access Control: allows a RESTful client application to authenticate itself and to provide a server application with identity information. Then the server can

use this information to decide what this application is allowed to do.

2.6.2. Windows Azure Inside

Windows Azure does two main things: runs application and stores data.

2.6.2.1. *Running Applications*

An application on Windows Azure typically has multiple instances, each running a copy of all or part of the application's code. Each instances runs in its own Windows virtual machine (VM) which are provided by a hypervisor specifically designed for cloud usage.

The developer does not have to explicitly know how to create VMs, or worry about run and maintain Windows OS. He can create applications using roles (Web roles and/or Worker roles) to tell Windows Azure how many instances of each role are needed. Windows Azure will do the left, in which it generates a VM for each instances, run application in the corresponding VMs.

The process is shown in Figure 26. Windows Azure provides built-in load balancing to spread requests across Web role instances that are part of the same application. The Web role instance accepts incoming HTTP (or HTTPS) requests via Internet Information Services (IIS) 7. It can be implemented using ASP.NET, WCF, or another technology that works with IIS. The Worker role instance is quite similar to a Web role instance. The key difference is that a Worker role doesn't have IIS pre-configured to run in each instance, and aren't hosted in IIS. A Worker role can still accept requests from the outside world, however, and developers can even run another Web server, such as Apache, in a Worker role instance. The comparison

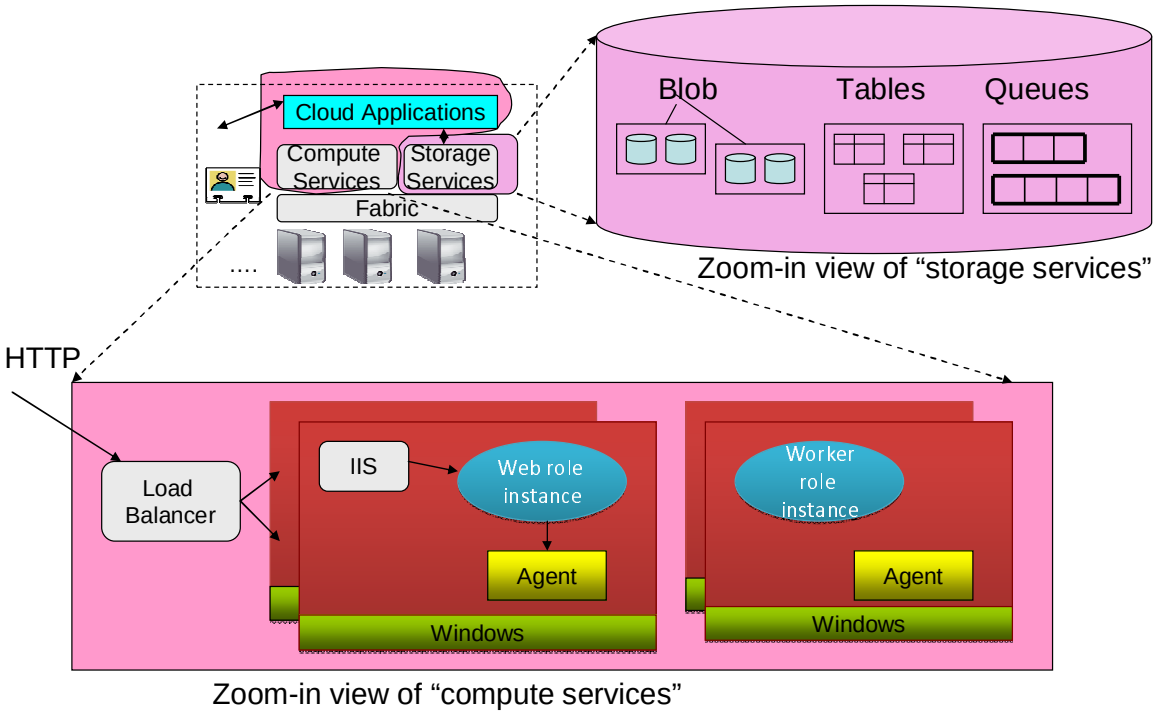


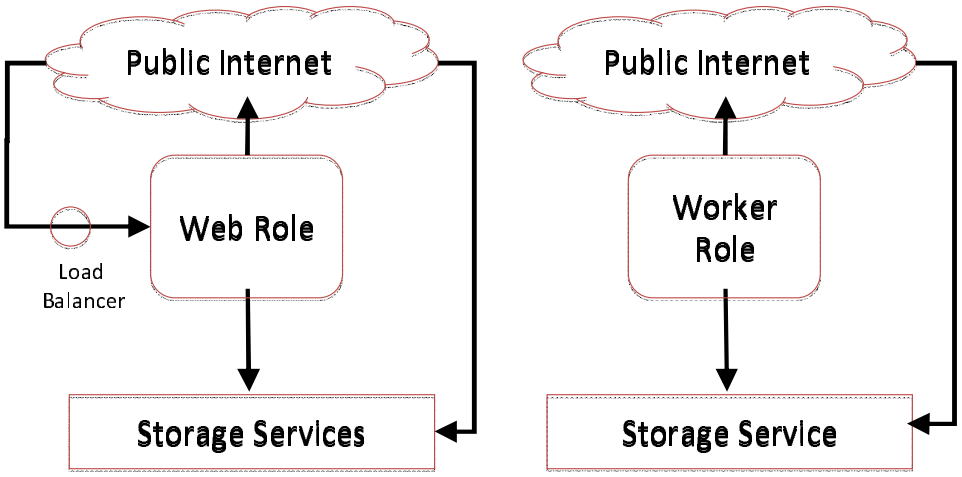
Fig. 26. Running Windows Azure Applications

of web role and worker role is shown in Figure 27.

Worker role instances can communicate with Web role instances in various ways:

- Use Windows Azure storage queues. A Web role instance can insert a work item in a queue, and a Worker role instance can remove and process this item.

- Worker roles and Web roles can set up direct connections via Windows Communication Foundation (WCF) or another technology. No matter it runs a Web role instance or a Worker role instance, each VM also contains a Windows Azure agent that allows the application to interact with the Windows Azure fabric. The agent exposes a Windows Azure-defined API that lets the instance do things such as find the root of a local storage resource in its VM



(a) Web role (b) Worker role

Fig. 27. Web Roles and Worker Roles

instance.

The size of VMs are four types: one core, two cores, four cores, and eight cores. Since each VM is assigned one or more cores, applications can have predictable performance. The application's owner can increase the number of running instances by modifying the application's configuration file to increase the performance. Then the Windows Azure fabric will spin up new VMs, assign them to cores, and start running more instances for this application. This process is not automatically, in which the fabric doesn't do this automatically with load changing, but provides APIs to support an application do this itself. The fabric can also detect the failure of a Web role or Worker role instance, then start a new one.

To be scalable, Windows Azure Web role instances must be *stateless*. The client-specific state would be written to Windows Azure storage, sent to SQL Azure Database, or passed back to the client in a cookie. The stateless requirement is also

mandated by Windows Azure's built-in load balancer since it doesn't allow creating an affinity with a particular Web role instance. No guarantee that multiple requests from the same tenant will be sent to the same instance. Both Web roles and Worker roles are implemented using standard Windows technologies. Hence moving existing applications to Windows Azure requires a few changes.

2.6.2.2. *Data Processing*

There are three types of data are stored and managed in "storage services" as a zoom-in view in Figure 26:

- Blobs: A storage account can have one or more containers in a hierarchy structure, each of which holds one or more blobs which can be as big as a terabyte. To make transferring large blobs more efficient, blobs can be subdivided into blocks. If a failure occurs, retransmission can resume with the most recent block rather than sending the entire blob again. Blobs can also have associated metadata. A content delivery network (CDN) is provided to make distributed access to blob data more efficient, which can store frequently accessed data at locations closer to the applications using it. The other way to use Blobs is through Windows Azure XDrives, which can be mounted by a Web role instance or Worker role instance. The underlying storage for an XDrive is a blob, and so once a drive is mounted, the instance can read and write file system data that gets stored persistently in a blob.

- Tables: which are not classic relational tables. The data is stored in a set of entities with properties without defined schema. The properties can have

various types, such as int, string, boolean, or DateTime. In stead of using SQL, an application can access a table's data using ADO.NET Data Services or LINQ. A single table can have a large size, with billions of entities holding terabytes of data, and Windows Azure storage can partition it across many servers if necessary to improve performance.

- Queues: provide a way for Web role instances to communicate with Worker role instances. For example, a user might submit a request to perform some compute-intensive task via a Web page implemented by a Windows Azure Web role. The Web role instance that receives this request can write a message into a queue describing the work to be done. A Worker role instance which is waiting on this queue can then read the message and carry out the task it specifies. Any results can be returned via another queue or handled in other way.

All data in Windows Azure storage is replicated three times. This replication allows fault tolerance. The system guarantees consistency, hence an application that reads data it has just written will get what it expects. Windows Azure storage can be accessed either by a Windows Azure application or by an application running somewhere else. In both cases, all three styles(blobs, tables, and queues) use the conventions of REST to search and expose data. Everything is named using URIs and accessed with standard HTTP operations.

2.6.3. SQL Azure Inside

An application using SQL Azure Database can run on different platforms, such as Windows Azure, an enterprise's data center, on a mobile device, or somewhere else. Wherever it runs, the application accesses data via a protocol, Tabular Data Stream (TDS), which is the same protocol used to access a local SQL Server database, and so a SQL Azure Database application can use any existing SQL Server client library (e.g. ADO.NET, ODBC, and PHP). It is more like an ordinary SQL Server system, standard tools can also be used, including SQL Server Management Studio, SQL Server Integration Services, and BCP for bulk data copy.

The SQL Azure administration is handled by Microsoft, and the service doesn't expose physical administrative functions. A customer can not shut down the system or interact directly with the hardware it runs on. It is more robust than a single instance of SQL Server providers, since all data are duplicate three times as discussed earlier in this section. It is strong consistency, when a write returns, the data is made persistent.

The maximum size of a single database in SQL Azure Database is 10 gigabytes. An application whose data is within this limit can use just one database, while an application with larger data needs to create multiple databases. Figure 28 shows two applications with different database size. With a single database, an application can only see one set of data, and SQL queries can be used as usual. With multiple databases, the application must divide its data among them. Each tenant can only operate/search its own data, and can no longer issue a single SQL query that accesses all data in all databases. For the application that works with multiple databases, it

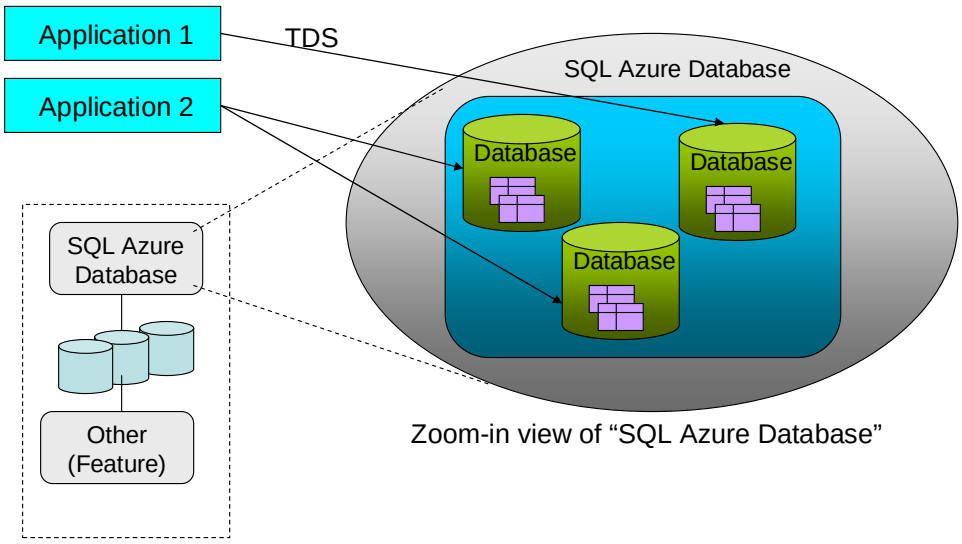


Fig. 28. SQL Azure

will need to be aware of how that data is divided.

To support parallel queries, applications with smaller amounts of data can also choose to use multiple databases. Multi-tenant application might choose multiple databases.

2.6.4. Windows Azure Platform AppFabric Inside

The goal of Windows Azure platform AppFabric is to help make applications connections with others. It contains two main components: service bus and access control.

2.6.4.1. *Service Bus*

After building up an application inside one organization, the next step is to connect this service through the internet with outside organizations. There are many challenges here, for example, how can clients in other organizations find endpoints to connect to inside organization? Registry may be need to in this case. Once outside organization finds the service, how can requests get through the inside service? Due

to the network address translation, some applications usually does not have a fixed IP address, how can external application get and go through the firewall? Service bus provides solutions.

Web services are built with Windows Communication Foundation (WCF), the main process works in the following steps:

- Step 1: WCF service registers one or more endpoints with Service Bus
- Step 2: For each registered endpoint, Service Bus exposes its own corresponding endpoint. Service Bus also assigns a URI root for the application, which can be named in a hierarchy way and allow endpoints to be assigned a discoverable URIs. Network Address Translation can be solved since traffic on the open connection with Service Bus will always be routed to a specific application. Also there would be no problem to pass firewall which would not be blocked.
- Step 3: search Service Bus registry to find out the endpoint using Atom Publishing Protocol, and returns an AtomPub service document with references to the endpoints Service Bus
- Step 4: The client can invoke operations on the services exposed through these endpoints
- Step 5: For each request Service Bus receives, it invokes the corresponding operation in the endpoint exposed by WCF service
- Step 6: Service Bus establishes a direct connection between an application

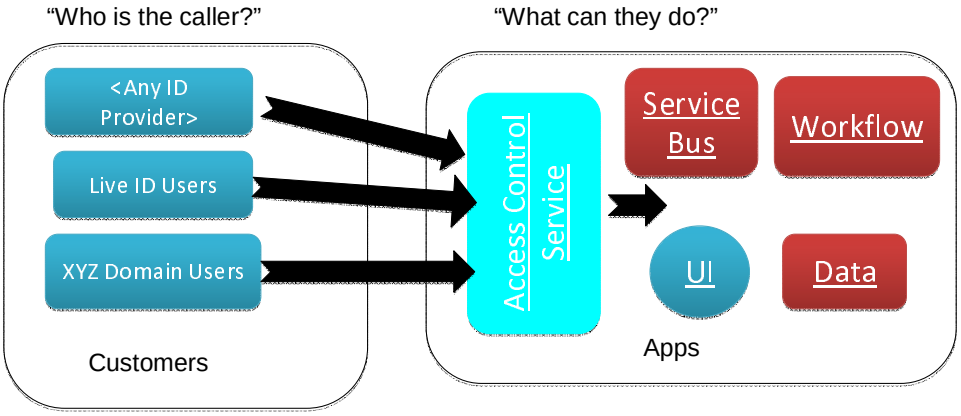


Fig. 29. Access Control of Windows Azure

and its client whenever possible, making their communication more efficient.

2.6.4.2. *Access Control*

The access control of Azure is shown in Figure 29. A UI(User Interface) for creating and managing collections of access control rules. The client API provides a programmatic way to manage collections of access control rules. The service (STS) which is a hosted service that issues tokens can developer interact with the service via the "Geneva" framework.

To communicate with a particular server application, the access control works as follows:

- Step 1: The client must first get a token issued by that contains identity information about this client. This information is expressed as one or more claims, each of which describes the client application in some way.
- Step 2: Once the client application has authenticated itself, the Access Control service creates another token containing identity information for this client
- Step 3: Once the new token is created, its sent back to the client application

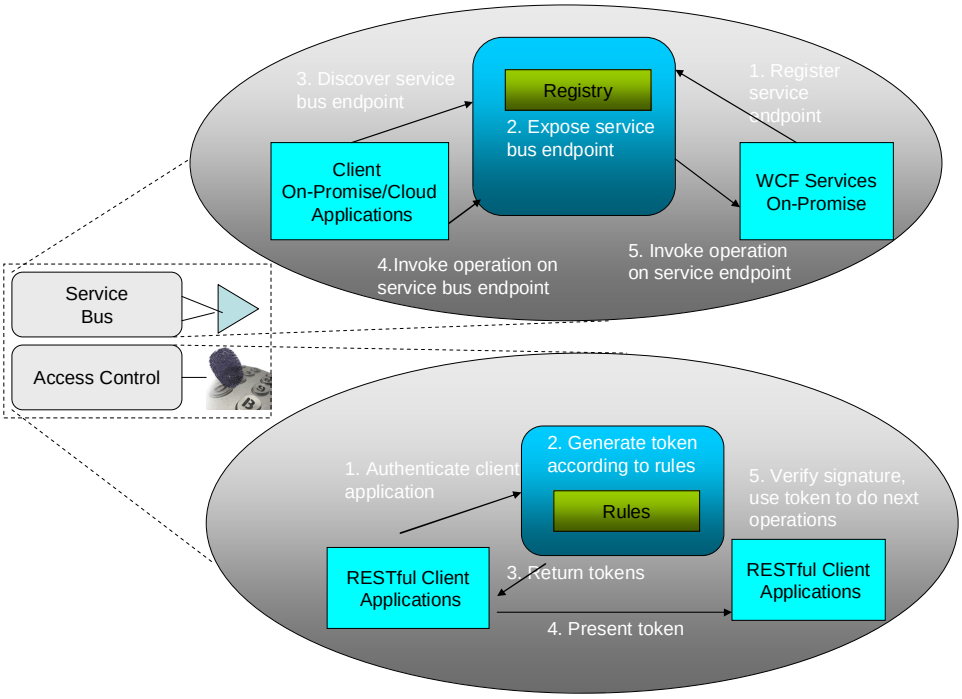


Fig. 30. Windows Azure Platform AppFabric Inside

- Step 4: The client sends this signed token to the server.

- Step 5: Validates the tokens signature and uses the claims it contains

2.7. Amazon's EC2(Elastic Compute Cloud)

Amazon EC2 is at one end of the spectrum. An EC2 instance looks much like physical hardware, and users can control nearly the entire software stack, from the kernel upwards. This low level makes it inherently difficult for Amazon to offer automatic scalability and fail-over, because the semantics associated with replication and other state management issues are highly application-dependent. At the other extreme of the spectrum are application domain specific platforms such as Google App Engine.

As a successful example, Elastic Compute Cloud (EC2) from Amazon Web

Services (AWS) sells 1.0-GHz x86 ISA "slices" for 10 cents per hour, and a new "slice", or instance, can be added in 2 to 5 minutes. Amazon's Scalable Storage Service (S3) charges $0.12 to 0.15$ per gigabyte-month, with additional bandwidth charges of $0.10 to 0.15$ per gigabyte to move data in to and out of AWS over the Internet. Amazon's bet is that by statistically multiplexing multiple instances onto a single physical box, that box can be simultaneously rented to many customers who will not in general interfere with each others' usage.

The API exposed is "thin": a few dozen API calls to request and configure the visualized hardware. There is no a priori limit on the kinds of applications that can be hosted; the low level of virtualizationraw CPU cycles, block-device storage, IP-level connectivity allow developers to code whatever they want. On the other hand, this makes it inherently difficult for Amazon to offer automatic scalability and failover, because the semantics associated with replication and other state management issues are highly application-dependent.

3. AN EFFECTIVE SERVICE PRIORITIZATION MODEL

Service providers receive multiple requests from customers(consumers), how to prioritize those service requests to maximize the business values and minimize customers' dissatisfaction is one of the most important issues in cloud computing.

In this chapter, we proposal an innovative prioritization model, which uses different types of information, including customer, service, environment and workflow information to optimize the performance of the system. The large state-space of the workflow makes the ranking problem challenging. We propose a workflow attribute with a reduced state-space based on the number of visits to a particular step or re-work. We find that incorporating this workflow attribute results in improvement of the density modeling techniques that we develop over those that incorporate only customer and service specific attributes by $8-9$ percent in Average Precision. We apply this model to a real application, an end-to-end mortgage origination process and evaluate the performance of the model. Further, our results indicate that these models perform better than classification based models like Support Vector Machines for ranking, showing a 8 percent improvement in Average Precision.

3.1. Introduction

Cloud Computing is an on-demand process, in which service providers received multiple service requests from consumers and process them in a first in first serve way at most cases. Unfortunately, this is not the desirable way to handle the customers' requests for the following reasons: first of all, some requests should have a high priority than others, since they can bring more business profits to the service providers than other requests, e.g. banks should process the big customer

Channel	Pull-through
Branch	0.33-0.56
Phone Banking	0.24-0.52
Internet Banking	0.06-0.51
Broker	0.76

Table 1

Typical Pull-through Rates

with millions dollars first, in stead of small individual transaction, which can bring more benefit to the bank. Secondly,customer service providers should reduce the dissatisfaction of customers as much as possible, which requires in-time service. More and more service providers realize the importance of prioritization service requests, and they call for effective prioritization models in real applications. We investigate a real application in mortgage services, and propose an efficient model which can provide an effective solution to service prioritization.

Mortgage Origination (MO) is the end-to-end process beginning with the submission of a mortgage application to a lender and ending in *closing* (lender approves, applicant accepts and lender funds the loan) or *non-closing* (either lender disapproves, or applicant withdraws or refuses approved offer by lender). Below, we use the terms mortgage application and loan interchangeably to refer to an application submitted by a customer to a lender for approval. We are interested in developing models for ranking applications, taking into account customer and product-specific attributes of the applications as well as their history or workflow. Developing ranking models not only enables process efficiency but also allows for identification of applications that may have a high likelihood of non-closing but whose likelihood of closing may be improved through "corrective action". The specific nature of correc-

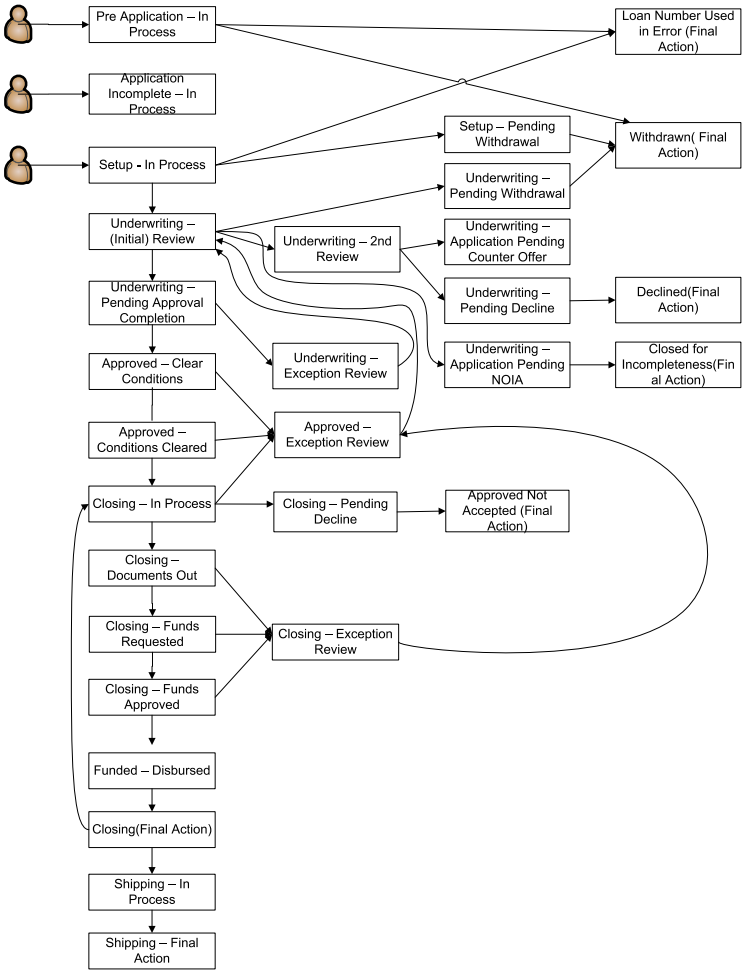


Fig. 31. Lending Process Overview

tive action is dependent on the lending institution. It could include change in the attributes of the mortgage product being offered. It is hoped that such an action would lead to a higher conversion rate of applications submitted into applications closed. This *pull-through rate* is defined as the ratio of the number of applications that close to those that are submitted.

Our study is motivated by the following two observations: (1) The typical pull-through rates in the industry may be quite low for some channels (see Table 1) thus offering considerable scope for improvement. (2) The MO process may involve

several dozens of tasks or status' and thus the problem of ranking applications in the process in order of likelihood of closing, at an intermediate status, may be non-trivial. In fact, as we will see, several applications re-visit their status, making the ranking problem involved.

As stated above, each application contains customer attributes like Credit Score and product specific attributes such as Interest Rate, Loan Amount, etc. However, what makes the problem of ranking applications particularly interesting is the workflow history that we incorporate to rank applications. Let us describe this in more detail through an illustrative example. Consider the Underwriting-Pending Approval Completion task in the MO process of a service provider (see Figure 31 for an overview of the process). Suppose that there are two applications, A and B, waiting to be reviewed at this status. For simplicity, we consider only credit score and workflow history while comparing the two applications. Suppose that application A has a credit score of 712 while application B has a credit score of 662. However, the history of application A, so far, reveals that it has undergone considerable re-work, i.e., it has traversed the loop, Underwriting-(Initial) Review, Underwriting - Pending Approval Completion and Underwriting - Exception Review two times, possibly due to insufficient employment proof. On the other hand, application B has undergone no re-work. The question arises as to how we might compare the likelihood of closing of applications A and B. Thus, the large state-space of the workflow attribute makes the problem of ranking applications, in order of likelihood of closing, challenging.

We now summarize the main contributions: we propose a workflow attribute

with a reduced state-space based on the number of visits to a particular step or re-work. We find that incorporating this workflow attribute results in improvement of the density modeling techniques that we develop over those that incorporate only customer and product specific attributes by $8-9$ percent in Average Precision. The simple and scalable density modeling techniques allow for easy identification of applications that are likely to non-close and consequent corrective action such as change in the attributes of the mortgage product being offered. Further, our results indicate that these models perform better than classification based models like Support Vector Machines for ranking, showing a 8 percent improvement in Average Precision.

3.2. Problem Statement

In this section, we introduce the relevant notation and state the problem of ranking applications at any status taking into account the customer, product and workflow attributes of applications. We first represent the MO process by a directed graph whose vertices are the status' or tasks of the process and edges are possible transitions between status'. Associated with each application is a unique identifier. We present the problem of prioritizing applications at each status as an optimization problem, whose objective function is a metric for ranking models and decision variables are ranks associated with each application waiting at the status of interest. In order to state the problem, we introduce the notion of history of the applications at an epoch of time to be a set that contains information pertaining to the sequence of status' visited by all applications up to that time.

Let us now state the problem formally. We represent the workflow of the MO

process by a strict digraph, G. Let $V(G)$ and $E(G)$ denote the set of vertices and edges of G respectively. We define the history of applications at time T, $\mathcal{H}_{\mathcal{T}}$ to be the set of triplets of the unique identifier corresponding to the loan, status' and entry times into those status'. $\mathcal{H}_{\mathcal{T}}$ uniquely determines the history of all applications that have been processed and those that are being processed till time T. We refer to $\mathcal{H}_{\mathcal{T}}$ as the state of the process at time T:

$$\mathcal{H}_{\mathcal{T}} \quad = \quad \{(i, v, t_{iv}) \mid t_{iv} \leq T \;\; i \in \mathcal{S}, v \in V(G)\}, \tag{3.1}$$

where $\mathcal{S}$ is the set of unique identifiers of all applications.

Let $\mathcal{P}_{Tv}$ be the set of unique identifiers of applications waiting to be processed at status v at time T. Let the cardinality of $\mathcal{P}_{Tv}$ be n. Let $1 \leq y(i) \leq n$, where $y(i)$ is an integer, be the rank of application i[1]. In particular, if $y(i) < y(j)$, then application i has a higher rank than application j at status v.

To measure the performance of a ranking model, several metrics have been proposed. We will discuss some of these metrics in detail later in the paper. Let M be a metric that we are interested in maximizing. Then, the objective is to assign a rank to each application at status v at time T in order to maximize M, i.e.,

$$\max_{y(i), i \in \mathcal{P}_{Tv}} M.$$

subject to

$$y(i) \neq y(j) \;\forall\; i, j \in \mathcal{P}_{Tv}.$$

[1]Allowing abuse of notation, we refer to the unique identifier i of an application as application i.

3.3. Process and Data

The data used for the experiments in this paper comes from a real lending process. Details of the data set source is withheld for confidentiality reasons. In section 3.3.1, we describe the process that we study in detail. Section 3.3.2 describes the data that we use for the analysis.

3.3.1. Process Description

The end-to-end MO process of the lender that we study involves 57 status'. Figure 32 is a simplified representation of the process flow with all the closing and non-closing status'. There is one closing status (Shipping - Final Action, status 49) and five non-closing status':

- 43. Loan Number Used in Error.
- 45. Withdrawn.
- 44. Approved Not Accepted.
- 46. Closed for Incompleteness.
- 47. Declined.

3.3.2. Data Description

The analysis and models that we develop are based on 279 mortgage applications that were made available to the authors. For each of these applications, it is known whether the application has closed or not. Customer specific attributes that are available for analysis do not include any attributes that are related to the identity of the customer, such as such as name, age, address, etc. The data attributes that

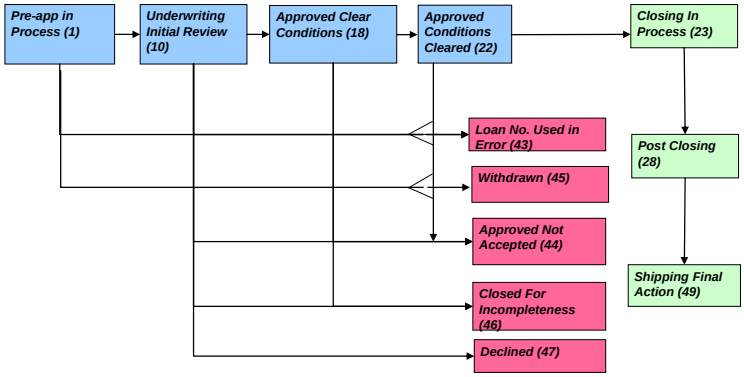


Fig. 32. Simplified Workflow Representation

Unique Identifier	**Credit Score**	**Interest Rate**	**Loan Amount**	**...**	**Outcome**
14652345	777	5.625	170,905		49
3540600	661	5.875	253,700		44
54482483	675	5.625	215,718		49
45615239	790	5.5	239,400		49
90006327	741	5.875	159,315		46

Table 2

Sample Data Set

are available to us may be classified into three types (an illustration of the data set in provided in Table 2):

1. Customer-specific attributes: (1) Credit score. (2) (Assets - Liabilities) / Income. If income is considered on a per monthly basis, then (Assets - Liabilities) / Income corresponds to the number of months of income that are required to accrue the net assets of the individual. (3) Appraised Value - Sale Price. The appraised value of a property corresponds to its assessed value by a qualified appraiser. The sale price pertains to the price that is being paid for the property. Thus, the difference between Appraised Value and Sale Price, i.e., Appraised Value - Sale Price corresponds to the "benefit" that is realized by paying less for the property than what it is worth.

2. Product-specific attributes: (1) Rate Type. A variable interest rate is one that is linked to the movement of an index of interest rates. A fixed interest rate, on the other hand, is pre-determined and does not change during the tenure of the loan. Rate type is a binary variable corresponding to whether a loan has a variable interest rate or a fixed interest rate. (2) Interest Rate. (3) Property Type. Applying for a secured loan to pay off a different loan secured against the same asset is called refinancing. Property type is a binary variable corresponding to whether a loan is purchase or refinance. (4) Loan amount is the amount of the loan requested. (5) Cashout is a binary variable corresponding to whether the applicant receives money or pays money at the end of the transaction, if accepted.

3. Process history attributes. Data pertaining to the history of status changes of applications along with the time that status is entered is available. Consider, for example, the sequence of status changes of an application up to a certain epoch of time (see Table 3) along with a description of the status' and corresponding entry times.

As a first step to building models to rank applications at any status, we first study how to rank applications at the initial status. The reader may note that all applications are processed through the initial status and work flow related information is unavailable at that point. At this status, applications contain information pertaining only to customer and product-specific attributes. Based on the insights that we derive from the initial status analysis, we study ranking models at any in-

Unique Identifier	Status	Description	Date & Time
40267891	7	Set up - In process	10/8/2008 14:00:07
40267891	10	Underwriting - Initial Review	10/8/2008 18:30:29
40267891	11	Underwriting - Second Review	10/9/2008 16:02:01
40267891	15	Underwriting - Pending Approval Com	10/9/2008 19:40:26
40267891	18	Approved - Clear Condition	10/22/2008 13:24:59
40267891	21	Approved - Exception Review	10/22/2008 19:34:16
40267891	10	Underwriting - Initial Review	10/22/2008 19:34:29
40267891	11	Underwriting - Second Review	10/22/2008 19:41:27
40267891	13	Application Pending	10/22/2008 20:00:48
40267891	18	Approved - Clear Conditions	10/22/2008 20:05:48
40267891	22	Approved - Condition Cleared	10/24/2008 20:09:45
40267891	23	Closing - In Process	10/24/2008 20:09:58
40267891	50	Closing - Document Out	10/27/2008 21:12:20
40267891	51	Closing - Funds Requested	10/28/2008 14:06:45
40267891	52	Closing - Funds Approved	10/28/2008 17:01:15
40267891	56	Closing - Exception Review	12/1/2008 14:51:26
40267891	52	Closing - Funds Approved	12/1/2008 14:55:36
40267891	53	Funded - Funds Disbursed	12/1/2008 14:56:09
40267891	28	Post Closing - In process	12/1/2008 14:56:18

Table 3

Workflow of a Sample Loan

Credit Score	#Closing Loans	#Non-closing Loans	Fraction of Closing Loans
(450,500]	0	1	0.00
(500,550]	0	2	0.00
(550,600]	0	1	0.00
(600,650]	2	11	0.15
(650,700]	12	8	0.60
(700,750]	10	15	0.40
(750,800]	29	17	0.63
(800,850]	26	6	0.81

Table 4

Loan Outcome by Credit Score

termediate status incorporating attributes based on the history of the applications.

3.4. Customer and Product Attributes Based Ranking Analysis

We begin with a discussion of the problem of ranking applications at the initial status with customer and product attributes of applications. The results and analysis that we present in this section are based on the data set that is available to us, which is randomly partitioned into a training data set and a test data set which contain 141 and 138 applications respectively[2].

[2]The test data set contains 59 non-closing applications and 79 closing applications.

We outline our approach and then detail the specifics. If the training data set is large enough, we can estimate the joint cumulative distribution function (c.d.f.) of $X_1, X_2, ...X_m$ of all applications that close, where $\{X_1, X_2, ..., X_m\}$ is the set of customer and product specific attributes. We can leverage this distribution to prioritize the applications using an appropriate *Scoring function* and sorting the applications according to their scores. One such intuitive Scoring function may be the joint c.d.f. or probability density function (p.d.f.) itself. We will discuss the functional form of the scoring function in detail below.

Firstly, given the limited size of our data set, we assume that $X_1, X_2, ...X_m$ are independent in order to estimate the joint c.d.f. For simplicity of discussion, let us restrict ourselves to one variable, for example, Credit Score and ask whether it might be better to use either the c.d.f. or p.d.f. for scoring. We first note that Credit Score is positively correlated with a Bernoulli random variable which equals one when the outcome is close and zero if it is non-close (Table 4 provides the fraction of closing applications, by Credit Score intervals, in the training data set.). Consequently, using a scoring function that is monotone increasing in the credit score may be preferable for prioritizing applications. Similarly, for some other variables (for example, interest rate), using a scoring function that is monotone decreasing in the interest rate may be preferable for prioritizing applications (Table 5 provides the fraction of closing applications, by Interest Rate, in the training data set.).

The set of customer and product specific attributes may be partitioned into two sets, $\mathcal{I}$ and $\mathcal{D}$, such that the scoring function is increasing (decreasing) in each variable in $\mathcal{I}$ ($\mathcal{D}$) independently. Thus, based on our assumption of independence

of attributes, we score an application with $X_i = x_i, i = 1, 2, ..., m$ by

$$\prod_{i \in \mathcal{I}} F_{ic}(x_i) \cdot \prod_{i \in \mathcal{D}} (1 - F_{ic}(x_i)), \tag{3.2}$$

where $F_{ic}(.)$ is the c.d.f. of X_i estimated from the training data set and corresponding to the applications that close only. We refer to this ranking method as *Multivariate Likelihood* (MV).

The reader may note that (MV) does not consider the distribution of non-closing applications in the scoring function. This could lead to unwarranted results, in come cases. Consequently, we may also score an application with $X_i = x_i, i = 1, 2, ..., m$ by

$$\frac{\prod_{i \in \mathcal{I}} F_{ic}(x_i) \cdot \prod_{i \in \mathcal{D}} (1 - F_{ic}(x_i))}{\prod_{i \in \mathcal{I}} F_{in}(x_i) \cdot \prod_{i \in \mathcal{D}} (1 - F_{in}(x_i))}, \tag{3.3}$$

where $F_{in}(.)$ is the c.d.f. of X_i estimated from the training data set and corresponding to the applications that non-close only. We refer to this ranking method as *Multivariate Likelihood Ratio* (LR).

The reader may note that (MV) and (LR) are simple and scalable ranking models. These models also allow for easy identification of attributes that cause an application to non-close with a high likelihood and suggest corrective action. We will discuss this in detail later in the paper (see Section 6).

We compare the performance of MV and LR with Support Vector Machines(SVM)[3] and that of a perfect ranking model, i.e., a model that ranks all non-closing applications above closing applications and a first-in-first-out (FIFO) model, i.e., a model

[3]We use classification methods such as SVM to train, ranking applications in the test data set by their likelihood of closing.

that ranks applications at a status in the order in which they are received at that status. We compute non-parametric estimates of the attributes to estimate the performance of (MV) and (LR). We also examine whether some attributes belong to well-known parametric distribution families. A chi-squared test at 95 percent significance reveals that $(Assets - Liabilities)/Income$ is normally distributed. However, we are unable to determine the distribution of several attributes at 95 percent significance. We also compute estimates of the distribution of all attributes, assuming them to be Normal and independent. Since the performance of our models with the the non-parametric assumption is inferior to that with the Normal distribution assumption, the results for that case are omitted. Below, we refer to models with the Normal distribution assumption as LR and MV.

To evaluate the performance of our ranking methods, we use Precision-Recall curves. Precision and Recall are two widely used measures for evaluating the quality of results in information retrieval and statistical classification. *Precision* represents the ability to retrieve top-ranked applications that are relevant while *recall* evaluates the ability of the search to find all of the relevant applications in the corpus. Formally, precision and recall are defined as follows:

$$\begin{aligned} Precision &= \frac{\text{Number of relevant applications retrieved}}{\text{Number of applications retrieved}}, \\ Recall &= \frac{\text{Number of relevant applications retrieved}}{\text{Total number of relevant applications}}. \end{aligned}$$

In our setting, we are interested in retrieving two ranked lists, one with non-closing applications as relevant and the other with closing applications as relevant. In the main body of the paper, we restrict ourselves to the first case, i.e., the non-closing

Interest Rate	# Closing Loans	# Non-closing Loans	Fraction of Closing Loans
3.75	1	0	1.00
3.875	1	0	1.00
5	7	0	1.00
5.125	5	2	0.71
5.25	11	0	1.00
5.375	2	4	0.33
5.5	9	6	0.60
5.625	8	4	0.67
5.75	17	7	0.71
5.875	8	6	0.58
6	3	5	0.38
6.125	3	3	0.50
6.25	2	9	0.18
6.375	0	2	0.00
6.5	2	11	0.15

Table 5

Loan Outcome by Interest Rate

applications are relevant. The insights for the second case are similar in terms of which models perform better. Therefore, we merely include the results for this case in the Appendix, for ease of presentation.

In order to enable comparison of two Precision-recall curves, I and II, we note that curve I dominates curve II if the precision, at every value of recall, is lower for curve II than curve I (see [113] and [38]).

We now discuss the performance of MV, LR and SVM. The Precision-recall curves for these three methods, along with FIFO and Perfect is provided in Figure 33. Our results indicate that each of the three methods, MV, LR and SVM dominate FIFO. Among the three methods, SVM is dominated by both MV and LR.

Different metrics have been used for evaluating the performance of ranking models, the two most popular of which are Average Precision and R-precision (see [7]). In our setting, Average precision is the average of the precisions of all non-

	Average Precision	R-Precision
Perfect	1.00	1.00
MV	0.85	0.48
LR	0.85	0.50
SVM	0.78	0.47
FIFO	0.47	0.45

Table 6

Performance of Ranking Models at Initial Status

closing applications while R-precision is the precision at rank R. We present our results for the case where R is the number of non-closing applications. The results corresponding to evaluation of the metrics for MV, LR and SVM are provided in Table 6. The insights from this table are similar to those derived from Figure 33.

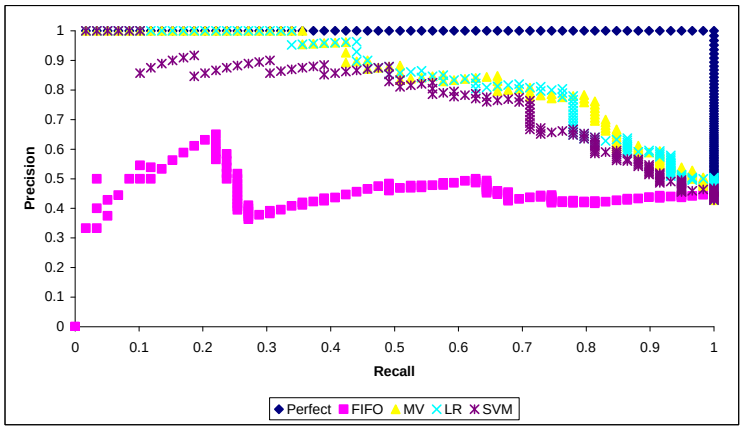


Fig. 33. Precision-Recall Curves at Initial Status

Based on the ranking models for the initial status, we next develop models for intermediate processing. The analysis incorporates not only the product and customer specific attributes that were used for the initial status analysis but also workflow attributes. We discuss this in detail in the next section.

3.5. Customer, Product and Workflow Attributes Based Ranking Analysis

While the results of the previous section are encouraging, an interesting question arises while ranking applications at an intermediate status - Can we improve the performance of our ranking models that use only customer and product-specific attributes by leveraging historical state information? The following analysis answers this question. Throughout this section, we use status u as a generic status for which we are developing a ranking model.

Recall that the historical information available at time T is given by

$$\mathcal{H}_T \quad = \quad \{(i, v, t_{iv}) \mid t_{iv} \leq T \;\; i \in \mathcal{S}, v \in V(G)\}. \tag{3.4}$$

The large state space of the history of the applications makes the analysis intractable. Thus, we collapse the state space to a single dimension for each application. The reduced state space that we consider captures information pertaining to the number of visits to status u, the status for which we are building the ranking model. Clearly, the reduced state space that we consider may not be a sufficient statistic.

Define the number of visits to status u by application i as

$$l_{iu} = |\{(i, u, t_{iu}) \mid t_{iu} \leq T \;\; i \in \mathcal{S}, u \in V(G), (i, u, t_{iu}) \in \mathcal{H}_T\}|,$$

where $|Y|$ denotes the cardinality of set Y. In the sequel, we drop the subscripts u and i and refer to l_{iu} as *Visits*, for ease of presentation.

The attribute *Visits* was chosen for two reasons: (1) Domain experts that we interacted with pointed out that applications which have considerable re-work are the ones that are likely to non-close, and (2) the data supported this hypothesis for

several status' that we considered (A table supporting observation (2) is excluded for confidentiality reasons).

We first discuss how to construct the test data set. We begin with the test data set, $\mathcal{DA}$ that we used for evaluating our models in Section 5. For ease of exposition, we do not introduce additional notation to explain this construction. Recall that we are building our ranking models at time T at status u. Thus, we are interested in constructing a subset of the applications of $\mathcal{DA}$ that that are waiting to be processed at status u at time T, irrespective of whether they have been processed at status u before time T or not. Unfortunately, this constraint reduces the number of applications in the test data set $\mathcal{DA}$ significantly since (1) the data set contains applications over a six month period and, at any time, only a fraction of applications are being processed, and (2) the process contains over four dozen tasks and at any time, of the fraction being processed, only a fraction are at status u.

To alleviate this problem, we construct an approximate test data set. We relax the constraint (2) and consider all applications that are being processed at time T and belong to $\mathcal{DA}$. We could adopt an alternative approach, i.e., relax constraint (1) and consider all applications with either one entry or multiple entries in the test data set corresponding to each visit to status u. However, the alternative approach introduces additional uncertainty without any obvious benefits over the first approach (If we were to include one entry corresponding to each visit to status u for each application, the question of which entry it would be has to be addressed. Similarly, if we were to include all entries in the test data set, it becomes biased towards applications with multiple visits to status u).

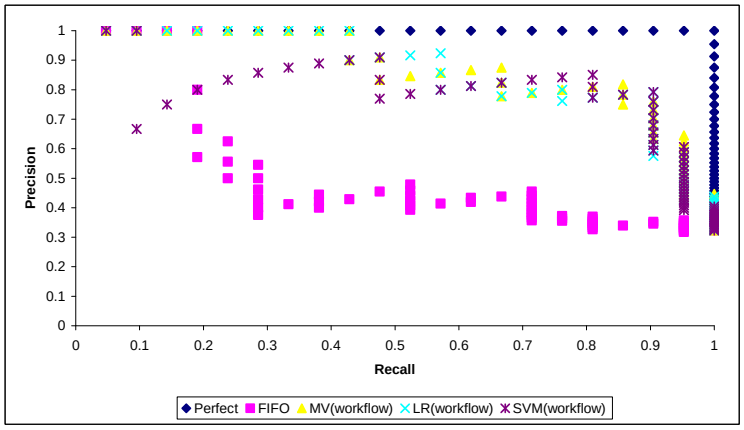


Fig. 34. Precision-Recall Curves with Workflow Attributes

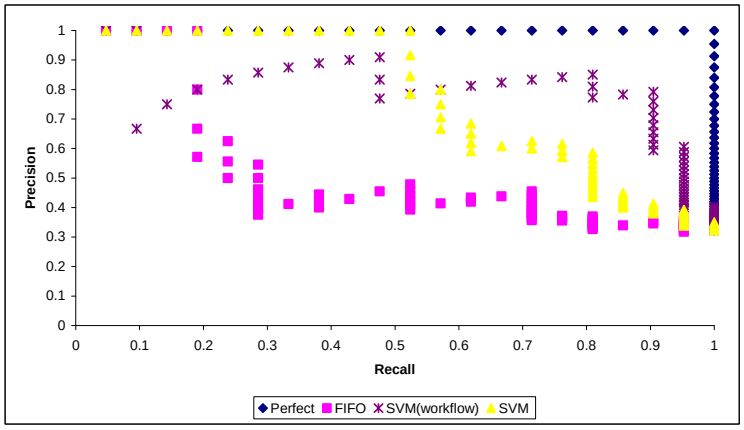


Fig. 35. Precision-Recall Curve for SVM

Let us now discuss how to incorporate the workflow information pertaining to *Visits* into the MV and LR models that we discussed in Section 5. For the purpose of building the model, *Visits* may be thought of "just another independent attribute" and added to these models. However, while evaluating the score of an application at time T, the fact the the number of visits information that is available is "partial" needs to be considered, i.e., if we know at time T, an application has made two prior visits to status 10, all we know is that it will make two or more visits by the time it is processed. Thus, while evaluating the score of an application based on this partial information, the conditional probability of the number of visits to state

u has to be incorporated. Suppose that l visits were made to status u prior to time T. Then, the following term when multiplied by the score of the MV model, i.e., expression 3.2, provides the new score:

$$\sum_{j=l}^{\infty} P(L_c = j/L_c \geq l) \cdot Score_v(L_c = j), \tag{3.5}$$

where L_c is a random variable corresponding to the number of visits made by a closing application to status u and $Score_u(L_c = j)$ is the score if exactly j visits were made to status u. Similarly, the following term when multiplied by the score of the LR model, i.e., expression 3.3, provides the new score:

$$\frac{\sum_{j=l}^{\infty} P(L_c = j/L_c \geq l) \cdot Score_u(L_c = j)}{\sum_{j=l}^{\infty} P(L_{nc} = j/L_{nc} \geq l) \cdot Score_u(L_{nc} = j)}, \tag{3.6}$$

where L_{nc} is a random variable of the number of visits made by a non-closing application to status u and $Score_u(L_c = j)$ is the score if exactly j visits were made to state u. Similar to the logic that we applied in Section 5, we define

$$Score_u(X = x) = P(X \geq x).$$

Next, we describe how to incorporate the workflow information pertaining to *Visits* into the SVM model that we discussed in Section 5. As in the case of MV and LR, for the purpose of training the model, *Visits* may be thought of "just another attribute". However, while evaluating the score of an application at time T which has made l visits to state u and whose predicted score by the trained model is $Score_{svm}(L = l)$, we make the following correction to account for the partial information observed:

$$\sum_{j=l}^{\infty} P(L = j/L \geq l) \cdot Score_{svm}(L = j), \tag{3.7}$$

where L is a random variable corresponding to the number of visits made by an application to status u.

We now discuss the performance of the three methods, MV, LR and SVM with workflow attributes[4] for one epoch of time at status 10 at which the test data set was constructed[5] (see Figures 34 - 37). In each of these figures, when the legend has workflow included in parenthesis, it refers to the model with workflow attributes; otherwise, the model does not incorporate workflow attributes. We first note that Precision-Recall curve of none of these methods with the workflow attribute dominates the curve without the workflow attribute. However, for both the metrics that we consider, the Average Precision and R-precision, each method performs better with the workflow attribute (see Table 7), with some metrics giving as much as a nine percent improvement. We thus conclude that the workflow attribute that we consider, *Visits* has significant explanatory power for ranking.

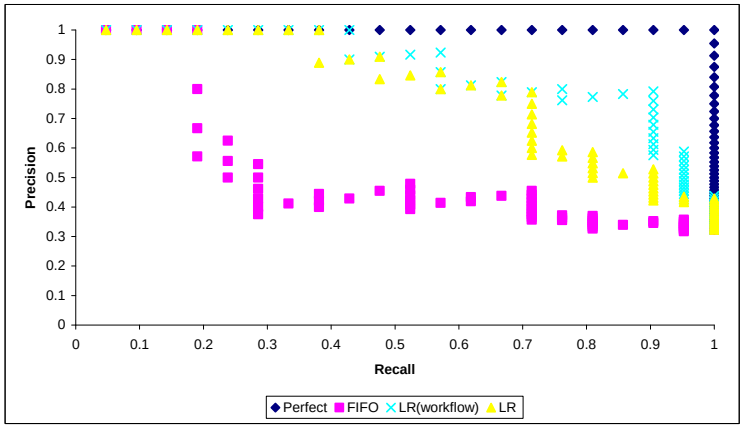


Fig. 36. Precision-Recall Curve for LR

[4]We find that *Visits* is a Geometric random variable with parameter 0.7 through a chi-squared test.

[5]The test data set contains 21 non-closing applications and 44 closing applications.

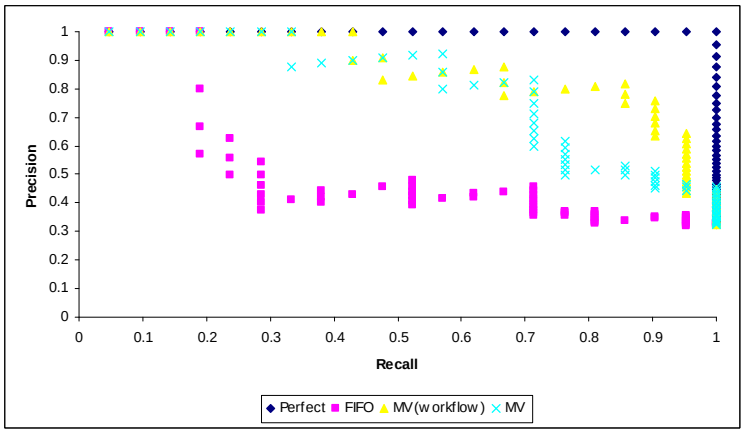


Fig. 37. Precision-Recall Curve for MV

Model	Average Precision	R-precision
Perfect	1.00	1.00
MV(workflow)	0.88	0.45
LR(workflow)	0.87	0.44
SVM(workflow)	0.82	0.40
MV	0.81	0.44
LR	0.81	0.43
SVM	0.79	0.35
FIFO	0.54	0.33

Table 7

Metrics for Ranking Models at Status 10

Finally, recall that the results pertaining to the ranking models that we presented above were for one snapshot of time. To evaluate the robustness of our results, we repeat our experiments at four epochs, say, t_1, t_2, t_3, t_4 at status 10 for (MV) with the workflow attribute. A summary of the results (see Table 8) shows that the performance is comparable at all four epochs.

3.6. Discussion

Not only are MO service providers interested in developing models for ranking but also see value in identifying attributes that are "responsible" for an application to non-close. Further, if the identified attributes were actionable, i.e., they are

Epoch	Average Precision	R-precision
t_1	0.92	0.65
t_2	0.81	0.45
t_3	0.78	0.32
t_4	0.65	0.50

Table 8

Robustness of Ranking Results

Attribute	Attribute Value	Score
Credit Score	595	8E(-5)
Property Type	1	0.56
Interest Rate	5	0.72
Cashout	1	0.70
Loan amount	120000	0.18
Appraised Value - Sale Price	0	0.44
(Assets -Liabilities)/Income	-49.93	0.19

Table 9

(MV) model, by attribute, for application X

not exogenous, then appropriate corrective action be suggested so as to increase the likelihood of closing, thus improving the pull-through rate. Consequently, we examine this issue in the context of our analysis and results.

The results presented in Sections 5 and 6 clearly demonstrate that the likelihood and likelihood ratio based distribution function estimation models outperform

Attribute	Attribute Value	Score
Credit Score	736	0.24
Property Type	1	0.56
Interest Rate	6.5	0.01
Cashout	0	0.03
Loan amount	160000	0.32
Appraised Value - Sale Price	0	0.44
(Assets -Liabilities)/Income	11.54	0.83

Table 10

(MV) model, by attribute, for application Y

classification techniques such as Support Vector Machines. Not only do these models perform better but also have the added advantage of allowing "attribute-wise comparison". We explain what we mean by this below.

We begin by suggesting one intuitive scheme for resolving the problem of attribute identification for (MV) at the initial status: Let s_i be the score associated with attribute $X_i = x_i$, $i = 1, 2, ..., m$ for some application, i.e., $s_i = F_{ic}(x_i)\ \forall\ i \in \mathcal{I}$ and $s_i = 1 - F_{ic}(x_i)\ \forall\ i \in \mathcal{D}$. For ease of presentation and without loss of generality, we assume, in this section, that $s_1 \leq s_2 ... \leq s_m$. The corrective action that we suggest is to increase / decrease the value of attribute X_1, keeping other attribute values fixed, so that $s_1 = s_2$.[6]

We discuss how this scheme works through two examples below: Consider two applications, say X and Y, which have been deemed likely to non-close at the initial status. The attribute values and the corresponding scores are provided in Tables 9 and 10. Based on the scheme suggested above, Credit Score and Interest Rate are the identified attributes for applications X and Y respectively. However, since Credit Score is exogenous, we suggest lowering the interest rate offer for application Y and no action for application X. Clearly, one can think of other schemes as well.

A similar scheme may be suggested for the (LR) model by defining the individual attribute scores in terms of their likelihood ratios. The details directly follow from our discussion above and are, hence, omitted.

[6]If X_1 is a discrete random variable, then we would change the value of the attribute just enough such that $s_1 \geq s_2$. For ease of exposition, this issue is ignored in the paper.

3.7. Related Work

There is a vast literature on ranking models. Both supervised and unsupervised techniques have been widely used for classification and ranking. We refer the reader to [54] and [133] for details of two popular supervised techniques, regressions and support vector machines respectively. Likelihood and likelihood ratio based models have also been widely used. In fact, likelihood ratio is the minimum probability-of-error decision scheme for classification (see [111]). Unsupervised techniques have been applied to a number of different domains as well. For example, [65] use Behavior Shift Models to determine exceptions in the Travel and Entertainment Expenses of a company.

Much of the literature applying classification techniques to the mortgage industry focuses on mortgage delinquency. [166] study the problem using Logistic Regressions. [69] discusses the performance the Radial Basis Function (RBF), which combines the mathematical complexity of neural networks with a comprehensive visualization in IBM's Intelligent Miner for mortgage scoring. [50] is a case study of various data mining models to assess mortgage risks pertaining to delinquency.

The extant literature on modeling, execution and optimization of workflows is vast (see [27, 2, 34]). For example, [130] study the problem of optimizing workflow by reducing the number of steps to resolution in the context of problem tickets and resolution groups.

Related to our study is the problem of prioritizing multi-class applications or jobs that arrive to a queue. Associated with each class are due dates and service level penalties. Most of these models either leverage scheduling heuristics such as

Shortest Processing Time, Earliest Due date First, etc. (see [108]) or asymptotic properties of Service Systems (see, for example, [134]).

3.8. Conclusion and Future Work

In this chapter, we present an investigation of different ranking models that incorporate customer, product and workflow attributes at any status in the MO process. We propose a workflow attribute with a reduced state-space based on the number of visits to a particular step or re-work. We find that incorporating this workflow attribute results in improvement of the density modeling techniques that we develop over those that incorporate only customer and product specific attributes by $8-9$ percent in Average Precision. The simple and scalable density modeling techniques allow for easy identification of applications that are likely to non-close and consequent corrective action such as change in the attributes of the mortgage product being offered. Further, our results indicate that these models perform better than classification based models like Support Vector Machines for ranking, showing a 8 percent improvement in Average Precision.

A promising future research direction is the development of other attributes with a reduced state-space apart from *Visits*. For example, consider the number of status' visited by an application prior to time T or the duration the application has been in process. Extending the reduced state-space beyond a single dimension to two dimensions would also be of interest.

The prioritization model can be applied to other application, such as credit card processing as well. Correspondingly, we can consider different features to improve the prediction accuracy and maximize the business benefit.

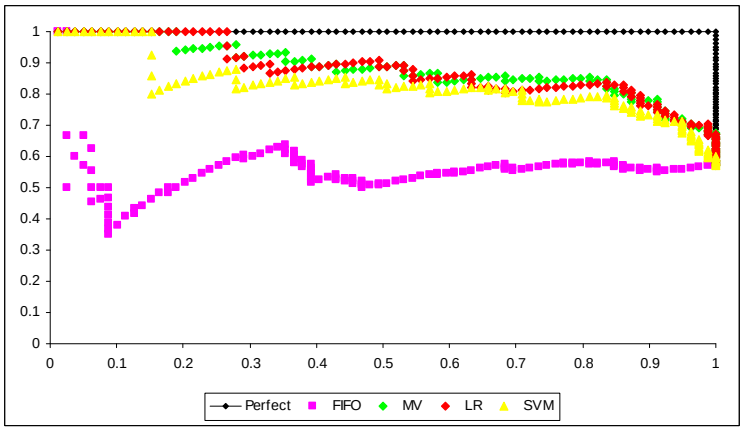


Fig. 38. Precision-Recall Curve for Initial Status with Relevant Applications as Closing

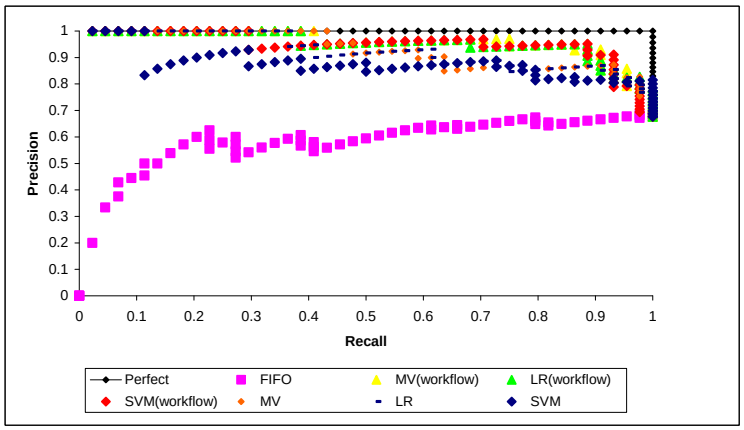


Fig. 39. Precision-Recall Curve for Status 10 with Relevant Applications as Closing

4. AN EFFICIENT SERVICE DEMAND FORECASTING MODEL

We study the problem of forecasting service request volumes, by task, of a cloud service process. We model the evolution of service volumes as a discrete time stochastic process. We develop two predictors to forecast requests volumes. The first predictor, which we refer to as Semi-markovian Predictor (SMP) is based on the assumption that the underlying stochastic process is Semi-markovian. Since the standard approach to evaluate forecasts under a Semi-markovian assumption requires either a recursive computation or simulation, we develop an approximate expression to evaluate the forecasts easily. The second predictor, which we refer to as Weighted Markvovian Predictor (WMP), is based on classifying loans into two types, those that are predicted to close and those that are predicted to not close, and assuming each of these loan types follows a Markovian process. We benchmark the performance of the two predictors that we develop against a predictor based on the assumption that the underlying stochastic process is Markovian (MP). Our numerical experiments indicate that SMP performs better than MP when the forecast duration is small (Average improvement percentage in Mean Square Root Error of SMP over MP is 42.5 %). A mortgage origination is applied as the application scenario in this chapter.

4.1. Introduction

An end-to-end Mortgage Origination (MO) process begins with the submission of a mortgage application by an applicant to a lender and ends with one of the following outcomes: closing, i.e., loan approved by the lender and accepted by the applicant or non-closing, i.e., loan either rejected by the lender, or approved by the lender and

not accepted by the applicant. The outcome of the loan is primarily dependent on the characteristics of the loan such as Credit Score, Income, Assets, etc. (a detailed discussion of the characteristics is provided later in the paper). The MO process is composed of several pre-defined tasks, each of which may require specialized skills. A mortgage application passes through different tasks during its lifecycle, the next task possibly dependent on the current task (for example, a loan may be routed to third party evaluators for independent assessment in case of insufficient proof of income).

MO managers are interested in long-term loan volume forecasts to make intelligent hiring decisions as well as short-term forecasts for shift scheduling. Long-term loan volume forecasts at a task-level depend on exogenous factors such as federal interest rate, since the number of loans at a particular task would depend on the rate at which new loans enter the MO process. On the other hand, short-term weekly forecasts may be assumed to be independent of exogenous factors. The reader may note that hiring decisions, usually made quarterly and shift scheduling, typically done weekly, are critical activities for MO managers as the skills required for the different tasks are not always transferrable. The focus of this paper is the development of *easy-to-evaluate short-term forecasts* of loan volumes for the MO process.

The remainder of the section is organized as follows: In Section 4.1.1, we review the relevant literature. We summarize the main contributions of our work in Section 4.1.2. In Section 4.1.3, we introduce the notation and formally state the problem.

4.1.1. Related Literature

In this section, we briefly summarize the relevant literature on forecasting. We first summarize the work on Markovian predictors. We then review papers that assume a Semi-markov process for forecasting.

The Markovian assumption is widely used in many domains such as as telecommunications and manufacturing systems. Here, we briefly refer to two such papers. Chong et al. [28] use a Markov model to predict the demand of storage space in a soft drink company. Mannila et al. [88, 89] also use a Markovian approach for telecommunication alarm management.

A Semi-markov process based predictor has also been widely used by researchers. For example, Mcclean et al. [91] use a Semi-markovian approach to model a human resource system, which combines a duration dependent stay in a grade with a transition matrix of movements between grades, to predict the system's future development. Trivedi et al. [148] develop a Semi-markov model to study transitions of physicians, nurse practitioners, and physician assistants between different settings and locations within a geographic area. Lee et al. [82] model user mobility in WLANs by a Semi-markov process, and obtain the transition probability matrix from the association history of WLAN users. With the steady-state characterization of user mobility in WLANs, they estimate the long-term wireless network usage among different access points. Several papers provide comparisons of Semi-markov

and Markov based predictors, for example, see Valliant et al. [160].

We also allude to some classification methods that have been widely used (see, for example, Linear Regression [54], SVM (Support Vector Machines) [133], and Likelihood Ratio [131]) for discrete choice modeling (for example, to predict whether a loan closes or not).

4.1.2. Our Contributions

We develop two new predictors to forecast loan volumes in this paper. We now discuss how these predictors contribute to the literature on forecasting. Our first predictor, SMP, is based on the assumption that the underlying stochastic process in Semi-markovian. While several papers make this assumption for forecasting, they employ either a recursive computation or a simulation driven approach to estimate the forecasts. Our approach, on the other hand, is focused on developing an approximate expression, which is easy to evaluate for forecasting. Our second predictor, which we refer to as WMP, incorporates a classification model such as Regression, SVM, etc. into standard classical forecasting models under the Markovian assumption. To the best of our knowledge, this is the first such attempt in the literature.

4.1.3. Problem Definition and Notation

The MO process consists of several tasks that may be thought of as the nodes of a directed graph whose edges are transitions between tasks. The reader may note that a loan may be in only one task at a time and may transition to another task

depending on the current task.

Let the MO process consist of n tasks, which are indexed as $1, 2, , n$. Associated with each loan is a loan number, which is a unique identifier for the loan. On any given day, a loan is at one of the tasks (we are interested in forecasting loan volumes at the granularity of one day) in the process. We are interested in studying the evolution of the number of loans at each task with time. In order to study this, we define a stochastic process whose state is the n-dimensional vector of volumes at each task. We first introduce some notation. Let $S_{i,t}$ be the set of loan numbers at task i on day t. Let S_t be the set of all loans on day t, i.e.,

$$S_t = \bigcup_{i \in \{1,2,...,n\}} S_{i,t}.$$

Let loan k be at task Z_t^k on day t. Then, the volume of loans at task i on day t is given by

$$X_{i,t} = \sum_{k \in S_t} I\{Z_t^k = i\}, \tag{4.1}$$

where $I(.)$ is the indicator function defined as $I(x = y) = 1$ if $x = y$ and 0 otherwise. We use

$$Y_t = (X_{1,t}, X_{2,t}, ..., X_{n,t})$$

to denote the evolution of the n-dimensional vector of loan volumes by task.

On day T, we have historical information regarding the task associated with each loan as well as the characteristics of the loan. Given the loan history up to time

T, the problem is to forecast the loan volume $m > 0$ days hence. We will denote the estimate of loan volumes on day $T + m$ by

$$\hat{Y}_{T+m} = (\hat{X}_{1,T+m}, \hat{X}_{2,T+m}, ..., \hat{X}_{n,T+m}) \tag{4.2}$$

where $\hat{X}_{i,T+m}$ is the forecast of loan volumes at task i m days into the future (on day $T + m$).

Several metrics have been used to evaluate the performance of forecasting policies, the most popular of which is Mean Square Root Error (MSRE). We use this metric to evaluate the predictors that we propose. The expression for MSRE is given by

$$\frac{\sqrt{\sum_{i=1}^{n} \left(X_{i,t} - \hat{X}_{i,t}\right)^2}}{\sum_{i=1}^{n} X_{i,t}}. \tag{4.3}$$

The reminder of the paper is organized as follows: In Section 4.2, we introduce three predictors. We evaluate the performance of the predictors that we develop numerically in Section 4.3.

4.2. Markovian, Semi-Markovian and Weighted Markovian Predictors

In this section, we introduce several predictors to forecast loan volumes. Since the stochastic process Y_t may depend on the history of sequence of tasks of loans as well as the loan characteristics, estimating the transition probabilities is intractable without making assumptions on the evolution of the stochastic process Y_t. In Section 4.2.1, we introduce a forecasting model assuming that Y_t is Markovian. Section 4.2.2

assumes that Y_t follows a Semi-markov process. Finally, in Section 4.2.3, we classify the loans into two classes, close and non-close, and incorporate the likelihood-to-close (Given the loan characteristics, how likely is the loan to close) information to forecast loan volumes.

4.2.1. Markovian Predictor

We first introduce a simple Markovian model in order to predict loan volumes. Under the assumption that Y_t is Markovian, we estimate the m day steady-state transition probability matrix, $P = \{p_{ij}\}_{n\times n}$ of Y_t. We can then forecast loan volumes (we refer to this forecasting rule as MP) on day $T + m$ by

$$\hat{X}^{MP}_{i,t+m} = \sum_{j=1}^{n} p_{ji} \cdot X_{i,t}. \tag{4.4}$$

The advantage of MP is that it is easy to evaluate but this comes at the cost of making strong assumptions on Y_t. Such an approach does not incorporate the fact that loans are heterogeneous, i.e., they transition from one task to another depending on their likelihood-to-close. In fact, as noted by Shao et al 2009 [131], the sequence of paths followed by a loan is correlated with whether the loan will close or not. Thus, incorporating this information may reduce forecast errors. Further, the Markovian assumption itself may be poor. Below, we introduce two forecasting models, the first relaxes the Markovian assumption while the second incorporates the fact that the loans are not homogeneous.

4.2.2. Semi-Markovian Predictor

In this section, we assume that Y_t follows a Semi-markov process. Recall that a Semi-markov process is a continuous time stochastic process in which the embedded

jump chain (the discrete process that determines the next task for any given task) is a discrete time Markovian process and the time between transitions (holding times or time between between jumps) are generally distributed random variables (which may depend on the two tasks between which the move is made).

Before we introduce our Semi-markov process based loan volume predictor, we first introduce an assumption and some notation. We assume that the duration a loan spends at a task is independent of the characteristics of the loan as well as next task that a loan transitions to. This assumption does not always hold - for example, discussions with Subject Matter Experts have revealed that Underwriters spend more time on loan applications with less documentation on applicant assets. However, this assumption simplifies our analysis significantly.

Let $F_i(.)$ be the cumulative distribution function of the random variable D_i corresponding to the duration (in days) that is spent by a loan at task i. Given that a loan k has been in state i for d_k days and is currently in state i, the probability that the loan will still be in state i after m days is given by

$$\alpha_k^i(m) = \frac{P\{D_i \geq d_k + m\}}{P\{D_i \geq d_k\}} = \frac{1 - F_i(d_k + m)}{1 - F_i(d_k)}.$$

Let $\beta_k^j(u)$ be the probability that loan k will be in task j on day $T+1$, $T+2$,..., $T + u - 1$ and will transit from task j on day $T + u$, given that it was in task j on day T, i.e.,

$$\beta_k^j(u) = \alpha_k^j(u-1) - \alpha_k^j(u).$$

We may predict loan volumes without making any further assumptions. However, such an approach would require a recursive computation or a simulation driven approach. Since we are interested in developing easy-to-evaluate predictors, we next make another assumption that will allow evaluation of our forecasts easily, for example, using a spreadsheet. We only consider loans that were at task i on day T or loans that transitioned from any task j directly to task i. In particular, we do not consider loans that transitioned from task $j \neq i$ to task $k \neq i$ to task i before $T + m$. With this assumption, we estimate loan volumes on day $T + m$ by

$$\begin{aligned} \tilde{X}^{SMP}_{i,t+m} &= \sum_{k \in S_{i,T}} \alpha^i_k(m) \\ &+ \sum_{j=1, j\neq i}^{n} \sum_{k \in S_{j,T}} \sum_{1 \leq u \leq m} \beta^j_k(u) \cdot p_{ji} \cdot \alpha^i_k(m-u). \end{aligned}$$

The first term on the right hand side of the above equation corresponds to the event that a loan will stay at task i up to $T+m$, given that it was at task i on day T. The second term corresponds to the event that a loan will transition to task i from any other task. We note that the above expression is not exact since it ignores events with more than one transitions from task j to task i between time T and time $T+m$. The reader may recall that since we are interested in making near-term forecasts only, we expect the above approximation to be good, especially for shorter forecast durations.

In order to develop a predictor that is easy to evaluate, we develop an upper

bound for $\tilde{X}^{SMP}_{i,t+m}$. Note that

$$\begin{aligned} A &:= \sum_{1\le u\le m} \beta^j_k(u)\cdot\alpha^i_k(m-u) \\ &= \sum_{1\le u\le m} (\alpha^j_k(u-1)-\alpha^j_k(u))\cdot\alpha^i_k(m-u) \\ &\le \sum_{1\le u\le m} (\alpha^j_k(u-1)-\alpha^j_k(u))\cdot 1 \\ &= 1-\alpha^j_k(m), \end{aligned}$$

where the second equality follows from the definition of $\beta^j_k(u)$, the inequality from the fact that $P\{D_i \ge d_k+m\} \le P\{D_i \ge d_k\}$ and the last equality from a recursive computation. The above analysis allows us to develop the following predictor (we refer to this forecasting rule as SMP):

$$\begin{aligned} \tilde{X}^{SMP}_{i,t+m} &\le \hat{X}^{SMP}_{i,t+m} \\ &= \sum_{k\in S_{i,T}} \alpha^i_k(m) \\ &+ \sum_{j=1,j\ne i}^{n} \sum_{k\in S_{j,T}} (1-\alpha^j_k(m))\cdot p_{ji}. \end{aligned}$$

4.2.3. Weighted Markovian Predictor

In this section, we incorporate the heterogeneous nature of loans in our predictions by considering two classes of loans: those that close and those that non-close. Our approach allows us to use any classification model that predicts whether a loan will close or not depending on the attributes of the loan. Several classification and ranking models have been used in the context of loan origination (see Shao et al. 2009). Below, we present an approach that combines the likelihood-to-close estimates of loans and the attribute independent Markovian forecasting predictor

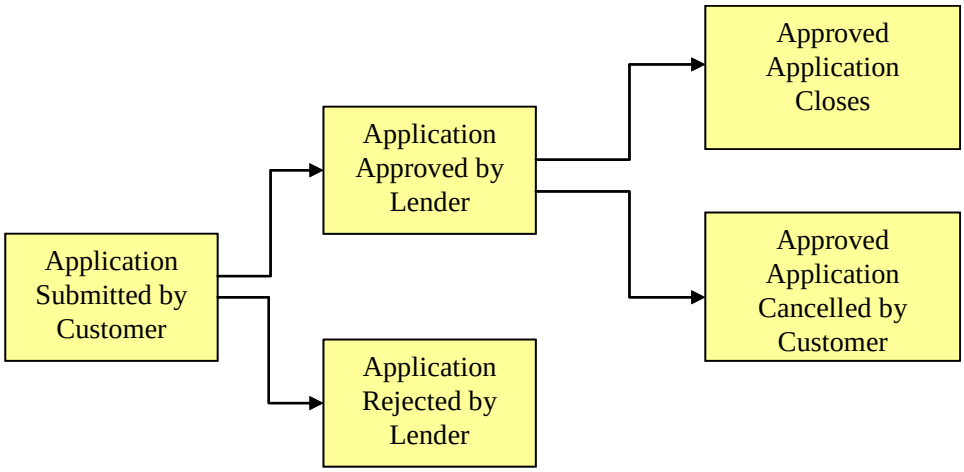


Fig. 40. Flowchart of MO Process

MP. We estimate two transition probability matrices, one for loans that close and one for loans that do not close. We denote the estimates of the m-day transition probability matrix of loans that close and those that do not by $P^c = \{p^c_{ij}\}_{n\times n}$ and $P^{nc} = \{p^{nc}_{ij}\}_{n\times n}$ respectively. Let $X^c_{i,t}$ and $X^{nc}_{i,t}$ be the volume of loans at task i that are predicted to close and non-close respectively. We suggest the following rule (we refer to this rule as the Weighted Markovian Prediction or WMP) for estimating the loan volume at day $T + m$:

$$\hat{X}_{i,t+m} = \sum_{j=1}^{n} (p^c_{ji} \cdot X^c_{i,t} + p^{nc}_{ji} \cdot X^{nc}_{i,t}). \tag{4.5}$$

4.3. Numerical Results

In this section, we investigate the efficacy of the predictors that we develop for a real lending process. The forecasting methods that we develop in this paper are evaluated for a MO process consisting of 58 tasks. A graphical representation of the process is provided in Figure 49.

The analysis that we present is based on 1332 mortgage applications that were made available to the authors. For each of these applications, it is known whether the application has closed or not as well as the duration each loan spent at a particular task. For each application, we have data pertaining to three types of attributes: Customer, Product and Environment. Below, we merely summarize the attributes that are available for our analysis. We refer the readers to Shao et al [131] for details on attribute definitions.

- Customer-specific attributes: (1) Credit score. (2) (Assets - Liabilities) / Income. (3) Appraised Value - Sale Price. (4) Debt to Income.
- Product-specific attributes: (1) Rate Type (variable inter-est rate or a fixed interest rate) (2) Interest Rate. (3) Property Type (Purchase or refinance). (4) Loan amount. (5) Cashout. (6) Loan to Value. (7) Finance charge.
- Environment attributes: The U.S. federal interest rate is extracted from the Federal Reserve website biweekly.

Shao et al [131] compare different classification and ranking techniques, for example, SVM, Regressions and Likelihood-ratio in order to predict whether loans are likely to close or not using the above attributes. For this data set, they find that the Likelihood-ratio based technique performs better than SVM and Regressions. We use the same technique to evaluate SMP in this paper.

We evaluate the performance of the SMP (using the non-parametric duration distribution estimates) and WMP predictors (using the Likelihood Ratio classifier) on the metric that we discussed in Section 1.3: MSRE. We benchmark the performance of these predictors against the MP predictor. Our numerical experiments are presented for five different dates as the snapshots to make forecasts. Table 1 provides a summary of the results, providing average improvements in SMP and WMP over MP for a given forecast duration across all snapshots that we consider (for each snapshot, we refer to these percentage improvements as PSMP and PWMP). The detailed error percentages for each predictor for one, two, three, four and five day forecasts for each snapshot is provided in the appendix.

We first note that, on average, each of the SMP and WMP predictors performs better than MP for some forecast durations and worse for others. However, the better of the SMP and WMP predictors performs at least as well as the MP predictor.

We now comment on the performance of both our policies with the forecast duration. We note that the average error percentage improvement of SMP over MP is decreasing with the forecast duration. The reader may recall that we ignore more than one transitions between tasks during the forecast horizon when developing the expression for the SMP predictor. This assumption is good for small forecast durations but poor for larger forecast durations. Also, we observe that the average error percentage improvement of WMP over MP is increasing with the forecast duration. The intuition behind this observation is that the sequence of future tasks depends on

Forecast Duration (days)	Average PSMP	Average PWMP
1	59	-29
2	24	-12
3	- 3	0
4	-18	9
5	-28	14

Table 11

Average MSRE over Five Snapshots

the characteristics of the loan. Further, this information captures greater predictive power when the forecast durations are larger since more loans transition during a larger forecast duration.

We summarize our results by forecast duration interval as follows: For small forecast durations (1-2 days), the average percentage improvement in MSRE of SMP over MP is 42.5%. For large forecast durations (4-5 days), the average percentage improvement in MSRE of WMP over MP is 11.5%. Based on these results, we recommend using SMP when the forecast duration is small, WMP when the forecast duration is large and MP for medium forecast durations.

Snapshot	Forecast duration (days)	MP	SMP	WMP	PSMP	WSMP
1	1	7.14	1.40	9.66	80.39	-35.29
1	2	6.04	2.81	7.60	53.48	-25.83
1	3	5.15	4.29	6.38	16.70	-23.88
1	4	4.15	4.51	4.21	-8.67	-1.45
1	5	4.38	4.54	3.46	-3.65	21.00
2	1	7.25	2.07	9.73	71.45	-34.21
2	2	7.58	5.00	9.01	34.04	-18.87
2	3	5.76	5.34	5.87	7.29	-1.91
2	4	4.91	5.35	4.49	-8.96	8.55
2	5	4.79	6.17	4.25	-28.81	11.27
3	1	7.45	3.48	9.89	53.29	-32.75
3	2	6.23	5.00	6.94	19.74	-11.40
3	3	6.55	7.50	6.09	-14.50	7.02
3	4	6.28	8.24	5.69	-31.21	9.39
3	5	6.00	8.54	5.16	-42.33	14.00
4	1	5.75	2.21	7.21	61.57	-25.39
4	2	6.72	6.99	6.39	-4.02	4.91
4	3	6.09	7.06	5.59	-15.93	8.21
4	4	6.71	8.31	5.69	-23.85	15.20
4	5	6.74	8.88	5.73	-31.75	14.99
5	1	6.04	4.27	7.05	29.30	-16.72
5	2	5.55	4.55	5.91	18.02	-6.49
5	3	8.17	8.80	7.35	-7.71	10.04
5	4	7.85	9.34	6.98	-18.98	11.08
5	5	8.20	10.87	7.60	-32.56	7.32

Table 12

Mean Square Root Error

5. ONTOLOGY-BASED INTELLIGENT CUSTOMIZATION FRAMEWORK FOR SAAS

Software as a Service (SaaS) with multi-tenancy architecture is a popular approach. To support a significant number of tenants, SaaS applications need be customizable to fulfill the various functional and quality requirements of individual tenants. This paper presents a unified and innovative multi-layered customization framework, to support and manage the variability of SaaS applications and tenants-specific requirements. Ontology is used to derive customization and deployment information for tenants cross layers. This framework also has an intelligent recommendation engine to support new tenants to deploy using information from existing deployed SaaS applications. A case study in mortgage application is used to demonstrate the proposed model.

5.1. Introduction

Software as a Service (SaaS) with multi-tenancy architecture (MTA) is a model for software delivery where a software provider publishes a copy of their software on the Web Internet to support multiple tenants or customers in a cloud environment. The cloud environment centrally operates, maintains and supports its customers via SaaS. Notable Saas applications include Salesforce.com provides on-demand Customer Relationship Management (CRM) services; People-Soft On-Demand from Oracle provides SaaS infrastructure for enterprise applications; Google maps and apps (mails, docs, and sites.) supports millions of customers; Microsoft announces Office Web Apps in early 2010.

MTA allows multiple tenants to share a software service with customization so

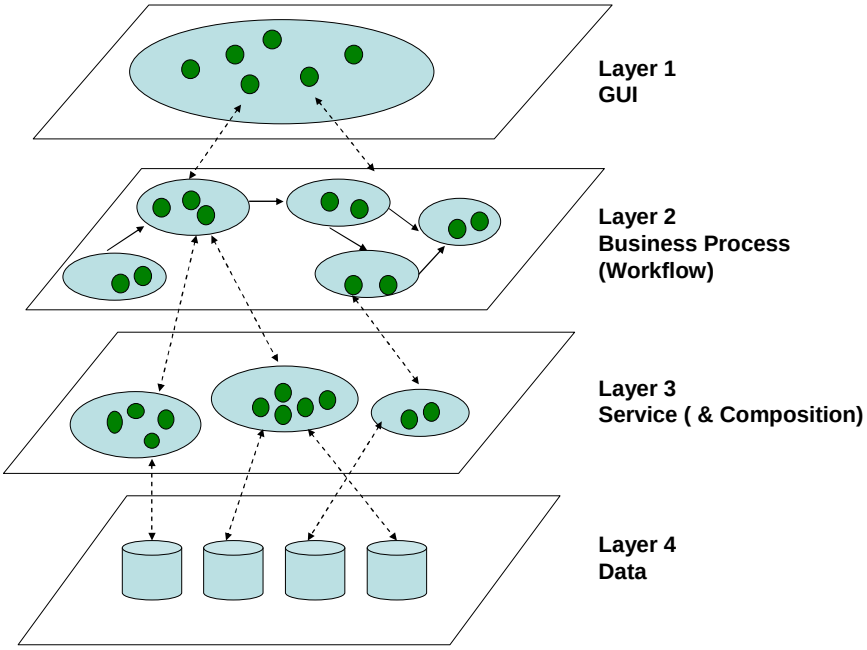


Fig. 41. Multi-Layered Architecture for SaaS Customization

that each tenant may have its own GUI, data, and user interaction. As a consequence, the SaaS software may appear to each tenant as if it is the sole tenant (e.g., keeping confidential data private), while allowing multiple tenants to use the same software (to achieve economy of scale). A maturity model with four levels is proposed [29], and the highest level of SasS is configurable and scalable, and has MTA. A configurable SaaS is often achieved by customization, and a scalable SaaS by duplicating software to meet the increased load. This chapter addresses customization mechanisms with MTA.

The SaaS customization need to meet multiple goals: SaaS providers need to support tenants/customers with a multitude of options and variations using a single code base, such that it is possible for each tenant to have a unique software configuration. Also they need to ensure that the configuration is simple and easy

to satisfy tenants' specific requirements without extra development or operation costs. The SaaS customization is not only related to functionality but also related Quality-of-Services (QoS), e.g., some tenants require an application to be highly available and are willing to pay a premium for it, while other tenants are not interested in high availability but care more about the price. As the functionality and quality requirement from individual tenants can be different, SaaS providers face two goals and try to balance between them: cater more tenants (with varying requirements by providing tenant-specific adaptation and deployment), and maintain enough commonalities (to exploit the economies of scale).

Indeed, a fully customizable SaaS application has a layered architecture, as shown in Figure 41. All the aspects for an application can be configured through a platform, from the top to the bottom layer, including Graphic User Interfaces (GUI), workflows, services and data respectively. Tenant-specific customization of a SaaS application affects all layers, from functional requirements such as designs and texts in GUI, customized business processes to database schemas design.

One solution is to develop a set of customization mechanisms to allow tenants expressing their requirements with the following features: easy to use (template objects are provided as default); layered architecture (from GUI, workflow, service and data layers); semantic oriented (using domain ontology to help the customization process); intelligent (recommendation supported by mining knowledge from tenants community and profiling); and adaptable (periodic maintain the framework to improve performance).

Existing SaaS customization solutions do not satisfy these requirements. They

do not provide a unified and mechanism for SaaS customization. It is difficult for tenants to generate individual applications, especially the new application scope of organization may be broad, and the customer requirements are diverse. They require that tenant organizations to master customized points, and this can be difficult. Domain knowledge such as ontology is not well integrated into the customization process. In addition, the customization process needs to be an incremental process with automation support.

Take a scenario of creating a mortgage application as example. A new mortgage company customizes its own business requirements using mortgage SaaS may encounter difficulty due to the intricacy relationship among layers. It is difficult for the company to customize its GUI, business processes, services and data if the customization process is complicated.

Furthermore, MTA brings in new challenges as the same code base will be used for all tenants with individual customizations. There are several challenges, such as how to recommend candidate components to tenant based on his profile information? How to use domain knowledge, e.g. ontology information to help with customization?

This chapter proposes OIC, a multi-layers ontology based intelligent customization framework. Four layers from GUI, business process, service and data layers, are modeled in a multi-layered architecture, and ontology information is used at each layer to guide the customization process. According to the multi-layered model, a framework is provided which can be applied in the development of SaaS. Based on this framework, the SaaS providers are supported in their decision. The proposed

ontology-based layered framework for customization with the following features:

- This chapter exploits a multi-layer structure of SaaS applications, analyzes their inherent relationships, as well as cross-layer relationship using ontology information. To the best of our knowledge, this is the first customization framework that uses ontology to guide the customization process.

- This chapter uses template objects as default, and recommends candidate components at different possible layers (GUI, workflow, service and data) using collaborative filtering techniques to provide a cost effective way to customize tenants' specific individual requirements.

- This chapter uses a case study in mortgage application to demonstrate the proposed framework which can effectively improve customization in SaaS.

The chapter is organized as the following: Section 5.2 develops customization algorithms with each layer using ontology; Section 5.3 uses data mining algorithms to recommend similar components for tenants as references; Section 5.4 presents the the multi-layered framework; Section 5.5 discusses the adaptive process of the proposed framework; Section 5.7 discusses the related work in customization; and Section 5.8 concludes this chapter.

5.2. Ontology based Customization

The framework has four layers: data layer, service layer, process layer, UI layer respectively. The data layer and the service layer are the foundation and they establish the data structure and operations for applications. The process layer manages collaboration mechanisms, organizes services into the process to achieve

complex tasks. The GUI layer provides the interface between systems and end users, accepts input from users and returns results back to users.

The customization process is assisted by domain *Ontology*[99], and this specifies domain vocabulary and their relationships. All layers have their own ontology information, thus data ontology, service ontology, business process (workflow) ontology, and GUI ontology, which describes the concepts and relations in that layer. The following sections will discuss issue in each layer first, then discuss cross-layer relationships and customization granularity.

5.2.1. Template Objects

It is helpful if a user can search objects (data, service, work-flows) in a repository, and then reuse, include or modify them as needed when designing new ones, so that the design phase will be easier and be shortened comparing with designing new ones from scratch. But it is impossible to build generalized SaaS software to fulfill all needs from diverse tenants due to the complexity.

To deal with the commonality of tenants, a set of templates (standard) objects is provided for designers to assist SaaS customization. The template objects store at different repositories at all layers (including data repository, service repository, workflow repository and UI repository). [1] Given ontology information for a particular application domain, OIC use template objects as an initial starting point, and support customization in a cost effective way. Also the recommendation engine can

[1]Some SaaS providers, e.g. saleforce.com can provide built-in objects, for things as name, address, phone numbers, and email address, contains almost all possible features in a domain and can be customized to meet individual special requirements. OIC can easily integrate and reuse them. The notion of templates this chapter is more general.

provide a list of candidate according to tenant's profiling.

5.2.2. Data Layer

In a domain, multiple data ontology systems can be defined by different communities [48]. Existing research have discussed how to define ontology in a specific domains, compare and integrate ontology systems between different communities. For example, ACM and IEEE are two large communities and each has its own standards and practices. Even though they are similar but still distinct, and thus if the corresponding ontology systems will be similar but distinct, Ontology integration [109] is developed to solve these heterogeneities, which refers to build a larger and complete ontology at a higher level using existing ontology systems.

As a concept structure of domain knowledge, ontology is usually represented as a tree. Comparing the concepts of two nodes in the tree, can be easily estimated by domain experts. For instance, "people" and "human being" are referring to the same meaning with a similarity degree of 1. "faculty" and "professor" are very similar in university domain, with similarity degree 0.95, which means around 95% occasions these two are describing the same concept. Some research have been done in determining conceptual similarity in a knowledge context[162].

Data layer customization is guided by ontology information. After searching for the domain ontology, e.g. mortgage domain, one can find the template, and customize it using ontology to guide the customization process. Take a mortgage application as an example, Figure 42 shows a sample of ontology-based customization process. As one can see, given the ontology tree T (Figure 42(a)), two mortgage companies A and B can customize their own templates by picking up desirable com-

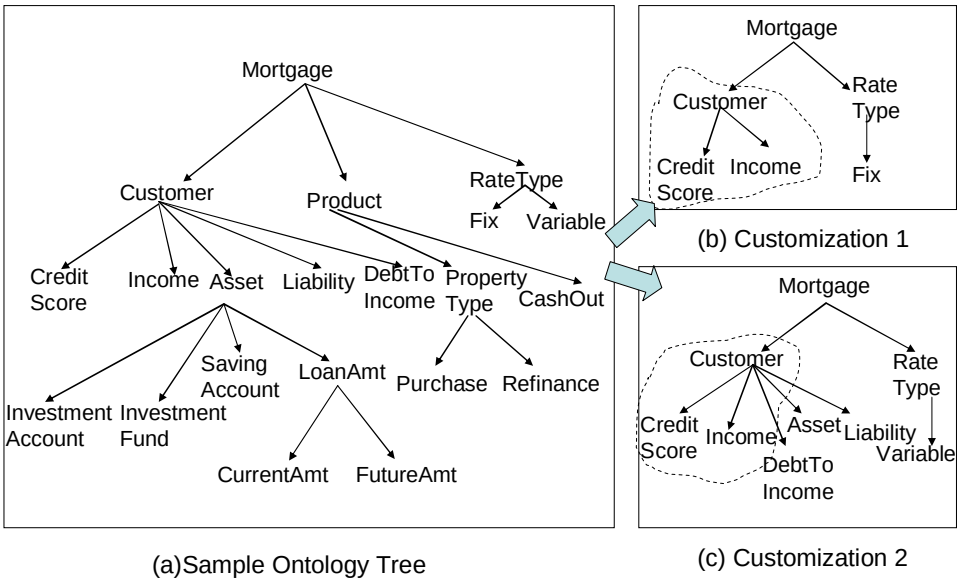


Fig. 42. Sample Ontology Tree and Customizations in Mortgage Applications

Mortgage Company A

Customer	Credit Score	Income
1	675	80,000
2	712	110,000

Mortgage Company B

Customer	Credit Score	Asset	Liability	Income	DebtTo Income
1	720	120,000	280,000	70,000	4
2	590	58,000	200,000	56,000	3.57

Multi-tenancy Data Schema

Tenant ID	Customer	Credit Score	Income	Sparse					
1	1	675	80,000	(NULL)					
1	2	712	110,000	(NULL)					
2	1	720	70,000	Asset	120,000	Liability	280,000	DebtToIncome	4
2	2	590	56,000	Asset	58,000	Liability	200,000	DebtToIncome	3.57

Fig. 43. Sample Database Schemas for Mortgage Applications

ponents in T optionally, the customized template for A is shown in (Figure 42(b)), while company B in (Figure 42(c)). Both of them have some similarity, e.g. credit score and income (in dashed circle), as well as differences where company A has more attributes to consider, including asset, liability and debtToIncome.

Furthermore, the ontology semantic information can be matched to database logic designs. The domain objects can represent a large proportion of meta-data that are serialized into the data repository. Multiple database schemas can be used in MTA[8], such as XML, sparse table, views, tenants can choose any database schemas as needed.

Example 1: The mortgage loan application process, each mortgage company(lender) is treated as independent *tenant* and processes loan applications from end customers everyday. Suppose two mortgage companies try to build their own mortgage business applications, where company A choose the default template, and company B uses customized data schemas (mapping to ontology tree in Figure 42). Sample data tables for $\{customer\}$ entity is shown in Figure 43 for each company (on top), as well as the multi-tenancy database schema support (at bottom). Note that to explicitly demonstrate the customization process in the data schema, we choose sparse representation for demonstration purpose, all other possible schema designs, e.g. XML can also be used here.

5.2.3. Service Layers

The service could be atomic services and composite services. An atomic service is the basic service which accomplishes fundamental operations, while a composite service involves several related atomic services to conduct more complex tasks.

The atomic service customization is the easiest case. To differentiate a service, one has to understand the unique parts in a service in a description/profile. Every service represents an underlying capability under specific terms and conditions. The capability offered can satisfy the customer's requirements with certain constraints. The terms to offer including several aspects as properties [102], e.g. cost, discounts, availability, QoS, convenience to use and etc.

More complicated task can be achieved by ontology. Semantic Web community has made efforts in the development of expressive languages to describe web service ontology based on artificial intelligence technology, including RDF, DAML+OIL, and OWL.Researchers have developed automated reasoning machinery to address more difficult tasks including automated Web Service discovery, semantic translation and automated Web Service composition.

Example 2: The mortgage company tries to "find the closest bank partner to finish the financial status checking step for a loan application". The domain ontology for bank service is shown in Figure 44. The functionality branch contains a classification of service capabilities, one for finding bank partner and the other for calculating distances between two locations. Using the sample service ontology, the task can be customized and automated. The right services can be selected automatically from a collection of services. Ontology provides a flexible selection which can retrieve services that partially matching a request but are potentially interesting.

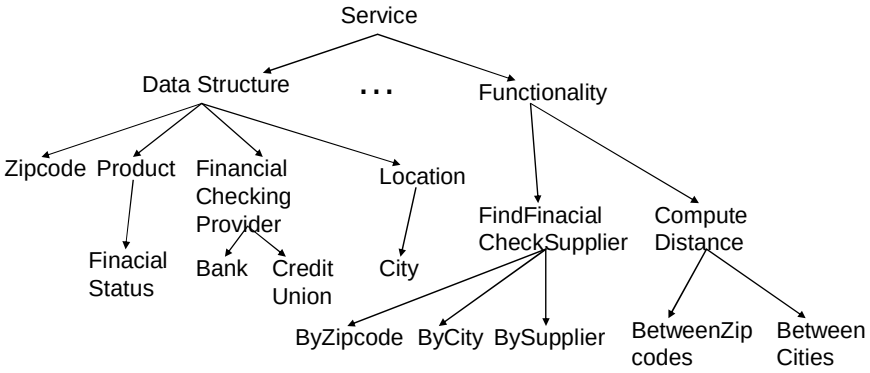
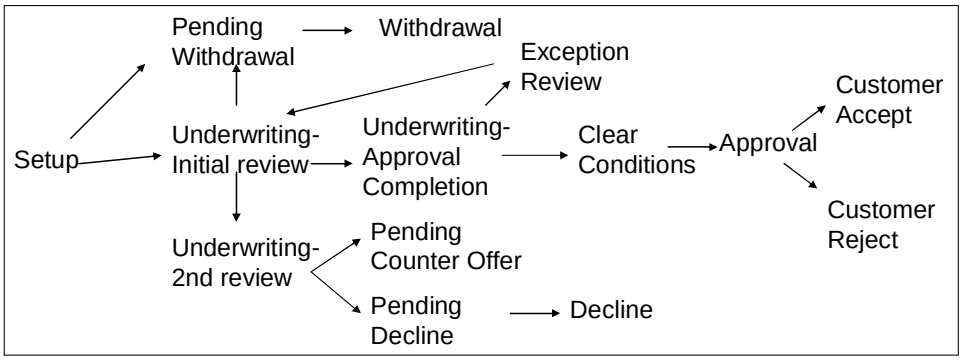
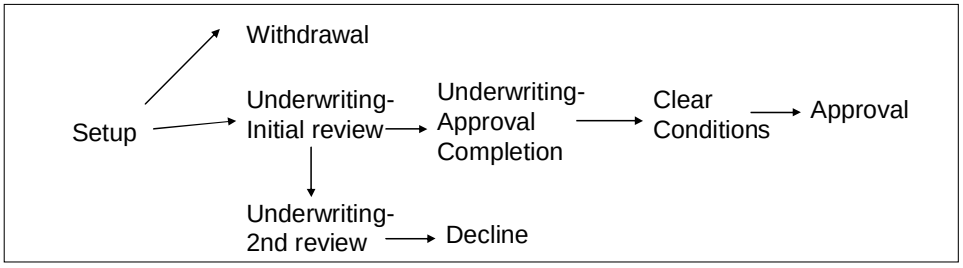


Fig. 44. Sample Mortgage Service Domain Ontology



(a) Loan Application Workflow



(b) Simplified Version with Customization

Fig. 45. Sample Mortgage Workflow Customization

5.2.4. Business Process Layer

In this layer, the services and participants have been organized to achieve more complex business tasks, workflows, which consist by a set of activities and represent business processes. Tenants can search a workflow repository using keywords and retrieve the relevant ones according to their interests. The customization process is based on the business domain knowledge in multi-layered workflow with a series of steps or transformation from template objects. Data pass information through flows, and act as the step transformation conditions.

Example 3: The mortgage origination is an end-to-end process beginning with the submission of a mortgage application to a lender and ending in closing (lender approves, applicant accepts and lender funds the loan) or non-closing (either lender disapproves, or applicant withdraws or refuses approved offer by lender). A sample workflow template is shown in Figure 45(a). As a new open mortgage company comes in, it may only need some basic functionality, hence it can utilize workflow search engine [132] to get a customized workflow as in Figure 45(b), which returns informative and concise search results, defined as the minimal views of the most specific workflow hierarchies. The choice of workflow is made depending on domain ontology information.

5.2.5. GUI Layer

A UI ontology can be built to provide the concepts, relationships, reasoning, and searching for UI-related elements. The ontology should include UI classification information includes[151]: data collection, data presentation, monitoring, command-and-control, and hybrid (combination of two or more types). Much easier UI

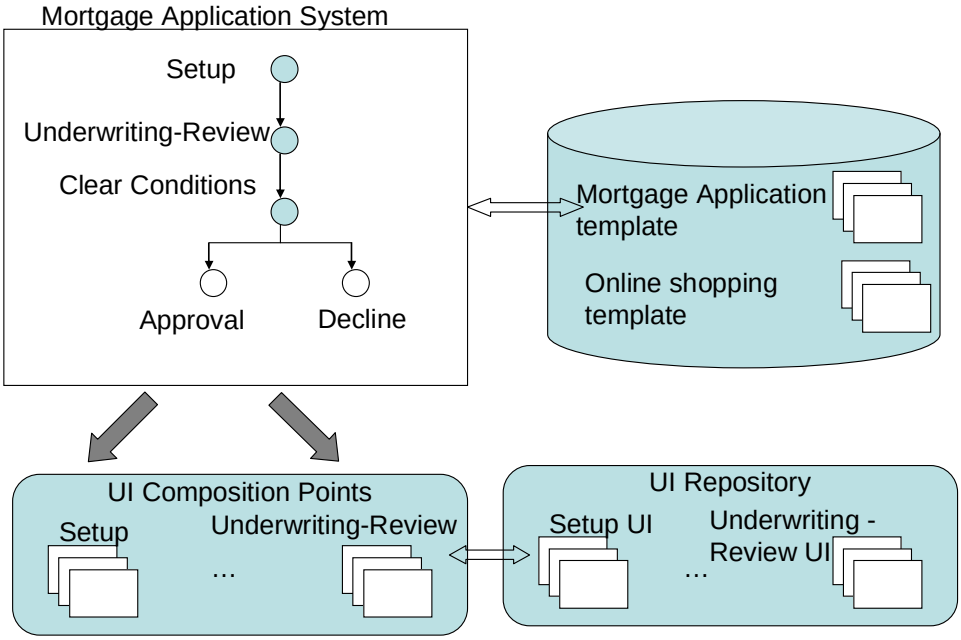


Fig. 46. Sample Mortgage UI Customization

customization is to change and configure the appearance and the UI available to the users, including adding/editing/deleting the icons, colors, fonts, titles in pages, menus and page-sections.

Example 4: A sample UI customization is shown in Figure 46 outlines for mortgage application. The application template on the upper right is specified by the workflow at upper left. The UI Composition points are where the application interact with tenants and accept customization. Searching UI repository, one can find candidate template objects according to individual preference.

5.2.6. Cross-Layer Relationship

The cross-layer relationships appear between data layer and service layer, service layer and workflow layer, UI layer and other three layers. Recall the hierarchy model in Figure 41, three types of relationship are embedded in the multi-layered model, include: (1) Feed: between data and service layer, representing the data information

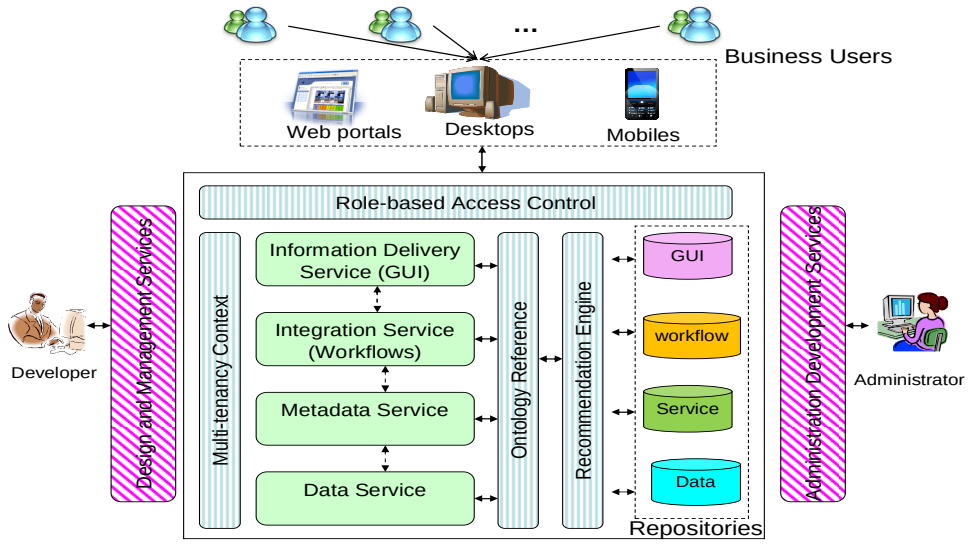


Fig. 47. System Architecture of OIC

as the input of certain services and the storage of results from them. For example, the data mining service needs plenty of data and writes the analysis results back the data storage. (2)Composition: Data and services serve as the participants of business process. Data acts as the information passed through the workflow, and the conditions judged by the transformation between steps. Services are the composite units for the business process. (3)Interaction: Data services and processes have to interacted with users. Pages acquire the input information and injunction, expose the result of services and examine the states of the executing process.

Ontologies are related to and referenced each other as shown in Figure 48. For example, a service ontology as candidate solutions for the workflow ontology. Ontologies also cross references with each other. Specifically, a workflow ontology may reference specific services in service ontology that are applicable to the business application. For example, a mortgage application may reference the following services: underwriting review, clear conditions, and etc. In this way, a tenant can identify

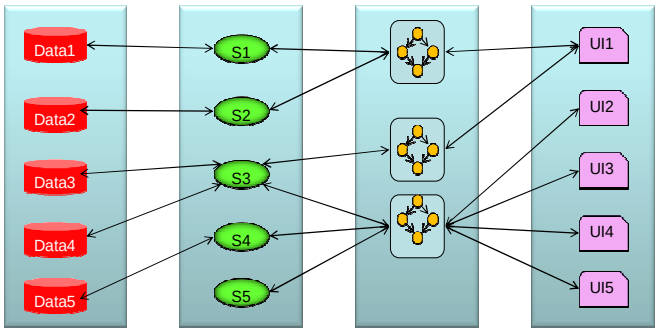


Fig. 48. Example of Ontology Cross Layer Reference

the related objects quickly for customization.

5.2.7. Customization Granularity

Two types of variants for customization: lightweight and heavyweight. For *lightweight* variants, e.g. service flavor changes [67], in which the same enterprise service can be offered to diverse customers. The capability in lightweight variants remains the same, while the terms at which the object is offered becomes varied. Another variations is *heavyweight*, in which the inherent variability in the underlying business process, requirement specifications, customer requirement and etc. are all different. It causes all layers changed. The heavyweight variants bring more challenges in SaaS model and result in inherent variation in the business process, rules, industry requirements and tenant specific requirements.

From the lightweight to heavy weight, the customization granularity can be classified into four categories, as the following four levels: *Level 1 Parameter customizable*: simplest case, least flexibility. Tenants can only utilize the templates by editing parameters of entities. *Level 2 Entity customizable*: Tenants are authorized to create their own business entities or fields with the help of templates and wizards. More difficult than level 1, with more flexibility. *Level 3 Composition customizable*:

Tenants can compose a new business process using relative data and services to accomplish new tasks. Tenants have to think over more relations during the customization process. *Level 4 Implementation customizable*: professional tenants can develop their own objects with the toolkit. Tenants are have the greatest freedom of customization process. The most comprehensive flexibility and complicity layer in all.

Based on the multi-layered customization model, the SaaS providers can have a better understanding of the essences for SaaS customization. Before starting their own SaaS design, they can consider existing resources and make design choices. A hybrid SaaS service provides different customization granularity according to business requirements.

5.3. Intelligent Recommendation

To provide more intelligence support, customization can get help from recommendation, which works at all layers to provide more candidate templates for tenants. The knowledge can be mined from similar community in the repository or tenants' own history behaviors. The opinions of a community to help individuals in the community identify content of interest from a potentially large set of choices more effectively. Industry partners, such as Amazon have shown that a retail experience can be substantially enhanced by statistically correlating macro patterns in buying and browsing behavior. Tenant's behavior history captures his preferences information well and can be used to predict his future actions. Basically, two types of methods can be used here:

Collaborative Filtering (CF) [53], the process of filtering information/patterns

using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. The intuition of CF is that similar users vote similarly on similar items. Hence if similarity is determined between users and items, a potential prediction can be made for the vote of a user for a given items. The common insight is that personal tastes are correlated: for example, both mortgage companies A and B prefer templates $\{T_1, T_2\}$ and A prefers T_3 then B is more likely to prefer T_3. Given tenant B's existing template choices, the system can recommend new templates to him with high confidence to cater for his taste.

As the preparation for CF, one has to explore tenants similarity, social networks can be used to identify potential friends and community of interest for tenants. Several memory-based algorithm can be used, including k-Nearest Neighborhood (kNN)[126], Pearson Correlation, and Cosine Similarity [138]. Formally, given $v_{i,j}$ is vote of tenant i on template j, I_i represents items for which tenant i has voted, the mean vote for i is $\overline{v_i} = \frac{1}{I_i}\sum_{j \in I_i} v_{i,j}$. To predict an active tenant a's vote, which is defined as

$$p_{a_j} = \overline{v_a} + \kappa \sum_{i=1}^{n} \omega(a,i)(v_{i,j} - \overline{v_i}) \tag{5.1}$$

where κ is normalizer, and $\omega(a,i)$ is the weights of n similar users. Specially, the $\omega(a,i)$ can be defined by kNN (in Eq.5.2), Pearson Correlation (in Eq.5.3), and Cosine Similarity in (Eq.5.4) as following:

$$\omega(a,i) = \begin{cases} 1 & \text{if } i \in neighbour(a), \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

$$\omega(a,i) = \frac{\sum_j (v_{a,j} - \overline{v_a})(v_{i,j} - \overline{v_i})}{\sqrt{\sum_j (v_{a,j} - \overline{v_a})^2 \sum_j (v_{i,j} - \overline{v_i})^2}} \tag{5.3}$$

$$\omega(a,i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}} \tag{5.4}$$

Content-Based Recommendation uses the tenants' profiling[20], such as tenant configuration files, and tenant context (initial loaded from tenant's configuration files and updated accompany with tenant's information) to make recommendations. Many classification algorithms, e.g. naive bayesian, decision tree, and supported vector machine (SVM), can be used to predict the category of a tenant. For instance, tenants can be classified into different clusters, and a new tenant from mortgage domain will share similar features from tenants in the same domain. The system can predict new tenant's preference accordingly using classification algorithms.

Example 5: Take service layer as an example, a tenant provides his requirements using a semantic document which may vary from service description and QoS parameters. The system compares the tenant's request with existing service templates in the repository, provides a list of available candidate services for the tenant to choose according his preference using either CF or content methods. Based on users' feedback, the recommendation system orders the list and presents to the tenant with a desirable service. At last, the user may provide a rating to this service using given metric which indicates user's satisfaction level. It can be stored in a repository and used as an input for the recommendation in future.

5.4. OIC System Architecture

The OIC architecture is shown in Figure 77 with four main parts for different roles in SaaS applications: (i) for developers, the design and management layer(left most); (ii)for administrator, the infrastructure administration and configuration layer (right most);(iii) for business users, the end-users access tools layer (top); and (iv) the core services layer (central rectangle).

The design and management layer contains services (left most virgule) to design models and manage development projects for developers. This layer offers an on-demand design in order to ensure platform integrity which simplifies the deployment process and easy access to the development environment. It reduces the installation time and cost for development infrastructures.

The infrastructure administration and configuration layer (right most virgule) offers a web-based tool for administrators to manage users accounts, configure services and report platform usage and performance information.

The end-users access tools layer (top most) contains client applications for business users to access the platform. Many different applications can be supported, e.g. mobile technologies, web services, and desktop tools.

The core services layer support customization requirements used by different users. Four essential services (in light green) mapping to the proposed multi-layered model: (i) the information delivery service(IDS) is an abstraction level to support multi-tenants interfaces and technologies (e.g., web browser, mobile, office tools). (ii) the integration service (IS), which offers an ad-hoc way to define data integration jobs, workflows, etc. (iii) the meta-data service (MS), which facilitates information

sharing and exchange between all services, with meta-data and business information definition support. (iv) data services(DS), which explores business data information, and communicate with other services to support business requirements.

Several customization components(in blue vertical virgule): (i) role-based access control: determines which resources and functions each user should be allowed to access lies with the tenant. Users are grouped into different roles according to the organizational structure and ontology, and the access control privileges can be configured accordingly (More details in [153]). (ii)multi-tenancy context: analyzes the context information of new SaaS application, e.g. location, requester information, application domains and etc. All those can be used later for recommendation and mining. (iii)ontology reference: provides the fundamental semantic support for customization. Cross layer relationship has hidden correlation by ontology references. (iv)recommendation engine: use data mining techniques to recommend the most similar services/templete in the repository. (v) repositories: each component has a repository in SaaS platform, which store template objects, including workflow repository, service repository, and etc.

5.5. Adaptive Customization Process

The OIC framework works in an adaptive way, as shown in Figure 49. At different layer, customers can customize and modify the "component" (e.g. datatable attributes, service components and etc.) in the template according to individual requirement. The customization process can be subdivided into two major parts, template retrieval and adaptation process. The first part is to retrieve templates from the repository using intelligent recommendation or default setting by keyword

search. The second step is an iterative procedure which performs the necessary adaptation of the retrieved base-template if it does not fully fulfill customer's specific demands.

In most cases, template retrieved may not match the customer's specific demand, the components in template which is not desirable as tenant requirements, named as " undesirable components (UC)", can be replaced with other suitable components. An iterative procedure then works through all UCs and retrieves the more suitable components from component databases. If needed, the component can be configured, updated and altered to satisfy with desirable features. Specially, to ensure the quality of the customization, a validation process is also offered after each new component is updated. Finally, after all the components are configured, updated and validated, the new product is formed.

5.6. Case Study

This section uses a mortgage application system to illustrate how to customize SaaS in real business scenarios. Assume three mortgage companies S, M, L are trying to use OIC to customize their own mortgage application systems with diverse requirements as follows:

- S, a small start-up company, wants to build up a simple mortgage system in a short time, and can afford limited cost. Hence, it can use all default templates in different layers provided by OIC easily.

- M, a middle size mortgage company, owns hundreds of customers and increases its business scale gradually recently. It decides to move to SaaS model with

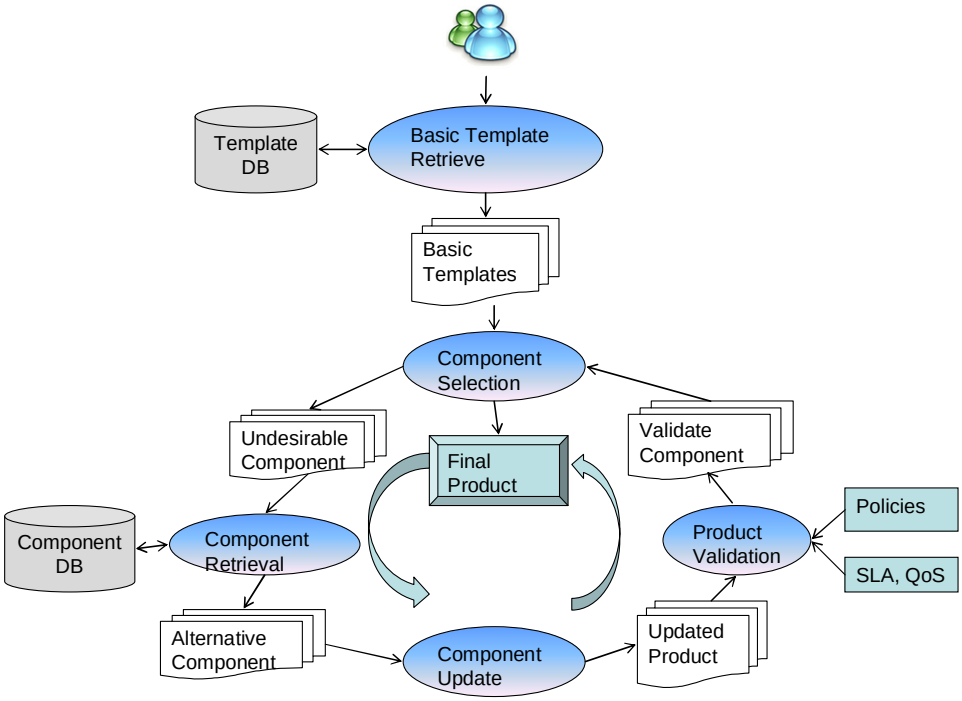


Fig. 49. Adaptive Recommendation Process

some unique requirements in mind, represented as ontology at different layers. OIC can support its customization from data layer, up to UI layer in a cost effective way.

- L, a large mortgage company, has more than thousands of customers every month, with a high availability and scalability requirement. It can afford high cost of usage for high performance services.

The template choices sample in each layer for all three companies are shown in Figure 50. Take L as an example, given its business domain of mortgage application, OIC starts customization from data layer, matches its ontology to database schemas, and extended database templates with more customized tables and attributes as described in Section 5.2.2. Then OIC searches the service repository, and pick more

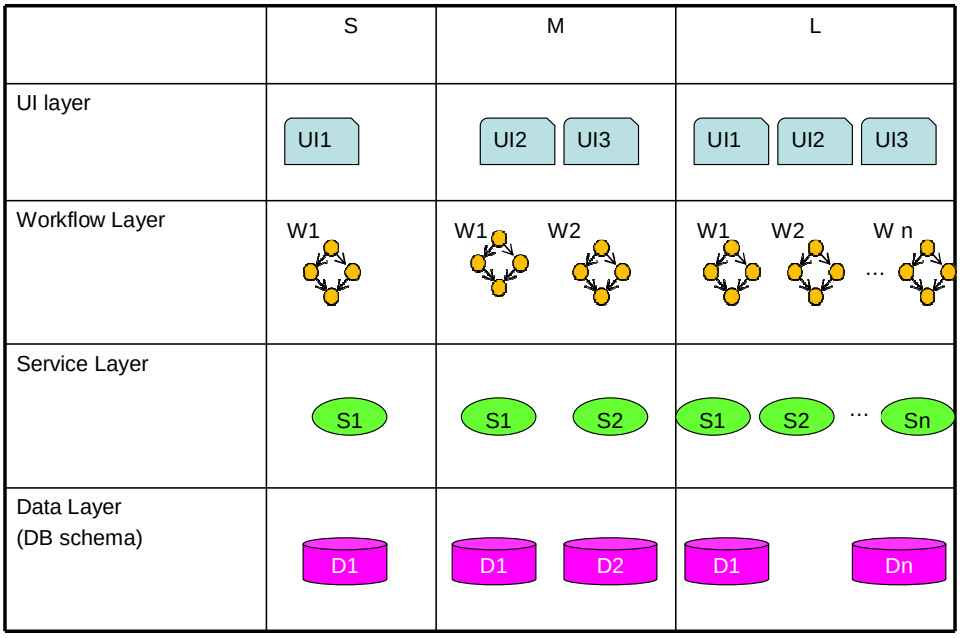


Fig. 50. Comparison of Three Tenants Customization Samples

advanced services, such as "approval but reject", "approval and accept" to satisfy his unique requirements as in Section 5.2.3. Up to workflow layer, by searching the working repository, L can modify the workflow according to its advanced business requirements using methods proposed in Section 5.2.4, finally OIC searches for UI repository to get the matching templates with more modification methods as in Section 5.2.5.

5.7. Related Work

Software customization has been a research problem for a long time, but SaaS customization using the same code base is a new issue. Traditional customization include parametrization, AI rules, file-based customization, and template-based customization such as Template Method in OO design patterns [49]. Customization may be done manually, automatically, and adaptively[92]. Customization may be based on software running history, current workload including file size and memory

requirements, or user experience such as GUI and user interactions.

Traditional customization is often reflected in products offered in different packages, e.g., one package offers one set of features, while the other different set of features. Service-oriented architecture (SOA) offer a new way of customization as a new application may be re-composed from an existing application by using different services. Furthermore, this can be extended to GUI composition, test script composition, and dynamic process composition in a community ([157, 152]). Thus it introduces a new customization mechanism, e.g. service replacement.

SaaS customization has mainly addressed data-level customization, specifically data schema design. Specifically, the data architecture of multi-tenant [29] is identified as three distinct approaches (Separate Databases, Shared Database Separate Schemas and Shared Databases Shared Schema). [164] proposed a new schema-mapping technique for multi-tenant data named Chunk Folding. But they have not touched other layers in customization. Another group of researches touched service level, for example, Zhang [170] proposed a novel SaaS customization framework using policy through design-time tooling and a runtime environment. Mietzner [95] described the notion of a variability descriptor which defines variability points for the process layer and related artifacts in a service-oriented manner. Li [85] considered the multi-layer and cross layer relationship, and used multi-granularity models to compose customization tasks. Essaidi [44] presented an open source infrastructure to build and deliver on-demand customized business intelligence services.

These existing projects either discussed solutions in certain layer of customization or did not investigate the relationship cross layers based on ontology informa-

tion. Also, no recommendations are provided to tenants as the guide to configure the process. Thus, the task of customization is still difficult.

Existing industry partners support SaaS customization in their own ways. For instance, Google App Engine (GAE) [43] offered the customization at the program layer using a deployment description file in which users can change the parameters such as servlet, URL paths, jsps, and security methods. GAE also supports service layer customization in which users can set up service names and domains, control the billing budgets. However, workflows and date are not customizable at GAE. Amazon EC2 [41] offers similar customization capabilities as GAE, but it does not offer the workflow layer. Salesforce.com [46] offers a flexible customization framework, in which users can customize the UI, workflow and data inside their framework, but semantic information, e.g., ontology is not integrated into customization.

Ontology based customization has been applied to other domain such as Retschitzegger [119], and often they used a context reasoning module to analyze ontology information to guide customization rule engine. Yarimagan [167] proposed a customization tool for the Universal Bushiness Language. But these are not related to SaaS.

5.8. Conclusion

In this chapter, we have introduced OIC, a multi-layer ontology-based intelligent customization framework to support and manage the variability of SaaS applications and tenants-specific requirements. Ontology is used to derive customization and deployment information for tenants cross layers. Intelligent recommendation engine is used to support new tenants to deploy using information from existing

deployed SaaS applications. For the future, we plan to investigate the database partitioning and scheduling algorithm to work together with the proposed customization framework, as well as loan balancing, recovery mechanism in SaaS.

6. TOWARDS A SCALABLE MULTI-TENANCY SAAS

Software-as-as-Service (SaaS) is a new approach for developing software, and it is characterized by its multi-tenancy architecture and its ability to provide flexible customization to individual tenant. However, the multi-tenancy architecture and customization requirements introduce many new issues in software, such as database design, database partition, scalability, recovery, and continuous testing. This chapter proposes a hybrid test database design to support SaaS customization with two-layer database partitioning. The database is further extended with a new built-in redundancy with ontology so that the SaaS can recover from ontology, data or metadata failures. Furthermore, constraints in metadata can be used either as test cases or policies to support SaaS continuous testing and policy enforcement.

6.1. Introduction

Software-as-a-service (SaaS) is software that deployed over the internet and often run on a cloud platform. With SaaS, a software provider licenses an application to customer as a service on demand, through a subscription or a "pay-as-you-go" model. A common SaaS application is CRM (Customer Relationship Management). Notable SaaS applications include Salesforce.com which provides on-demand CRM; Peoplesoft from Oracle which provides SaaS infrastructure for enterprise applications; Google maps and Google apps (such as Google docs, gmail); and Microsoft Office Web Apps. As a proliferating software model, more application developers will embrace the SaaS model. However, it also faces new challenges.

(1) Multi-Tenancy Architecture (MTA) Support: Multi-tenancy refers to a principle where a single instance of the software runs on a server, serving

multiple client organizations (tenants), which is often used in SaaS. With MTA, a software application is designed to virtually partition its data and configuration, so that each client works with a customized virtual application instance. Although all tenants share the same software, they feel like they are the sole user of the software. A maturity model for SaaS with four levels is proposed in [29] with, the highest level of SaaS being configurable, scalable, and having MTA. A configurable SaaS is often achieved by customization and a scalable SaaS is often achieved by duplicating software to meet the increased load. The MTA requires SaaS providers to support a huge number of applications. In 2009, Salesforce.com [163] reported that it is supporting 100,000 distinct applications using 10 databases running on top of 50 servers in two mirror-sites with just one software base. Due to the unique features of MTA, a realistic SaaS application needs to address the following issues:

- Scalability: The algorithm should adjust according to the changing load of system. Specifically, if the workload increases, resources (such as processors, memory, and disk space) should be allocated to handle the task, and if the workload decreases, resources should be re-allocated to other tasks. In this way, resources can be dynamically allocated and re-allocated at runtime. Ideally, the increase (decrease) in resources should be proportional to the increase (decrease) in workload, while keeping the performance of each task at an acceptable level;

- Database partitioning and consistency: Tenant data need to be partitioned well in the back-end database to support real-time high performance comput-

ing;

- Fault-tolerant computing: The failures of processors or storage should not affect the operation or data because they may represent business transactions that will bring in revenue for the company.

- Security and fairness: Tenant data should be isolated from each other, and tenants of the same priority should receive the same level of services and resources as multiple tenants will share the same software and possibly also the same database in MTA;

- Parallel processing: It is highly desirable that the tasks can be processed in parallel such as done by Map-Reduce to take advantage of the massive number of processors, memory and storage units available in a cloud environment;

- Isolation: Any changes in a tenant's data should not affect any other tenants;

- Performance and availability: As one SaaS program potentially needs to serve hundreds of thousands or even millions of tenants or applications, it is critical that SaaS software can provide real-time performance and availability with automated data migration, backup and restoration and isolation.

Note that cloud computing has changed computing significantly due to massive number of processors will be used to support real-time applications that require high availability and reliability. Data must be reliably stored and restored in case of data failure. A cloud need to allocate resources to a computing task in case of raising workload and re-allocate resources when the workload decreases. In cloud

computing, system availability, performance, reliability and security will be more important than minimizing system resources. To appreciate the scale scope of cloud computing, one can take look of the existing cloud infrastructure. For example, Google data center [21] reports that it can host 45K servers in 45 containers in one single data center as reported in 2009. Microsoft [63] also reported their data center version 4.0 recently where there will be no side walls so that processors can be exposed to air, and containers of processors can be added to the data center to meet the increased workload. These massive numbers of servers will be used to provide real-time computing with automated reconfiguration and recovery mechanisms.

In [156], we proposed a four-layered architecture for SaaS customization. This chapter further extends the four-layer architecture, with an additional data layer, to provide a scalable framework for SaaS. This chapter discusses the possible database partitioning choices for SaaS in the data layer. For each choice, this chapter discusses what types of application is suitable for taking advantage and their trade-offs. This chapter also proposes the related load-balancing algorithms to address the scheduling problem when data are partitioned. More details will be discussed in Section 6.4.

(2)SaaS testing with new challenges: Testing SaaS is different from testing software services as SaaS involves customization, configuration, and scalability while services involves only calling and responding with QoS constraints. Currently the testing of SaaS software often uses the traditional software testing practices. As only a single copy of the software is maintained for multiple sharing tenants and each having different requirements. Furthermore, a new tenant may be added into

SaaS, and bring new requirements after SaaS deployment, and thus SaaS testing may need to continue after deployment. In fact, as the SaaS being deployed, it needs to be continuously verified to support the SaaS. This feature has been used by Google Chrome OS where continuous verification is a key feature. As SaaS being developed, it also needs to be tested and evaluated, and thus SaaS requires a continuous testing model throughout its entire lifecycle. iTKO [64] is a new SaaS testing enterprise that uses the continuous testing when building up their DevTest Cloud. iTKO's continuous validation service (CVS) feature orchestrates the testing and validation aspects of IT, Integration workflows and SOA Governance, to ensure reliability and instill trust throughout the lifecycle of the application.

This chapter proposes a built-in continuously testing SaaS testing framework from ontology and metadata to support QoS (Quality of Services) and SLA (Service Level Agreement). Besides continuous testing, testing SaaS software can also be collaborative by nature since it is usually developed with a service oriented architecture. In this chapter, a collaborative testing environment by generating test scripts in a collaborative manner. The schema integrates continuous testing with the storage layer, by leveraging database triggering rules. Integrating testing and intelligent testing can also be conducted within the framework.

(3) Ontology-based analysis and development: Ontology can have a significant role in service applications and SaaS development, and it can be used for specification, references, reasoning, and even customization [156]. In the customization framework, each layer is supported with its own ontology for discovering similar templates and conducting intelligent mining.

As a summary, cloud computing and SaaS provide new requirements for scalability, and robustness. Knowing the key differences between cloud computing and traditional computing can help to better understand the issues in SaaS. Comparing with traditional computing, cloud computing has several major differences, as shown in Table 51: In terms of scalability, cloud computing provides on-demand resource scaling, which allows the cloud users to scale their applications dynamically according to application workloads. It is the most cost-effective way for users to scale since they do not need to worry about the management of resources and hardware costs. While in traditional computing, users need to support scalability either by scaling up (upgrading the configuration of the hardware) or scaling out (buying and adding more compute nodes into the system). This is not cost-effective when what the users want to deal with is only temporary workload changes. In traditional computing, usually one issue is considered when conducting the computing. For example, fault-tolerant computing concerns more about availability, while real-time computing concerns more about application performance. However in cloud computing, all the issues, such as fault-tolerance, reliability, application performance, resource management, need to be taken into consideration at the same time. This is a new type of software engineering. In traditional computing, people concerns more about efficient resource utilization and management, while in cloud computing, the reliability of the cloud system becomes a serious concern. The importance of the read/write operations is also changed. Write operations are considered to be more important in cloud computing because a write update may represent a customer order [39], while it is the opposite case in traditional computing where efficient

	Traditional Computing	Cloud Computing
Scalability	Provides scalability by scaling-up or scaling-out. Need to buy / update hardware to scale to satisfy increasing demand. Resource might be wasted as idle.	Provides "on-demand" scalability to users of the cloud. The resources can be dynamically allocated and released. User can scale with low hardware costs.
Software Architecture	Usually focuses on a single facet, such as fault-tolerant computing, real-time computing. Traditional software engineering principles can be applied.	A new kind of software engineering: Many separate issues, such as fault-tolerant computing, real-time computing, database, are all involved together in the cloud.
Frequency and priority of operations (Read/Write)	Read operations are considered more important than write operations	Write operations are considered more important than read operations
System Metrics	Resources are considered to be more important. The efficient management of resources is highly valued.	Reliability is considered to be more important than resources. Efficient reallocation of the resources is highly valued.

Fig. 51. Major Differences of Cloud Computing and Traditional Computing

execution is the primary concern.

The contribution of this chapter is five-fold:

- This chapter tackles the scalability problem in SaaS framework by extending the previous four-layer SaaS customization framework with additional data layer.
- This chapter investigates a two-layer partitioning schema with effective index to support scalability in SaaS applications.
- This chapter incorporates the feature of continuous testing in the proposed SaaS framework, which provides embed testing support.
- This chapter exploits the usage of ontology in providing support for customization, recovery and continuous testing in SaaS.

This rest of the chapter is structured as follows: Section 6.2 discusses the

related works. Section 6.3 presents the SaaS framework with data layer to support scalable SaaS. Specific database partitioning schemes and related issues are discussed in Section 6.4. Section 6.5 concludes this chapter.

6.2. Related Work

This chapter is related to several perspectives, including SaaS customization, database partitioning, SaaS recovery and testing. The following sections will discuss these topics in turn.

6.2.1. SaaS Customization

Existing SaaS customization has mainly addressed data-level customization, specifically data schema design. Specifically, the data architecture of multi-tenant [29] is identified as three distinct approaches (Separate Databases, Shared Database Separate Schemas and Shared Databases Shared Schema). [164] proposed a new schema-mapping technique for multi-tenant data named Chunk Folding. But they have not touched other layers in customization. Another group of researches touched service level, for example, Zhang [170] proposed a novel SaaS customization framework using policy through design-time tooling and a runtime environment. Mietzner [95] described the notion of a variability descriptor which defines variability points for the process layer and related artifacts in a service-oriented manner. Li [85] considered the multi-layer and cross layer relationship, and used multi-granularity models to compose customization tasks. Essaidi [44] presented an open source infrastructure to build and deliver on-demand customized business intelligence services.

These existing projects either discussed solutions in certain layer of customization or did not investigate the relationship cross layers based on ontology informa-

tion. In our previous work [156], we addressed the problem of customizable SaaS with MTA by providing a four-layer architecture. This chapter further extends the four-layer architecture, with an additional data layer, to provide a scalable framework for SaaS.

Existing industry partners support SaaS customization in their own ways. For instance, Google App Engine (GAE) [43] offered the customization at the program layer using a deployment description file in which users can change the parameters such as servlet, URL paths, jsps, and security methods. GAE also supports service layer customization in which users can set up service names and domains, control the billing budgets. However, workflows and date are not customizable at GAE. Amazon EC2 [41] offers similar customization capabilities as GAE, but it does not offer the workflow layer. Salesforce.com [46] offers a flexible customization framework, in which users can customize the UI, workflow and data inside their framework, but semantic information, e.g., ontology is not integrated into customization.

6.2.2. Scalability and Database Partitioning

Data partitioning is a well-studied problem in database systems (e.g., [94, 121, 22, 87, 40]). In the literature, many partitioning schemes have been studied; e.g., vertical partitioning vs. horizontal partitioning, round-robin vs. hashing vs. range partitioning [23]. Past work has noted that partitioning can effectively increase the scalability of database systems, by parallelizing I/O [87] or by assigning each partition to separate workers in a cluster [94]. H-Store [141] presents a framework for a system that uses data partitioning and single-threaded execution to simplify concurrency control. G-Store [37] extend this work by proposing several schemes

for concurrency control in partitioned, main memory databases, concluding that the combination of blocking. The discussion of vertical partitioning, e.g. column based database has been introduced recently, including MonetDB [18], C-Store [140], which can provide performance improvement on read-intensive analytical processing workload, such as in data warehouse.

To support scalability of cloud, data partitioning becomes a widely accepted solution. Parallel DBMSs with emerging cloud data management platforms is provided by industry partners, such as ([19, 35, 39, 98]) (such as efficient data partitioning, automatic fail over and partial re-computation, and guarantees of complete answers). In the database community recent work compared the performance of Hadoop versus the more traditional (SQL-based) database systems [106] which focusses on read-only, large scale OLAP workloads, and [77, 78] focused on OLTP workloads. Berkeley's Cloudstone [137] specifies a database and workload for studying cloud infrastructures and defines performance and cost metrics to compare alternative systems.

In this chapter, a novel two-layer model for partitioning is provided, which first partitions horizontally by tenants, and then vertically partitions by columns. The model can benefit both read and update operations comparing with horizontal-partitioning only or vertical-partitioning only methods. Effective indexes, DHT and B-tree are used at each layer respectively to help with the load balancing and scheduling.

6.2.3. Recovery Mechanism

Traditionally, data recovery is achieved by adding / storing some redundant information so that whenever the data is corrupted / lost, it can be reconstructed using the redundant information. For example, in different filesystems, checksum is usually computed for each data block to verify that the operation is performed correctly. In RAID 1, it simply uses mirroring to store additional copy of the data. In RAID 2, 3, 4, 5 and 6, they use different kinds of parity as redundant information so that the failure of one disk can be recovered. However, there are disadvantages in these approaches. First, the filesystem approach can not tolerate the failure of the entire disk. Second, the hardware is expensive. Most of the cloud computing platform use cheap commodity machines as nodes for computation, and thus RAID is usually not available on these commodity machines. Third, the redundant information is centralized. If power outage happens to the node having the RAID array, all the data will be unavailable.

To provide better availability, replication is a frequently adopted technology in the cloud. However, such recoverability is provided outside of the SaaS framework. In some cases it might be desirable that recoverability can be embedded in the SaaS framework. This chapter proposes a solution which can recover different data type with the assist of ontology information.

6.3. SaaS Customization Framework

A multi-layer customization framework *OIC* is proposed in [156]. In the chapter, all the aspects for an application can be configured through a platform, and tenant specific customization of a SaaS application affects all layers, from functional require-

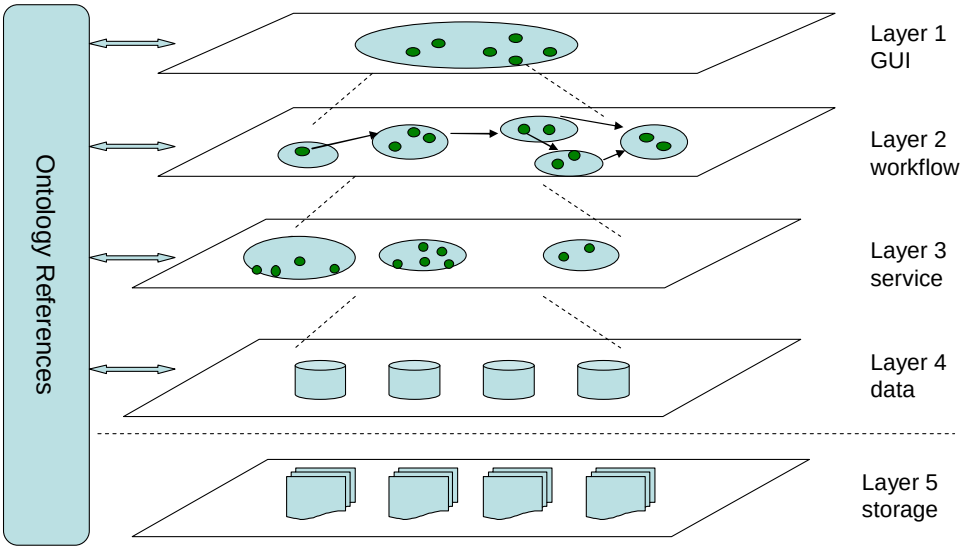


Fig. 52. Multi-Layered Architecture for SaaS Customization

ments in Graphic User Interface(GUI), customized business processes to database schemas design. The customization process is assisted by domain ontology [99], that specifies domain vocabulary and their relationships. All layers have their own ontology information, thus data ontology, service ontology, business process (workflow) ontology, and GUI ontology, that describes concepts and relations in that layer. *OIC* allows users to search for objects (data, service, workflows) in a repository, and then reuse, include or modify them as needed when designing new ones, so that the design phase will be easier and be shortened comparing with designing new ones from scratch. To deal with the commonality of tenants, a set of templates (standard) objects is provided for designers to assist SaaS customization. The template objects are stored at different repositories at all layers (including data repository, service repository, workflow repository and GUI repository). Given ontology information for a particular application domain, *OIC* uses template objects as an initial starting point, and support customization in a cost effective way. Also the recommendation

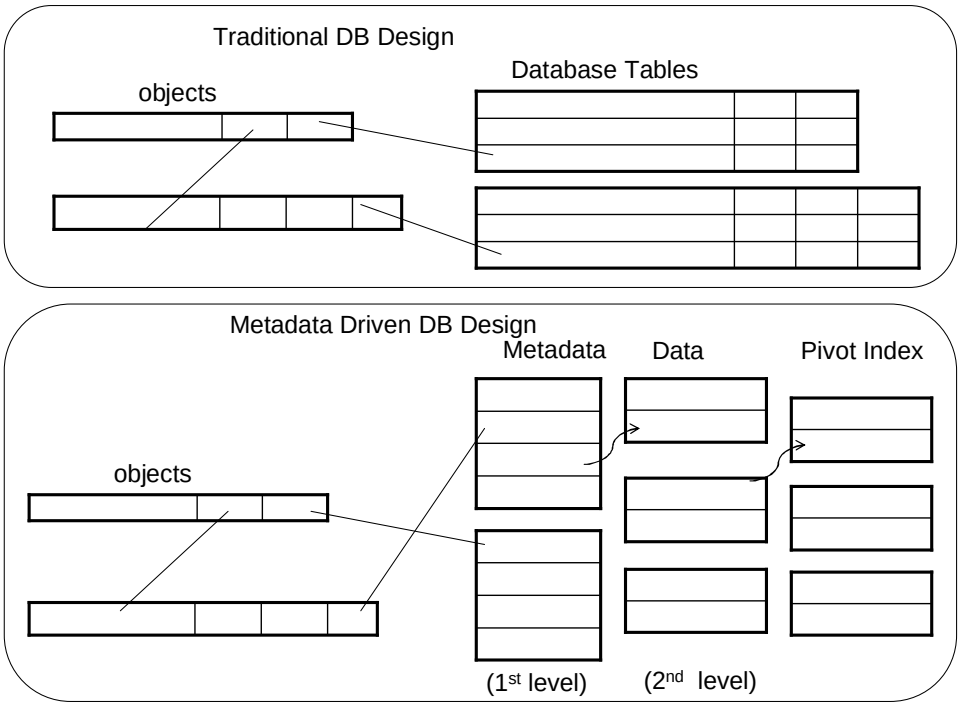


Fig. 53. Metadata Driven Database Design

engine can provide a list of candidates according to tenant's profiling.

This chapter further extends *OIC* with an additional storage management layer, as shown in Figure 77, responsible for database partitioning, load balancing, and scheduling.

6.3.1. Ontology Driven Meta-data Customization

In traditional database design, objects and fields are defined to provide abstractions of the real-world entities that they represent. Separate database tables are created for each type of object represented. Specific attributes are represented by fields within the tables. Object instances are represented by rows within the tables. Actual data is placed into a database by inserting rows into the database tables. Relationships are represented by fields in one table referring to a key field in another table.

To support MTA, a metadata-driven databases operate somewhat differently. Objects and their fields are mapped to metadata tables. Actual data is stored in either in a single data table, or, for large text objects such as documents, in a separate character large object storage (Clobs) area. A series of index tables is created to make accessing the data within the single data table more efficient. To support multiple tenants, the object and field metadata contains information about the fields, and also about the tenants. The comparison of metadata driven databases and traditional database designs are shown in Figure 53, similar as in Salesforce.com.

In details, three types of data in MTA with diverse features[165]:

- Metadata: Objects and their fields are mapped to metadata tables.
- Data: Actual data is stored in either in a single data table, or, for large text objects such as documents, in a separate character large object storage (Clobs) area.
- Pivot Index: make accessing the data within the single data table more efficient. To support multiple tenants, the object and field metadata contains information about the fields, and also about the tenants.

Ontology semantic information can be matched to database logic designs and help metadata generation. The domain objects can represent a large proportion of meta-data that are serialized into the data repository. Multiple database schemas can be used in MTA[8], such as XML, sparse table, views. Tenants can choose any database schemas as needed.

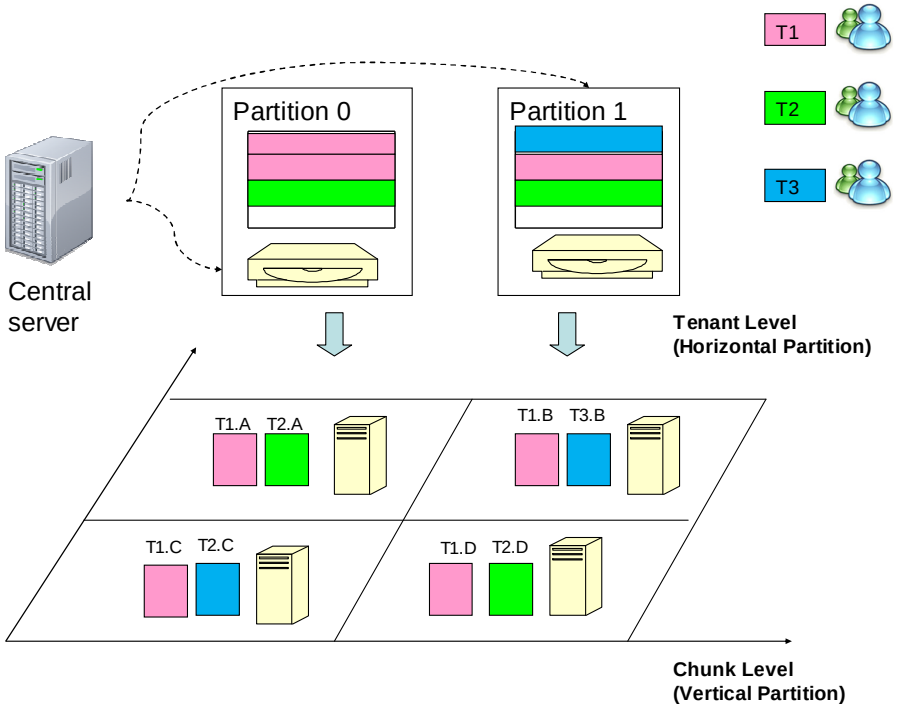


Fig. 54. Two Layer Partitioning Model

6.4. Scalable SaaS with Database Partitioning

In MTA, the shared database architecture design [29] calls for effective scalability support. In the ideal case, the maximum number of tenants should be proportional to the increase of resources, while keeping the performance metrics of each tenant at an acceptable level. There are two types of scaling: scale-up and scale-out. The scale-up or vertical scaling is done by adding additional resources, such as CPUs, memory, and disks into a single node in a clustered system. In this way, a node becomes more powerful by having more resources. The scale-out or horizontal scaling is done by adding additional nodes to an existing clustered system. For example, instead of a cluster of thirty nodes, the system may have fifty nodes instead. The scale-up is easy to use but may not provide linear scalability increase due to the

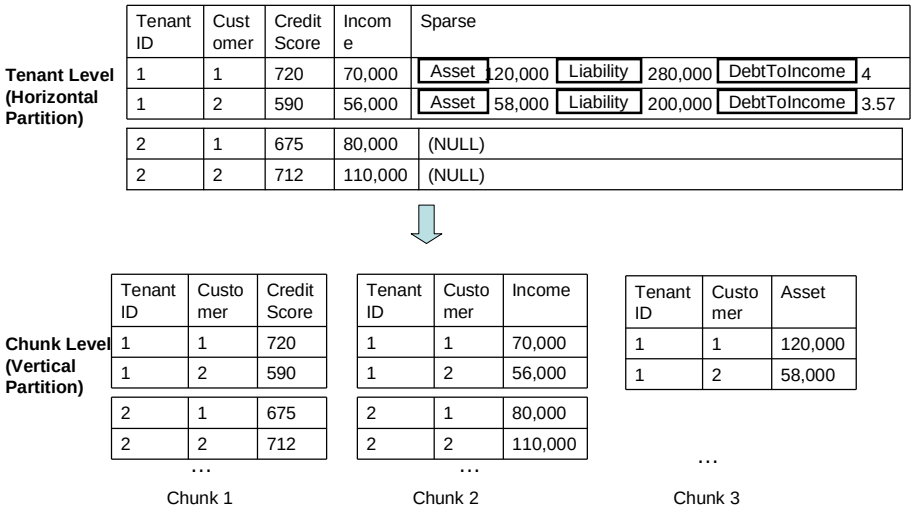


Fig. 55. Example for Two Layer Partitioning Model for Figure 54

overhead in resource management. The scale-out provides a more cost-effective way, where it can incrementally extend the system by adding more resources to a low-cost hardware set. Furthermore, it can improve the reliability and availability of the system due to the redundancy. In the scale-up scenario, one can create more than one database partition on the same physical machine, while in the scale-out scenario, partitions can be created in multiple physical machines, and each partition has its won common memory, CPUs, and disks.

With the increase of tenant's traffic, SaaS application can be easily scaled out by adding new instances, but database server becomes the bottleneck of the system scalability [37]. While most traditional database systems (e.g., DB2, Oracle 11, SQL Server, MySQL, Postgres) uses traditional data structures (e.g., dynamic programming, B-tree indexes, write-ahead logging), the differences in the implementation of SaaS are immense.

Database Partitioning [165] can improve the system performance, scalability

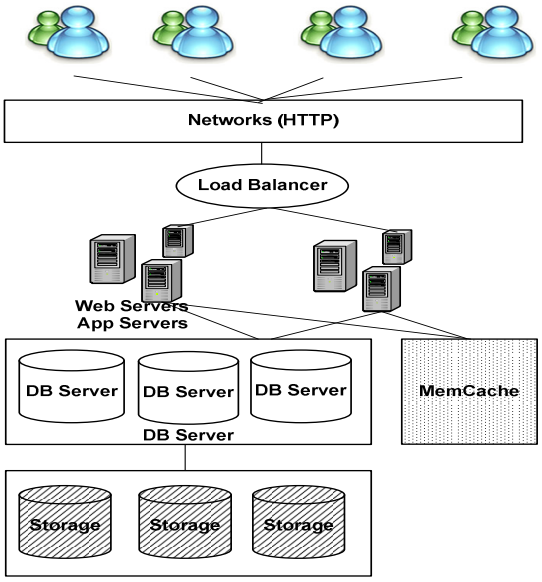


Fig. 56. Scheduling System Architecture

and availability of a large database system in a multi-tenant way. For example, given a tenant's information, the query optimizer only has to access the partitions containing the tenant's data rather than the entire table or index, using "partition pruning". Data partitioning is a proved technique that database systems provide to physically divide large logical data structures into smaller and easy manageable pieces(chunks). The data inside a database can be distributed across one or more partitions. A distribution key is the column used to determine the partition in which a particular row is stored. Instead of having one database server controlling the whole system, the database is logically partitioned and each of them can be controlled by a separate server. Indexes play an important role in improving overall performance together with partitioning. Different types of indexes are built to provide efficient query processing for different applications.

6.4.1. Review of Database Partitioning Choices

Many partitioning schemes have been studied; e.g., vertical partitioning vs. horizontal partitioning, round-robin vs. hashing vs. range partitioning [23]. Two most widely used methods are horizonal partitioning and vertical partitioning.

Row Stores and Horizontal Partitioning Key-value stores (row stores) is inherent to be the preferred data management solutions in cloud, such as Bigtable [25], PNUTS [35], Dynamo [39], and their open-source HBase [59]. These systems provide various key-value stores and are different in terms of data model, availability, and consistency guarantees. The common property of these system is the key-value abstraction where data is viewed as key-value pairs and atomic access is supported only at the granularity of single keys. This single key atomic access semantics naturally allows efficient *horizontal data partitioning*, and provides the basis for scalability and availability in these systems.

Horizontal partitioning is widely used in existing cloud computing products, such as IBM DB2 V9 [31], Force.com [46] and etc. Two horizontal database partitioning approaches are available: application-based distribution keys (choosing one or more attributes as a distribution key according to domain knowledge) and tenant-used distribution keys(stores each tenant's data in a single partition).

Update in row partition is simple and supported as follows: the storage key (SK), for each record is explicitly stored in each partition. A unique SK is given to each "insert" of a tuple in a table T.

Column Store and Vertical Partitioning Column store is a read-optimized solution, any fragment of projections can be broken into its constituent columns,

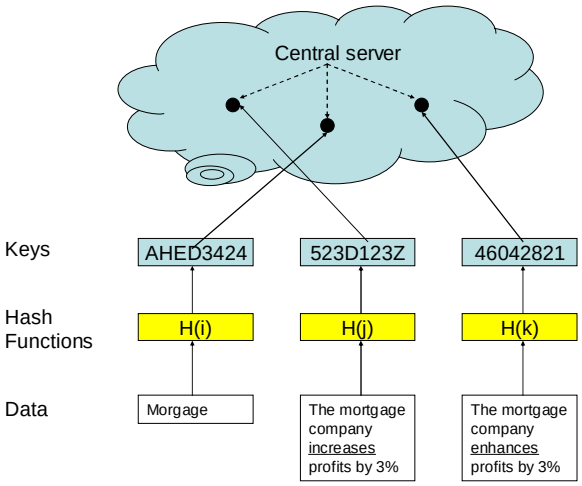


Fig. 57. Sample of DHT (Distributed Hash Tables)

and each column is stored in order of the sorted key for the projection. There are several possible encoding schemas considering the ordering and proportion of distinct values it contains, including:

1. Self-order, few distinct values: represented using triple $(val, 1^{st}, occur)$ such that val is the value stored in the column, 1^{st} is the position where val first appears, $occur$ is the number of occurrence of val in the column.

   Clustered B-tree indexes can be used over this type of columns. With large disk blocks (e.g. 128k), the height of this index can be small.

2. Foreign-order, few distinct values, represents as (val, bmp) such that val is the value stored in the column, bmp is a bitmap index, which can indicate the positions the value is stored. Each bitmap is sparse, one can run length encode to save space. To find the i^{th} value, "offset indexes" B-tree can be used to map values contained in the column.

3. Self-order, many distinct values: represent $delta$ value of the previous value

in the column. The first entry of every block is a value in the column and its associated storage key.

4. Foreign-order, many distinct values: not necessary to encode.

Update in column store is more complicated, one has to join values cross columns, in which join indexes are used to connect various projection in the same table.

Chunk is a single physical unit: the logical tables are partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. For example, in BigTable, GFS, and other similar Web databases or file system, each chunk is about 64 MB. Tenants can have multiple chunks distributed at different databases and share resources.

As a summary, row store and horizontal partitioning is writeable operation preferable, while column store and vertical partitioning is optimal for read operations. This chapter proposes a hybrid approach as SaaS involves both read and write operations.

6.4.2. P^2: Two-Layer Partitioning Model

The hybrid two-level scheme combines both read-optimized column store and an update oriented writeable store as shown in Figure 54: At the top level, there is a partition for each tenant, which can support high performance inserts and updates. At the lower level, a larger component for column partitions are supported, which can optimize for reading and batching with the tenant's attribute level. As one can see, tenant A, B, C share same physical databases, and each of them has its own

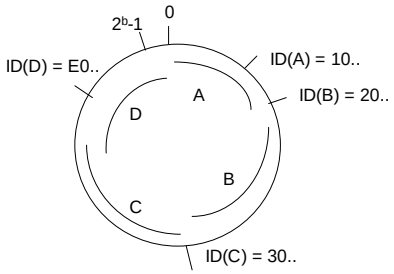


Fig. 58. Balanced Range Allocation

physical chunks associate with it respectively.

Originally, a master server is used to maintain the global index. All queries are sent to the master server to search the global index and then forwarded to corresponding servers. The size of the global index is proportional to the size of the data and concurrent requests, the master server risks being a bottleneck, hence one can further distributes the global index across servers. Each server only maintains a portion of the global index. The distributed approach improves scalability and fault tolerance. The global index is build on top of the local indexes. To search local data efficiently and make the local balancing, B-tree is used for local chunks. In the global index, a DHT index is used to make the uniform distribution among servers.

6.4.3. Scheduling and Load Balance

To do better load balancing among partitions to optimize the overall database performance, an effective algorithm is highly desirable, that can migrate, distribute and duplicate tenants among partitions through monitoring the load. Most cloud scheduling algorithms and database solutions address their problems independently. However, most of cloud components and functionalities are interconnected. Specifically, a task scheduling algorithm need to consider database partitioning to provide

DB Schema

OrgID	TableName	ColumnName	Type	Length	IsIndexed	MaxValue	MinValue
1	Customer	Income	Integer	64	0		10,000
1	Customer	CreditScore	Integer	32	1	1000	0
2	Cusomter	DebtToIncome	Float	64	0	100.0	0.0
2	Customer	Liability	Integer	64	1	1,000,000	1000
…	…	…	…	…	…	…	…

Triggers

Trigger_Schema	Trigger_Name	Created	Action_Condition	Action_Statement	Definer
Customer	Test_Income	08/01/10	Income < 10,000	Warning	Paul
Customer	Test_CreditScore	07/25/09	CreditScore < 0	Warning	Tim
Customer	On_Insert_Liability	06/30/10	Insert	Check_Liability	Tim
…	…	…	…	…	…

Fig. 59. The Metadata Table

an efficient solution for performance and scalability. More specific, a task assigned to a processor should host the appropriate data partitions otherwise data updates and migration among caches and processors can be expensive.

The most scalable MTA requires a SaaS scheduler that can dispatch tasks to multiple copies of the same software in a data center [30]. As the same version of the software is used, user customization must be stored in databases, and thus an integrated solution must address both scheduling and database partitioning together as shown in Figure 56.

Different strategies have been adopted to allocate data partitions in the cloud. One allocation strategy permits a single copy of the database to be stored in the network, non duplication. The partitions are allocated to the nodes to minimize the overall system communication cost, query response time, and other criteria depending upon the objective of the designer[22, 121]. Another strategy is to store multiple

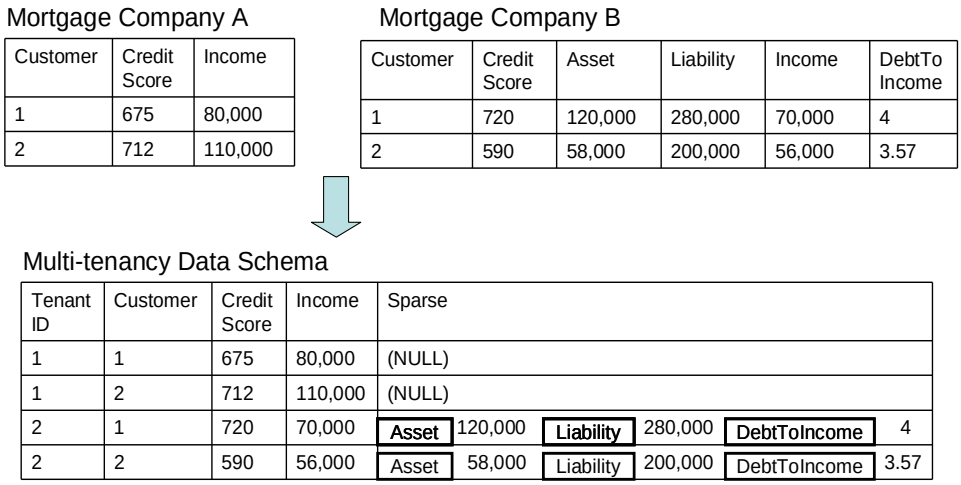


Fig. 60. The Data Table

copies of all or a part of duplications. Although this reduces transmission cost and improve response time, it increases data redundancy, storage costs, and update costs to keep data consistency. To solve this problem, sharing everything among tenants provides a solution. This chapter adapts a sharing everything framework to support scheduling and load balancing in a cost effective way.

6.4.4. Two-Layer Index for P^2

To scheduling requests and balance loads using our P^2 model in Section 6.4.2, a correspondingly two-layer index mechanism is proposed as follows:

DHT at Tenant Partitioning Level : DHT(Distributed Hash Tables) [146] can be adopted in the upper layer of partitioning for nodes among tenants.

Given a key, DHT can map the key onto a tenant's data block as shown in Figure 57. Inherent from consistent hashing [72, 139] to assign keys to blocks, the consistent hashing supports balance load, since each node received roughly the same number of keys, and involves relatively little movement of keys when add or

delete chunks from the system. Several good features are maintained in DHT, e.g. balances load with high probability (all nodes receive roughly the same number of keys), minimize maintain cost (when an N^{th} node added/deleted, only $O(1/N)$ faction of the keys are moved to a different node). Each node maintains information only about $O(logN)$ other nodes, and a lookup takes $O(logN)$ time.

One can use unsigned integers to match to the output of cryptographic hash function. It is convenient to visualize the key space as a ring of values, support b bit in the ring, starting at 0 and increase clockwise until they get to $(2^b - 1)$ and then overflow back to 0. Figure 58 shows a ring representation of Pastry-style routing [120], in which key space is divided into evenly sized sequential ranges, each node has one range, and ranges are assigned in the order of nodes, sorted by hash ID. Hence data are uniformly distributed among the nodes.

B-Tree Index at Chunk Partitioning Level To allocate and schedule chucks at the second level at least the following approaches can be applied:

(a) Allocate tenant's data with fixed partitions periodically or asynchronously: given the number of tenants k in a cloud, and the number of partitioning blocks at each tenant, it can partition the available database chunks into groups based on resource constraints and user requests. This method will decrease the contention, and each partition will be allocated to a certain copy of the software. As the workload changes, a re-allocation needs to be done. One way is to perform the update periodically, whenever the changes or the rates of changes exceed certain thresholds, or when the system slows down significantly due to unbalanced workload among different tenants.

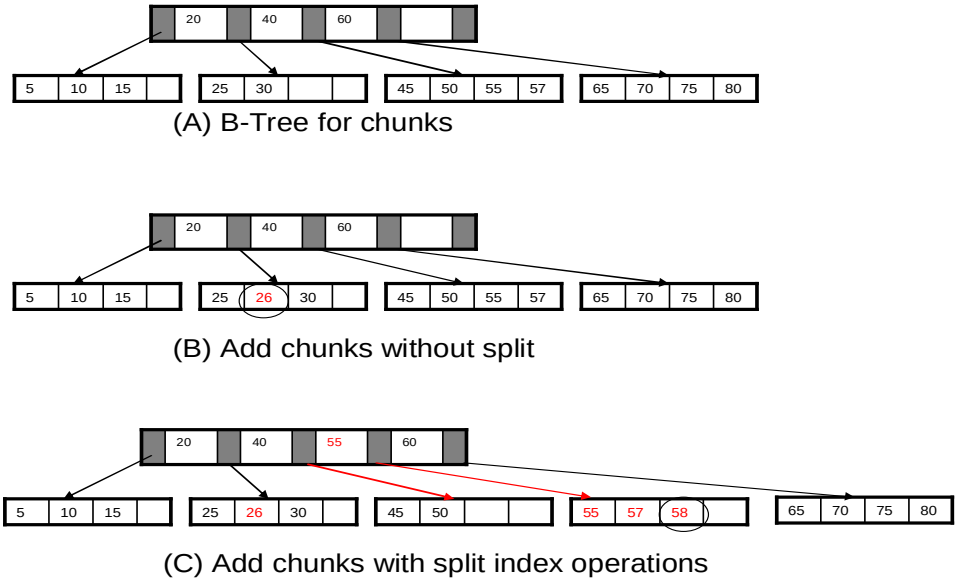


Fig. 61. Sample B-Tree for Chunks

(b) Flexible partitioning and re-partitioning. Unlike the previous approach, the partitions will be dynamically maintained as the workload changes. For example, one may use a scheme similar to B-tree to organize data partition accordingly. A B-tree allows a congested partition to double the resource, and a lightly loaded partition to reduce its resources by half, and it may be served together with another light partition in the same processor. In this way, a busy tenant can have its needs met, and a light tenant will not occupy idle resources by sharing with fellow light tenants. By using this approach, the resource can be automatically maintained and balanced.

Example: a sample B-tree for chunk partitioning is shown in Figure 61. The B-tree is used to maintain all chunks. At the beginning, each tenant can be allocated 20 chunks, and more chucks can be allocated to them when necessary. To add a chuck, either simple add operations (in Figure 61(B)) or split operations on

the index page in B-tree (in Figure 61(C)) are needed.

6.4.5. Performance Analysis for P^2

To analyze the performance of P^2, this chapter first analyzes the performance of row-store and column-store. Then it further compares the performance of P^2 with these two schemes.

Operations Composition Based Analysis: The performance of a storage mechanism can be evaluated using the access patterns. There are three types of access pattern: read mostly, write (update) mostly, and a hybrid of read and write operations.

To have a better understanding of P^2, one can start from the simple cases, and analyze the performance of row-store and column-store. The comparison of row-store and column-store is easier. At the beginning, all are update operations with no read operations (read = 0%, update = 100%), row-store has a shorter average response time than column-store, since update operations are only influence certain tuples in row-store, while column-store needs to update multiple columns in different chunks with join operations. As the read operation percentage increases to an extreme degree, all read operation and no update (read = 100%, update = 0%), column store beats row store due to its easy access to certain columns, especially when queries are focus on some specific attributes, e.g. credit score. When the distributions of read and update operations are close to each other, there would be some points, row-store and column-store have similar performances.

The case for two-layer is more complicated:

1. Mostly write-only operations: When update operations dominated the whole set of operations, the first layer of row-partitioning data will be used to store update changes. To ensure data consistency at the chunk level, similar as Dynamo's "always writeable" strategy, one can be maintained and postponed the chunk level update to back-end. At the meanwhile, the query operations are supported by two-level indexes, which can easily find out matching data, hence the average response time for the whole operation sets(both read and update operations) are shorter than row-store only, even much better than column-store.

2. Mostly read-only queries: When read operations dominate the whole operation sets, two-level indexes can support better query response time than others, and considering the mix of column store's high update costs, the average of two-layer model is better than column-store, even much better than row-store.

3. Mix of read and write operations: write operations are few but important, while read operations are frequent but less important. To satisfy the priority requirements of write operations, one can optimize write operations by adapting to some database transaction isolation models, to change the execution order of read/write operations without violating the consistency constraints.

Hence, using the proposed two-layer partition model with two-level indexes, one can improve the overall system performance.

Specially, some interesting observations here: suppose the total number of write operations is $|w|$ and the total number of reading operations is $|r|$, it is easy to see

that $|w| \leq |r|$, since most of operations for tenants are getting data and few update to save their utility costs. After commit the initial data, the tenant may seldom update the data when necessary. On the other hand, the priority of write operations can be represented as $p(w)$, and the priority of read operations is $p(r)$, as we discussed in Section 6.1, since write operations have a high priority than read operations, one can get $p(w) > p(r)$. There is a conflict between the two arguments, as shown in Eq. 6.1, in another words, write operations are few but important, while read operations are frequent but less important. Balancing the frequency and importance of these two types of operations is an interesting problem. Readers may notice that P^2 is a natural solution for this conflict: the first level horizontal partitioning can fit for the priority requirement of $p(w) > p(r)$, since update operations are favored at this level, hence one can write easily. While the second level of vertical partitioning works for $|w| \leq |r|$, in which read operations dominate the system in most of the time.

$$Conflict(occur, p) = \begin{cases} |w| \leq |r| \\ p(w) > p(r). \end{cases} \tag{6.1}$$

To satisfy the priority requirements of write operations, one can optimize write operations by adopting to some database transaction isolation models, to change the execution order of read/write operations without violating the consistency constraints. New business requirements in cloud applications force service providers to loosen the rigid constraints and adopt a more relaxable approach in transaction isolation. It is important to ensure that the relaxation of isolation does not cause

difficult-to-find problems.

The write priority optimization algorithm is suitable for cloud applications for the following reasons: In some circumstances, read operations are not necessary to get the most recent updates or it can even read data which are entered later. For example, in an online shopping application, reading customer's product list will not affect the shopping cart data. Hence one can move write operations forwards, in another words, change the order of read/write operations to adapt to high priority write operations.

On the other hand, there are specific read operations could not be postponed, due to the isolation affect. For example, a customer may want to know many books have been ordered in the shopping cart, and he/she can use "read" before issuing another write, but the write can move forward, and the read will read the most recently updated data. Another example, if a customer found that an order has been placed (for a read operation issued before the write) or an order has been removed, the customer can re-issue the read for double confirmation. This type of read as "double-confirmation-read" instead of read. Hence, three kinds of operations existing in cloud applications, including read, double-confirmation-read, and write. It is easy to see that write operations can move forward before any read operation, but not before double-confirmation-read. As we do not have too many double-confirmation-read, the optimization algorithm can be easily developed. Double-confirmation-read can be easily identified by checking whether a read operation is issued after the write operation from the same user on the same data.

When adjust the execution order of read/write operations, one can explore the

traditional database concurrency issues [115] and design an optimization algorithm for write-priority operations without violating certain constraints accordingly. The three concurrency issues includes:

- Dirty Reads: one transaction reads data written by another uncommitted transaction
- Non-repeatable Reads : one transaction read the same data twice and one write operation modify the data in between the two reads, which cause the 1st read operation got the non-repeatable value
- Phantom Reads : when one read operation gets a range of data more than once and a write operation inserts/deletes rows that fall within that range between the first transaction's read attempts, hence "phantom" rows appear/disappear

Usage View Analysis: There are two types of usage views for P^2, one is tenant-specific view for each customers, the other is cross-tenant view by cloud service providers, such as system monitoring, auditing, and performance control.

For the cross-tenant system level view, the three types of operations co-exist as well, and read operations dominate the system in most of the time. Most of operations for service providers are monitoring, and auditing, hence the system can get the statistic information of any chunks easily using P^2's partitioning model.

For the tenant-specific operations, as we discussed earlier, there are three types of operations including read/write/mix operations, and one can find out the benefit of using P^2 easily.

6.5. Conclusion

SaaS is characterized by its multi-tenancy architecture and its ability to provide flexible customization to individual tenant, which brought up various challenging problems, such as the testing of software developed with the SaaS model and built-in recoverability. This chapter presents a unified and innovative multi-layered customization framework supporting continuous testing and recoverability. Different database partitioning strategies are offered for customization. Ontology is used to derive customization and deployment information to tenants and to support continues testing and recoverability. In the future, more testing techniques will be investigated to further improve the robustness of SaaS framework. A simulation of two-layer partitioning model will be investigated to further evaluate the proposed model performance.

7. TESTING SAAS APPLICATIONS

Cloud computing has attracted significant attention recently and the issue of testing cloud applications also becomes important because many mission-critical applications will be deployed on the cloud. This chapter first discussed the unique features and challenges in testing cloud applications, and then proposed a novel SaaS testing framework that has considered three dimensions: 1) the SaaS maturity level model 2) the platform support and 3) the methodology used. The framework supports continuous testing, intelligent testing, multi-tenancy testing, scalability testing and customization testing. Scalability testing is illustrated in detail to demonstrate the advantage of the proposed framework.

7.1. Introduction

Cloud computing has received significant attention recently as it is a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and visualized manner. SaaS (Software as a Service), that is often deployed on a cloud, is a new way to deliver software. In SaaS, software is maintained and updated on a cloud, and presented to the end users as services on demand, usually in a browser. With SaaS, a software provider licenses an application to customer as a service on demand, through a subscription or a "pay-as-you-go" model. Saas also involves difficult design issues such as customization, multi-tenancy architecture, and scalability, and these three features are represented in the three maturity levels for SaaS proposed in [29].

Cloud computing has not only changed the way of obtaining computing resources, infrastructure, platform, and software services, but also changed the way

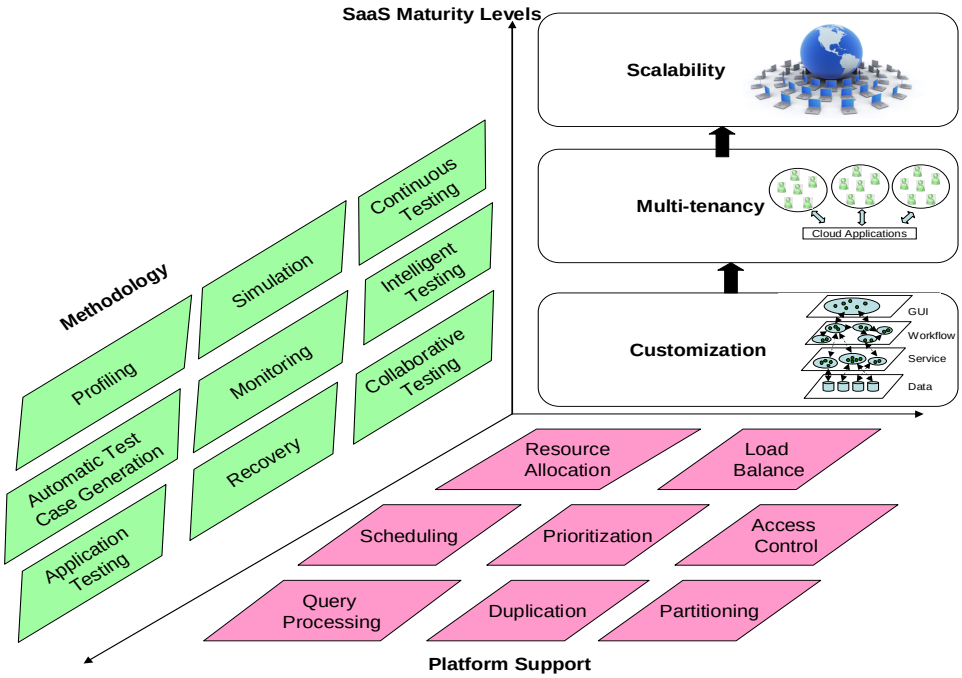


Fig. 62. SaaS Testing Framework

of managing technology and solutions for all participants, which inherently leads to new challenges and opportunities for software testing and maintenance. Requiring up to 70% of software development costs, testing has always been a cost-intensive operation in the software development process, and even more for mission-critical applications. While most research communities start to investigate the cloud architecture, technology, data model and design, as well as management in cloud application, how to effectively test cloud applications is still an open question, and a new subject.

Comparing with traditional testing and web-based service systems, testing cloud applications has several unique advantages: cost-effective (cloud computing reduces hardware and software costs by leveraging cloud resources in a pay-as-you-

go way using virtual resources), on-demand (real time large scale online validation and verification), automatic (dynamic online testing without manual cost), scalable (supporting multi-tenancy), and continuous (testing services work at anytime as "365 days-24 hours").

Testing cloud applications involves testing and measurement activities in a cloud environment by leveraging cloud technologies and solutions. Four different levels of testing in cloud are considered:

- Testing SaaS - this ensures the quality of a SaaS on its functional and non-functional requirements, e.g. SLA, and QoS.
- Testing cloud from an external system - this validates the quality of a cloud by evaluating specified capabilities and service features.
- Testing clouds from the internal - this checks the internal infrastructures of a cloud and specified cloud capabilities. Only cloud providers can perform this type of testing because they are the ones having accesses to internal infrastructures and SaaS.
- Testing over clouds - this tests cloud-based service applications over clouds, including private, public, and hybrid clouds based on system-level application service requirements and specifications.

Currently testing cloud applications often uses the traditional software testing practices. However, current software testing frameworks have not considered the features brought by SaaS and cloud computing which include:

- Scalability: The algorithm should scale up or out as the task requests may change significantly at runtime. Ideally, the increase in resources should be proportional to the increase in tenants or their requests, while keeping the performance of each task at an acceptable level;
- Database partitioning and consistency: Tenant data should be partitioned well in the backend database to support real-time high performance computing;
- Fault-tolerant aspects: The failures of processors or storage should not affect the operation or data.
- Security and fairness: Tenant data should be isolated from each other, and tenants of the same priority should receive the same level of services and resources;
- Parallel processing: It is highly desirable that the tasks can be processed in parallel such as done by Map-Reduce;
- Isolation: One tenant's change should not affect all other tenants;
- Performance and availability: The data migration, backup and restore is isolated between tenants.

This chapter proposes a novel SaaS testing framework that considers three dimensions: 1) SaaS maturity levels, 2) testing methodology, and 3) platform support for testing. There three dimensions work together in a *"cross product combination"* way, where components/methodologies in each dimension can be combined with components from the other two dimensions dynamically to get specific support

accordingly. For example, to support customization feature in the first level in the SaaS maturity model, one can choose to use any methodology, such as continuous testing, intelligent testing, and collaborative testing in the entire SaaS application life cycle. Similarly, to support customizable SaaS, multiple platform support components can be considered, such as different mechanism to allocate resources to multiple tenants, prioritize customer requests, and schedule their tasks.

The rest of the chapter is structured as follows: a novel SaaS framework is proposed in Section 7.2. Section 7.4 discusses a sample study of scalability testing in the framework. Section 7.5 concludes the chapter.

7.2. SaaS Testing Framework

The testing framework has three dimensions as shown in Figure 7.1. Each dimension has a cross product combination relationship with other dimensions. More items can be added in each dimension as more technical issues are discovered later.

7.2.1. SaaS Maturity Levels

A maturity model for SaaS with four levels is proposed in [29] with, the highest level of SaaS being configurable, support Multi-tenancy architecture (MTA), and scalable. A configurable SaaS is often achieved by customization and a scalable SaaS is often achieved by duplicating software to meet the increased load. To support customization, an ontology based intelligent customization framework is proposed in [156], which considers the customization process from GUI, business process, services and data layers using ontology information. The MTA refers to a principle where a single instance of the software runs on a server, serving multiple client organizations (tenants). With MTA, a software application is designed to virtually

partition its data and configuration, so that each client works with a customized virtual application instance. Although all tenants share the same software, they feel like they are the sole user of the software.

7.2.2. Methodology

Diverse testing methodologies can be used including continuous testing, intelligent testing, collaborative testing, profiling, and monitoring. Specially, the following methodology are essentially helpful in testing cloud applications:

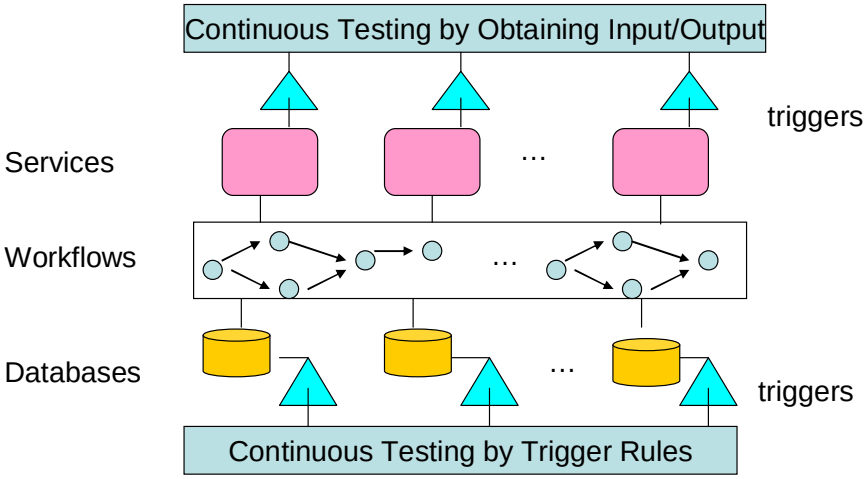
- **Continuous Testing**: This feature is needed as a cloud system often keeps on changing. The number of users and the number of tenants keep on changing, and new features and new software services will arrive at the cloud. These new features and new services need to be tested continuously to ensure the quality of software and services. For example, if a new software service is available on the cloud, and its specifications indicate that it can be used in 100 cloud applications. If any of the application uses the new service, the new application needs to re-validated with the new software service. As a cloud platform may receive many new services on a daily basis, the cloud essentially performs testing continuously. As shown in Figure 63, a cloud can use excess cycles on a developer's workstation to continuously run tests in the background, providing rapid feedback about test failures. Continuous testing is a feature introduced by Google Chrome OS, a new network system.

  An effective model of automated testing is continuous testing. It can also be part of the TDD (Test-Driven Development) process. Continuous testing

implements continuous processes of applying quality control - small pieces of effort applied frequently, in the process of software development. Continuous testing has been proposed and can be applied in various aspects in software development. For example, as proposed in [136], tests run 24 hours a day, 7 days a week and the results of these testings are efficiently processed. While in [122], continuous testing is integrated into eclipse as a tool for continuous code verification when source code changes. It uses excess cycles on a developer's workstation to continuously run tests in the background, providing rapid feedback about test failures as source code is edited. A radical design choice in the Google Chrome OS is its incorporation of continuous verification. Given the extensive usage of continuous testing, its desirable that the SaaS framework also provides built-in continuous testing capability.

Besides continuous testing, testing SaaS software can also be collaborative by nature since it is usually developed with a service oriented architecture. In the framework proposed in this chapter, we proposed to embed built-in testing capability in the SaaS framework. We provide a collaborative testing environment by generating test scripts in a collaborative manner. We integrate continuous testing with the storage layer, by leveraging database triggering rules. We also propose algorithms so that integrating testing and intelligent testing can be conducted within our framework. Figure 63 show the evolution of different models, from SOA, SaaS to continues testing model.

- **Collaborative Testing**: Testing SaaS software can be collaborative by na-



(c) Continuous Testing in SaaS

Fig. 6

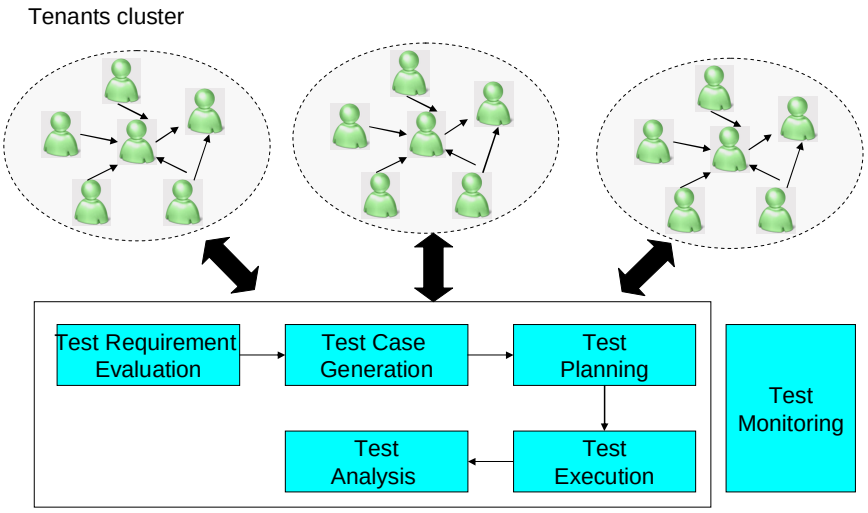


Fig. 64. Collaborative Testing

ture since it is usually developed with a service oriented architecture. A collaborative testing environment can generate test scripts in a collaborative manner as shown in Figure 64. Test scripts can be contributed by different parties or automated generated in a multi-tenancy way.

- **Intelligent Testing**: SaaS data(such as code bases, execution logs, mailing lists, and bug databases) is a good wealth of information about an appli-

cation's lifecycle. Using data mining techniques, one can fully explore the potential of this valuable data, and manage their projects in a cost effective way, produce higher-quality software systems with less bugs. Two types of information are available as data resources as shown in Figure 65: (1)Historical repositories: including source control repositories, bug repositories, and communications records of project evolution and etc. It captures dependencies between project artifacts (e.g. functions, documentation files, and configuration files). Not only handling static or dynamic code dependencies, one has to consider implicitly dependency, e.g. change of writing data may require reading data code change implicitly. Also it can be used to track the history of a bug or a feature, determine the expected resolution time according to previously closed bug resolution history. (2)Real-time repositories: including deployment logs with execution information and system usage logs from multi-tenancy. By monitoring the execution, one can find out the dominant execution or usage from logs, and tune the system performance accordingly. Similarly, one can mine the dominant APIs usage patterns by monitoring code repositories.

- **Automatic Test Cases Generation from Metadata:** Test cases can be generated by examining metadata, e.g. Income length of customer must be 64 bits or so, hence some simple test cases will be randomize with 64 bits. One can generate a collection of customers of 64 bits, another collection with 128 bits or any other bits. Random number from $0 - (2^{64} - 1)$, e.g. another

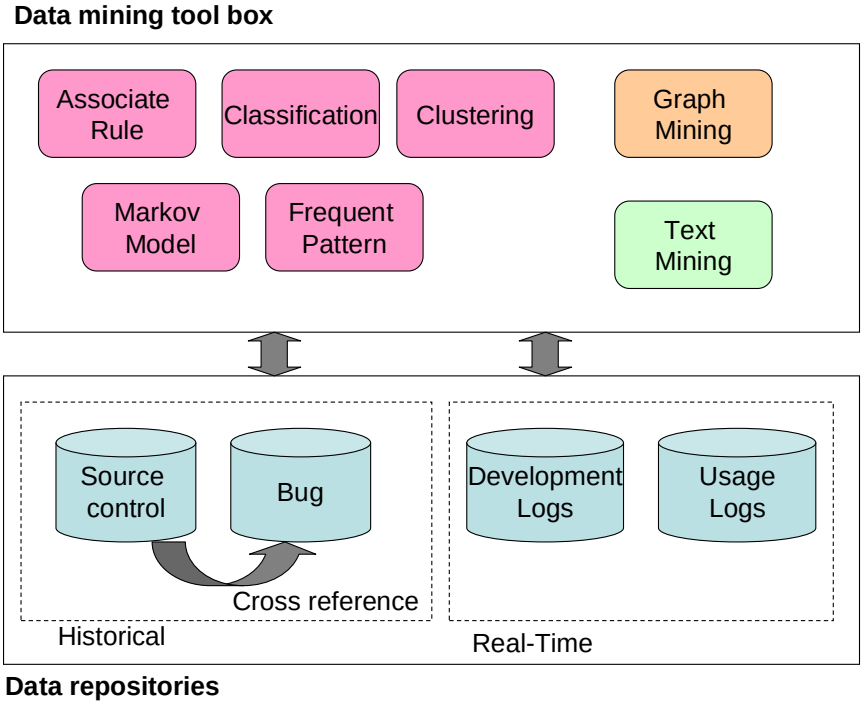


Fig. 65. Tool Box of Data Mining Algorithms and Data Repositories

set is negative numbers, and greater than $2^{64} - 1$, so we have three set of values, one valid and other two are invalid. The boundary value test cases can be generated from {-2, -1, 0, +1, +2} around boundary of the constraints, specifies by the metadata. For example, credit score > 0 is an invalid testing put, credit score $= 0$, 1 are boundary test values. Several test case generations from ontology including constraints has been proposed [11, 10] and can be used in our framework.

- **Test Cases Ranking** Based on the WebStra's framework, test cases can be ranked[159], and based on the importance, and history, test result oracle can be established by voting[12], test case dependency can be automatically analyzed using the test results based on statistic techniques. without canalizing software

structures, a large collection of test cases can be constructed, ranked and evaluated on a continuous bases.

7.2.3. Platform Support

To support the other two dimension, platform has to provide sufficient supports as shown in Figure 7.1. Take scheduling and recover as examples:

- **Scheduling:** Manage processors to handle testing tasks properly. Different scheduling strategies can be suitable for a local cluster managed by a processor or a virtual machine (VM) to improve its SLAs with users, such as FCFS (First Come First Serve), EDF (Earliest Deadline First), weighted queue (with priority), selective (based on ratio of (wait time + run time) and run time) and etc. Map-Reduce can be used to scheduling testing jobs in a decentralized way.

- **Prioritization:** Request prioritization presents a challenge in MTA Tenant may have individual (local) prioritization requirements, and these requirements can be different for different tenants. The shared application must use a global priority scheme for requests from all the tenants. [154] proposes an effective model to prioritize service requests from multiple tenants while preserving local priorities from individual tenant requests. The Crystalline Mapping (CM) algorithm which maps local priorities from individual tenants to global priorities. The algorithm also maximizes revenues within the local to global priority mapping constraints.

- **Recovery:** SaaS calls for its built-in recoverability, a tripartite scheme [155],

i.e., ontology, metadata and data, is proposed to support recoverability. Specically, any information in one aspect, such as ontology, metadata and data, can be used to recover data in other aspects. For example, ontology information can be used to recover metadata, and metadata information can be used to recover ontology, and data can be used to recover ontology and metadata.

In the following discussion, a sample case of scalability testing with intelligent, and platform support are provided.

7.3. Policy Enforcement

Policies represents the expected software behavior, which are enforced at runtime to ensure that the software execution conforms to the requirements. They are derived from business goals and service level agreements(SLA) in enterprises, which are "rules governing the choices in behavior of a system" [135]. Policies includes obligation policies(event triggered condition-action rules), authorization policies(define what services or resources a subject can access) and etc. This chapter is focus on obligation policy to manage SaaS testing process. The *Obligation Policy (OP)* defines a tenant's responsibilities, what activities a subject he must(or must not) do. In general, obligation policies are event-condition-action rules (ECA) as trigger rules, in the format of

On *Event* ***If*** *Condition* ***Do*** *Action*

The event part specifies when the rule is triggered; the condition part determines if the data are in a particular state, in which case the rule fires; the action part describes the actions to be performed if the rule fires. ECA systems receive

inputs (mainly in the form of events) from the external environment and react by performing actions that change the stored information (internal actions) or influence the environment itself (external actions).

Two general ways to address the faults using trigger rules, one emphasized prevention, e.g. developing a formal trigger rule to ensure the decidability and completeness of a trigger rule system, which prevents anomalies. The second is to design a mechanism for handling various faults or failures during the execution of trigger rules, e.g. develop sophisticated plans for any possible results, which either eliminate the adverse effects or minimize the bad effects. This chapter uses the second method, and proposes policy enforcement framework, which not only uses trigger rules, but also contingency plans.

SaaS are applied to increasing complicated, non-conventional application areas with real-time constraints, the probability of faults during the execution of trigger rules increase greatly. A trigger service in SaaS become increasingly complicated in handling the faults, such as failures and aborts, which may occur during the execution of SaaS customization. This chapter models failures, aborts and other fault situations as events in the ECA paradigm, hence the contingency plans for handling fault events can be modeled as trigger rules.

7.3.1. Policy Enforcement Triggering Rules

Policies are often enforced in service application when a service is been involved, for example, WS policy, XACML [96] and other policy standards, however, the policy used in the chapter are derived from constraints, in the metadata and they may need to be enforced whenever data are changed, other than a service is involved[150], and

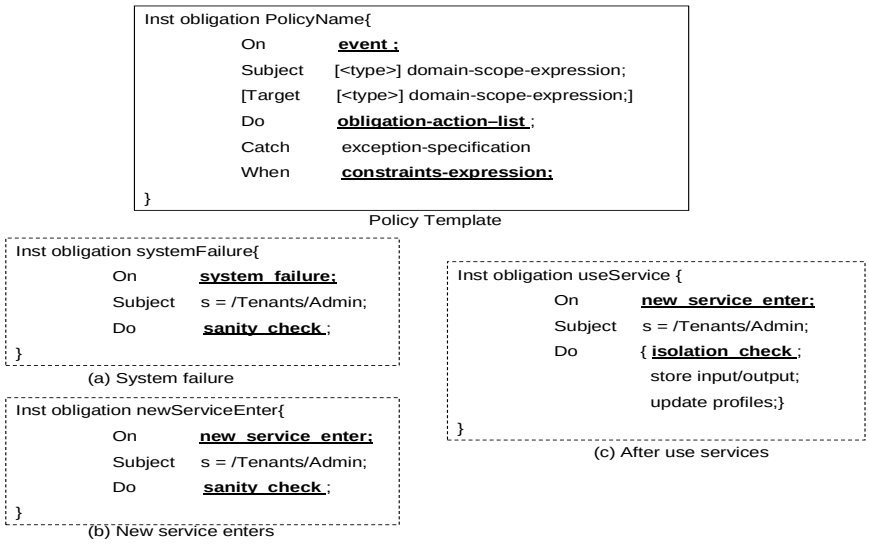


Fig. 66. Sample Policy Specifications

also in the SaaS environment, multiple threads and services may be active at a given time, and may cause multiple data to be access or updated concurrently, and thur one needs different policy triggering rules, other than the traditional service invocation events. The following events are selected sample of policy enforcement triggering events.

1. There is a failure in the system somewhere, this is for sanity check; (Figure66 (a))

2. Before a service will be used (to ensure that the service is in a good shape, this is similar to acceptance testing); (Figure66 (b))

3. after a service has been just used (to ensure that the running does not affect the software), and store the input/output pair, to update the profile;(Figure66 (c))

4. Whenever a new service with the same service specification arrives;

5. Whenever a new application is created to specify that it intends to use the service (this is equivalent to testing during development)

6. If the service is replaced by another one as the previous one has some bugs or performance issues;

7. Certain time period has passed. For example, one week has passed, and the system is not sure that something is wrong, this is more like a sanity checking;

8. Whenever the new resources are added into SaaS during execution; new resources may cause issues, and need to work on scalability issues (scale out);

9. Whenever an existing resources is removed from the SaaS during execution, a reduce resource may cause issues, and need to work on scalability issues (scale in);

10. Whenever the cloud platform has a change in configuration: to ensure scalability issues (scale up and down), to maintain performance and so on

11. Whenever the output produced des not match with the predicted output;

12. Whenever a new input that has not occurred before arrive, and the new input may reveal new bugs not known;

Some sample rules can be specified as shown in Figure 66.

7.4. Sample Study: SaaS Scalability Testing

One of the important issues and challenges in cloud computing and SaaS is system scalability and performance testing. In the past decades, there were numerous

researches focusing on scalability and performance testing and evaluation ([79, 68, 105, 142, 144, 145, 143]). Most frequently used three related kinds of metrics reported in [68] are: speedup (how the rate of doing work increases with the number of processors R, compared to one processor), efficiency (the work rate per processor ($E(R) = S(R)/R$)) and scalability (from one scale R_1 to another scale R_2 is the ratio of the efficiency figures for the two cases: $\psi(R_1, R_2) = E(R_2)/E(R_1)$.). Most of them are not suitable for cloud-based application systems since they mostly address the scalability evaluation and performance validation for software applications in pre-configured homogeneous or heterogeneous distributed and/or parallel environments.

There are several reasons why these metrics are not enough for scalability testing in the cloud. First, the complexity in the cloud environment is not considered in these metrics. Second, different workload runs on the cloud application might demonstrate different performance, the scalability metrics should keep these performance variance in consideration. More specifically, the following facts of SUT (SaaS Under Test) and applications in cloud infrastructures make the cloud-based testing more complicated than traditional testing: (1) Operated under a scalable testing environment - For a cloud-based application, a testing environment usually is online and scalable environment in a third-party cloud or private cloud. It must be supported with scalable computing resources auto-provisioned by a cloud or over clouds. (2) Applied with virtual and real-time traffic data - Each SaaS and cloud-based application must be validated online with both simulated and real-time traffic data and user accesses. (3) Targeted at any evolved software and its opera-

tion contexts - In cloud testing, the validation target is not only software itself but also its operation contexts, including software hardware/software configurations and organizations, its meta-data, and supporting database.

To address all these challenges in cloud application, this chapter proposes a set of preliminary metrics for testing cloud applications.

7.4.1. Testing Scalability of Cloud Applications

Different scalability metrics can be used to measure the scalability of a system. Following is a preliminary set of possible scalability metrics.

1. T (processing time), which reflects the traditional speed-up
2. $R * T_r$ (resouce consumption, which reflects the resouce usage in the system
3. PRR (performance resource ratio), which will be defined in the following and reflects the relationship between performance and the resources used.
4. Metric variance, such as the variance of speed-up, variance of resource consumption and variance of PRR. The variance fo PRR will be defined later in this chapter.

7.4.1.1. *Performance/Resource Ratio (PRR) Measurement.*

The performance of system running on the cloud needs to be measured considering not only the time it requires to do the computing, but also the resources consumed in the process. Therefore, this chapter considers the performance/resource ratio (PRR) for measuring the performance of the SUT in cloud. To define PRR, T_w and

C_R are defined as follows.

$$T_w = T_q + T_e$$

$$C_R = \sum R_i * T_i$$

where T_w represents the waiting time, T_q the queueing time and T_e the execution time, R_i is the allocation of resource i, which can be I/O bandwidth, CPU and memory usage and T_i is the time resource i is used. PRR is defined as:

$$PRR = \frac{1}{T_w} * \frac{1}{C_R}$$

Given PRR, the scalability of the SUT is measured by the PC (Performance Change) when workload changes.

$$PC = \frac{PRR(t)W(t)}{PRR(t')W(t')}$$

with the ideal PC equals to unity.

However, the cloud system is so complex that the PC measured from the SUT might vary between different test runs. Therefore, not only should PC be considered, but also the performance variance PV. Given the performance change PC, the performance variance PV effectively measures the scalability of the SUT when the workload changes. In the ideal case, after multiple run of the same testing workload, a small PV shows that the SUT has a good scalability. A truly scalable system should have both PC close to unity and PV close to 0. The PV can be computed by the standard variance of the PC in multiple runs of the same workload as follows.

$$PV = E[(PC_i - \frac{1}{n}\sum_{i=1}^{n} PC_i)^2]$$

Now we illustrate the metrics used in this chapter through several examples.

Example 7.4.1 *Suppose that the system have a total work of* $W = 100$ *to be processed. The waiting time* $T_w = 2$ *and the resource consumption* $C_R = 2$*, the PRR in this case is* $\frac{1}{2} * \frac{1}{2} = \frac{1}{4}$*. When the workload increases to* $W = 200$*, suppose* T_w *is now* 4 *and* C_R *stays the same, then the PRR is* $\frac{1}{8}$*. In this case, then* $PC = \frac{\frac{1}{4}*100}{\frac{1}{8}*200} = 1$*, which shows that the SUT has a good scalability since intuitively, if the workload increases without adding resource, the performance should degrade proportionally. Measuring the performance variance of PC shows whether the performance change is stable for the SUT.*

Example 7.4.2 *Considering the case when workload doubles, in this case* $W(t') = 2W(t)$*. If the system needs to allocate 4 times resources to keep the waiting time the same,* $P(t') = \frac{1}{4}P(t)$*. According to the definition of PC, now the value of PC is 2, which shows the scalability of the system is not good enough.*

Example 7.4.3 *Considering the case when workload does not change, and twice resources are allocated and the waiting time remains the same. In this case* $W(t') = W(t)$ *and* $P(t') = \frac{1}{2}P(t)$*. According to the definition of PC, the value of PC is 2, which shows the scalability of the system is not good enough. This is consistent with the intuition that adding resource should improve the system performance*

Given measurement x of the PC of a system, the scalability of them system can be observed by the regression of x. The distribution of x might be the indicator of the scalability mechanisms in the cloud. Different cloud scalability mechanism exist, such as

1. Lazy and mean allocation (on demand), which allocates resources as late as possible and as little as possible.

2. Lazy but generous (on demand), which allocates resources as late as possible, but more generous such as doubling the resources each time (like B-tree).

3. Lazy but intelligent (on demand), which allocates resources as late as possible, and allocates by estimating the workload using profiling information.

4. Estimate, which estimates the resource needed and allocates accordingly.

The list above describes just some possible mechanisms. The increase function of PC might have some correlations with the scalability mechanisms used in the system. By finding the correlations between the trend of PC and the scalability mechanism, useful information might be discovered for guidance of the scalability mechanisms should be used in the given system.

Example 7.4.4 *For example, for the first lazy and mean allocation scalability mechanism, the observed performance change might be quadratic to x while the lazy but generous mechanism can have a linear performance change. However, resources come at a price. The price factor might also be added into the scalability metric.*

7.4.1.2. *Significant Test for Scalability Analysis*

To determine whether model M(before the scalability test) has significant difference with model M' (after the scalability test) , one can adopt standard T-test. Let X represent the distribution of model M on $PRR(M)$, where $X \sim N(\mu_1, \sigma_1^2)$, and Y to represent the distribution of M' on $PRR(M')$ where $Y \sim N(\mu_2, \sigma_2^2)$, and

$\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ are unknown variables. First, make a hypothesis: the means of these two normally distributed model M and M' are equal, which is $H_0 : \mu_1 = \mu_2$. $PRR(M)$ and $PRR(M')$ can be treated as a sample from the whole space. The number of independent instances of X is denoted as m, which is equal to the size of $PRR(M)$; and the number of independent instances of Y is n, which is equal to the size of $PRR(M')$. Their means and variances as following: $\overline{X} = \frac{1}{m}\sum_{i=1}^{m} X_i$, $\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$, $S_{1m}^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \overline{X})^2$, $S_{2n}^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^2$.. The t statistic can be calculated as follows:

$$t = \frac{\overline{X} - \overline{Y}}{\sqrt{mS_{1m}^2 + nS_{2n}^2}} \sqrt{\frac{mn(m+n-2)}{m+n}}, \tag{7.1}$$

where s_{1m}^2 and s_{2n}^2 are unbiased estimators of the variances of two sets. Compare the calculated t-value with the threshold chosen for statistical significance α (usually α is 0.10, 0.05, or 0.01 level). If $|t| \leq t_{\frac{\alpha}{2}}$, then the hypothesis that the two models do not differ is accepted.

7.4.2. Methodology Dimension: Intelligent Testing to Assist Scalability Testing

Given the history of system scalability testing, many intelligent testing can be applied to further assist testing process. One can find out the bottleneck of the scalability using feature selection algorithms, also association rule can used to mine the correlation among parameters, as well as mining input-out relationship to reduce test case numbers.

Case 1: Feature Selection to mine the bottleneck in scalability testing Let's consider a sample result from scalability testing using PRR measurement in Section 7.4. There are more than 50 features (CPU, memory, network bandith, database

partitioning, and etc.) which could affect the scalability, how can one decide which ones are key features decides the scalability performance, and only testing those features can reduce the complexity of testing scalability, as well as find out the solution to improve the system performance. One may have a better understanding of the cloud applications by using intelligent testing in this way.

Case 2: Association Rule to decide correlation among parameters and scalability One can mine the association rule of different parameters and their relationships with scalability using testing data. For example, when both workloads of network and number of tenants increase, the scalability decreases.

Case 3: Mining Input-output relationship to reduce test cases A list of input attributes relevant to a single output (in a single-objective model) or to several outputs (in a multi-objective model). This list can be usually derived from the structure of the induced model. As shown in [128], the knowledge of input-output relationships can significantly reduce the amount of test cases.

7.4.3. Platform Support Dimension: Partitioning to Assist Scalability Testing

At the platform support dimension, different components can be applied to help with the scalability testing. To support scalability testing in cloud, data partitioning becomes a widely accepted solution. A novel two-layer model for partitioning is provided [155], which first partitions horizontally by tenants, and then vertically partitions by columns. The model can benet both read and update operations comparing with horizontal- partitioning only or vertical-partitioning only methods. Effective indexes, DHT and B-tree are used at each layer respectively to help with the load balancing and scheduling.

7.5. Conclusion

This chapter proposes a novel testing framework for SaaS, which has considered three dimensions: 1) the SaaS maturity level model 2) the platform support and 3) the methodology used. Testing capability is built-in in the SaaS testing framework. A collaborative testing environment by generating test scripts in a collaborative manner. Scalability testing is illustrated in detail to demonstrate the advantage of the proposed framework. In future, we will further investigate the correlations of different dimensions, as well as more testing methodology to support SaaS maturity model.

8. ROLE-BASED ACCESS-CONTROL USING REFERENCE ONTOLOGY IN CLOUDS

In cloud computing, security is an important issue due to the increasing scale of users. Current approaches to access control on clouds do not scale well to multi-tenancy requirements because they are mostly based on individual user IDs at different granularity levels. However, the number of users can be enormous and causes significant overhead in managing security. RBAC (Role-Based Access Control) is attractive because the number of roles is significantly less, and users can be classified according to their roles.

This chapter proposes a RBAC model using a role ontology for Multi-Tenancy Architecture (MTA) in clouds. The ontology is used to build up the role hierarchy for a specific domain. Ontology transformation operations algorithms are provided to compare the similarity of different ontology. The proposed framework can ease the design of security system in cloud and reduce the complexity of system design and implementation.

8.1. Introduction

Cloud computing receives significant attention recently. Public clouds are available from Amazon, Google, Yahoo!, Microsoft, Salesforce.com and others. Private cloud technologies, in which the cloud software is loaded locally are available from VMware, Eucalyptus, Citrix, and there are thousands of vendors offering "cloud solutions".

However, storing valuable business data online creates a situation similar to storing "money", attracting frequent assaults by malicious attackers. As a result,

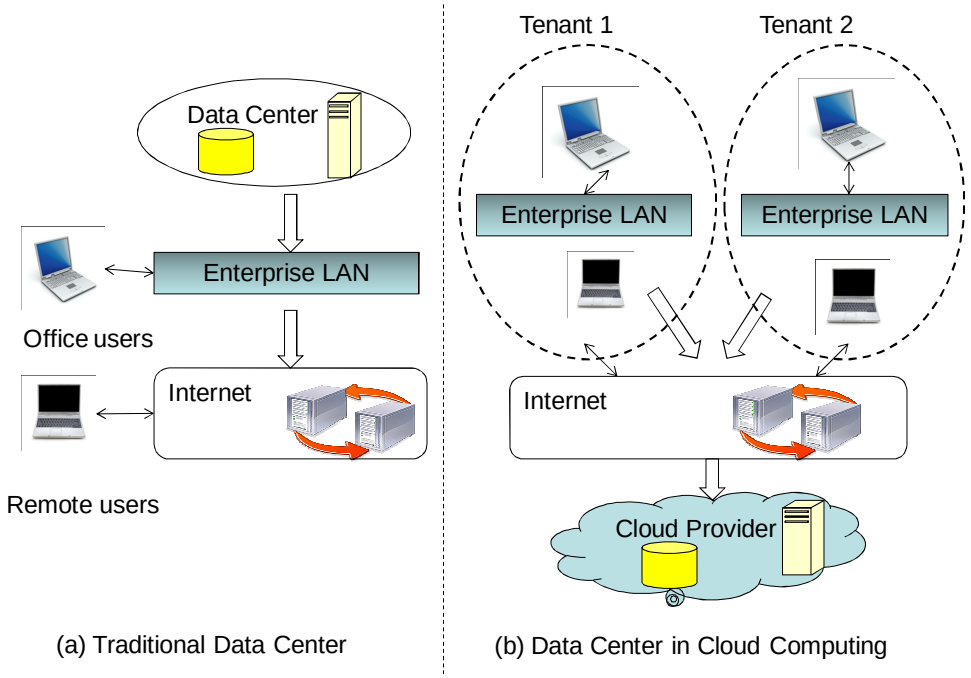


Fig. 67. Difference of Cloud Computing with Traditional Data Centers

security is a high priority issue in clouds. Several interesting concerns are often embedded in customers' mind, such as: Can cloud employees/administrators be trusted to not to look at private data or change it? Can other customers of the cloud access private data by any means including hacking? The security and privacy violations of business data can be devastating. Several cloud security accidents had already happened. One of the notable security incidents occurred in March 2009 with Google Docs, when a system failure allowed the content of private documents to be exposed to everyone for a brief period of time. As a result of this security breakdown, The Electronic Privacy Information Center (EPIC), filed a detailed complaint with the Federal Trade Commission to request an injunction against Google offering their cloud service until "safeguards are verifiably established" claiming. Google's inadequate security is a deceptive business practice.

Cloud security and vulnerability are similar to the traditional issues in networking and applications. In a cloud environment, security mostly depends on the security mechanisms supplied by cloud providers. They control the hardware and the hypervisors on which data are stored and applications are run. A cloud and conventional data center share many characteristics. However, in the cloud, due to multi-tenancy architecture (MTA), data from multiple clients are stored and managed by the *same* software [5]. When the software makes a mistake, potentially millions of clients may access private data of other clients. Furthermore, data stored in a cloud may be available to cloud administrators and they may access or modify data for their own benefits. Figure 77 illustrates these issues.

The MTA has increased the security risk due to the sharing of software, data and data schemas by multiple tenants. As these collocated tenants may be competitors, if the barriers between tenants are broken down, one tenant may access another tenant's data or interfere with their applications. The cloud providers are responsible for ensuring that one customer cannot break into another customer's data and applications.

A simple access control mechanism is *user based access control (UBAC)* as shown in Figure 68 (A). An authorization states whether a subject can perform a particular action on an object which are stated according to the access control policies of the organization. The system can accept query "Can user U perform action A on resource R?" and return Yes(Y) or No(N) answer. Unfortunately, UBAC is not suitable for cloud computing. As cloud applications usually contain millions of users and resources, instead of specifying policies for individual tenants,

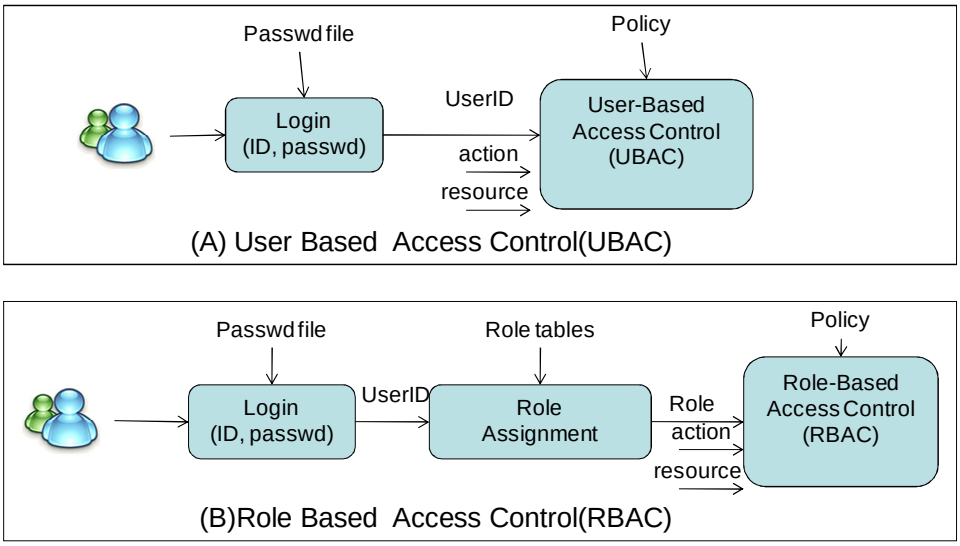


Fig. 68. User Based Access Control vs. Role Based Access Control

it is more practical to specify policies relating to groups of entities with similar functionalities. It is helpful to cluster the policies pertaining to the duties of a role within an organization such as a project manager, and senior developers.

Another approach is to use *role-based access control (RBAC)* as shown in Figure 68 (B). A RBAC system has two phases in assigning a privilege to a user: in the first phase, the user is assigned one or more roles; and in the second phase, the roles are checked against the requested operations. In RBAC, permissions are associated with roles rather than users, thus separating the assignment of users to roles from the assignment of permissions to roles. Users acquire access rights by their roles, and they can be dynamically re-assigned or removed from roles without changing the permissions associated with roles. The number of roles is typically much smaller than the number of users.

Roles may have a hierarchical structure, and it reflects the organization's lines of authority and responsibility. For example, Figure 69 is a sample fragment of role

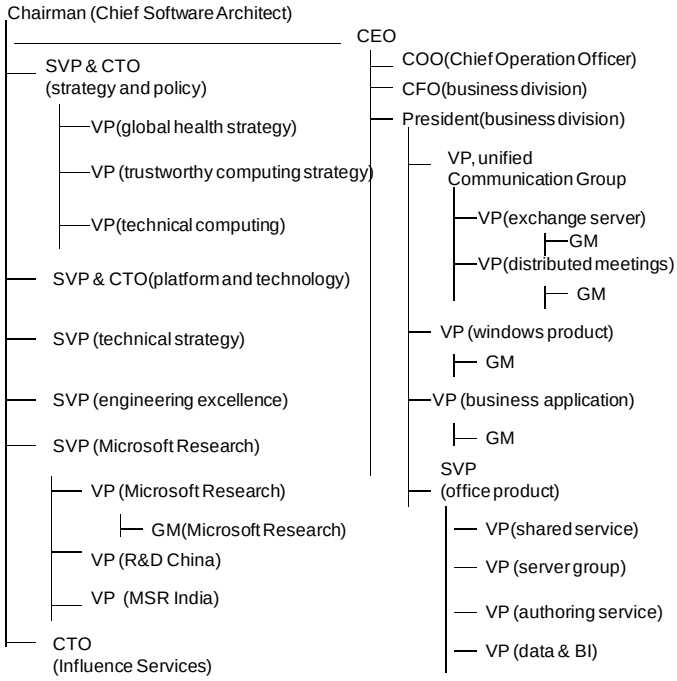


Fig. 69. A Sample Fragment of Role Hierarchy from Microsoft

hierarchy from Microsoft. Different roles, such as CEO, CTO, and VP are arranged in the diagram, where junior roles appear at the bottom and senior roles at the top.

However, it is not clear how to define roles for a specific application domain. For example, if a tenant from a start-up company tries to build up its own access control model for its application, it is difficult to start from scratch and define role hierarchy and related policies.

Our strategy is to use RBAC model to control MTA in cloud. Intuitively, one may find that ontology information can be used to map users to roles and build up the role hierarchy. This chapter proposes a reference ontology framework, in which users can search ontology database given a specific domain to find out relevant candidate role hierarchy templates, and further get the corresponding policies associated with the templates to help with their own designs.

There are several challenges in applying RBAC model in this problem. For instance, how to define and manage the roles in a cloud? Given a specific domain, there are more than one ontology systems provided by different applications, how to compare the similarity between ontology systems? How to transfer one ontology to another? How to define the policies associated with different roles? This chapter investigates these issues and designs a reference ontology framework using ontology for RBAC that generates good recommendations and ease the security management process. In summary:

- This chapter proposes a reference ontology framework for access control in a cloud to facilitate the design of security system and reduce the complexity of system design and implementation.
- This chapter exploits the possibility of RBAC to support MTA in a cloud. Ontology information is used to build up the role hierarchy. Ontology transformation operations algorithm is used to compare the similarity of different ontology.
- This chapter investigates different policy models and each of them can be used as a component in the proposed framework.
- This chapter discusses impact of RBAC in applications using a case study.

The chapter is organized as the following. Section 8.2 discusses RBAC and its components; Section 8.3 presents the architecture of reference ontology framework; Section 8.4 define the role definition using ontology information, and the transformation operations in ontology trees; Section 8.5 investigates the possible policy

strategies can be used in a cloud; Section 8.7 discuss the related work; and Section 8.8 concludes this chapter.

8.2. Role-based Access Control

Motivated by the need to simplify authorization administration and to directly represent access control policies of organizations, RBAC was first proposed by Sandhu [125] and it has been further developed [101, 124, 117, 104, 3] ever since.

A *role* represents a specific function within an organization and can be seen as a set of actions or responsibilities associated with this function. In a RBAC model, all grant authorizations deal with roles, rather than being granted directly to users. Users are then made members of roles, thereby acquiring the roles' authorizations. User access to resources is controlled by roles; each user is authorized to play certain roles and, based on his own role he can perform accesses to the resources and operate them correspondingly. As a role organizes a set of related authorizations together, it can simplify the authorization management. Whenever a user needs a certain type of authority to perform an activity, s/he only has to be granted the authority of a proper role, rather than directly assigned the specific authorizations. Furthermore, when she changes her function inside the organization, she needs to revoke the permission function of the role. Complicated cascaded authorization revoke operations are no longer needed.

RBAC ensures that only authorized users are given access to certain data or resources. It also supports three well-known security principles: information hiding, least-privilege, and separation of duties.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the orga-

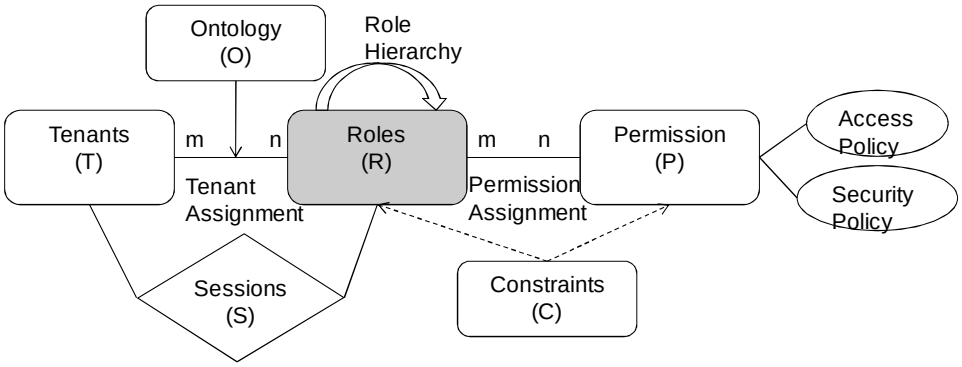


Fig. 70. O-RBAC: Using Ontology for Role-Based Access Control Model

nization's lines of authority and responsibility. By convention, junior roles appear at the bottom of the hierarchic role diagrams and senior roles at the top. The hierarchic diagrams are partial orders, so they are reflexive, transitive, and antisymmetric.

Several RBAC models are provided when integrate constraints, sessions and other information into the basic model. A general family of RBAC models was defined in [125].

8.3. Reference Ontology for RBAC in Clouds

This chapter proposed a reference ontology framework using Role-Based Access Control (O-RBAC) model which provides an appropriate policy with a specific *role* instead of specific *tenant*.

Figure 70 shows the proposed O-RBAC model with basic components, includes:

- Tenant: a user in the cloud or a human being
- Role: a named job function within the company which describes the authority and responsibility conferred on a member of the role. A role is classified according to the security requirement in the system.
- Permission(authorization, access right, privilege): an approval of a particular

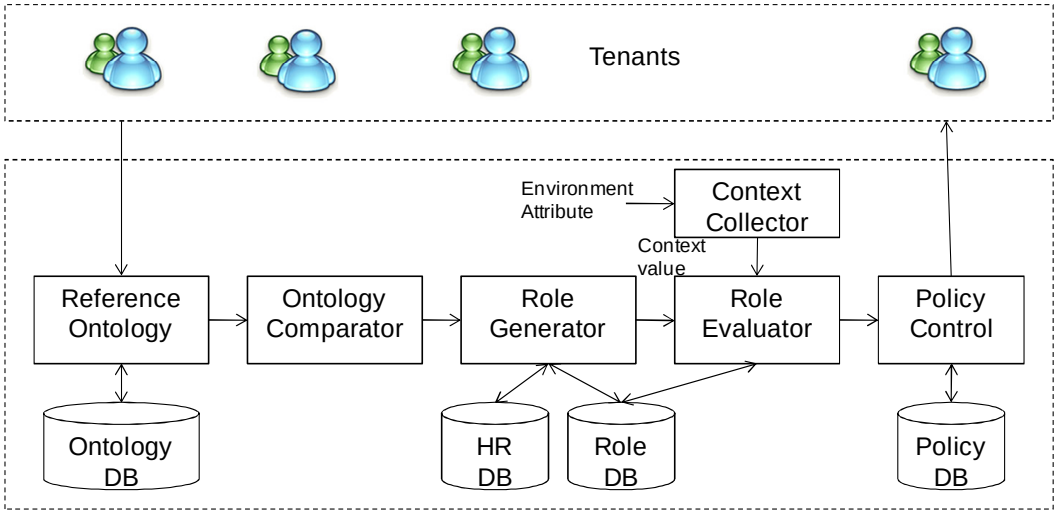


Fig. 71. RBAC using Reference Ontology Framework in Cloud

mode of access to one or more objects in the system. Policy is used as an extention of permission in the framework, including access policy, security policy are etc.

- Constraints: restricting conditions which might be applied to the policies. When applied, constraints are predicates that return a value of acceptable or not acceptable.

- Sessions: users establish sessions during which they may activate a subset of the roles they belong to. Each session maps one user to possibly many roles.

Note that one tenant can have multiple roles in different sessions, and it could happen in reality, for example, a tenant becomes sick or out of town for business, also some employee may work in branch A in the morning and branch B in the afternoon, in that case, a role is defined in certain session, the model can avoid conflict of interest very well.

Several differences from traditional RBAC: the role hierarchy is based on do-

main ontology, and can be transferred between different ontology. Also, the permissions are composed of policies, including access policy and security policy, and specific policies may become components of a role according to the role's characteristics, such as priority and business values. The access authority of each tenant can be assigned to tenants by various policies without any changes, the appropriate policies can be operated/updated accompany to tenant changes.

The cloud service provider consists of two modules: the real service module which provides tenants various types of services such as e-commerce, and the security module which offers security check functionalities before providing services.

The overall architecture of the proposed security module is shown in Figure 71. To build up a new security model for access control of an application in cloud, instead of from scratch, one can search for the ontology database according to its specific application domain, e.g. IT company, business company, academic university.

"*Reference Ontology*" module provides a list of candidate ontology template from the existing ones to the tenants. If the tenant already has his own role architecture design, the tenant may import the his ontology template into the system, and use "*Ontology Comparator*" model to compare the similarity of its own ontology with other candidates. He can refer to the most similar one with highest score, and reuse the policy template from the existing ones. On the other hand, if the tenant is totally new, without his own ontology well defined, a default ontology template, as well as policy template in his specific domain is provided for reference.

When a tenant is trying to access to a protected service/data, the *Context Collector* module collects various contextual information from both the environment

and tenants. The *Role Evaluator* module uses context information quantify these values and interact with role databases and policy database to determine the security level. According to the security level, role, and access policy, the *Policy Controller* determines the appropriate security services, includes granting, denying or revoking access. And then, the result of this security service can be delivered to the service model, and perform actions according to this security checking process.

8.4. Using Ontology for RBAC

8.4.1. Define Roles with Semantic Information

The first question is how to define roles given a specific domain. Ontology[99], a conceptual structure which contains knowledge in a domain and their relationships, provides useful and valuable information for cloud computing. It specifies a conceptualization of a domain in terms of concepts and their relationships, which is used to generate a commonly agreed vocabulary for information exchange without ambiguity.

Consider an IT company in cloud, in which access control is critical issue in the customers data. In general, according to the semantic in a specific domain, roles are defined as a combination of the official positions, job functions, and etc. Typical official positions could be that of the ordinary member, group manager, regional manager and etc. Functions represent the user's daily duties such as being a developer, testing engineer and etc. Additionally the organizational unit to which a user belongs is used as an access control criterion for certain applications. All these data are defined and maintained in the human resources (HR) database as shown in Figure 77. Thus, a RBAC system has an accurate image of the current

RoleID	Description
1	Chairman (Chief Software Architect)
2	SVP & CTO (strategy and policy)
3	CEO
4	COO(Chief Operation Officer)
5	CFO(business division)
6	VP (global health strategy)
7	...

Table 13

Sample Role Definition in IT company

organizational status and existing roles. Each employee can be assigned to one or more roles. A sample role definition is shown in Table 13, e.g, CEO has a unique roleID = 3.

8.4.2. Manage Roles Hierarchy with Ontology

In a domain, multiple possible role hierarchies are defined by ontology systems from different communities[48]. Examples of possible role hierarchies in an IT company are shown in Figure 72.

Existing research have discussed how to define an ontology in a specific domains, compare and integrate ontology systems between different communities.

In practices, for a given domain, multiple reference ontology systems from various communities may in that domain. For example, ACM and IEEE are two large communities and each has its own standards and practices, and they are similar but still distinct, and thus if the corresponding ontology systems will be similar but distinct. Such heterogeneity is common. Ontology integration [109, 110] is developed to solve these heterogeneities, which refers to build a larger and complete ontology at a higher level using existing ontologies.

As a concept structure of domain knowledge, ontology is usually represented as a tree. A formal definition of ontology tree will be discussed in Section 8.4.3. Comparing the concepts of two nodes in the tree, can be easily estimated by domain experts. For instance, "people" and "human being" are referring to the same meaning with a similarity degree of 1. "faculty" and "professor" are very similar in university domain, with similarity degree 0.95, which means around 95% occasions these two are describing the same concept. Some research have bee done in determining conceptual similarity in a knowledge context[162, 58].

Previous researchers use editing cost from one tree to another to measure two trees [42, 4, 55], which focus on the structural and geometrical features of trees, considering the number of nodes affected when editing the trees.

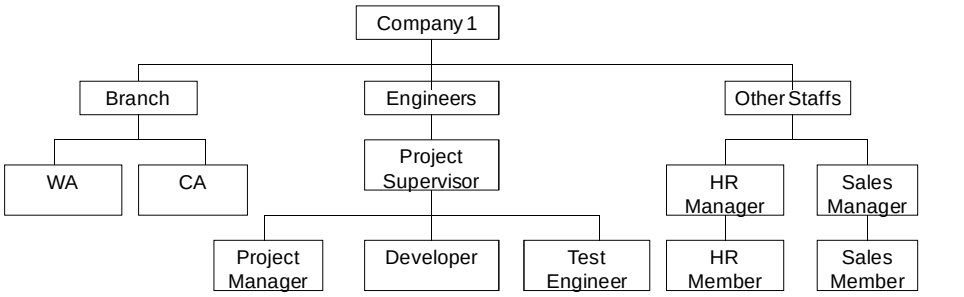
In the following discussion, a formal definition of ontology tree, similarity of ontology tress, and transformation operation between ontologies tree will be discussed.

8.4.3. Solve Role Hierarchy using Ontology Trees

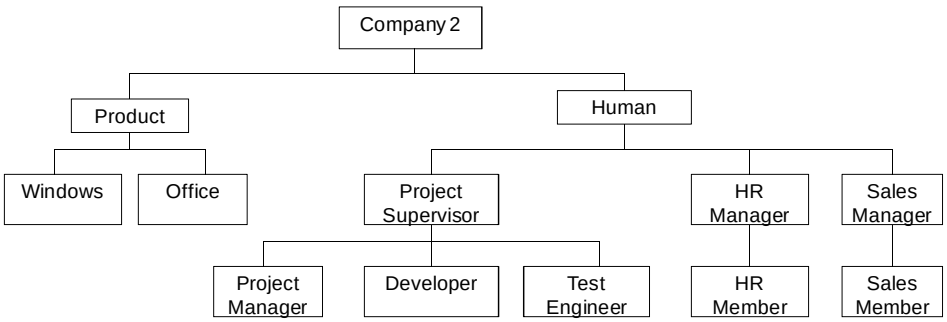
Role hierarchies impose restrictions which can generate a simpler tree structure (i.e., a role may have one or more immediate ascendants, but is restricted to a single immediate descendant). To extend the traditional definition of trees for an ontology in a specific domain, formally one can define the ontology tree as below:

Definition 8.4.1 *Ontology Tree(OT).*

An unordered and labeled Ontology Tree is a tuple $OT = (V, E)$ *where* V *is a finite set of nodes,* E *is a set of edges where* $E \subset V \times V$ *represents relationship*



(a) Ontology tree T1



(b) Ontology tree T2

Fig. 72. Sample Ontology Trees Companies

between nodes. If $(u, v) \in E$*, u is the parent of v, denoted as* $u = parent(v)$ *and v is the child of u, as* $v = child(u)$*. The ancestor and descendent relationship can be defined similarly. Any node in* V *except root node, has one and only one unique parent node.*

In addition, several auxiliary notations are used, include L^V, a set of labels for nodes, M is the injective mapping from V to label set L^V, $M : V \to L^V$, each node v_i has a unique label L^{v_i}.

Definition 8.4.2 *Similarity of OTs (*$Sim(OT_1, OT_2)$*).*

$Sim(OT_1, OT_2)$ *is a real number, defined as* $Sim(OT_1, OT_2) : L^{V_1} \times L^{V_2} \to R \in (0, 1]$.

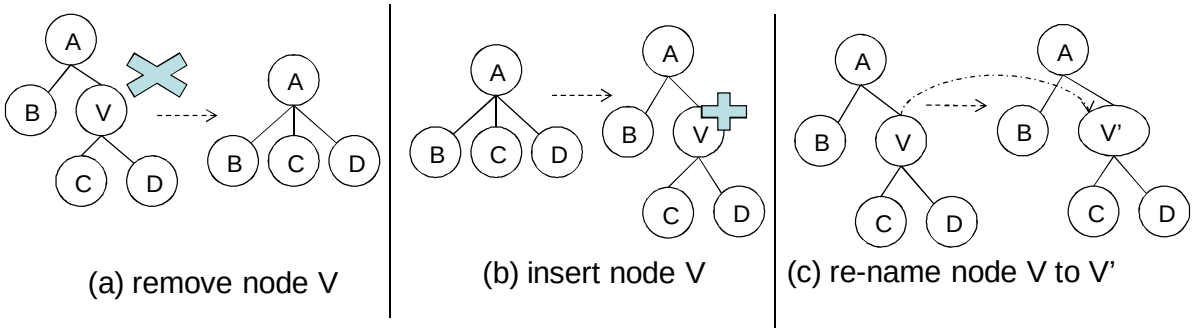


Fig. 73. Ontology Tree Transformation Operations

As one can see, the larger the $Sim(OT_1, OT_2)$ value, the closer the two ontology trees and $Sim(OT_1, OT_2) = 1$ means the trees are identical.

Ontology Tree Transformation Operations

To transform one tree to another tree or compare the similarity of two ontology trees, three basic operations are needed [17] as shown in Figure 73. All other more complex operations can be treated as compositions of these three operations.

1. Delete node v: eliminate the node from the tree, make its children nodes be direct children of parent(v).

2. Insert node v: add a new node, some of child(u)(represented as $CS'(u)$) become child(v), which can be decided according to context information.

3. Rename v: relabel v with a new name v', the tree structure does not change.

Formally, the tree transformation operations map tree OT to OT'. Suppose the children set of v is $CS(v)$.

- Delete node v: $OT' = (V', E')$ where $V' = V - \{v\}, E' = E - \{(u, v)|u = parent(v) - \{(v, v_c)|v_c \in CS(v)\}\} + \{(u, v_c)|u = parent(v) \wedge v_c \in CS(v)\}$.

- Insert node v: $OT' = (V', E')$ where $V' = V + \{v\}, E' = E + \{u, v\} + \{(v, u_c)|u_c \in CS'(u)\} - \{(u, u_c)|u_c \in CS'(u)\}, CS(u)' \subseteq CS(u)$

- Rename v: $L^{V'} = L^V + l'_v - l_v$, where l_v is the old label of v, and l'_v is the new label.

[17] defines cost of each transformation operations, this chapter uses consistent notations and definitions as follows: suppose the labels in OT are chosen from a finite alphabet Σ. Let $\lambda \notin \Sigma$, which is a special blank symbol and $\Sigma_\lambda = \Sigma \cup \lambda$. Cost function γ is defined as $\gamma : (\Sigma_\lambda \times \Sigma_\lambda) \setminus (\lambda, \lambda) \to R$. For $l_1, l_2, l_3 \in \Sigma_\lambda$, the following conditions are satisfied: (1) $\gamma(l_1, l_2) \geq 0, \gamma(l_1, l_1) = 0$ (2) $\gamma(l_1, l_2) = \gamma(l_2, l_1)$ (3) $\gamma(l_1, l_3) \leq \gamma(l_1, l_2) + \gamma(l_2, l_3)$.

The cost of a sequence $S = s_1, ...s_k$ of operations is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. Hence, the edit distance between OT_1 and OT_2 is formally defined as

$$\delta(OT_1, OT_2) = min\{\gamma(S)\} \tag{8.1}$$

where S is a sequence of edit operations transforming OT_1 to OT_2.

Indeed, the transformation of two OTs can be treated as an "ordered edit distance problem" which was introduced by Tai[147], further improved by others[169]. Klein [75] used dynamic programming to solve the problem.

8.4.4. Role Numbers and Scalability

In general, the total number of possible roles is the product of every category dimension, for example, in an IT company, one needs to consider official positions, and their job functions. However, the actual number of roles is a subset of these combinations, because some roles will not be needed.

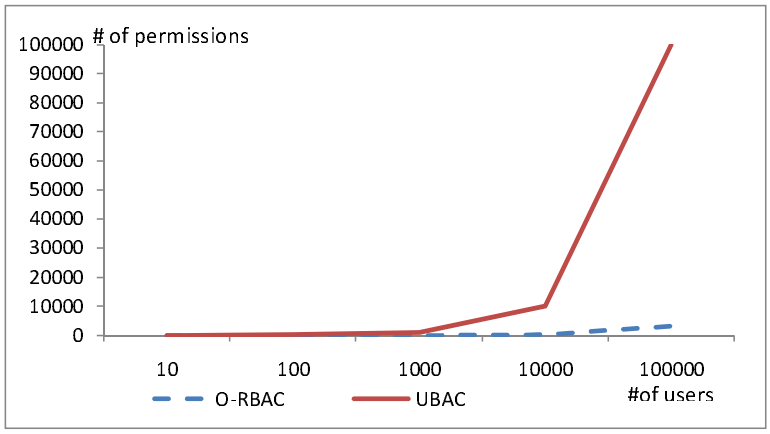


Fig. 74. # of Permissions using O-RBAC vs. UBAC

For example, as reported from a Enropean bank[127], there are around 65 positions that can range from an ordinary clerk in a branch, though the department manager, to the super manager, in a partial order of hierarchy structure. And there are around 368 different job functions provided by HR database, the total possible role set 23920(= 65 * 368), but the actual number of roles is around 1300, which is much smaller than expectation. Also, RBAC2000 Workshop[117] suggests that the number of roles in a role-based system is approximate 3 - 4% of the user population.

To learn the benefit of O-RBAC, this chapter simulates the scale change of role based model using a random number between 3 - 4% and compare the total number of permissions using O-RBAC with UBAC, the result is as shown in Figure 74. To simplify the comparison, assume each user/role needs a single permission. As one can see, with the increase of number of users, O-RBAC (in dashed line) can reduce the number of permissions operations greatly, comparing with user based access control. In another words, using O-RBAC, one can make the access control process in cloud much easier and efficient than using UBAC.

RoleID	Application	Access Right
...	...	...
100	Market Instruments	{1,2,3,4}
101	Customer Instruments	{1,2,3,7,10}
102	Employee Instruments	{1,4,8,12}
...	...	

Table 14

Roles, Applications and Access Rights

8.5. Policy Specification and Management

As another improvement of traditional RBAC, the proposed framework uses policies, as an extension of permissions in the access control. In fact, user membership is inherited top-down, and role permissions are inherited bottom-up. Both of them have a co-related ontology architecture behind.

Policies are derived from business goals and service level agreements(SLA) in enterprises, which are "rules governing the choices in behavior of a system" [135]. Policies includes obligation policies(event triggered condition-action rules), authorization policies(define what services or resources a subject can access) and etc.

The *Authorization Policies (AP)* define a tenant's rights which give him permissions to perform certain actions. In general, an authorization policy could be positive(permitting) or negative(not permitting, prohibiting). These policies are defined together with roles. The policy enforcer uses there properties to decide whether access are allowed or denied. For example, Table 14 shows that one with roleID= 100 has as many ore more access rights in the market instruments and etc.

The *Obligation Policy (OP)* defines a tenant's responsibilities, what activities a subject he must(or must not) do. In general, obligation policies are event-

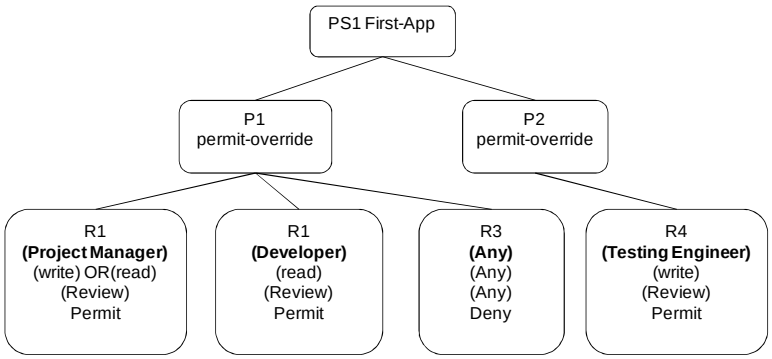


Fig. 75. An Example Policy in XACML Tree Presentation

condition-action rules (ECA), in which obligate a tenant to perform action A is triggering event E occurs and condition C is satisfied. The OASIS standard XACML[52] is an expressive, general purpose XML-based language (with significant deployment1) that is used to specify policies on web resources. XACML enables the use of arbitrary attributes in policies, allows for expressing negative authorization, conflict resolution algorithms and enables the use of hierarchical Role Based Access Control, among other things. Figure 75 shows a sample of policy using XACML presented in graphical form. In this example, there are three security roles, Project Manager, Developer and Testing Engineer; one resource: Review; and two actions: read, write.

The proposed reference framework support sophisticated authorization policy specification and management, which is particularly useful for cloud computing applications. Multiple policies may apply to the proposed O-RBAC model, including authorization policies, obligation policies and etc.

8.6. Answering Requests using O-RBAC

A O-RBAC system has two phases in assigning a privilege to a user: in the first phase, the user is assigned one ore more roles, and in the second phase, the roles are checked against the requested operation. When a user starts an application the O-RBAC delivers the security profile that tells the application which individual access rights the user possesses.

For example, in Figure 76, an existing mortgage loan applicant wants to discuss his personal loan application situation with the branch's financial advisor. The advisor and the client sit in the same office in the mortgage company with a personal computer. The advisor identifies and authenticates himself to the machine using an employeeID and his password. He launches an application that allows him to enter the records of his client which are stored on a central server in cloud.

When the application is launched it issues a request to the host, querying which rights the advisor has within the application domain. The application request contains the personnel number, which was obtained during the identification and authentication process. Also the application identifier is submitted to obtain the relevant authorization profile for the application.

Once the O-RBAC has used these data to deliver the security profile, the application knows which access rights are assigned to the role of the user and allows him to execute his access rights accordingly. In this particular case information about the relevant organizational unit to which the advisor belongs will prohibit him from accessing account data outside his branch. His access rights are confined within the organizational domain of the branch. However, other applications can be used from

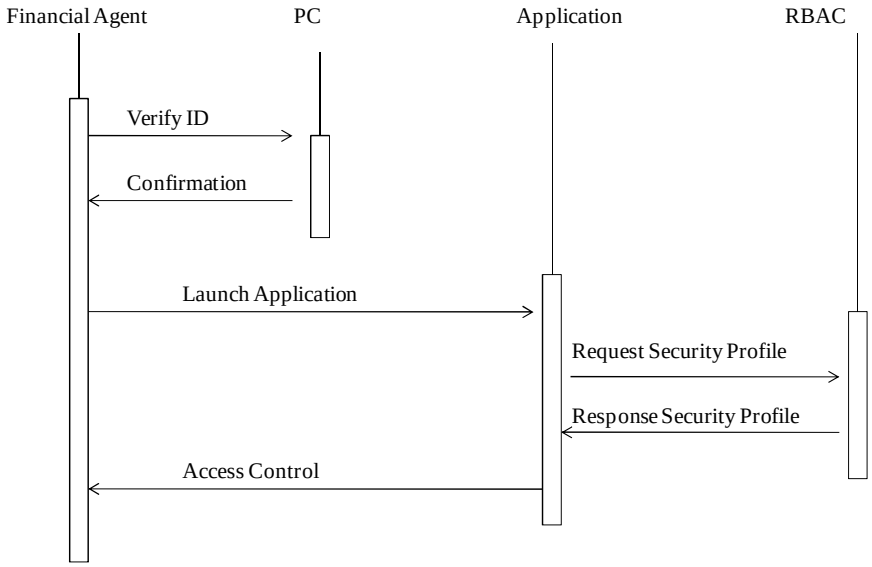


Fig. 76. Process of Service Requests

access points all over the mortgage company, as the access rights which are granted for them do not depend on any local information.

The proposed reference ontology framework works together with business service modules to support secure access control in cloud.

8.7. Related Work

8.7.1. Research Community on Security

Database Security The multi-tenancy architecture have increased the risk exposure of databases and, thus, data protection is today more crucial than ever. Three types of data security problems: *confidentiality* (protect unauthorized data observation) using encryption techniques [66]; *integrity* (prevent incorrect data modification) using semantic correctness; *availability* (recover from malicious attack) using recovery and concurrency control mechanism.

There are mainly two different views of access control models: role based access control, and content based fine-grained access control model.

1)Role Based Access Control (RBAC) models represent arguably the most important recent innovation in access control models. RBAC has been motivated by the need to simplify authorization administration and to directly represent access control policies of organizations.

RBAC models have been widely investigated. A standard has been developed [124] as well as an XML-based encoding of RBAC [16]. Relevant extensions of RBAC include: administration models development [36, 74, 76]; temporal constraints of TRBAC model [14, 70]; and security analysis techniques [86]. RBAC models are also supported by commercial DBMSs [116]. However, commercial implementations provided as part of DBMSs are very limited and only support a simple version of RBAC, referred to as flat RBAC, that does not include role hierarchies or constraints. RBAC is also used in Web-service architectures, such as the Permis system [24].

2)Content-based and Fine-Grained Access Control requires access control decision based on data contents. To support content-based access control, views is organized into prtection views and sharehand views [15], and filtering based label are used in access control. Fine-grained mechanisms are able to support access control from object to tuple level [100]. The object-level security is provided by profiles which is represented by metadata to group users with common data access requirements. It contains a set of permissions for every object defined in the system. These permissions determine the privilege of access control for each object. Record-level security is layered on top of object-level security, which restricts access to data based on record ownership.

Access Control Policy There has been a great amount of attention to access control policy languages for web services which accommodate large, open, distributed and heterogeneous environments like the Web. Policy languages, such as WS-Policy [129] (a W3C submission), which specifies the constraints and capabilities of web services, and the more general eXtensible Access Control Markup Language (XACML)[52].

Many extension of XACML are investigated by research community. Zhao et al [171] present a formalization of RBAC based on the description logic ALCQ. Massacci [90] formalizes RBAC using multi modal logic and presents a decision method based on analytic tableaux. Hughes et al. [62] propose a framework for automated verification of access control policies based on relational First-Order Logic.

Network Security Both authentication and encryption techniques are widely discussed in the current literature on computer network security and we refer the reader to [73] for details on such topics. We will, however, discuss the use of encryption techniques in the context of secure outsourcing of data, as this is an application of cryptography which is specific to database management.

8.7.2. Current Industry Security Support in Cloud

Saleforce.com Force.com provides a multilayered approach to data security[103], from object-level to record-level. It can support four level of filters: (1) Object Permissions, which ensures that the requesting user is authorized by its profile to take the desired action on this object.(2) Field Accessibility, in which he requesting user's profile is consulted again to determine whether there are fields included in the request that are read-only or hidden.(3) Sharing Model, which evaluates whether the

user is not the owner of this record or otherwise privileged with an administrative profile, organization-wide defaults are applied. (4)Sharing Reasons, which overrides the organization-wide defaults. The owner of the requested record is matched against a list of sharing reasons relevant to its group affiliation. If a sharing reason is found, access is granted. Groups are defined as simple lists of users and other groups or as a hierarchy, allowing permissions of subordinates to be inherited by their superiors.

IBM DB2 V9 uses two approaches to realize data security, filter-based approach at the application level (based on tenant ID) and permission-based approach at the DBMS level (based on dedicated DB access account) or row-level access control, e.g. Label-Based Access Control (LBAC). The advantage of LBAC is that it controls cross-tenant data access at the DBMS level instead of the application level. However, the maximum number of tenants is limited and could not support multi-tenancy requirement.

Amazon EC2 Security within Amazon EC2 is provided on multiple levels: The operating system (OS) of the host system, the virtual instance operating system or guest OS, a stateful firewall and signed API calls. Each of these items builds on the capabilities of the others. The goal is to ensure that data contained within Amazon EC2 cannot be intercepted by non-authorized systems or users and that Amazon EC2 instances themselves are as secure as possible without sacrificing the flexibility in configuration that customers demand.

Amazon SimpleDB Security APIs provide domain-level controls that only permit authenticated access by domain creator, therefore the customer maintains full

control over who has access to their data. SimpleDB access can be granted based on an AWS Account ID. Once authenticated, a subscriber has full access to all user operations in the system. Access to each individual domain is controlled by an independent Access Control List (ACL) that maps authenticated users to the domains they own.

As one can see, no matter what techniques applied in those cloud providers, either RBAC model or UBAC, none of them can provide a reference model to end users. The proposed framework can benefit users with recommendation role and policy template given an application domain and greatly shorten the design time.

8.8. Conclusion

A reference ontology role-based access model is proposed in this chapter. To design a security mechanism in a multi-tenancy architecture, instead of starting from scratch, a reference role-based access control template is provided. Specially, Ontology information are used as heuristic to build up the role hierarchy. An effective ontology transformation operations algorithms are provided to compare the similarity of different ontology. The impact of role based access control model in real application, a case study of IT company is provided.

As future works, a new back-end database schema to support role-based access control will be investigated. Also it would be interesting to analyze the role/user ratio according to the position hierarchy, to measure the scalability of the O-RBAC, with respect to number of roles, number of permissions, size of role hierarchy, limits on tenant-role assignments, and etc.

9. EASYSAAS: A NEW SAAS ARCHITECTURE

Software as a Service (SaaS) with multi-tenancy architecture is a popular approach. To support a significant number of tenants, an easy to use SaaS construction framework is highly desirable. This paper introduces an easy SaaS constructing architecture: an automatic SaaS construction framework. In this architecture, in stead of starting from scratch and customize applications, the tenant can publishing their requirements into the global SaaS platform in the form of application requirement and specification with their unique business requirements, as well as their expectation of the SaaS outcome and test scripts. The SaaS providers proposes their SaaS products, customize their services to meet tenant's requirements. This framework releases the workload of tenants, and provide an easier way to customize tenants' business requirement in a collaborative way. The SaaS providers also get benefits from the shared platform, and fast the development process. A hierarchy global index is used to support the matching and customization process.

9.1. Introduction

Cloud computing has received significant attention recently as it is a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and visualized manner. Public clouds are available from Amazon, Google, Amazon, Microsoft, Salesforce.com and others. Private cloud technologies, in which the cloud software is loaded locally are available from VMware, Eucalyptus, Citrix, and there are thousands of vendors offering "cloud solutions".

SaaS (Software as a Service), that is often deployed on a cloud, is a new way to deliver software. In SaaS, software is maintained and updated on a cloud, and

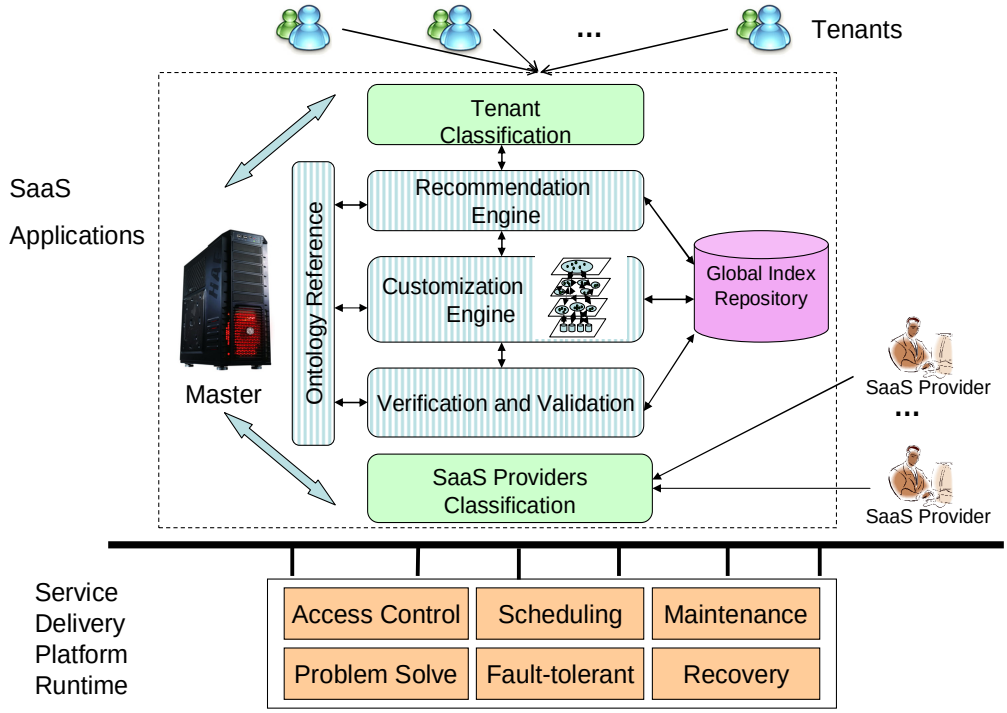


Fig. 77. System Architecture of EasySaaS

presented to the end users as services on demand, usually in a browser. With SaaS, a software provider licenses an application to customer as a service on demand, through a subscription or a "pay-as-you-go" model. Saas also involves difficult design issues such as customization, multi-tenancy architecture, and scalability, and these three features are represented in the three maturity levels for SaaS proposed in [5].

To generate a SaaS application, different providers try to easy the life of customers(tenants) by providing templates, such as Salesforce.com [46] proposed a set of objects, such as customer, and account. Customers can reuse and customize the objects according to their own business requirements. However, the workload of customization and SaaS construction is still on the customer side.

This paper proposes a novel SaaS constructing framework, in which can leverage customers in a collaborative way. Different from the conventional SaaS framework as presented in [46], in which SaaS providers develop and publish their services, and consumers are responsible for searching and designing the desirable services as well as for customize their own applications.

In EasySaaS system, customers can publish their application requirement together with associated service specifications including the workflow and cooperation requirement. Once such structures and templates are published, any SaaS providers can submit their software or services to meet the application requirements. This way of computing is in a *consumer-centric* way, since SaaS providers will actively search for customers' requirements and needs. In stead of passively waiting for requests, SaaS providers are compete in an active way.

There are several challenges in EasySaaS, for example, how to provide a global platform for SaaS providers and tenants to communicate? How to make recommendation according to tenant's specific requirement? How to use the profiling of tenant and SaaS providers to easy the constructing process? How to search for the desirable workflows/servuces/data schemas in the customization process. This paper will propose solutions for these questions in turn.

In summary:

- This paper proposes an automatic SaaS construction framework. In stead of starting from scratch and customize applications, the tenants can publishing their requirements into the global SaaS platform in the form of application

requirement and specification with their unique business requirements, as well as their expectation of the SaaS outcome and test scripts. The SaaS providers proposes their SaaS products, customize their services to meet tenant's requirements. This framework releases the workload of tenants, and provide an easier way to customize tenants' business requirement in a collaborative way.

- This paper designs a hierarchy global index which is used to support the matching and customization process. By searching the global index, one can find out the desirable workflow/services/database design in a cost-efficient way.

- This paper provides intelligent support for constructing process, including profiling mining of tenants/SaaS providers, clustering services, as well as service classification. With these intelligent support, EasySaaS can leverage the constructing process.

- This paper uses customization and recommendation engine to support the SaaS construction, a four layer customization model with keyword search engine of semantic dewey index is provided to fast the query processing.

The paper is organized as the following. Section 9.2 provides an overview of EasySaaS; Section 9.3 discusses its key components in the framework; Section 9.4 investigate the core algorithms in EasySaaS; Section 9.6 discuss the related work; and Section 9.7 concludes this paper.

9.2. EasySaaS Overview

EasySaaS provides a framework for collaboration-oriented service specification, discovery, matching, verification, validation, and composition. The EasySaaS architec-

ture is shown in Figure 77.

A EasySaaS 's global index stores not only service specifications, but also application templates and collaboration patterns. Once a tenant publishes an application template to the platform, the SaaS providers will be informed and they can develop SaaS for the new application. Once a new SaaS is developed for a published template, the tenant will be informed to test and evaluate the newly available SaaS. The detail steps are listed below:

1. A tenant(consumer) who needs a SaaS application, first develops an application template, which includes information on customization information includes GUI specification, workflow specification, service specification, database design, service acceptance criteria, and application acceptance criteria.

2. The application template is published and stored to the global index database.

3. SaaS providers subscribed to the application registry are informed the new requirement from customer's new templates.

4. The ontology are used by the customization engine, which automatic matches between the requested and registered application templates.

5. The SaaS provider develops a SaaS application according to the application template and submits it to the platform administrator. Each service submitted will be evaluated by validation and verification module, to meet the service acceptance criteria.

6. If a service passes the evaluation, the SaaS providers will be notified.

7. Using the binding information from the coordinator, the application builder (customer) test and evaluate the service.

8. If the services pass the application acceptance testing, the application builder(customer) will bind the service into the target application.

9. If all required services are available, the application building is completed.

10. At both the SaaS provider and customer sides, a classification module is used to easy the search and customization process. Both of them can be classified by their profiling, e.g. mortgage domain, health care domains, and etc.

9.3. Key Components in EasySaaS

9.3.1. Tenants' Requirements

Requirement from each tenant is one of the most important issues in EasySaaS in which it defines customer's requirement precisely. EasySaaS , several aspects are considered, each aspect a certain functionality of the SaaS application. For example, a mortgage SaaS, EasySaaS will consider both the applicants and lenders: for the applicants, the submission loan application process dataflow, for the lender, the approvel/denial workflow are considered.

Dividing applications into aspects have the following benefits: first it simplify the the specification process in a divide and conquer way; secondly, given a service one can focus on a specific functionality and avid the ambiguity caused by other services; thirdly, different aspects may have different impact in the application, and can be weighted. The more important ones will be arranged a higher and more

strict in the verification and validation module, while less important one can be loose managed.

9.3.2. Service Specification

In EasySaaS , current service specification can be further extended as follows:

- Use scenarios: Collaboration among tenants can be described using service use scenario Under EasySaaS architecture, a use scenario for a service specifies how a service could be used by other applications and the requirements when using the service.

- Workflow specification : OWL-S and PSML-S can be used to integrate modeling and specification language supporting a set of analyze capabilities such as C&C analysis, model checking, simulation, and etc.

- Service property specification: Certain service properties, such as the service classification information, service provider information with its "service history", and constraints that the application requires for the service. All these properties together with the other parts of the service specification mentioned above contribute to the final service collaboration and discovery/matching process.

9.3.3. Intelligent Clustering, Classification and Profiling Mining

SaaS is usually composed of a large number of services, data mining can be applied to two sets of data and assist the process of SaaS construction. Two types of data can be used: service metadata in the service registry, and service access logs with

pre-gathered tenants' profiles. Hence two types of mining algorithms can be applied, service mining (searching for appropriate services based on metadata) and service usage mining (searching for associations, usage pattern). By utilizing knowledge mined from those patterns, intelligent SaaS providers can generate desirable services more quickly with less effort.

9.3.3.1. ***Profiling Mining***

In addition to the application aspect workflow description, one also need the application classification information in order to classify the application in an ordered way. Profiling of each requirement can be analyzed by different data mining methods.

Application requirement content contains descriptions of the SaaS application requirement, such as what is the functionality of the SaaS, which area it will be applied to, etc. Predicting the SaaS application category and reuse, customize existing applications, given only its profile, can be treated as a classification problem, where the profile content can be parsed with well-known algorithms (e.g. support vector machine (SVM)[133], k-nearest neighbor[71]) to create a feature vector, which in turn can be mapped to an application domain the class label.

Furthermore, to capture the semantic similarity between requirement contents, one can use the Vector Space Model (VSM), a commonly used method in information retrieval [123]. With VSM, the content of each specification is represented by a vector in a multi-dimensional space, where each dimension corresponds to a natural language unit, such as a word or a phrase. In this study, unigram model is used in which each unique word is a dimension.

As a first step, specification contents are preprocessed with, for example, stop-

word removing, word stemming [1], etc. After preprocessing, one can use the bag-of-words approach to convert each specification to a vector. Formally, assume there are $|\mathcal{D}|$ specifications in the training data set, and V represents the specification content vocabulary. For each specification content τ_i in $\mathcal{D}$, one can have a $|V|$-dimensional vector $\overrightarrow{\tau}_i = \langle v_{i1}, ..., v_{i|V|} \rangle$, where

$$v_{ij} = \log(c(\tau_i, t_j) + 1) \log(\frac{|\mathcal{T}| + 1}{df_j}).$$

Here, $c(\tau_i, t_j)$ is the frequency of term t_j occurring in content τ_i; df_j is the number of specifications in the training data that contain term t_j.

Given two vectors derived from contents, τ_i and τ_j, one can compute their cosine similarity as:

$$s_{ij} = cos(\tau_i, \tau_j) = \frac{\overrightarrow{\tau}_i \cdot \overrightarrow{\tau}_j}{||\overrightarrow{\tau}_i|| \cdot ||\overrightarrow{\tau}_j||}. \tag{9.1}$$

It is obvious that $0 \leq s_{ij} \leq 1$, where $s_{ij} = 1$ indicates that the two specifications are about the same requirement, while $s_{ij} = 0$ means they have no keywords in common.

9.3.3.2. ***Service Classification***

Classification first categorizes previously unseen data, based on a model built with the existing data set. A model usually contains a function of several important attributes and can be built by classification techniques such as decision tree[168] and Bayesian Networks [107] using training set. Using classification techniques, one can classify services based on their contents(service category, service description) and metadata.

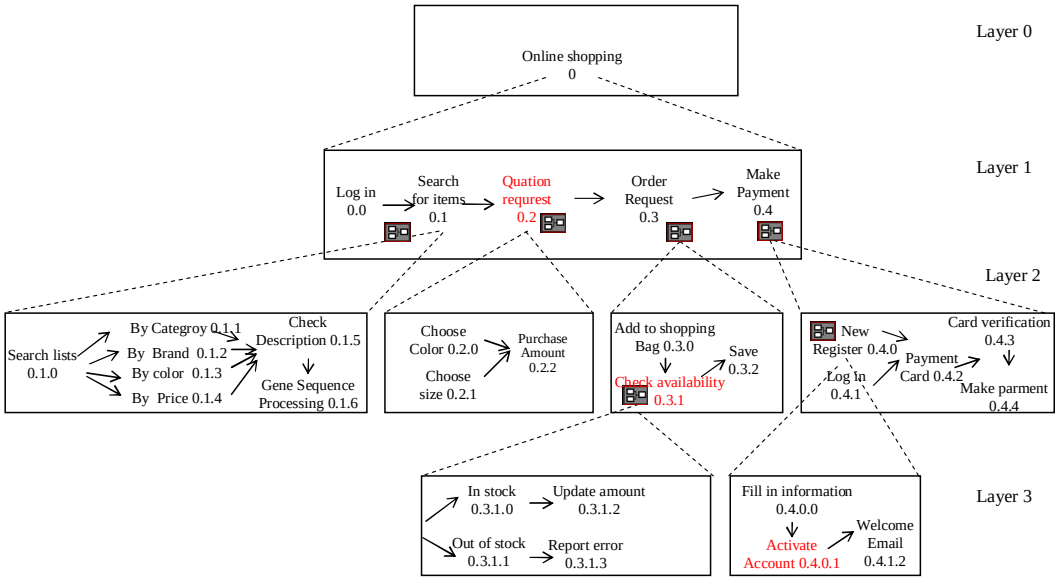


Fig. 78. Sample Online Shopping Application with Hierarchy Workflow Structures

9.3.3.3. ***Service Dependency and Clustering***

EasySaaS has a 1:N mapping between application and SaaS implementation. Similarly a service implementation may subscribe to multiple applications, hence EasySaaS infrastructure can maintain this M:N mapping mode.

Clustering algorithms can be used in linking relationship between services. Given service is chosen, one can find out its closely related services by examining possible inclusions in the new applications. Furthermore, one can form a "service pool" in the platform, and set up a "service cluster master" to integrate services in the cluster. The most important part in clustering algorithms is the distance calculation. Two common calculations are the Manhattan distance[169] and the Euclidean distance[75]. The Manhattan Distance (MD) is the sum of the differences of their corresponding components, e.g. MD between a point $X = (X_1, X_2, ...X_n)$ and a point $Y = (Y_1, Y_2, ..., Y_n)$ is $MD = \sum_{i=1}^{n} |x_i - y_i|$. The Euclidean dis-

tance(ED) function measures the 'as-the-crow-flies' distance, e.g. ED between a point $X = (X_1, X_2, ...X_n)$ and a point $Y = (Y_1, Y_2, ..., Y_n)$ is $ED = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

Clustering techniques can be utilized in EasySaaS to identify similar services and similar service consumers(tenants).

9.3.4. Recommendation Engine: Discovery and Matching

In addition to the service discovery and matching in traditional SOA[157], EasySaaS has discovery and matching also for application templates and/ collaboration patterns. The discovery/matching process has a four-layer architecture: interface level (GUI), process level(workflow), service level, and data level.

9.3.4.1. *GUI Discovery*

Interface level discovery is based on the service interface specification, which is similar to WSDL approach. It is required to have an exact match between the interfaces of the service implementation and the service specification in the application template. Indeed this ensures the essential syntax validity for the application integration.

9.3.4.2. *Workflow and Service Discovery*

Workflow level discovery and matching is based on service process specification. Workflow is composed of services and has a hierarchy structure, named as workflow hierarchy, which provide multi-resolution views of workflows in order to ease the analysis and maintenance of workflows. Figure 78 is a sample online shopping service workflow for illustration. As one can see, a workflow hierarchy contains nested service graphs in multiple layers, thus forms a three dimensional structure. For instance, in layer 1 of the online shopping process, after log in to the system, the

customers can search for items, then quote their request, make an order and at last check out with payment. A service in the hierarchy can be atomic or composite. A composite task can be zoomed-in to reveal the detailed procedure of how to perform it, represented by a directed graph consisting of a set of services and their dataflows shown in the layer immediately beneath. Dotted lines, connecting a composite task to its expansion, are referred as expansion edges. A composite service "search for items(0.1)" is zoomed into a graph as pictured in the leftmost box in layer 2 which consists of seven services. Each node n in the workflow hierarchy is assigned a unique label $NID(n)$. To explore the ancestor-descendant relationships of nodes to generate results, one can use the Dewey labeling scheme, as shown underneath each node in Table 78 for the sample application.

One can use keyword search to find out the desirable workflow or services, e.g. one would issue a query to the service repository using "quotion request, check availability, activate account" Workflow matching is done when the application template places a process requirement on the service. Thus, a process level requirement is placed on the candidate services: only the services whose process specification match the application's requirement will be selected.

9.3.5. Customization Engine

EasySaaS has a customization engine, as in [156], an ontology intelligent customization framework is proposed. which can customize application templates and collaboration patterns. The customization process has a five-layer architecture under EasySaaS: interface level, process level, service level, data level, as well as and collaboration level (both use scenario and collaboration specification matching).

With the customization mechanism in [156], tenants can express their requirements with the following features: easy to use (template objects are provided as default); layered architecture (from GUI, workflow, service and data layers); semantic oriented (using domain ontology to help the customization process); intelligent (recommendation supported by mining knowledge from tenants community and profiling); and adaptable (periodic maintain the framework to improve performance).

To meet all these requirements, OIC, a multi-layers ontology based intelligent customization framework. Four layers from GUI, business process, service and data layers, are modeled in a multi-layered architecture, and ontology information is used at each layer to guide the customization process. According to the multi-layered model, a framework is provided which can be applied in the development of SaaS. Based on this framework, the SaaS providers are supported in their decision. The proposed ontology-based layered framework for customization with the following features: (1) a multi-layer structure of SaaS applications, developers can analyze their inherent relationships, as well as cross-layer relationship using ontology information. (2) use template objects as default, and recommend candidate components at different possible layers (GUI, workflow, service and data) using collaborative filtering techniques to provide a cost effective way to customize tenants' specific individual requirements.

9.3.6. Verification and Validation

Before a SaaS provider can register a service into global index database to support an application template/collaboration pattern published by an application tenant, the service need to be verified and validated by the master against the service acceptance

Index	Service_Name	Service_Description
0	Online shopping	Generate an online shopping service
0.0	Log in	Take username and password to logo into system
0.1	Search for item	Give the item name, search for the item
0.1.0	Search lists	List all possible search options
0.1.1	By category	Search item by product category
0.1.2	By brand	Search item by product brand
…	…	…
0.2	Question Request	Pick up the desire sub categories, e.g. size, color ..
0.2.0	Choose color	Make selection of product color
…	…	….

Fig. 79. Sample Global Service Index Table for Figure 78

criteria.

The master has multiple Service Verification and Validation Agent (SVVA) that can verify and validate services. When one service provider requests to subscribe to an application template/collaboration pattern, the SVVA firstly retrieves the test cases provided by the application builder or other parties from the repository. Then, it performs unit testing on the service being registered. In addition to general unit testing for functional validation, the test agent may need to perform other property-specific testing according to different requirements from the application builder. The property-specific testing could include reliability testing, security testing, robust testing, and performance testing. Once unit testing and property-specific testing are completed, the test agent performs inter-operability testing to validate the collaboration capabilities of this service. Only if the subscribing service passes both phases of testing, it can register to the application template/collaboration pattern.

Whenever changes are made to the services, regression testing needs to be performed to validate the revised component services. When composing a new

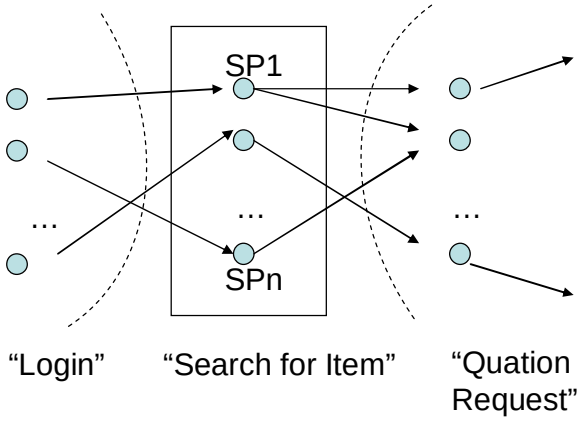


Fig. 80. Dependency Based Testing in EasySaaS

service, SOSE (Service-Oriented System Engineering) [149] tools can be applied for integration testing and evaluation in a distributed environment, and the V&V can be done on all the domains of the EasySaaS framework.

9.3.6.1. ***Dependency Based Testing***

When a new service joins EasySaaS, it first has to pass the verification embedded. There are multiple testing methods can be used in the process, e.g. unit testing, regression testing, mutation testing. One kind of valuable information here is the dependency between services, which can be applied to evaluate the new coming services, as shown in Figure 80.

Basically, there are three types of dependencies [97]: Input Dependency (ID), Input/Output Dependency (IOD), and Output Dependency (OD). Service Dependency Graph[80] is used to capture the service dependency using an AND/OR graph. Tsai. [26] further consider the dependency in a user-centric SOA.

An interesting way to validate a new coming service, is to plug it into the existing workflows to test its collaborative with other services. For example, as

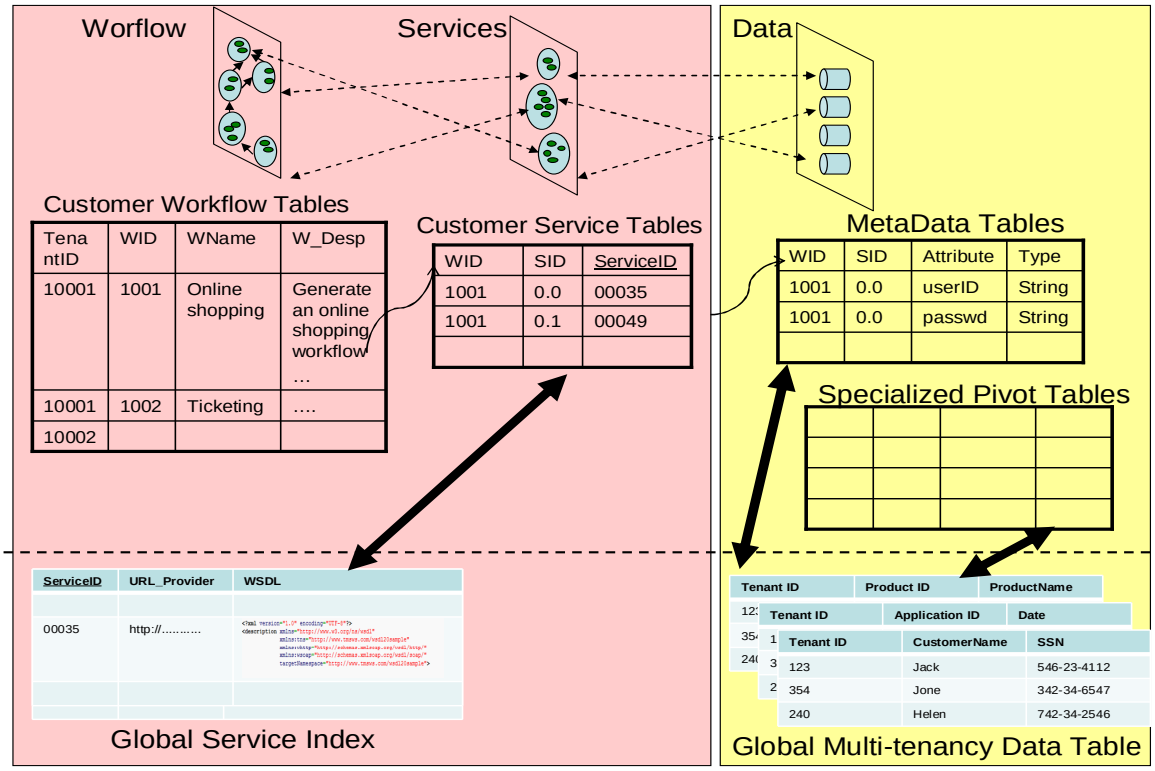


Fig. 81. EasySaaS Global Index Data Model Design

shown in Figure 80, support a new "search for item" service, named as X, enters EasySaaS, before allowing it join the global platform, X has to pass the validation first. Suppose X has the same interface as all other services in "search for item" service set S ($SP_1, ...SP_n$), it has to test with all the dependency of S. As long as X passes all tests, it can be proceeded to the next step.

9.4. Core Designs in EasySaaS

9.4.1. EasySaaS Global Model Design

Hierarchical Global Index (HGI)

To search and match services from different service provides in EasySaaS, global index is used to assist in the whole lifecycle. This paper proposes a three-level hierarchy analogous to that of a B+- tree to store global index at each layer (workflow

index, service index and data schema index.) as shown in Figure 81.

HGI is built on several other pieces of services at the infrastructure level. It uses the distributed storage services to store log and data files. A HGI can be operated in a shared pool of machines that run other types of applications. HGI also depends on other services, such as scheduling services to manage resources, recovery services dealing with machine failures, monitoring services for machine status, maintenance and continuous testing.

Figure 81 shows the basic components in HGI. The first level is workflows index that contains the information of the workflows. Each workflow tablet contains the location of a set of user workflows. The second level is service index, matching to the services layer in the OIC framework, which composes workflows. As one can see, each workflow in the first level has a composite graph of services, as shown in Figure 78, hence one workflow instance has multiple tuples in the service index tables. The third level is metadata schema index, matching to the data layer in the customization framework, which records both the workflow and services information, as well as the database schema design for a specific service.

The global cache is maintained. If the client does not know the location of a component(workflow, service, database schema), or if it discovers that cached information is incorrect, then it recursively moves up the index hierarchy. If the client's cache is empty, the searching algorithm requires three network round-trips. If the client's cache is stale, the searching algorithm could take up to six round-trips, because stale cache entries are only discovered upon misses. Although global index are stored in memory, also can be pre-fetched to cache to fast the query processing

time.

Global Service Sharing: A global service index(GSI) is maintained, all verified services are stored in the table, with its provider information and WSDL to uniquely identify a service from similar services.

Considering the customization process, when SaaS providers build up the workflow and decide the composed services, they can access the GSI to search for service components, they can simply issue queries in for GSI to get all candidate services, and pick up one according his specific requirement from end customers. Then SaaS providers can build up customizable services in a cost effective way. One service in GSI can be used by multiple SaaS providers in diverse customizable SaaS applications.

Specialized Pivot Table When SaaS providers create custom application (tables), metadata table in the third level concerning their objects, their fields, relationship and other characteristics. Meanwhile, specialized pivot tables can used to maintain de-normalized data that makes the combined data set extremely functional. For example, one can build up *UniqueFields* pivot table to capture the feature that an object containing unique values, which can fast the query processing, as well as secure the system, e.g. when an application attempts to insert a duplicate value into a field which marked as uniqueness, it would report errors. *Cross reference* tables can be user to provide "relationship" among object/attributes.

9.4.2. Analysis of EasySaaS Global Design

Multi-level Customization Support: as one can see in Figure 81, the proposed global design can effective support customization process cross workflow, services,

and data layer. The top blocks shows the semantic model cross layers, while the middle blocks shows the logic design of the tables, including customization workflow tables, customizer services tales, metadata tables, as well as data tables. The mapping from the semantic model and the logic model are explicitly demonstrated in the figure.

UCSOA Style Support in Service Evolution: In UCSOA [26] framework, it maintains the link between the solution specification and implementations. Since multiple implementations (workflows) can serve the same specification, the UCSOA infrastructure maintains this one-to-many links from the solution specification to multiple implementations. Similarly, a service implementation may contribute to multiple applications(workflow), and UCSOA also maintain this one-to-many links from a service implementation to applications.

The EasySaaS global design uses one-to-many links in a similar way as UCSOA, as shown in the left (pink) shadowed part with dashed circle . The customization workflows and services can use any services in the global services in the repository. In this case, one can easily find the cross reference relationships between workflows and services.

Multi-tenancy Database Design Support: In saleforce.com[103], a multi-tenancy database with metadata tables, data tables and pivot tables are supported, which is metadata-driven. Figure 82 contrasts a traditional database with a metadata-driven database. In traditional database design, objects and fields are defined that represent abstractions of the real-world entities that they represent. Separate database tables are created for each type of object represented. Specific attributes

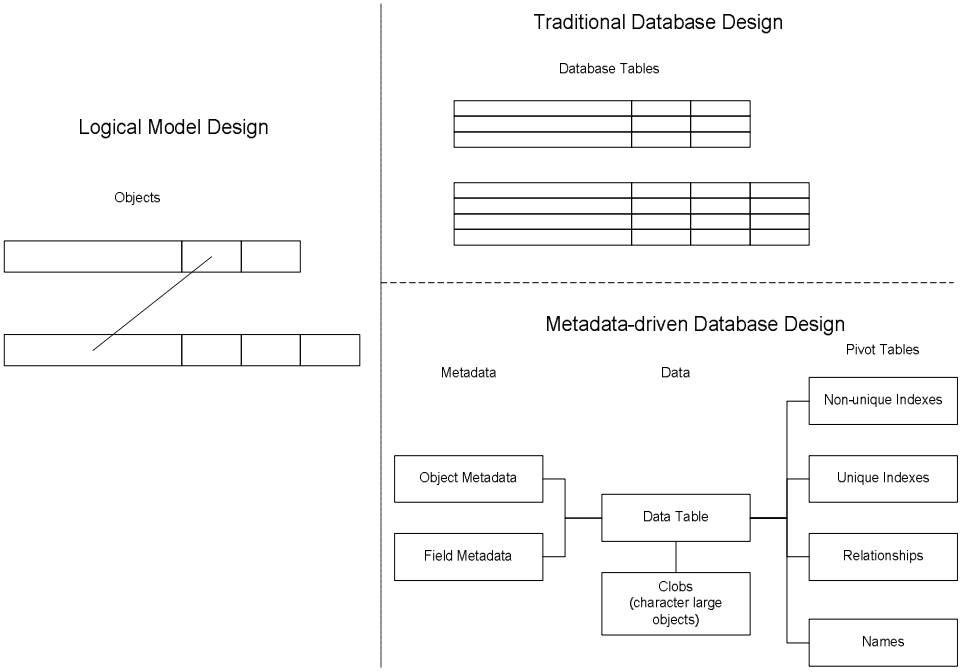


Fig. 82. Metadata-Driver Multi-tenancy Database Design

are represented by fields within the tables. Object instances are represented by rows within the tables. Actual data is placed into a database by inserting rows into the database tables. Relationships are represented by fields in one table referring to a key field in another table.

The EasySaaS global design can also support the multi-tenancy database design as shown in Figure 81 right (yellow) shadowed part with dashed circle.

Dynamic Code Generation with flexibility : The code generated from EasySaaS is in a dynamic way, in which workflows, services, and data are dynamic constructed in the customization process.

9.4.3. Chunk Partitioning

Database Partitioning can improve the system performance, scalability and availability of a large database system in a multi-tenant way. When using HGI, one

has to consider partition the tables into chunks, and distribute chunks into different nodes in cloud. For example, given a tenant's information, the query optimizer only has to access the partitions containing the tenant's data rather than the entire HGI table, using "partition pruning". Data partitioning is a proved technique that database systems provide to physically divide large logical data structures into smaller and easy manageable pieces(chunks). The data inside a database can be distributed across one or more partitions. A distribution key is the column used to determine the partition in which a particular row is stored. Instead of having one database server controlling the whole system, the database is logically partitioned and each of them can be controlled by a separate server. Indexes play an important role in improving overall performance together with partitioning. Different types of indexes are built to provide efficient query processing for different applications.

9.5. EasySaaS Hosting Platform Support

To support the SaaS application constructing, service delivery platform has to provide sufficient functionality, including access control, scheduling, maintenance, problem solving, fault-tolerant, and recovery, as shown in Figure 83.

Access Control: Security is an important issue due to the increase scale of users. Current approaches to access control on clouds do not scale well to multi-tenancy requirements because they are mostly based on individual user IDs at different granularity levels, however, the number of users can be enormous and causing significant overhead in managing security. RBAC (Role-Based Access Control) is attractive because the number of roles is significantly less, and users can be classified according to their roles. Tsai. [153] proposed a RBAC model using a role

ontology for Multi-Tenancy Architecture (MTA) in clouds. The ontology is used as to build up the role hierarchy for a specific domain. An ontology transformation operations algorithms are provided to compare the similarity of different ontology. The proposed framework can easy the design of security system in cloud and reduce the complexity of system design and implementation.

Scheduling: To do better load balancing among partitions to optimize the overall system performance, an effective algorithm is highly desirable, that can migrate, distribute and duplicate tenants among partitions through monitoring the load. Most cloud scheduling algorithms and database solutions address their problems independently. However, most of cloud components and functionalities are interconnected. Specifically, a task scheduling algorithm need to consider database partitioning to provide an efficient solution for performance and scalability. More specific, a task assigned to a processor should host the appropriate data partitions otherwise data updates and migration among caches and processors can be expensive. Tsai[154] proposed a crystalline mapping method which can prioritize service request in a fair way.

Problem Solving: services are working in a collaborative way, and when an error occurs, it is mainly caused by one service failure during the process. How to identify the root cause of the service failure calls for effective resolution. [130] investigate a real application in IT problem management, and address the possibility of improving the efficiency of problem solving by mining resolution sequences and text contents.

Fault Tolerant: Traditional testing practices need to conduct testing activ-

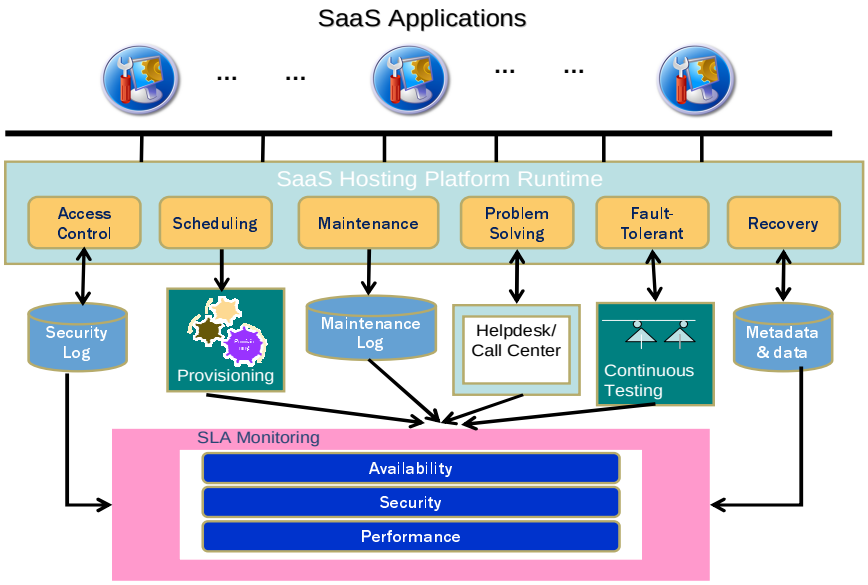


Fig. 83. SaaS Hosting Platform Support

ities after all development activities are completed. To test the modified part of a software can cost longer time, as well as costs more labors. Such a sequential develop-test process is insufficient to satisfy the requirement fast involvement posted by the MTA and SaaS model. Therefore, this paper proposes an embedded capability of continuous testing in the SaaS framework, to address the testing challenges introduced by SaaS model.

Recovery: The tripartite recovery model (TRM) [155] uses three major components: OC (ontology component), MC (metadata component) and DC (data component) for data and metadata recovery. In a traditional cloud environment, data and meta- data are often at least triplicated into different chunks to ensure reliability and availability. This is good as long as at least one chunk is available in case of failures, while data lost can be troublesome, metadata lost can cause significant issues for a cloud. This paper extends the traditional approach by having

redundant metadata information among the three components so that metadata can be recovered in case all the metadata chunks are lost. Any information stored in each component will have redundancy. Note that currently, a cloud environment does not have an OC, but it can be added for customization, service specification, classification, and now metadata and data recovery.

9.6. Related Work

SOA application construction

The conventional SOA is producer-centric because service providers publish their services and service consumers must search available services to compose their applications. consumer-centric service-oriented architecture (CCSOA) [157]is different as it allows consumers publish their needs including workflows and services, and let producers to produce services to meet the requirements. Based on CCSOA, [26] introduces a new user-centric service oriented architecture (UCSOA) that allows end users to compose applications. UCSOA is an extension of CCSOA, which is an extension of conventional SOA. UCSOA provides support for end users. An application builder is an engineer who has both domain and programming knowledge, while an end user has little knowledge on programming and thus UCSOA needs to allow nontechnical persons to compose their applications. Global Software Enterprise (GSE)[158] proposed a new software constructing architecture: the software construction starts from the consumers' publishing their requirements in the form of application or service specification which they exactly want. The consumer also publishes the expectation for the services the application needs in the form of collaboration specification and test scripts. The service providers then produce services

that meet the needs of the applications. This new approach reduces the workload for the consumer and the communication between the consumer and provider. It also extends the design and code sharing, and thus further improves the software productivity.

Dependency Analysis and Testing

[45] uses a functional tree method to test the input parameters with constraints, which is based on the traditional boundary value analysis in black-boxing testing. This method solves the limitation of the traditional approach which does not cover the dependency relationship between different parameters. [114] proposed a dynamic dependency analysis framework which is used to extract dependencies in component-based system. It can analyze different components' dependencies, which were developed by different developers. Based on the dynamic characteristics, it can capture runtime information between different components. [56] analyzed the dependency at the code level, the formal analysis has been proposed to identify input-data dependencies in the code based on abstract interpretation. A formal description of the abstract-interpretation framework is provided by the whole language. It also demonstrates the process of finding input dependencies.

In this chapter, EasySaaS uses dependency between services to validate the new coming services, which can easy the testing process.

9.7. Conclusion

This chapter proposes a new SaaS framework EasySaaS, which consider the consumers as the center of the development by leverage the workload of searching and customization. The framework is different from current SaaS framework, in stead

of starting from scratch and customize applications, the tenant can publishing their requirements into the global SaaS platform in the form of application requirement and specification with their unique business requirements, as well as their expectation of the SaaS outcome and test scripts. The SaaS providers proposes their SaaS products, customize their services to meet tenant's requirements. This framework releases the workload of tenants, and provide an easier way to customize tenants' business requirement in a collaborative way. The SaaS providers also get benefits from the shared platform, and fast the development process. A hierarchy global index is used to support the matching and customization process.

10. CONCLUSION AND FUTURE WORK

My thesis discusses several critical research problems in supporting effective and intelligent multi-tenancy SaaS, including:

1. *Service Requests Prioritization*: service providers receive multiple requests from customers, how to prioritize those service requests to maximize the business values and minimize customers' dissatisfaction is one of the most important issues in cloud. An innovative prioritization model is proposed, which uses different types of information, including customer, service, environment and workflow information to optimize the performance of the system. The model is applied to a real end-to-end mortgage origination process and evaluate the performance of the model.

2. *Service Demand Forecasting*: most services experience seasonal or other periodic demand variation as well as some unexpected demand bursts due to external events. The only way to provide "on-demand" services, is to provision in advance. Accurate demand prediction and provision become critical for the successful of the cloud computing, which reduces the waste of utility purchase and can therefore save money using utility computing. An effective demand prediction model is proposed, and apply it to a real mortgage application.

3. *SaaS Customization*: to support a significant number of tenants, SaaS applications need be customizable to fulfill the various functional and quality requirements of individual tenants. A unified and innovative multi-layered

customization framework is proposed to support and manage the variability of SaaS applications and tenants-specific requirements. Ontology is used to derive customization and deployment information for tenants cross layers. This framework also has an intelligent recommendation engine to support new tenants to deploy using information from existing deployed SaaS applications. A case study in mortgage application is used to demonstrate the proposed model.

4. *Scalable and Robust SaaS*: The multi-tenancy architecture and customization requirements have brought up new issues in software, such as database design, database partition, scalability, recovery, and continuous testing. A hybrid test database design to support SaaS customization with two-layer database partitioning is proposed. Furthermore, constraints in metadata can be used either as test cases or policies to support SaaS continuous testing and policy enforcement.

5. *Secure SaaS*: security is an important issue due to the increase scale of users. Current approaches to access control on clouds do not scale well to multi-tenancy requirements because they are mostly based on individual user IDs at different granularity levels, however, the number of users can be enormous and causing significant overhead in managing security. RBAC (Role-Based Access Control) is attractive because the number of roles is significantly less, and users can be classified according to their roles. A RBAC model is proposed using a role ontology for Multi-Tenancy Architecture (MTA) in clouds. The ontology is used as to build up the role hierarchy for a specific domain. An

ontology transformation operations algorithms are provided to compare the similarity of different ontology. The proposed framework can easy the design of security system in cloud and reduce the complexity of system design and implementation.

6. EasySaaS: To support a significant number of tenants, an easy to use SaaS construction framework is highly desirable. An easy SaaS constructing architecture is proposed: an automatic SaaS construction framework. In the architecture, in stead of starting from scratch and customize applications, the tenant can publishing their requirements into the global SaaS platform in the form of application requirement and specification with their unique business requirements, as well as their expectation of the SaaS outcome and test scripts. The SaaS providers proposes their SaaS products, customize their services to meet tenant's requirements. This framework releases the workload of tenants, and provide an easier way to customize tenants' business requirement in a collaborative way. The SaaS providers also get benefits from the shared platform, and fast the development process. A hierarchy global index is used to support the matching and customization process.

There are many interesting research problems in multi-tenancy SaaS, for example, how to further extend the prioritization framework to Hadoop and evaluate the general performance of the model. Also, integrating this scheme with diverse possible scheduling policies in the cloud to find solutions for prioritizing requests. How to design a collaborative database partitioning and scheduling algorithm to

work together with the proposed customization framework, as well as loan balancing, recovery mechanism in SaaS. How to further improve the robustness of SaaS framework. I will continue work on these interesting research topics in the future.

REFERENCES

[1] Kjersti Aas and Line Eikvil. Text categorisation: A survey., 1999.

[2] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *EDBT*, pages 469–483, 1998.

[3] Gail-Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, 2000.

[4] Julien Allali and Marie france Sagot. Novel tree edit operations for rna secondary structure comparison. In *Proceedings of the 4th Workshop on Algorithms in BioInformatics (WABI),Lecture Notes in Computer Science*, pages 412–425. Springer-Verlag, 2004.

[5] Microsoft Multi-Tenancy Architecture. http://msdn.microsoft.com/en-us/library/aa479086.aspx.

[6] Arithum. http://www.arithum.com/, 2010.

[7] J. Aslam, E. Yilmaz, and V. Pavlu. A geometric interpretation of r-precision and its correlation with average precsision. *SIGIR*, 2005.

[8] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD '08*, pages 1195–1206, New York, NY, USA, 2008. ACM.

[9] Windows Azure. http://www.microsoft.com/windowsazure/windowsazure/, 2010.

[10] Xiaoying Bai, Shufang Lee, Wei-Tek Tsai, and Yinong Chen. Ontology-based test modeling and partition testing of web services. In *ICWS '08*, pages 465–472, Washington, DC, USA, 2008. IEEE Computer Society.

[11] Xiaoying Bai, Yongli Liu, Lijun Wang, Wei-Tek Tsai, and Peide Zhong. Model-based monitoring and policy enforcement of services. In *SERVICES I*, pages 789–796, 2009.

[12] Xiaoying Bai, Yongbo Wang, Guilan Dai, Wei-Tek Tsai, and Yinong Chen. A framework for contract-based collaborative verification and validation of web services. In *CBSE'07*, pages 258–273, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] Ryan Barrett. 2008 google i/o session videos and slides.

[14] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[15] Elisa Bertino and Laura M. Haas. Views and security in distributed database management systems. In *EDBT '88*, pages 155–169, London, UK, 1988. Springer-Verlag.

[16] Rafae Bhatti, Elisa Bertino, Arif Ghafoor, and James B. D. Joshi. Xml-based specification for web services document security. *Computer*, 37(4):41–49, 2004.

[17] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

[18] Peter A. Boncz, Marcin Zukowski, and Niels Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, 2005.

[19] Matthias Brantner, Daniela Florescu, David Graf, Donald Kossmann, and Tim Kraska. Building a database on s3. In *SIGMOD '08*, pages 251–264, New York, NY, USA, 2008. ACM.

[20] Hong Cai, Ke Zhang, MingJun Zhou, Wei Gong, JunJie Cai, and XinSheng Mao. An end-to-end methodology and toolkit for fine granularity saas-ization. In *CLOUD '09*, pages 101–108, 2009.

[21] Google Data Center. http://www.youtube.com/watch?v=zrwpsfplx8i.

[22] S. Ceri, S. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Trans. Softw. Eng.*, 9(4):487–504, 1983.

[23] Stefano Ceri and Giuseppe Pelagatti. *Distributed databases principles and systems*. McGraw-Hill, Inc., New York, NY, USA, 1984.

[24] David W. Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with x.509 attribute certificates. *IEEE Internet Computing*, 7(2):62–69, 2003.

[25] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06*, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[26] Mark Chang, Jackson He, Wei-Tek Tsai, Bingnan Xiao, and Yinong Chen. Ucsoa: User-centric service-oriented architecture. In *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 248–255, Washington, DC, USA, 2006. IEEE Computer Society.

[27] I-Min A. Chen and Victor M. Markowitz. Modeling scientific experiments with an object data model. In *International Conference on Data Engineering*, pages 391–400, 1995.

[28] Wai-Ki Ching, Eric S. Fung, and Michal K. Ng. A multivariate markov chain model for categorical data sequences and its applications in demand predictions. *IMA Journal of Management Mathematics*, 2002.

[29] Frederick Chong and Gianpaolo Carraro. Architecture strategies for catching the long tail. 2006.

[30] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-tenant data architecture, June 2006.

[31] IBM Cloud. http://www.ibm.com/developerworks/data/library/techarticle/dm-0712taylor/index.html.

[32] CloudStore. http://kosmosfs.sourceforge.net/index.html.

[33] Cloud computing(wiki). http://en.wikipedia.org/wiki/cloud_computing.

[34] J. Cook and A. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[35] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, 2008.

[36] Jason Crampton and George Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security*, 6:201–231, 2003.

[37] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. G-store: a scalable data store for transactional multi key access in the cloud. In *SoCC '10*, pages 163–174, New York, NY, USA, 2010. ACM.

[38] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. *Technical Report, University of Wisconsin-Madison*, 2006.

[39] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07*, pages 205–220, New York, NY, USA, 2007. ACM.

[40] George Eadon, Eugene Inseok Chong, Shrikanth Shankar, Ananth Raghavan, Jagannathan Srinivasan, and Souripriya Das. Supporting table partitioning by reference in oracle. In *SIGMOD '08*, pages 1111–1122, New York, NY, USA, 2008. ACM.

[41] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2010.

[42] Marie Gu Egan, Marie Gugan, and Nicolas Hern. Recognizing textual parallelisms with edit distance and similarity degree, 2006.

[43] Google App Engine. http://code.google.com/appengine, 2010.

[44] Moez Essaidi. Odbis: towards a platform for on-demand business intelligence services. In *EDBT '10: ICDT Workshops*, pages 1–6, New York, NY, USA, 2010. ACM.

[45] Wenying Feng. A generalization of boundary value analysis for input parameters with functional dependency. *Computer and Information Science, ACIS International Conference on*, 0:776–781, 2010.

[46] Force.com. http://force.com/.

[47] Armando Fox. The potential of cloud computing: Opportunities and challenges.

[48] Mark S. Fox, Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modeling: preliminary concepts for linking structure and behaviour. *Comput. Ind.*, 29(1-2):123–134, 1996.

[49] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Professional, 1995.

[50] R. Gerritsen. Assessing loan risks: A data mining case study. *IT Pro*, 1999.

[51] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[52] S. Godik and T. Moses. Oasis extensible access control markup language (xacml) version 1.1. oasis committee specification. July 2003.

[53] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[54] W. Greene. Econometric analysis. *Prentice Hall, Inc*, 2000.

[55] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate xml joins. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 287–298, New York, NY, USA, 2002. ACM.

[56] J. Gustafsson, B. Lisper, R. Kirner, and P. Puschner. Input-dependency analysis for hard real-time software. pages 53 – 53, oct. 2003.

[57] Hadoop. http://hadoop.apache.org/.

[58] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[59] HBase. Hbase: Bigtable-like structured storage for hadoop hdfs, 2009.

[60] http://asusrl.eas.asu.edu/srlab/.

[61] http://csrc.nist.gov/groups/SNS/cloud computing/index.html.

[62] Graham Hughes and Tevfik Bultan. Automated verification of access control policies using a sat solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–520, 2008.

[63] Microsoft Building New Data Center in Quincy. http://www.datacenterknowledge.com/archives/2010/05/19/microsoft-building-new-data-center-in-quincy/.

[64] iTKO. itko lisa.

[65] V. Iyengar, I. Boier, K. Kelley, and R. Curatolo. Analytics for audit and business controls in corporate travel and entertainment. *Sixth Australasian Data Mining conference*, 2007.

[66] Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik, and Yonghua Wu. A framework for efficient storage security in rdbms. In *In EDBT*, 2004.

[67] Harshavardhan Jegadeesan and Sundar Balasubramaniam. Differentiating commoditized services in a services marketplace. In *SCC '08*, pages 153–160. IEEE Computer Society, 2008.

[68] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, 2000.

[69] G. John and Y. Zhao. Mortgage data mining. *Computational Intelligence in Finanacial Engineering*, 232-236, 1997.

[70] James B. D. Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. on Knowl. and Data Eng.*, 17(1):4–23, 2005.

[71] K. Schliep K. Hechenbichler. Weighted k-nearest-neighbor techniques and ordinal classification. Technical report, Ludwig-Maximilians University, 2007.

[72] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97*, pages 654–663, New York, NY, USA, 1997. ACM.

[73] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World, second ed.* 2002.

[74] Axel Kern, Martin Kuhlmann, Rainer Kuropka, and Andreas Ruthert. A meta model for authorisations in application security systems and their integration into rbac administration. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 87–96, New York, NY, USA, 2004. ACM.

[75] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *In Proceedings of the 6th annual European Symposium on Algorithms (ESA*, pages 91–102. Springer-Verlag, 1998.

[76] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Administrative scope in the graph-based framework. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 97–104, New York, NY, USA, 2004. ACM.

[77] Donald Kossmann, Tim Kraska, and Simon Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD '10*, pages 579–590, New York, NY, USA, 2010. ACM.

[78] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: pay only when it matters. *Proc. VLDB Endow.*, 2(1):253–264, 2009.

[79] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms.* The Benjamin/Cummings, 1994.

[80] Qianhui Althea Lang and et al. And/or graph and search algorithm for discovering composite web services, 2005.

[81] Stephen S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, Inc., Orlando, FL, USA, 1983.

[82] Jong-Kwon Lee and Jennifer C. Hou. Modeling steady-state and transient behaviors of user mobility: formulation, analysis, and application. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 85–96, New York, NY, USA, 2006. ACM.

[83] W. Lehner and K.-U. Sattler. Database as a service (dbaas). pages 1216 –1217, mar. 2010.

[84] Wolfgang Lehner and Kai-Uwe Sattler. Database as a service (dbaas). In *ICDE*, pages 1216–1217, 2010.

[85] Hongbo Li, Yuliang Shi, and Qingzhong Li. A multi-granularity customization relationship model for saas. In *WISM '09*, pages 611–615, Washington, DC, USA, 2009. IEEE Computer Society.

[86] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, 2006.

[87] Miron Livny, Setrag Khoshafian, and Haran Boral. Multi-disk management algorithms. *SIGMETRICS Perform. Eval. Rev.*, 15(1):69–77, 1987.

[88] H. Mannila and D. Rusakov. Decomposition of event sequences into independent components. In *Proc. 1st SIAM Conf. Data Mining*, pages 1–17, 2001.

[89] H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[90] Fabio Massacci. Reasoning about security: a logic and a decision method for role-based access control. In *In ECSQARU-FAPR*, pages 421–435, 1997.

[91] Sally Mcclean. Semi-markov models for human-resource modelling. *IMA Journal of Management Mathematics*, 4(4):307–315, 1992.

[92] M.O. McFaddin. Adaptive Customization: New Design Opportunities in Orthopedics, Driven by the Merging of Imaging and Surgery. 45(4), 2007.

[93] Marshall Kirk McKusick and Sean Quinlan. Gfs: Evolution on fast-forward. *Queue*, 7(7):10–20, 2009.

[94] Manish Mehta and David J. DeWitt. Data placement in shared-nothing parallel database systems. *The VLDB Journal*, 6(1):53–72, 1997.

[95] Ralph Mietzner and Frank Leymann. Generation of bpel customization processes for saas applications from variability descriptors. In *SCC '08*, pages 359–366, Washington, DC, USA, 2008. IEEE Computer Society.

[96] Tim Moses. extensible access control markup language tc v2.0 (xacml), February 2005.

[97] OASIS. http://www.oasisopen.org/committees/uddispec/doc/tcspecs.htm.

[98] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08*, pages 1099–1110, New York, NY, USA, 2008. ACM.

[99] Ontology. http://en.wikipedia.org/wiki/ontology_(information_science).

[100] Oracle. The virtual private database in oracle9ir2, 2008.

[101] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[102] Justin O'Sullivan, David Edmond, and Arthur H. M. ter Hofstede. Formal description of non-functional service properties. In *Technical Report FIT-TR-2005-01*, Queensland University of Technology, Australia., 2005.

[103] Jason Ouellette. *Development with the Force.com Platform: Building Business Applications in the Cloud*. Addison-Wesley Professional, 2009.

[104] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, 4(1):37–71, 2001.

[105] L. Pastor and J.L. Bosque. An efficiency and scalability model for heterogeneous clusters. In *cluster*, page 427. Published by the IEEE Computer Society, 2001.

[106] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD '09*, pages 165–178, New York, NY, USA, 2009. ACM.

[107] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, August 1985.

[108] M. Pinedo. Scheduling theory, algorithms and systems. *Prentice Hall, Inc*, 1995.

[109] Helena Sofia Pinto and Martins. Ontologies: How can they be built? *Knowl. Inf. Syst.*, 6(4):441–464, 2004.

[110] Helena Sofia Pinto and ao P Martins, Jo˙ A methodology for ontology integration. In *K-CAP '01: Proceedings of the 1st international conference on Knowledge capture*, pages 131–138, New York, NY, USA, 2001. ACM.

[111] V. Poor. An intorudction to signal detection and estimation. *Springer texts in Electrical Engineering*, 1994.

[112] Taverna Project. http://taverna.sourceforge.net/.

[113] F. Provost, T. Fawcett, and R. Kohavi. The case aainst accuracyestimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[114] Binbin Qu, Qian Liu, and Yansheng Lu. A framework for dynamic analysis dependency in component-based system. volume 4, pages V4–250 –V4–254, apr. 2010.

[115] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems.* 2007.

[116] Chandramouli Ramaswamy, Ravi Sandhu, Ramouli Ramaswamy, and Ravi S. Role-based access control features in commercial database management systems. In *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, pages 503–511, 1998.

[117] Role-Based Access Control 2000 Workshop RBAC, 2000.

[118] Salesforce.com real time feature. http://www.salesforce.com/platform/cloudinfrastructure/integration.jsp, 2010.

[119] Werner Retschitzegger, Franz Phretmair, and Gerhard Nussbaum. Towards an ontology-based cutomization approach for supporting people with special needs.

[120] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IN: MIDDLEWARE*, pages 329–350, 2001.

[121] Domenico Sacca and Gio Wiederhold. Database partitioning in a cluster of processors. *ACM Trans. Database Syst.*, 10(1):29–56, 1985.

[122] David Saff and Michael D. Emst. Continuous testing in eclipse. In *2nd Eclipse Technology Exahcange Workshop (eTX)*, Barcelona, Spain, March 2004.

[123] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[124] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The nist model for role-based access control: towards a unified standard. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, New York, NY, USA, 2000. ACM.

[125] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[126] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system- a case study. In *In ACM WebKDD Workshop*, 2000.

[127] Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The role-based access control system of a european bank: a case study and discussion. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 3–9, New York, NY, USA, 2001. ACM.

[128] Patrick J. Schroeder and Bogdan Korel. Black-box test reduction using input-output analysis. *SIGSOFT Softw. Eng. Notes*, 25(5):173–177, 2000.

[129] WS-Policy. Web services policy framework (ws policy). http://www-106.ibm.com/developerworks/library/specification/wspolfram/.

[130] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Efficient ticket routing by resolution sequence mining. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 605–613, New York, NY, USA, 2008. ACM.

[131] Qihong Shao, Anshul Sheopuri, Milind Naphade, Chitra Dorai, Daniel Johnson, and Jane Hoffman. Ranking mortgage origination applications using customer, product, environment and workflow attributes. *Cloud'09*, pages 198–205, 2009.

[132] Qihong Shao, Peng Sun, and Yi Chen. Wise: A workflow information search engine. In *ICDE '09*, pages 1491–1494, Washington, DC, USA, 2009. IEEE Computer Society.

[133] J. Shawe-Taylor and N. Cristianini. Support vector machines and other kernel-based learning methods. 2000.

[134] A. Sheopuri, S. Zeng, and C. Dorai. A new policy for the service request assignment problem with multiple severity level, due date and sla penalty service requests. *Winter Simulation Conference*, 2008.

[135] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[136] E. Smith. Continuous testing. In *Proceedings of the 17th International Conference on Testing Computer Software*, 2000.

[137] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, 2008.

[138] Ellen Spertus, Mehran Sahami, and Orkut Buyukkokten. Evaluating similarity measures: A large-scale study in the orkut social network. In *KDD '05*, pages 678–684, New York, NY, USA, 2005. ACM.

[139] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[140] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel R. Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alexander Rasin, Nga Tran, and Stan B. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, Trondheim, Norway, 2005.

[141] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1150–1160. VLDB Endowment, 2007.

[142] X.H. Sun. Scalability versus Execution Time in Scalable Systems* 1. *Journal of Parallel and Distributed Computing*, 62(2):173–192, 2002.

[143] X.H. Sun, Y. Chen, and M. Wu. Scalability of heterogeneous computing. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 557–564. IEEE, 2005.

[144] X.H. Sun and L.M. Ni. Scalable problems and memory-bounded speedup. *Journal of Parallel and Distributed Computing*, 19:27–37, 1993.

[145] X.H. Sun and D.T. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems*, pages 599–613, 1994.

[146] Distributed Hash Tables. http://en.wikipedia.org/wiki/distributed_hash_table.

[147] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[148] Vandan Trivedi, Ira Moscovice, Richard Bass, and John Brooks. A semi-markov model for primary health care manpower supply prediction. *Manage. Sci.*, 33(2):149–160, 1987.

[149] Wei-Tek Tsai. Service-oriented system engineering: a new paradigm. pages 3 – 6, oct. 2005.

[150] Wei-Tek Tsai, Yinong Chen, Ray Paul, Xinyu Zhou, and Chun Fan. Simulation verification and validation by dynamic policy specification and enforcement. *Simulation*, 82(5):295–310, 2006.

[151] Wei-Tek Tsai, Qian Huang, Jay Elston, and Yinong Chen. Service-oriented user interface modeling and composition. In *ICEBE '08: Proceedings of the 2008 IEEE International Conference on e-Business Engineering*, pages 21–28, Washington, DC, USA, 2008. IEEE Computer Society.

[152] Wei-Tek Tsai, Qian Huang, Jingjing Xu, Yinong Chen, and Ray Paul. Ontology-based dynamic process collaboration in service-oriented architecture. In *SOCA '07*, pages 39–46, Washington, DC, USA, 2007. IEEE Computer Society.

[153] Wei-Tek Tsai and Qihong Shao. Role-based access-control using reference ontology in clouds. Technical Report ASU-CS-TR-xxx, Arizona State University, June 2010.

[154] Wei-Tek Tsai, Qihong Shao, and Jay Elson. Prioritizing service requests on cloud with multi-tenancy. *Int. Conf. on e-Business Engineering (ICEBE'10)*.

[155] Wei-Tek Tsai, Qihong Shao, Yu Huang, and Xiaoying Bai. Towards a scalable and robust multi-tenancy saas. In *Internetware' 2010*, 2010.

[156] Wei-Tek Tsai, Qihong Shao, and Wu Li. Oic: Ontology-based intelligent customization framework for saas. In *SOCA*, 2010.

[157] Wei-Tek Tsai, Bingnan Xiao, Yinong Chen, and Raymond A. Paul. Consumer-centric service-oriented architecture: A new approach. In *SEUS-WCCIA '06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 175–180, Washington, DC, USA, 2006. IEEE Computer Society.

[158] Wei-Tek Tsai, Bingnan Xiao, Ray Paul, Qian Huang, and Yinong Chen. Global software enterprise: A new software constructing architecture. In *CEC-EEE '06: Proceedings of the The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, page 55, Washington, DC, USA, 2006. IEEE Computer Society.

[159] Wei-Tek Tsai, Xinyu Zhou, Raymond A. Paul, Yinong Chen, and Xiaoying Bai. A coverage relationship model for test case selection and ranking for multi-version software. *HASE*, 0:105–112, 2007.

[160] Richard Valliant and George Milkovich. Comparison of semi-markov and markov models in a personnel forecasting application. *Decision Sciences*, 8(2):465–477, 1977.

[161] Guido van Rossum. Google app engine presentation.

[162] Martin Warin, Henrik Oxhammar, and Martin Volk. Enriching an ontology with wordnet based on similarity measures. In *In: MEANING-2005 Workshop*, 2005.

[163] Craig Weissman. Behind the scenes: Salesforce.com chief technology officer on cloud architecture, 2009.

[164] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD '09*, pages 889–896, New York, NY, USA, 2009. ACM.

[165] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD '09*, pages 889–896, New York, NY, USA, 2009. ACM.

[166] J. Wong and L. Fung. Residential mortgage default risk and the loan-to-value ratio. *Hong Kong Monetary Authority Quarterly Bulletin*, 2004.

[167] Yalin Yarimagan and Asuman Dogac. Semantics based customization of ubl document schemas. *Distrib. Parallel Databases*, 22(2-3):107–131, 2007.

[168] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets Syst.*, 69(2):125–139, 1995.

[169] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[170] Kuo Zhang, Xin Zhang, Wei Sun, Haiqi Liang, Ying Huang, Liangzhao Zhen, and Xuanzhe Liu. A policy-driven approach for software-as-services customization. *ICEBE*, 0:123–130, 2007.

[171] Chen Zhao, Nuermaimaiti Heilili, Shengping Liu, and Zuoquan Lin. Representation and reasoning on rbac: A description logic approach. In *In ICTAC*, pages 381–393, 2005.

BIOGRAPHICAL SKETCH

Qihong Shao was born in Dandong city, Liaoning province, People's Republic of China. She attended Renmin University of China (RUC), where she earned the bachelor of engineering in computer science in 2003. She received scholarship every year during her undergraduate study in RUC. Subsequently she entered the graduate school of RUC waived of entrance exam due to excellent academic record, and obtained her master of science degree in computer science department in 2006.

Qihong further pursued her doctoral degree in the area of cloud computing at Arizona State University(ASU) under the supervision of Dr. Wei-Tek Tsai. She published around 20 papers in top conferences and journals.