**Unit 3**

**Infrastructure as a Service (IAAS) & Platform and Software as a Service (PAAS / SAAS)**

**Public Cloud and Infrastructure Services**

Public cloud or external cloud describes cloud computing in a traditional mainstream sense, whereby resources are dynamically provisioned via publicly accessible Web applications/Web services (SOAP or RESTful interfaces) from an off-site third-party provider, who shares resources and bills on a fine-grained utility computing basis, the user pays only for the capacity of the provisioned resources at a particular time.

There are many examples for vendors who publicly provide infrastructure as a service. Amazon Elastic Compute Cloud (EC2) is the best known example, but the market now bristles with lots of competition like GoGrid, Joyent Accelerator, Rackspace, AppNexus, FlexiScale, and Manjrasoft Aneka.

Here, we will briefly cover and describe Amazon EC2 offering. Amazon Elastic Compute Cloud (EC2) is an IaaS service that provides elastic compute capacity in the cloud. These services can be leveraged via Web services (SOAP or REST), a Web-based AWS (Amazon Web Service) management console, or the EC2 command line tools. The Amazon service provides hundreds of pre-made AMIs (Amazon Machine Images) with a variety of operating systems (i.e., Linux, OpenSolaris, or Windows) and pre-loaded software.

It provides you with complete control of your computing resources and lets you run on Amazon’s computing and infrastructure environment easily. Amazon EC2 reduces the time required for obtaining and booting a new server’s instances to minutes, thereby allowing a quick scalable capacity and resources, up and down, as the computing requirements change. Amazon offers different instances’ size according to (a) the resources’ needs (small, large, and extra large), (b) the high CPU’s needs it provides (medium and extra large high CPU instances), and (c) high-memory instances (extra large, double extra large, and quadruple extra large instance).

**Private Cloud and Infrastructure Services**

A private cloud aims at providing public cloud functionality, but on private resources, while maintaining control over an organization’s data and resources to meet security and governance’s requirements in an organization. Private cloud exhibits a highly virtualized cloud data center located inside your organi-zation’s firewall. It may also be a private space dedicated for your company within a cloud vendor’s data center designed to handle the organization’s workloads.

Private clouds exhibit the following characteristics:

Allow service provisioning and compute capability for an organization’s users in a self-service manner.

Automate and provide well-managed virtualized environments. Optimize computing resources, and servers’ utilization.

Support specific workloads.

There are many examples for vendors and frameworks that provide infrastruc-ture as a service in private setups. The best-known examples are Eucalyptus and OpenNebula (which will be covered in more detail later on).

It is also important to highlight a third type of cloud setup named “hybrid cloud,” in which a combination of private/internal and external cloud resources

exist together by enabling outsourcing of noncritical services and functions in public cloud and keeping the critical ones internal. Hybrid cloud’s main function is to release resources from a public cloud and to handle sudden demand usage, which is called “cloud bursting.”

**Distributed Management of Virtualization**

Virtualization’s benefits bring their own challenges and complexities presented in the need for a powerful management capabilities. That is why many commercial, open source products and research projects such as OpenNebula, IBM Virtualization Manager, Joyent, and VMware DRS are being developed to dynamically provision virtual machines, utilizing the physical infrastrcture. There are also some commercial and scientific infrastructure cloud computing initiatives, such as Globus VWS, Eucalyptus and Amazon, which provide remote interfaces for controling and monitoring virtual resources. One more effort in this context is the RESERVOIR initiative, in which grid interfaces and protocols enable the required interoper-ability between the clouds or infrastructure’s providers. RESERVOIR also, needs to expand substantially on the current state-of-the-art for grid-wide accounting, and to increase the flexibility of supporting different billing schemes, and accounting for services with indefinite lifetime, as opposed to finite jobs with support to account for utilization metrics relevant to virtual machines.

**High Availability**

High availability is a system design protocol and an associated implementation that ensures a certain absolute degree of operational continuity during a given measurement period. Availability refers to the ability of a user’s community to access the system—whether for submiting new work, updating or altering existing work, or collecting the results of the previous work. If a user cannot access the system, it is said to be unavailable. This means that services should be available all the time along with some planned/unplanned downtime according to a certain SLA (formalize the service availability objectives, and requirements) which often refers to the monthly availability or downtime of a service; to calculate the service’s credits to match the billing cycles. Services that are considered as business critical are often categorized as high availability services. Systems running business critical services should be planned and designed from the bottom with the goal of achieving the lowest possible amount of planned and unplanned downtime.

Since a virtual environment is the larger part of any organization, management of these virtual resources within this environment becomes a critical mission, and the migration services of these resources became a corner stone in achieving high availability for these services hosted by VMs. So, in the context of virtualized infrastructure, high availability allows virtual machines to automatically be restarted in case of an underlying hardware failure or individual VM failure. If one of your servers fails, the VMs will be restarted on other virtualized servers in the resource pool, restoring the essential services with minimal service interruption.

**Cloud and Virtualization Standardization Efforts**

Standardization is important to ensure interoperability between virtualization management vendors, the virtual machines produced by each one of them, and cloud computing. Here, we will have look at the prevalent standards that make cloud computing and virtualization possible. In the past few years, virtualization standardization efforts led by the Distributed Management Task Force (DMTF) have produced standards for almost all the aspects of virtualization technology. DMTF initiated the VMAN (Virtualization Management Initiative), which delivers broadly supported interoperability and portability standards for managing the virtual computing lifecycle. VMAN’s OVF (Open Virtualization Format) in a collaboration between industry key players: Dell, HP, IBM, Microsoft, XenSource, and Vmware. OVF specification provides a common format to package and securely distribute virtual appliances across multiple virtualization platforms. VMAN profiles define a consistent way of managing a heterogeneous virtualized environment.

**OCCI and OGF**

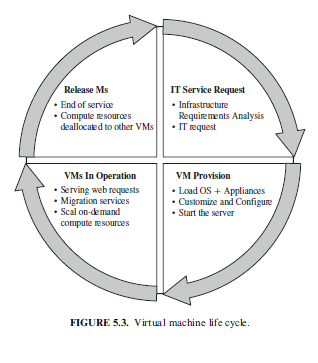
Another standardization effort has been initiated by Open Grid Forum (OGF) through organizing an official new working group to deliver a standard API for cloud IaaS, the Open Cloud Computing Interface Working Group (OCCI-WG). This group is dedicated for delivering an API specification for the remote management of cloud computing’s infrastructure and for allowing the development of interoperable tools for common tasks including deployment, autonomic scaling, and monitoring. The scope of the specification will be covering a high-level functionality required for managing the life-cycle virtual machines (or workloads), running on virtualization technologies (or containers), and supporting service elasticity. The new API for interfacing “IaaS” cloud computing facilities will allow :

Consumers to interact with cloud computing infrastructure on an ad hoc basis. Integrators to offer advanced management services. Aggregators to offer a single common interface to multiple providers. Providers to offer a standard interface that is compatible with the available tools. Vendors of grids/clouds to offer standard interfaces for dynamically scalable service’s delivery in their products.

**VIRTUAL MACHINES PROVISIONING AND MANAGEABILITY**

In this section, we will have an overview on the typical life cycle of VM and its major possible states of operation, which make the management and automation of VMs in virtual and cloud environments easier than in traditional computing environments.

As shown in Figure 5.3, the cycle starts by a request delivered to the IT department, stating the requirement for creating a new server for a particular service. This request is being processed by the IT administration to start seeing the servers’ resource pool, matching these resources with the requirements, and starting the provision of the needed virtual machine. Once it is provisioned and started, it is ready to provide the required service according to an SLA, or a time period after which the virtual is being released; and free resources, in this case, won’t be needed.



**VM Provisioning Process**

Provisioning a virtual machine or server can be explained and illustrated as in Figure 5.4: Steps to Provision VM. Here, we describe the common and normal steps of provisioning a virtual server:

Firstly, you need to select a server from a pool of available servers (physical servers with enough capacity) along with the appropriate OS template you need to provision the virtual machine.

Secondly, you need to load the appropriate software (operating system you selected in the previous step, device drivers, middleware, and the needed applications for the service required).

Thirdly, you need to customize and configure the machine (e.g., IP address, Gateway) to configure an associated network and storage resources.

Finally, the virtual server is ready to start with its newly loaded software.

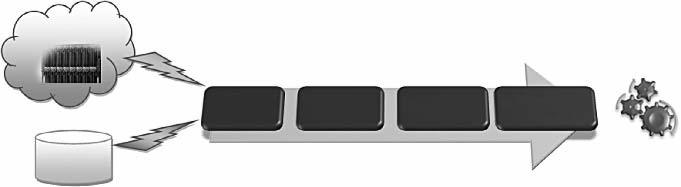
Typically, these are the tasks required or being performed by an IT or a data center’s specialist to provision a particular virtual machine.

To summarize, server provisioning is defining server’s configuration based on the organization requirements, a hardware, and software component (processor, RAM, storage, networking, operating system, applications, etc.). Normally, virtual machines can be provisioned by manually installing an operating system, by using a preconfigured VM template, by cloning an existing VM, or by importing a physical server or a virtual server from another hosting platform. Physical servers can also be virtualized and provisioned using P2V (physical to virtual) tools and techniques (e.g., virt-p2v).

After creating a virtual machine by virtualizing a physical server, or by building a new virtual server in the virtual environment, a template can be created out of it. Most virtualization management vendors (VMware, XenServer, etc.) provide the data center’s administration with the ability to do such tasks in an easy way. Provisioning from a template is an invaluable feature, because it reduces the time required to create a new virtual machine.

Administrators can create different templates for different purposes. For example, you can create a Windows 2003 Server template for the finance department, or a Red Hat Linux template for the engineering department. This enables the administrator to quickly provision a correctly configured virtual server on demand.

This ease and flexibility bring with them the problem of virtual machine’s sprawl, where virtual machines are provisioned so rapidly that documenting and managing the virtual machine’s life cycle become a challenge [9].

Servers Pool

Running Provisioned VM

Appliances

Repository

FIGURE 5.4. Virtual machine rovision proces

**VIRTUAL MACHINE MIGRATION SERVICES**

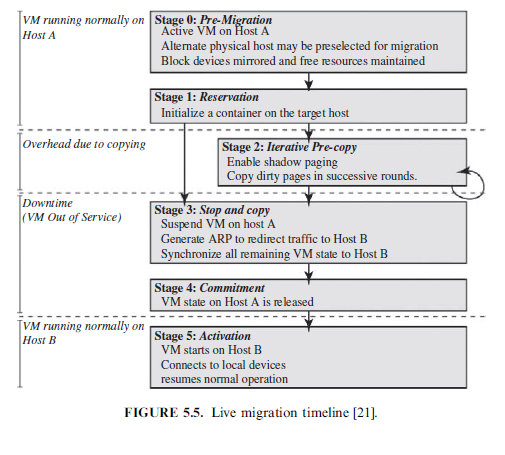
Migration service, in the context of virtual machines, is the process of moving a virtual machine from one host server or storage location to another; there are different techniques of VM migration, hot/life migration, cold/regular migration, and live storage migration of a virtual machine. In this process, all key machines’ components, such as CPU, storage disks, networking, and memory, are completely virtualized, thereby facilitating the entire state of a virtual machine to be captured by a set of easily moved data files. We will cover some of the migration’s techniques that most virtualization tools provide as a feature.

**Migrations Techniques**

Live Migration and High Availability. Live migration (which is also called hot or real-time migration) can be defined as the movement of a virtual machine from one physical host to another while being powered on. When it is properly carried out, this process takes place without any noticeable effect from the end user’s point of view (a matter of milliseconds). One of the most significant advantages of live migration is the fact that it facilitates proactive maintenance in case of failure, because the potential problem can be resolved before the disruption of service occurs. Live migration can also be used for load balancing in which work is shared among computers in order to optimize the utilization of available CPU resources.

Live Migration Anatomy, Xen Hypervisor Algorithm. In this section we will explain live migration’s mechanism and how memory and virtual machine states are being transferred, through the network, from one host A to another host B [21]; the Xen hypervisor is an example for this mechanism. The logical steps that are executed when migrating an OS are summarized in Figure 5.5. In this research, the migration process has been viewed as a transactional interaction between the two hosts involved:

Stage 0: Pre-Migration. An active virtual machine exists on the physical host A.

Stage 1: Reservation. A request is issued to migrate an OS from host A to host B (a precondition is that the necessary resources exist on B and on a VM container of that size).

Stage 2: Iterative Pre-Copy. During the first iteration, all pages are transferred from A to B. Subsequent iterations copy only those pages dirtied during the previous transfer phase.

Stage 3: Stop-and-Copy. Running OS instance at A is suspended, and its network traffic is redirected to B. As described in reference 21, CPU state and any remaining inconsistent memory pages are then transferred. At the end of this stage, there is a consistent suspended copy of the VM at both A and B. The copy at A is considered primary and is resumed in case of failure.

Stage 4: Commitment. Host B indicates to A that it has successfully received a consistent OS image. Host A acknowledges this message as a commit-ment of the migration transaction. Host A may now discard the original VM, and host B becomes the primary host.

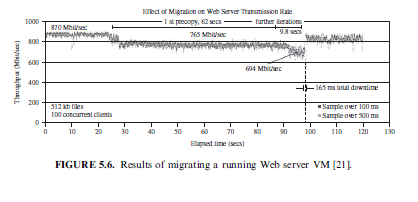
Stage 5: Activation. The migrated VM on B is now activated. Post-migration code runs to reattach the device’s drivers to the new machine and advertise moved IP addresses.

This approach to failure management ensures that at least one host has a consistent VM image at all times during migration. It depends on the assumption that the original host remains stable until the migration commits and that the VM may be suspended and resumed on that host with no risk of failure. Based on these assumptions, a migration request essentially attempts to move the VM to a new host and on any sort of failure, execution is resumed locally, aborting the migration.

Live Migration Effect on a Running Web Server. Clark et al. [21] did evaluate the above migration on an Apache 1.3 Web server; this served static content at a high rate, as illustrated in Figure 5.6. The throughput is achieved when continuously serving a single 512-kB file to a set of one hundred concurrent clients. The Web server virtual machine has a memory allocation of 800 MB. At the start of the trace, the server achieves a consistent throughput of approximately 870 Mbit/sec. Migration starts 27 sec into the trace, but is initially rate-limited to 100 Mbit/sec (12% CPU), resulting in server’s through-put drop to 765 Mbit/sec. This initial low-rate pass transfers 776 MB and lasts for 62 sec. At this point, the migration’s algorithm, described in Section 5.4.1, increases its rate over several iterations and finally suspends the VM after a further 9.8 sec. The final stop-and-copy phase then transfers the remaining pages, and the Web server resumes at full rate after a 165-msec outage.

This simple example demonstrates that a highly loaded server can be migrated with both controlled impact on live services and a short downtime. However, the working set of the server, in this case, is rather small. So, this should be expected as a relatively easy case of live migration.

Live Migration Vendor Implementations Examples. There are lots of VM management and provisioning tools that provide the live migration of VM facility, two of which are VMware VMotion and Citrix XenServer “XenMotion.”



VMware Vmotion. This allows users to (a) automatically optimize and allocate an entire pool of resources for maximum hardware utilization, flexibility, and availability and (b) perform hardware’s maintenance without scheduled downtime along with migrating virtual machines away from failing or underperforming servers.

Citrix XenServer XenMotion. This is a nice feature of the Citrix XenServer product, inherited from the Xen live migrate utility, which provides the IT administrator with the facility to move a running VM from one XenServer to another in the same pool without interrupting the service (hypothetically for zero-downtime server maintenance, which actually takes minutes), making it a highly available service. This also can be a good feature to balance the workloads on the virtualized environment.

Regular/Cold Migration. Cold migration is the migration of a powered-off virtual machine. With cold migration, you have the option of moving the associated disks from one data store to another. The virtual machines are not required to be on a shared storage. It’s important to highlight that the two main differences between live migration and cold migration are that live migration needs a shared storage for virtual machines in the server’s pool, but cold migration does not; also, in live migration for a virtual machine between two hosts, there would be certain CPU compatibility checks to be applied; while in cold migration this checks do not apply. The cold migration process is simple to implement (as the case for the VMware product), and it can be summarized as follows :

The configuration files, including the NVRAM file (BIOS settings), log files, as well as the disks of the virtual machine, are moved from the source host to the destination host’s associated storage area.

The virtual machine is registered with the new host.

After the migration is completed, the old version of the virtual machine is deleted from the source host.

Live Storage Migration of Virtual Machine. This kind of migration con-stitutes moving the virtual disks or configuration file of a running virtual machine to a new data store without any interruption in the availability of the virtual machine’s service. For more details about how this option is working in a VMware product.

**VM Migration, SLA and On-Demand Computing**

As we discussed, virtual machines’ migration plays an important role in data centers by making it easy to adjust resource’s priorities to match resource’s demand conditions.

This role is completely going in the direction of meeting SLAs; once it has been detected that a particular VM is consuming more than its fair share of resources at the expense of other VMs on the same host, it will be eligible, for this machine, to either be moved to another underutilized host or assign more resources for it, in case that the host machine still has resources; this in turn will highly avoid the violations of the SLA and will also, fulfill the requirements of on-demand computing resources. In order to achieve such goals, there should be an integration between virtualization’s management tools (with its migrations and performance’s monitoring capabilities), and SLA’s management tools to achieve balance in resources by migrating and monitoring the workloads, and accordingly, meeting the SLA.

**Migration of Virtual Machines to Alternate Platforms**

One of the nicest advantages of having facility in data center’s technologies is to have the ability to migrate virtual machines from one platform to another; there are a number of ways for achieving this, such as depending on the source and target virtualization’s platforms and on the vendor’s tools that manage this facility—for example, the VMware converter that handles migrations between ESX hosts; the VMware server; and the VMware workstation. The VMware converter can also import from other virtualization platforms, such as Micro-soft virtual server machines.

**VM PROVISIONING AND MIGRATION IN ACTION**

Now, it is time to get into business with a real example of how we can manage the life cycle, provision, and migrate a virtual machine by the help of one of the open source frameworks used to manage virtualized infrastructure. Here, we will use ConVirt (open source framework for the management of open source virtualization like Xen and KVM, known previously as XenMan).

Deployment Scenario. ConVirt deployment consists of at least one ConVirt workstation, where ConVirt is installed and ran, which provides the main console for managing the VM life cycle, managing images, provisioning new VMs, monitoring machine resources, and so on. There are two essential deployment scenarios for ConVirt: A, basic configuration in which the Xen or KVM virtualization platform is on the local machine, where ConVirt is already installed; B, an advanced configuration in which the Xen or KVM is on one or more remote servers. The scenario in use here is the advanced one. In data centers, it is very common to install centralized management software (ConVirt here) on a dedicated machine for use in managing remote servers in the data center. In our example, we will use this dedicated machine where ConVirt is installed and used to manage a pool of remote servers (two machines). In order to use advanced features of ConVirt (e.g., live migration), you should set up a shared storage for the server pool in use on which the disks of the provisioned virtual machines are stored. Figure 5.7 illustrates the scenario.

Installation. The installation process involves the following:

Installing ConVirt on at least one computer. Preparing each managed server to be managed by ConVirt.

Notes

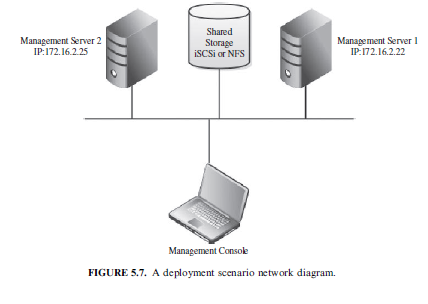
Make sure that the managed servers include Xen or KVM hypervisors installed. Make sure that you can access managed servers from your ConVirt management console through SSH. Environment, Software, and Hardware. ConVirt 1.1, Linux Ubuntu 8.10, three machines, Dell core 2 due processor, 4G RAM.

Adding Managed Servers and Provisioning VM. Once the installation is done and you are ready to manage your virtual infrastructure, then you can start the ConVirt management console (see Figure 5.8):

Select any of servers’ pools existing (QA Lab in our scenario) and on its context menu, select “Add Server.”

You will be faced with a message asking about the virtualization

platform you want to manage (Xen or KVM), as shown in Figure 5.9:



Choose KVM, and then enter the managed server information and credentials (IP, username, and password) as shown in Figure 5.10.

Once the server is synchronized and authenticated with the manage-ment console, it will appear in the left pane/of the ConVirt, as shown in Figure 5.11. Select this server, and start provisioning your virtual machine as in Figure 5.12: Fill in the virtual machine’s information (name, storage, OS template, etc.; Figure 5.13); then you will find it created on the managed server tree powered-off.

Note: While provisioning your virtual machine, make sure that you create disks on the shared storage (NFS or iSCSi). You can do so by selecting the “provisioning” tab, and changing the VM\_DISKS\_DIR to point to the location of your shared NFS.

Start your VM (Figures 5.14 and 5.15), and make sure the installation media of the operating system you need is placed in drive, in order to use it for booting the new VM and proceed in the installation process; then start the installation process as shown in Figure 5.16. Once the installation finishes, you can access your provisioned virtual machine from the consol icon on the top of your ConVirt management console. Reaching this step, you have created your first managed server and provisioned virtual machine. You can repeat the same procedure to add the second managed server in your pool to be ready for the next step of migrating one virtual machine from one server to the other.

**VM Life Cycle and VM Monitoring**

You can notice through working with ConVirt that you are able to manage the whole life cycle of the virtual machine; start, stop, reboot, migrate, clone, and so on. Also, you noticed how easy it is to monitor the resources of the managed server and to monitor the virtual machine’s guests that help you balance and control the load on these managed servers once needed. In the next section, we are going to discuss how easy it is to migrate a virtual machine from host to host.

**Live Migration**

ConVirt tool allows running virtual machines to be migrated from one server to another.This feature makes it possible to organize the virtual machine to physical machine relationship to balance the workload; for example, a VM needing more CPU can be moved to a machine having available CPU cycles, or, in other cases, like taking the host machine for maintenance. For proper VM migration the following points must be considered:

Shared storage for all Guest OS disks (e.g., NFS, or iSCSI). Identical mount points on all servers (hosts). The kernel and ramdisk when using para-virtualized virtual machines should, also, be shared. (This is not required, if pygrub is used.) Centrally accessible installation media (iso). It is preferable to use identical machines with the same version of virtualization platform. Migration needs to be done within the same subnet.

Migration Process in ConVirt. To start the migration of a virtual machine from one host to the other, select it and choose a migrating virtual machine, as shown in Figure 5.17. You will have a window containing all the managed servers in your data center (as shown in Figure 5.18). Choose one as a destination and start

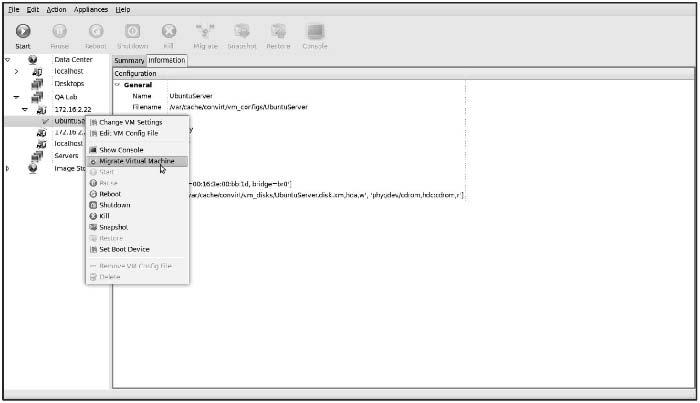


FIGURE 5.17. VM migration.

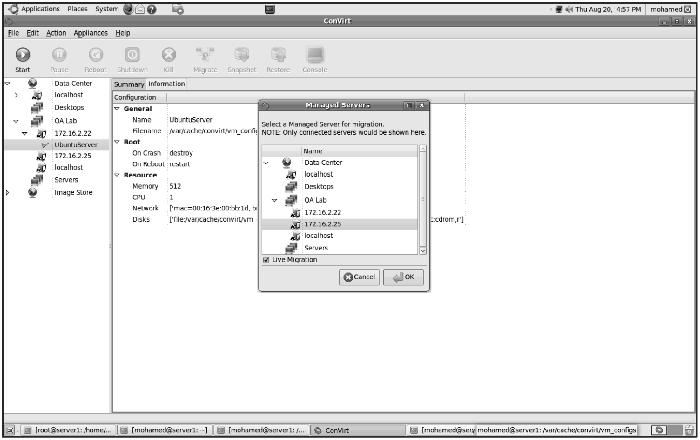


FIGURE 5.18. Select the destination managed server candidate for migration.

migration, or drag the VM and drop it on to another managed server to initiate migration. Once the virtual machine has been successfully placed and migrated to the destination host, you can see it still living and working (as shown in Figure 5.19).

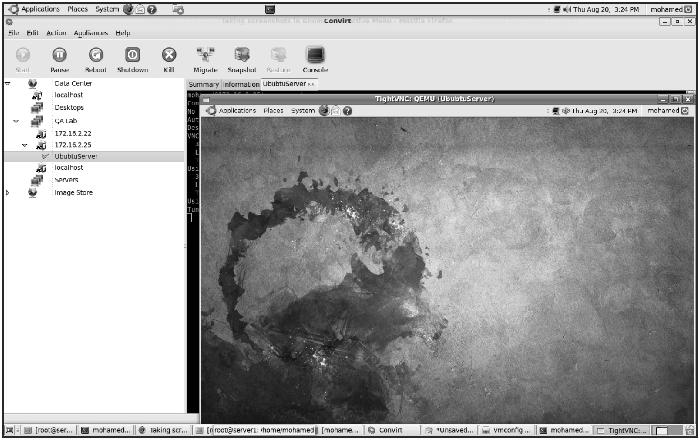


FIGURE 5.19. VM started on the destination server after migration.

**Final Thoughts about the Example**

This is just a demonstrating example of how to provision and migrate virtual machines; however, there are more tools and vendors that offer virtual infra-structure’s management like Citrix XenServer, VMware vSphere, and so on.

**PROVISIONING IN THE CLOUD CONTEXT**

In the cloud context, we shall discuss systems that provide the virtual machine provisioning and migration services; Amazon EC2 is a widely known example for vendors that provide public cloud services. Also, Eucalyptus and Open-Nebula are two complementary and enabling technologies for open source cloud tools, which play an invaluable role in infrastructure as a service and in building private, public, and hybrid cloud architecture.

Eucalyptus is a system for implementing on-premise private and hybrid clouds using the hardware and software’s infrastructure, which is in place without modification. The current interface to Eucalyptus is compatible with Amazon’s EC2, S3, and EBS interfaces, but the infrastructure is designed to support multiple client-side interfaces. Eucalyptus is implemented using commonly available Linux tools and basic Web service’s technologies [30]. Eucalyptus adds capabilities, such as end-user customization, self-service provisioning, and legacy application support to data center’s virtualization’s features, making the IT customer’s service easier. On the other hand, OpenNebula is a virtual infrastructure manager that orchestrates storage, network, and virtualization technologies to enable the dynamic placement of multi-tier services on distrib-uted infrastructures, combining both data center’s resources and remote cloud’s resources according to allocation’s policies. OpenNebula provides internal cloud administration and user’s interfaces for the full management of the cloud’s platform.

**Amazon Elastic Compute Cloud**

The Amazon EC2 (Elastic Compute Cloud) is a Web service that allows users to provision new machines into Amazon’s virtualized infrastructure in a matter of minutes; using a publicly available API (application programming interface), it reduces the time required to obtain and boot a new server. Users get full root access and can install almost any OS or application in their AMIs (Amazon Machine Images). Web services APIs allow users to reboot their instances remotely, scale capacity quickly, and use on-demand service when needed; by adding tens, or even hundreds, of machines. It is very important to mention that there is no up-front hardware setup and there are no installation costs, because Amazon charges only for the capacity you actually use.

EC2 instance is typically a virtual machine with a certain amount of RAM, CPU, and storage capacity. Setting up an EC2 instance is quite easy. Once you create your AWS (Amazon Web service) account, you can use the on-line AWS console, or simply download the offline command line’s tools to start provisioning your instances. Amazon EC2 provides its customers with three flexible purchasing models to make it easy for the cost optimization: On-Demand instances, which allow you to pay a fixed rate by the hour with no commitment. Reserved instances, which allow you to pay a low, one-time fee and in turn receive a significant discount on the hourly usage charge for that instance. It ensures that any reserved instance you launch is guaranteed to succeed (provided that you have booked them in advance). This means that users of these instances should not be affected by any transient limitations in EC2 capacity. Spot instances, which enable you to bid whatever price you want for instance capacity, providing for even greater savings, if your applications have flexible start and end times.

Amazon and Provisioning Services. Amazon provides an excellent set of tools that help in provisioning service; Amazon Auto Scaling is a set of command line tools that allows scaling Amazon EC2 capacity up or down automatically and according to the conditions the end user defines. This feature ensures that the number of Amazon EC2 instances can scale up seamlessly during demand spikes to maintain performance and can scale down automatically when loads diminish and become less intensive to minimize the costs. Auto Scaling service and Cloud Watch (a monitoring service for AWS cloud resources and their utilization) help in exposing functionalities required for provisioning application services on Amazon EC2.

Amazon Elastic Load Balancer is another service that helps in building fault-tolerant applications by automatically provisioning incoming application workload across available Amazon EC2 instances and in multiple availability zones.

**Infrastructure Enabling Technology**

Offering infrastructure as a service requires software and platforms that can manage the Infrastructure that is being shared and dynamically provisioned. For this, there are three noteworthy technologies to be considered: Eucalyptus, OpenNebula, and Aneka.

**Eucalyptus**

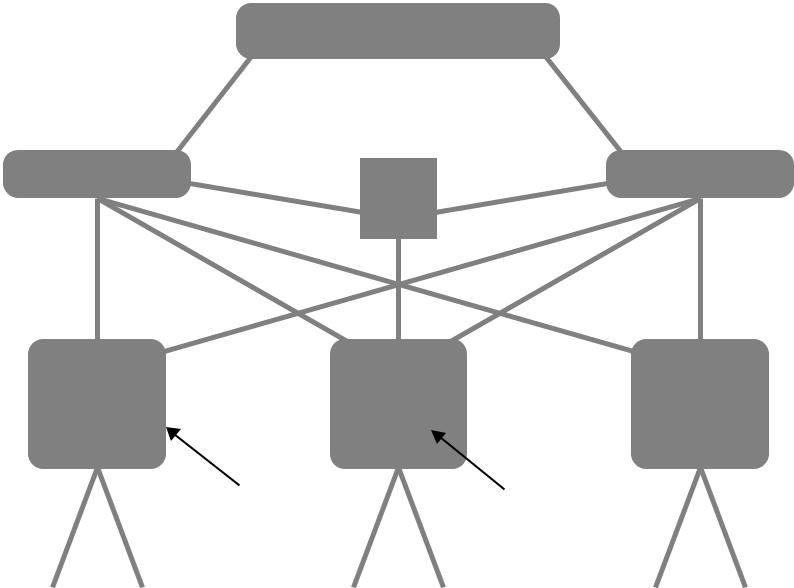
Eucalyptus is an open-source infrastructure for the implementation of cloud computing on computer clusters. It is considered one of the earliest tools developed as a surge computing (in which data center’s private cloud could augment its ability to handle workload’s spikes by a design that allows it to send overflow work to a public cloud) tool. Its name is an acronym for “elastic utility computing architecture for linking your programs to useful systems.” Here are some of the Eucalyptus features :

Interface compatibility with EC2, and S3 (both Web service and Query/ REST interfaces). Simple installation and deployment. Support for most Linux distributions (source and binary packages). Support for running VMs that run atop the Xen hypervisor or KVM. Support for other kinds of VMs, such as VMware, is targeted for future releases.

Secure internal communication using SOAP with WS security. Cloud administrator’s tool for system’s management and user’s accounting. The ability to configure multiple clusters each with private internal network addresses into a single cloud. Eucalyptus aims at fostering the research in models for service’s provisioning, scheduling, SLA formulation, and hypervisors’ portability.

Eucalyptus Architecture. Eucalyptus architecture, as illustrated in Figure 5.20, constitutes each high-level system’s component as a stand-alone Web service with the following high-level components.

Client-side Interface (via network)



Client-side API Translator

Database

Cloud Controller Walrus (S3)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Cluster Controller | |  | Storage Controller | |  |
|  |  |  |  |  | (EBS) | |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |



Node Controller

FIGURE 5.20. Eucalyptus high-level architecture.

Node controller (NC) controls the execution, inspection, and termination of VM instances on the host where it runs. Cluster controller (CC) gathers information about and schedules VM execution on specific node controllers, as well as manages virtual instance network. Storage controller (SC) is a put/get storage service that implements Amazon’s S3 interface and provides a way for storing and accessing VM images and user data. Cloud controller (CLC) is the entry point into the cloud for users and administrators. It queries node managers for information about resources, makes high-level scheduling decisions, and implements them by making requests to cluster controllers. Walrus (W) is the controller component that manages access to the storage services within Eucalyptus. Requests are communicated to Walrus using the SOAP or REST-based interface.

Its design is an open and elegant one. It can be very beneficial in testing and debugging purposes before deploying it on a real cloud. For more details about Eucalyptus architecture and design.Ubuntu Enterprise Cloud and Eucalyptus. Ubuntu Enterprise Cloud (UEC) is a new initiative by Ubuntu to make it easier to provision, deploy, configure, and use cloud infrastructures based on Eucalyptus. UEC brings Amazon EC2-like infrastructure’s capabilities inside the firewall. This is by far the simplest way to install and try Eucalyptus. Just download the Ubuntu server version and install it wherever you want. UEC is also the first open source project that lets you create cloud services in your local environment easily and leverage the power of cloud computing.

**VM Dynamic Management Using OpenNebula**

OpenNebula is an open and flexible tool that fits into existing data center’s environments to build any type of cloud deployment. OpenNebula can be primarily used as a virtualization tool to manage your virtual infrastructure, which is usually referred to as private cloud. OpenNebula supports a hybrid cloud to combine local infrastructure with public cloud-based infrastructure, enabling highly scalable hosting environments. OpenNebula also supports public clouds by providing cloud’s interfaces to expose its functionality for virtual machine, storage, and network management. OpenNebula is one of the technologies being enhanced in the Reservoir Project, European research initiatives in virtualized infrastructures, and cloud computing.

OpenNebula architecture is shown in Figure 5.21, which illustrates the existence of public and private clouds and also the resources being managed by its virtual manager. **OpenNebula** is an open-source alternative to these commercial tools for the dynamic management of VMs on distributed resources. This tool is supporting several research lines in advance reservation of capacity, probabil-istic admission control, placement optimization, resource models for the efficient management of groups of virtual machines, elasticity support, and so on. These research lines address the requirements from both types of clouds namely, private and public.

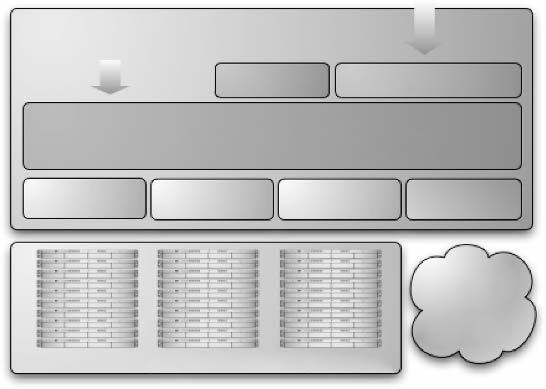
**OpenNebula and Haizea.** Haizea is an open-source virtual machine-based lease management architecture developed by Sotomayor et al; it can be used as a scheduling backend for OpenNebula. Haizea uses leases as a funda-mental resource provisioning abstraction and implements those leases as virtual machines, taking into account the overhead of using virtual machines when scheduling leases. Haizea also provides advanced functionality such as :

Advance reservation of capacity.

Best-effort scheduling with backfilling.

Cloud User

Interface



Local User and

Administrator Interface

Scheduler Cloud Service

**Virtual Infrastructure Manager**

Virtualization Storage Network Cloud

Public

Cloud

Local Infrastructure

FIGURE 5.21. OpenNebula high-level architecture.

Resource preemption (using VM suspend/resume/migrate). Policy engine, allowing developers to write pluggable scheduling policies in Python.

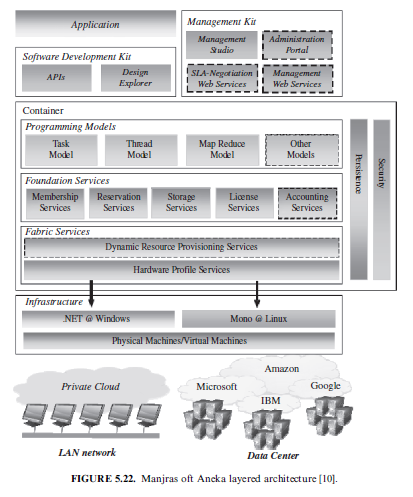
**Aneka**

Manjrasoft Aneka is a .NET-based platform and framework designed for building and deploying distributed applications on clouds. It provides a set of APIs for transparently exploiting distributed resources and expressing the business logic of applications by using the preferred programming abstractions. Aneka is also a market-oriented cloud platform since it allows users to build and schedule applications, provision resources, and monitor results using pricing, accounting, and QoS/SLA services in private and/or public cloud environments.

It allows end users to build an enterprise/private cloud setup by exploiting the power of computing resources in the enterprise data centers, public clouds such as Amazon EC2, and hybrid clouds by combining enterprise private clouds managed by Aneka with resources from Amazon EC2 or other enterprise clouds built and managed using technologies such as XenServer.

Aneka also provides support for deploying and managing clouds. By using its Management Studio and a set of Web interfaces, it is possible to set up either public or private clouds, monitor their status, update their configuration, and perform the basic management operations.

**Aneka Architecture**. Aneka platform architecture, as illustrated in Figure 5.22, consists of a collection of physical and virtualized resources connected through a network. Each of these resources hosts an instance of the Aneka container representing the runtime environment where the distributed applications are executed. The container provides the basic management features of the single node and leverages all the other operations on the services that it is hosting. The services are broken up into fabric, foundation, and execution services. Fabric services directly interact with the node through the platform abstraction layer (PAL) and perform hardware profiling and dynamic resource provisioning. Foundation services identify the core system of the Aneka middleware, providing a set of basic features to enable Aneka containers to perform specialized and specific sets of tasks. Execution services directly deal with the scheduling and execution of applications in the cloud.



On the Management of Virtual machines for Cloud Infrastructures

1. THE ANATOMY OF CLOUD INFRASTRUCTURES

There are many commercial IaaS cloud providers in the market, such as those cited earlier, and all of them share five characteristics: (i) They provide on-demand provisioning of computational resources; (ii) they use virtualization technologies to lease these resources; (iii) they provide public and simple remote interfaces to manage those resources; (iv) they use a pay-as-you-go cost model, typically charging by the hour; and (v) they operate data centers large enough to provide a seemingly unlimited amount of resources to their clients (usually touted as “infinite capacity” or “unlimited elasticity”). Private and hybrid clouds share these same characteristics but, instead of selling capacity over publicly accessible interfaces, focus on providing capacity to an organiza-tion’s internal users.

Virtualization technologies have been the key enabler of many of these salient characteristics of IaaS clouds by giving providers a more flexible and generic way of managing their resources. Thus, virtual infrastructure (VI) management—the management of virtual machines distributed across a pool of physical resources—becomes a key concern when building an IaaS cloud and poses a number of challenges. Like traditional physical resources, virtual machines require a fair amount of configuration, including preparation of the machine’s software environment and network configuration. However, in a virtual infrastructure, this configuration must be done on-the-fly, with as little time between the time the VMs are requested and the time they are available to the user. This is further complicated by the need to configure groups of VMs that will provide a specific service (e.g., an application requiring a Web server and a database server). Additionally, a virtual infrastructure manager must be capable of allocating resources efficiently, taking into account an organization’s goals (such as minimizing power consumption and other operational costs) and reacting to changes in the physical infrastructure.

Virtual infrastructure management in private clouds has to deal with an additional problem: Unlike large IaaS cloud providers, such as Amazon, private clouds typically do not have enough resources to provide the illusion of “infinite capacity.” The immediate provisioning scheme used in public clouds, where resources are provisioned at the moment they are requested, is ineffective in private clouds. Support for additional provisioning schemes, such as best-effort provisioning and advance reservations to guarantee quality of service (QoS) for applications that require resources at specific times (e.g., during known “spikes” in capacity requirements), is required. Thus, efficient resource allocation algorithms and policies and the ability to combine both private and public cloud resources, resulting in a hybrid approach, become even more important.

Several VI management solutions have emerged over time, such as platform ISF and VMware vSphere, along with open-source initiatives such as Enomaly Computing Platform and Ovirt. Many of these tools originated out of the need to manage data centers efficiently using virtual machines, before the Cloud Computing paradigm took off. However, managing virtual infra-structures in a private/hybrid cloud is a different, albeit similar, problem than managing a virtualized data center, and existing tools lack several features that are required for building IaaS clouds. Most notably, they exhibit monolithic and closed structures and can only operate, if at all, with some preconfigured placement policies, which are generally simple (round robin, first fit, etc.) and based only on CPU speed and utilization of a fixed and predetermined number of resources, such as memory and network bandwidth. This precludes extend-ing their resource management strategies with custom policies or integration with other cloud systems, or even adding cloud interfaces.

Thus, there are still several gaps in existing VI solutions. Filling these gaps will require addressing a number of research challenges over the next years, across several areas, such as virtual machine management, resource scheduling, SLAs, federation of resources, and security. In this chapter, we focus on three problems addressed by the Virtual Machine Management Activity of RESER-VOIR: distributed management of virtual machines, reservation-based provi-sioning of virtualized resource, and provisioning to meet SLA commitments.

**Distributed Management of Virtual Machines**

The first problem is how to manage the virtual infrastructures themselves. Although resource management has been extensively studied, particularly for job management in high-performance computing, managing VMs poses addi-tional problems that do not arise when managing jobs, such as the need to set up custom software environments for VMs, setting up and managing networking for interrelated VMs, and reducing the various overheads involved in using VMs. Thus, VI managers must be able to efficiently orchestrate all these different tasks.

The problem of efficiently selecting or scheduling computational resources is well known. However, the state of the art in VM-based resource scheduling follows a static approach, where resources are initially selected using a greedy allocation strategy, with minimal or no support for other placement policies. To efficiently schedule resources, VI managers must be able to support flexible and complex scheduling policies and must leverage the ability of VMs to suspend, resume, and migrate.

This complex task is one of the core problems that the RESERVOIR project tries to solve. In Section 6.2 we describe the problem of how to manage VMs distributed across a pool of physical resources and describe OpenNebula, the virtual infrastructure manager developed by the RESERVOIR project.

**Reservation-Based Provisioning of Virtualized Resources**

A particularly interesting problem when provisioning virtual infrastructures is how to deal with situations where the demand for resources is known before-hand—for example, when an experiment depending on some complex piece of equipment is going to run from 2 pm to 4 pm, and computational resources must be available at exactly that time to process the data produced by the equipment. Commercial cloud providers, such as Amazon, have enough resources to provide the illusion of infinite capacity, which means that this situation is simply resolved by requesting the resources exactly when needed; if capacity is “infinite,” then there will be resources available at 2 pm.

On the other hand, when dealing with finite capacity, a different approach is needed. However, the intuitively simple solution of reserving the resources beforehand turns out to not be so simple, because it is known to cause resources to be underutilized, due to the difficulty of scheduling other requests around an inflexible reservation.

As we discuss in Section 6.3, VMs allow us to overcome the utilization problems typically associated with advance reservations and we describe Haizea, a VM-based lease manager supporting advance reservation along with other provisioning models not supported in existing IaaS clouds, such as best-effort provisioning.

1. Provisioning to Meet SLA Commitments

IaaS clouds can be used to deploy services that will be consumed by users other than the one that deployed the services. For example, a company might depend on an IaaS cloud provider to deploy three-tier applications (Web front-end, application server, and database server) for its customers. In this case, there is a distinction between the cloud consumer (i.e., the service owner; in this case, the company that develops and manages the applications) and the end users of the resources provisioned on the cloud (i.e., the service user; in this case, the users that access the applications). Furthermore, service owners will enter into service-level agreements (SLAs) with their end users, covering guarantees such as the timeliness with which these services will respond.

However, cloud providers are typically not directly exposed to the service semantics or the SLAs that service owners may contract with their end users. The capacity requirements are, thus, less predictable and more elastic. The use of reservations may be insufficient, and capacity planning and optimiza-tions are required instead. The cloud provider’s task is, therefore, to make sure that resource allocation requests are satisfied with specific probability and timeliness. These requirements are formalized in infrastructure SLAs between the service owner and cloud provider, separate from the high-level SLAs between the service owner and its end users.

In many cases, either the service owner is not resourceful enough to perform an exact service sizing or service workloads are hard to anticipate in advance. Therefore, to protect high-level SLAs, the cloud provider should cater for elasticity on demand. We argue that scaling and de-scaling of an application is best managed by the application itself. The reason is that in many cases, resources allocation decisions are application-specific and are being driven by the application level metrics. These metrics typically do not have a universal meaning and are not observable using black box monitoring of virtual machines comprising the service.

RESERVOIR proposes a flexible framework where service owners may register service-specific elasticity rules and monitoring probes, and these rules are being executed to match environment conditions. We argue that elasti-city of the application should be contracted and formalized as part of capacity availability SLA between the cloud provider and service owner. This poses interesting research issues on the IaaS side, which can be grouped around two main topics:

SLA-oriented capacity planning that guarantees that there is enough capacity to guarantee service elasticity with minimal over-provisioning. Continuous resource placement and scheduling optimization that lowers operational costs and takes advantage of available capacity transparently to the service while keeping the service SLAs.

**DISTRIBUTED MANAGEMENT OF VIRTUAL INFRASTRUCTURES**

Managing VMs in a pool of distributed physical resources is a key concern in IaaS clouds, requiring the use of a virtual infrastructure manager. To address some of the shortcomings in existing VI solutions, we have developed the open source OpenNebula1 virtual infrastructure engine. OpenNebula is capable of managing groups of interconnected VMs with support for the Xen, KVM, and VMWare platforms within data centers and private clouds that involve a large amount of virtual and physical servers. OpenNebula can also be used to build hybrid clouds by interfacing with remote cloud sites. This section describes how OpenNebula models and manages VMs in a virtual infrastructure.

**VM Model and Life Cycle**

The primary target of OpenNebula is to manage VMs. Within OpenNebula, a VM is modeled as having the following attributes:

A capacity in terms of memory and CPU.

A set of NICs attached to one or more virtual networks.

A set of disk images. In general it might be necessary to transfer some of these image files to/from the physical machine the VM will be running in.

A state file (optional) or recovery file that contains the memory image of a running VM plus some hypervisor-specific information.

The life cycle of a VM within OpenNebula follows several stages: Resource Selection. Once a VM is requested to OpenNebula, a feasible placement plan for the VM must be made. OpenNebula’s default scheduler provides an implementation of a rank scheduling policy, allowing site administrators to configure the scheduler to prioritize the resources that are more suitable for the VM, using information from the VMs and the physical hosts. As we will describe in Section 6.3, OpenNebula can also use the Haizea lease manager to support more complex scheduling policies.

Resource Preparation. The disk images of the VM are transferred to the target physical resource. During the boot process, the VM is contextualized, a process where the disk images are specialized to work in a given environment. For example, if the VM is part of a group of VMs offering a service (a compute cluster, a DB-based application, etc.), contextualization could involve setting up the network and the machine hostname, or registering the new VM with a service (e.g., the head node in a compute cluster). Different techniques are available to contextualize a worker node, including use of an automatic installation system (for instance, Puppet or Quattor), a context server (see reference 15), or access to a disk image with the context data for the worker node (OVF recommendation). VM Creation. The VM is booted by the resource hypervisor. VM Migration. The VM potentially gets migrated to a more suitable resource (e.g., to optimize the power consumption of the physical resources).

VM Termination. When the VM is going to shut down, OpenNebula can transfer back its disk images to a known location. This way, changes in the VM can be kept for a future use.

**VM Management**

OpenNebula manages a VMs life cycle by orchestrating three different management areas: virtualization by interfacing with a physical resource’s hypervisor, such as Xen, KVM, or VMWare, to control (e.g., boot, stop, or shutdown) the VM; image management by transferring the VM images from an image repository to the selected resource and by creating on-the-fly temporary images; and networking by creating local area networks (LAN) to interconnect the VMs and tracking the MAC addresses leased in each network.

Virtualization. OpenNebula manages VMs by interfacing with the physical resource virtualization technology (e.g., Xen or KVM) using a set of pluggable drivers that decouple the managing process from the underlying technology. Thus, whenever the core needs to manage a VM, it uses high-level commands such as “start VM,” “stop VM,” and so on, which are translated by the drivers into commands that the virtual machine manager can understand. By decou-pling the OpenNebula core from the virtualization technologies through the use of a driver-based architecture, adding support for additional virtual machine managers only requires writing a driver for it.

Image Management. VMs are supported by a set of virtual disks or images, which contains the OS and any other additional software needed by the VM. OpenNebula assumes that there is an image repository that can be any storage medium or service, local or remote, that holds the base image of the VMs. There are a number of different possible configurations depending on the user’s needs. For example, users may want all their images placed on a separate repository with only HTTP access. Alternatively, images can be shared through NFS between all the hosts. OpenNebula aims to be flexible enough to support as many different image management configurations as possible.

OpenNebula uses the following concepts for its image management model (Figure 6.1):

Image Repositories refer to any storage medium, local or remote, that hold the base images of the VMs. An image repository can be a dedicated file server or a remote URL from an appliance provider, but they need to be accessible from the OpenNebula front-end.

Virtual Machine Directory is a directory on the cluster node where a VM is running. This directory holds all deployment files for the hypervisor to boot the machine, checkpoints, and images being used or saved—all of them specific to that VM. This directory should be shared for most hypervisors to be able to perform live migrations. Any given VM image goes through the following steps along its life cycle:

Preparation implies all the necessary changes to be made to the machine’s image so it is prepared to offer the service to which it is intended. OpenNebula assumes that the images that conform to a particular VM are prepared and placed in the accessible image repository. Cloning the image means taking the image from the repository and placing it in the VM’s directory in the physical node where it is going to be run before the VM is actually booted. If a VM image is to be cloned, the original image is not going to be used, and thus a copy will be used. There is a qualifier (clone) for the images that can mark them as targeting for cloning or not.

Save/remove. If the save qualifier is disabled, once the VM has been shut down, the images and all the changes thereof are going to be disposed of. However, if the save qualifier is activated, the image will be saved for later use.

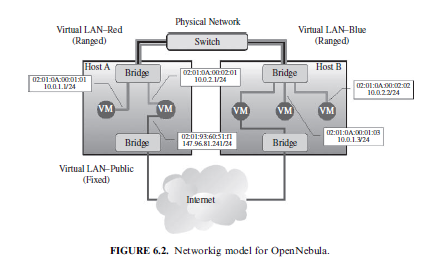
Networking. In general, services deployed on a cloud, from a computing cluster to the classical three-tier business application, require several inter-related VMs, with a virtual application network (VAN) being the primary link between them. OpenNebula dynamically creates these VANs and tracks the MAC addresses leased in the network to the service VMs. Note that here we refer to layer 2 LANs; other TCP/IP services such as DNS, NIS, or NFS are the responsibility of the service (i.e., the service VMs have to be configured to provide such services).

The physical hosts that will co-form the fabric of our virtual infrastructures will need to have some constraints in order to effectively deliver virtual networks to our virtual machines. Therefore, from the point of view of networking, we can define our physical cluster as a set of hosts with one or more network interfaces, each of them connected to a different physical network.

We can see in Figure 6.2 two physical hosts with two network interfaces each; thus there are two different physical networks. There is one physical network that connects the two hosts using a switch, and there is another one that gives the hosts access to the public Internet. This is one possible configuration for the physical cluster, and it is the one we recommend since it can be used to make both private and public VANs for the virtual machines. Moving up to the virtualization layer, we can distinguish three different VANs. One is mapped on top of the public Internet network, and we can see a couple of virtual machines taking advantage of it. Therefore, these two VMs will have access to the Internet. The other two are mapped on top of the private physical network: the Red and Blue VANs. Virtual machines connected to the same private VAN will be able to communicate with each other, otherwise they will be isolated and won’t be able to communicate.

1. Further Reading on OpenNebula

There are a number of scholarly publications that describe the design and architecture of OpenNebula in more detail, including papers showing perfor-mance results obtained when using OpenNebula to deploy and manage the back-end nodes of a Sun Grid Engine compute cluster and of a NGINX Web server on both local resources and an external cloud. The Open-Nebula virtual infrastructure engine is also available for download at http:// www.opennebula.org/, which provides abundant documentation not just on how to install and use OpenNebula, but also on its internal architecture.



**SCHEDULING TECHNIQUES FOR ADVANCE RESERVATION OF CAPACITY**

While a VI manager like OpenNebula can handle all the minutiae of managing VMs in a pool of physical resources, scheduling these VMs efficiently is a different and complex matter. Commercial cloud providers, such as Amazon, rely on an immediate provisioning model where VMs are provisioned right away, since their data centers’ capacity is assumed to be infinite. Thus, there is no need for other provisioning models, such as best-effort provisioning where requests have to be queued and prioritized or advance provisioning where resources are pre-reserved so they will be guaranteed to be available at a given time period; queuing and reservations are unnecessary when resources are always available to satisfy incoming requests.

However, when managing a private cloud with limited resources, an immediate provisioning model is insufficient. In this section we describe a lease-based resource provisioning model used by the Haizea2 lease manager, which can be used as a scheduling back-end by OpenNebula to support provisioning models not supported in other VI management solutions. We focus, in particular, on advance reservation of capacity in IaaS clouds as a way to guarantee availability of resources at a time specified by the user.

**Existing Approaches to Capacity Reservation**

Efficient reservation of resources in resource management systems has been studied considerably, particularly in the context of job scheduling. In fact, most modern job schedulers support advance reservation of resources, but their implementation falls short in several aspects. First of all, they are constrained by the job abstraction; when a user makes an advance reservation in a job-based system, the user does not have direct and unfettered access to the resources, the way a cloud users can access the VMs they requested, but, rather, is only allowed to submit jobs to them. For example, PBS Pro creates a new queue that will be bound to the reserved resources, guaranteeing that jobs submitted to that queue will be executed on them (assuming they have permission to do so). Maui and Moab, on the other hand, simply allow users to specify that a submitted job should use the reserved resources (if the submitting user has permission to do so). There are no mechanisms to directly login to the reserved resources, other than through an interactive job, which does not provide unfettered access to the resources.

Additionally, it is well known that advance reservations lead to utilization

problems, caused by the need to vacate resources before a reservation

can begin. Unlike future reservations made by backfilling algorithms, where the

start of the reservation is determined on a best-effort basis, advance reservations introduce roadblocks in the resource schedule. Thus, traditional job schedulers are unable to efficiently schedule workloads combining both best-effort jobs and advance reservations.

However, advance reservations can be supported more efficiently by using a scheduler capable of preempting running jobs at the start of the reservation and resuming them at the end of the reservation. Preemption can also be used to run large parallel jobs (which tend to have long queue times) earlier, and it is specially relevant in the context of urgent computing, where resources have to be provisioned on very short notice and the likelihood of having jobs already assigned to resources is higher. While preemption can be accomplished trivially by canceling a running job, the least disruptive form of preemption is checkpointing, where the preempted job’s entire state is saved to disk, allowing it to resume its work from the last checkpoint. Additionally, some schedulers also support job migration, allowing checkpointed jobs to restart on other available resources, instead of having to wait until the preempting job or reservation has completed.

However, although many modern schedulers support at least checkpointing-based preemption, this requires the job’s executable itself to be checkpointable. An application can be made checkpointable by explicitly adding that function-ality to an application (application-level and library-level checkpointing) or transparently by using OS-level checkpointing, where the operating system (such as Cray, IRIX, and patched versions of Linux using BLCR ) checkpoints a process, without rewriting the program or relinking it with checkpointing libraries. However, this requires a checkpointing-capable OS to be available.

Thus, a job scheduler capable of checkpointing-based preemption and migration could be used to checkpoint jobs before the start of an advance reservation, minimizing their impact on the schedule. However, the application-and library-level checkpointing approaches burden the user with having to modify their applications to make them checkpointable, imposing a restriction on the software environment. OS-level checkpointing, on the other hand, is a more appealing option, but still imposes certain software restrictions on resource consumers. Systems like Cray and IRIX still require applications to be compiled for their respective architectures, which would only allow a small fraction of existing applications to be supported within leases, or would require existing applications to be ported to these architectures. This is an excessive restriction on users, given the large number of clusters and applications that depend on the x86 architecture. Although the BLCR project does provide a checkpointing x86 Linux kernel, this kernel still has several limitations, such as not being able to properly checkpoint network traffic and not being able to checkpoint MPI applications unless they are linked with BLCR-aware MPI libraries.

An alternative approach to supporting advance reservations was propo-sed by Nurmi et al., which introduced “virtual advance reservations for queues” (VARQ). This approach overlays advance reservations over traditional job schedulers by first predicting the time a job would spend waiting in a scheduler’s queue and then submitting a job (representing the advance reservation) at a time such that, based on the wait time prediction, the probability that it will be running at the start of the reservation is maximized. Since no actual reservations can be done, VARQ jobs can run on traditional job schedulers, which will not distinguish between the regular best-effort jobs and the VARQ jobs. Although this is an interesting approach that can be realistically implemented in practice (since it does not require modifications to existing scheduler), it still depends on the job abstraction.

Hovestadt et al. proposed a planning-based (as opposed to queuing-based) approach to job scheduling, where job requests are immediately planned by making a reservation (now or in the future), instead of waiting in a queue. Thus, advance reservations are implicitly supported by a planning-based system. Additionally, each time a new request is received, the entire schedule is reevaluated to optimize resource usage. For example, a request for an advance reservation can be accepted without using preemption, since the jobs that were originally assigned to those resources can be assigned to different resources (assuming the jobs were not already running).

**Reservations with VMs**

As we described earlier, virtualization technologies are a key enabler of many features found in IaaS clouds. Virtual machines are also an appealing vehicle for implementing efficient reservation of resources due to their ability to be suspended, potentially migrated, and resumed without modifying any of the applications running inside the VM. However, virtual machines also raise additional challenges related to the overhead of using VMs:

Preparation Overhead. When using VMs to implement reservations, a VM disk image must be either prepared on-the-fly or transferred to the physical node where it is needed. Since a VM disk image can have a size in the order of gigabytes, this preparation overhead can significantly delay the starting time of leases. This delay may, in some cases, be unacceptable for advance reservations that must start at a specific time.

Runtime Overhead. Once a VM is running, scheduling primitives such as checkpointing and resuming can incur in significant overhead since a VM’s entire memory space must be saved to disk, and then read from disk. Migration involves transferring this saved memory along with the VM disk image. Similar to deployment overhead, this overhead can result in noticeable delays.

The Haizea project (http://haizea.cs.uchicago.edu/) was created to develop a scheduler that can efficiently support advance reservations efficiently by using the suspend/resume/migrate capability of VMs, but minimizing the overhead of using VMs. The fundamental resource provisioning abstraction in Haizea is the lease, with three types of lease currently supported:

Advanced reservation leases, where the resources must be available at a specific time.

Best-effort leases, where resources are provisioned as soon as possible and requests are placed on a queue if necessary.

Immediate leases, where resources are provisioned when requested or not at all.

The Haizea lease manager can be used as a scheduling back-end for the OpenNebula virtual infrastructure engine, allowing it to support these three types of leases. The remainder of this section describes Haizea’s leasing model and the algorithms Haizea uses to schedule these leases.

**Leasing Model**

We define a lease as “a negotiated and renegotiable agreement between a resource provider and a resource consumer, where the former agrees to make a set of resources available to the latter, based on a set of lease terms presented by the resource consumer.” The terms must encompass the following: the hardware resources required by the resource consumer, such as CPUs, memory, and network bandwidth; a software environment required on the leased resources; and an availability period during which a user requests that the hardware and software resources be available. Since previous work and other authors already explore lease terms for hardware resources and software environments, our focus has been on the availability dimension of a lease and, in particular, on how to efficiently support advance reservations.

Thus, we consider the following availability terms:

Start time may be unspecified (a best-effort lease) or specified (an advance reservation lease). In the latter case, the user may specify either a specific start time or a time period during which the lease start may occur.

Maximum duration refers to the total maximum amount of time that the leased resources will be available.

Leases can be preemptable. A preemptable lease can be safely paused without disrupting the computation that takes place inside the lease.

Haizea’s resource model considers that it manages W physical nodes capable of running virtual machines. Each node i has CPUs, megabytes (MB) of memory, and MB of local disk storage. We assume that all disk images required to run virtual machines are available in a repository from which they can be transferred to nodes as needed and that all are connected at a bandwidth of B MB/sec by a switched network.

A lease is implemented as a set of N VMs, each allocated resources described by a tuple (p, m, d, b), where p is number of CPUs, m is memory in MB, d is disk space in MB, and b is network bandwidth in MB/sec. A disk image I with a size of size(I) MB must be transferred from the repository to a node before the VM can start. When transferring a disk image to multiple nodes, we use multi-casting and model the transfer time as size(I)/B. If a lease is preempted, it is suspended by suspending its VMs, which may then be either resumed on the same node or migrated to another node and resumed there. Suspending a VM results in a memory state image file (of size m that can be saved to either a local filesystem or a global filesystem (f A {local, global}). Resumption requires reading that image back into memory and then discarding the file. Suspension of a single VM is done at a rate of s megabytes of VM memory per second, and we define r similarly for VM resumption.

**Lease Scheduling**

Haizea is designed to process lease requests and determine how those requests can be mapped to virtual machines, leveraging their suspend/resume/migrate capability, in such a way that the leases’ requirements are satisfied. The scheduling component of Haizea uses classical backfilling algorithms, extended to allow best-effort leases to be preempted if resources have to be freed up for advance reservation requests. Additionally, to address the pre-paration and runtime overheads mentioned earlier, the scheduler allocates resources explicitly for the overhead activities (such as transferring disk images or suspending VMs) instead of assuming they should be deducted from the lease’s allocation. Besides guaranteeing that certain operations complete on time (e.g., an image transfer before the start of a lease), the scheduler also attempts to minimize this overhead whenever possible, most notably by reusing disk image transfers and caching disk images on the physical nodes.

Best-effort leases are scheduled using a queue. When a best-effort lease is requested, the lease request is placed at the end of the queue, which is periodically evaluated using a backfilling algorithm—both aggressive and conservative backfilling strategies are supported—to determine if any leases can be scheduled. The scheduler does this by first checking the earliest possible starting time for the lease on each physical node, which will depend on the required disk images. For example, if some physical nodes have cached the required disk image, it will be possible to start the lease earlier on those nodes. Once these earliest starting times have been determined, the scheduler chooses the nodes that allow the lease to start soonest.

The use of VM suspension/resumption allows the best-effort leases to be scheduled even if there are not enough resources available for their full requested duration. If there is a “blocking” lease in the future, such as an advance reservation lease that would prevent the best-effort lease to run to completion before the blocking lease starts, the best-effort lease can still be scheduled; the VMs in the best-effort lease will simply be suspended before a blocking lease. The remainder of a suspended lease is placed in the queue, according to its submission time, and is scheduled like a regular best-effort lease (except a resumption operation, and potentially a migration operation, will have to be scheduled too).

Advance reservations, on the other hand, do not go through a queue, since they must start at either the requested time or not at all. Thus, scheduling this type of lease is relatively simple, because it mostly involves checking if there are enough resources available during the requested interval. However, the scheduler must also check if any associated overheads can be scheduled in such a way that the lease can still start on time. For preparation overhead, the scheduler determines if the required images can be transferred on time. These transfers are scheduled using an earliest deadline first (EDF) algorithm, where the deadline for the image transfer is the start time of the advance reservation lease. Since the start time of an advance reservation lease may occur long after the lease request, we modify the basic EDF algorithm so that transfers take place as close as possible to the deadline, preventing images from unnecessarily consuming disk space before the lease starts. For runtime over-head, the scheduler will attempt to schedule the lease without having to preempt other leases; if preemption is unavoidable, the necessary suspension operations are scheduled if they can be performed on time.

For both types of leases, Haizea supports pluggable policies, allowing system administrators to write their own scheduling policies without having to modify Haizea’s source code. Currently, three policies are pluggable in Haizea: determining whether a lease is accepted or not, the selection of physical nodes, and determining whether a lease can preempt another lease.

Our main results so far have shown that, when using workloads that combine best-effort and advance reservation lease requests, a VM-based approach with suspend/resume/migrate can overcome the utilization pro-blems typically associated with the use of advance reservations. Even in the presence of the runtime overhead resulting from using VMs, a VM-based approach results in consistently better total execution time than a scheduler that does not support task preemption, along with only slightly worse performance than a scheduler that does support task preemption. Measuring the wait time and slowdown of best-effort leases shows that, although the average values of these metrics increase when using VMs, this effect is due to short leases not being preferentially selected by Haizea’s backfilling algorithm, instead of allowing best effort leases to run as long as possible before a preempting AR lease (and being suspended right before the start of the AR). In effect, a VM-based approach does not favor leases of a particular length over others, unlike systems that rely more heavily on backfilling. Our results have also shown that, although supporting the deployment of multiple software environments, in the form of multiple VM images, requires the transfer of potentially large disk image files, this deployment overhead can be minimized through the use of image transfer scheduling and caching strategies.

**CAPACITY MANAGEMENT TO MEET SLA COMMITMENTS**

As was discussed in the previous section, when temporal behavior of services with respect to resource demands is highly predictable (e.g., thanks to well-known business cycle of a service, or predictable job lengths in computational service), capacity can be efficiently scheduled using reservations. In this section we focus on less predictable elastic workloads. For these workloads, exact scheduling of capacity may not be possible. Rather than that, capacity planning and optimizations are required.

IaaS providers perform two complementary management tasks: (1) capacity planning to make sure that SLA obligations are met as contracted with the service providers and (2) continuous optimization of resource utilization given specific workload to make the most efficient use of the existing capacity. It is worthy to emphasize the rationale behind these two management processes.

The first task pertains to the long-term capacity management aimed at cost-efficient provisioning in accordance with contracted SLAs. To protect SLAs with end users, elastic services scale up and down dynamically. This requires an IaaS provider to guarantee elasticity for the service within some contracted capacity ranges. Thus, the IaaS provider should plan capacity of the cloud in such a way that when services change resource demands in response to environment conditions, the resources will be indeed provided with the contracted probability. At the same time, the IaaS cloud provider strives to minimally over-provision capacity, thus minimizing the operational costs. We observe that these goals can be harmonized thanks to statistical multiplexing of elastic capacity demands. The key questions will be (a) in what form to provide capacity guarantees (i.e., infrastructure SLAs) and (b) how to control the risks inherent to over subscribing. We treat these problems in Sections 6.4.1 and 6.4.2, respectively.

The second task pertains to short- and medium-term optimization of resource allocation under the current workload. This optimization may be guided by different management policies that support high level business goals of an IaaS provider. We discuss policy-driven continuous resource optimization in Section 6.4.3.

From an architectural viewpoint, we argue in favor of a resource manage-ment framework that separates between these two activities and allows combination of solutions to each process, which are best adapted to the needs of a specific IaaS provider.

**Infrastructure SLAs**

IaaS can be regarded as a giant virtual hardware store, where computational resources such as virtual machines (VM), virtual application networks (VAN) and virtual disks (VD) can be ordered on demand in the matter of minutes or even seconds. Virtualization technology is sufficiently versatile to provide virtual resources on a almost continuous granularity scale. Chandra et al. quantitatively study advantages of fine-grain resource allocation in a shared hosting platform. As this research suggests, fine-grain temporal and spatial resource allocation may lead to substantial improvements in capacity utilization.

These advantages come at a cost of increased management, accounting, and billing overhead. For this reason, in practice, resources are typically provided on a more coarse discrete scale. For example, Amazon EC2 offers small, large, and extra large general purpose VM instances and high CPU medium and extra large instances. It is possible that more instance types (e.g., I/O high, memory high, storage high, etc.) will be added in the future should a demand for them arise. Other IaaS providers for example, GoGrid and FlexiScale follow similar strategy.

With some caution it may be predicted that this approach, as being considerably more simple management wise, will remain prevalent in short to medium term in the IaaS cloud offerings.

Thus, to deploy a service on a cloud, service provider orders suitable virtual hardware and installs its application software on it. From the IaaS provider, a given service configuration is a virtual resource array of black box resources, which correspond to the number of instances of resource type. For example, a typical three-tier application may contain 10 general-purpose small instances to run Web front-ends, three large instances to run an application server cluster with load balancing and redundancy, and two large instances to run a replicated database.

In an IaaS model it is expected from the service provider that it sizes capacity demands for its service. If resource demands are provided correctly and are indeed satisfied upon request, then desired user experience of the service will be guaranteed. A risk mitigation mechanism to protect user experience in the IaaS model is offered by infrastructure SLAs (i.e., the SLAs formalizing capacity availability) signed between service provider and IaaS provider.

There is no universal approach to infrastructure SLAs. As the IaaS field matures and more experience is being gained, some methodologies may become more popular than others. Also some methods may be more suitable for specific workloads than other. There are three main approaches as follows.

No SLAs. This approach is based on two premises: (a) Cloud always has spare capacity to provide on demand, and (b) services are not QoS-sensitive and can withstand moderate performance degradation. This methodology is best suited for the best effort workloads. Probabilistic SLAs. These SLAs allow us to trade capacity availability for cost of consumption. Probabilistic SLAs specify clauses that determine availability percentile for contracted resources computed over the SLA evaluation period. The lower the availability percentile, the cheaper the cost of resource consumption. This is justified by the fact that an IaaS provider has less stringent commitments and can over-subscribe capacity to maximize yield without exposing itself to excessive risk. This type of SLA is suitable for small and medium businesses and for many enterprise grade applications.

Deterministic SLAs. These are, in fact, probabilistic SLAs where resource availability percentile is 100%. These SLAs are most stringent and difficult to guarantee. From the provider’s point of view, they do not admit capacity multiplexing. Therefore this is the most costly option for service providers, which may be applied for critical services.

We envision coexistence of all three methodologies above, where each SLA type is most applicable to specific workload type. We will focus on probabilistic SLAs, however, because they represent the more interesting and flexible option and lay the foundation for the rest of discussion on statistical multiplexing of capacity in Section 6.4.2. But before we can proceed, we need to define one more concept, elasticity rules.

Elasticity rules are scaling and de-scaling policies that guide transition of the service from one configuration to another to match changes in the environ-ment. The main motivation for defining these policies stems from the pay-as-you-go billing model of IaaS clouds. The service owner is interested in paying only for what is really required to satisfy workload demands minimizing the over-provisioning overhead.

There are three types of elasticity rules:

Time-driven: These rules change the virtual resources array in response to a timer event. These rules are useful for predictable workloads—for example, for services with well-known business cycles.

OS Level Metrics-Driven: These rules react on predicates defined in terms of the OS parameters observable in the black box mode (see Amazon Auto-scaling Service). These auto-scaling policies are useful for transpar-ently scaling and de-scaling services. The problem is, however, that in many cases this mechanism is not precise enough.

Application Metrics-Driven. This is a unique RESERVOIR offering that allows an application to supply application-specific policies that will be transparently executed by IaaS middleware in reacting on the monitoring information supplied by the service-specific monitoring probes running inside VMs.

For a single service, elasticity rules of all three types can be defined, resulting in a complex dynamic behavior of a service during runtime. To protect elasticity rules of a service while increasing the multiplexing gain, RESER-VOIR proposes using probabilistic infrastructure availability SLAs.

Assuming that a business day is divided into a number of usage windows, the generic template for probabilistic infrastructure SLAs is as follows.

For each Wi, and each resource type rj from the virtual resource array, capacity range C 5 (r jmin, rjmax) is available for the service with probability pi.

Probabilistically guaranteeing capacity ranges allows service providers to define its needs flexibly. For example, for business critical usage window, availability percentile may be higher than for the regular or off-peak hours. Similarly, capacity ranges may vary in size. From the provider’s point of view, defining capacity requirements this way allows yield maximization through over-subscribing. This creates a win win situation for both service provider and IaaS provider.

**Policy-Driven Probabilistic Admission Control**

Benefits of statistical multiplexing are well known. This is an extensively studied field, especially in computer networking. In the context of CPU and bandwidth allocation in shared hosting platforms, the problem was recently studied by Urgaonkar et al. . In this work the resources were treated as contiguous, allowing infinitesimal capacity allocation. We general-ize this approach by means of treating each (number of instances of resource i in the virtual resources array) as a random variable. The virtual resources array is, therefore, a vector of random variables. Since we assume that each capacity range for each resource type is finite, we may compute both the average resource consumption rate and variance in resource consump-tion for each service in terms of the capacity units corresponding to each resource type.

Inspired by the approach of Guerin et al., we propose a simple management lever termed acceptable risk level (ARL) to control over-subscrib-ing of capacity. We define ARL as the probability of having insufficient capacity to satisfy some capacity allocation requests on demand. The ARL value can be derived from a business policy of the IaaS provider—that is, more aggressive versus more conservative over-subscription.

In general, the optimal ARL value can be obtained by calculating the residual benefit resulting from specific SLA violations. A more conservative, suboptimal ARL value is simply the complement of the most stringent capacity range availability percentile across the SLA portfolio.

An infrastructure SLA commitment for the new application service should be made if and only if the potential effect does not cause the residual benefit to fall below some predefined level, being controlled by the site’s business policy. This decision process is referred to as BSM-aligned admission control.3

Once a service application passes admission control successfully, optimal placement should be found for the virtual resources comprising the service. We treat this issue in Section 6.4.3.

The admission control algorithm calculates equivalent capacity required to satisfy the resource demands of the service applications for the given ARL. The equivalent capacity is then matched against the actual available capacity to verify whether it is safe to admit a new service.

In a federated environment (like that provided by RESERVOIR) there is potentially an infinite pool of resources. However, these resources should fit placement constraints that are posed by the service applications and should be reserved using inter-cloud framework agreements. Thus, the BSM-aligned admission control helps the capacity planning process to dimension capacity requests from the partner clouds and fulfill physical capacity requests at the local cloud.

The capacity demands of the deployed application services are being continuously monitored. For each application service, the mean capacity demand (in capacity units) and the standard deviation of the capacity demand are being calculated.

When a new service with unknown history arrives in the system, its mean capacity demand and standard deviation are conservatively estimated from the service elasticity rules and historic data known for other services. Then, an equivalent capacity is approximated using Eq. (6.1). The equivalent capacity is the physical capacity needed to host the new service and all previously deployed services without increasing the probability of congestion (acceptable risk level), ε.

Equivalent capacity is expressed in the form of resource array, where each element represents the number of instances of a resource of a specific type.4 To verify that physical capacity is sufficient to support the needed equivalent capacity, one may use either the efficient and scalable exact solution (via branch and bound algorithms) to the multiple knapsack problem or the efficient bin-packing approximation algorithm such as First-Fit-Descending, which guarantees approximation ratio within 22% of the optimal algorithm. Using multiple knapsacks is more appropriate when capacity augmentation is not an option. Assuming that value of the resources is proportional to their size, solving the multiple knapsack problem provides a good estimation of value resulting from packing the virtual resources on the given capacity. If capacity can be augmented—for example, more physical capacity can be obtained from a partner cloud provider or procured locally—then solving the bin packing problem is more appropriate since all items (i.e., resources comprising the service) are always packed.

4When calculating equivalent capacity, we do not know which service will use specific resource instances, but we know that it is sufficient, say, to be able to allocate up to 100 small VM instances and 50 large instances to guarantee all resource requests resulting from the elasticity rules application, so that congestion in resource allocation will not happen with probability larger than ε.

Note that this is different from computing the actual placement of services since at the admission control stage we have “abstract” equivalent capacity. Matching equivalent capacity against physical capacity, as above, guarantees that feasible placement for actual services can be found with probability 1 2 ε.

If the local and remote physical capacity that can be used by this site in a guaranteed manner is sufficient to support the equivalent capacity calculated, the new service is accepted. Otherwise, a number of possibilities exist, depend-ing on the management policy:

The service is rejected.

The total capacity of the site is increased locally and/or remotely (through federation) by the amount needed to satisfy the equivalent capacity constraint and the service is admitted.

The acceptable risk level is increased, and the service is accepted.

Our approach initially overestimates the average capacity demand for the new service. With the passage of time, however, as capacity usage statistics are being collected for the newly admitted application service, the mean and standard deviation for the capacity demands (per resource type) are adjusted for this service. This allows us to reduce the conservativeness when the next service arrives.

Service providers may impose various placement restrictions on VMs comprising the service. For example, it may be required that VMs do not share the same physical host (anti-affinity). As another example, consider heterogeneous physical infrastructure and placement constraints arising from technological incompatibilities.

From the admission control algorithm’s vantage point, the problem is that during admission control it may not know which deployment restrictions should be taken into account since which restrictions will be of relevance depends on the dynamic behavior of the services.

In general, to guarantee that a feasible placement for virtual resources will be found with controllable probability in the presence of placement restrictions, resource augmentation is required. The resource augmentation may be quite significant (see references 34 and 35). It is, therefore, prudent on the side of the IaaS provider to segregate workloads that admit full sharing of the infrastru-cture from those who do not and offer service provider-controlled deployment restrictions as a premium service to recover capacity augmentation costs.

**Policy-Driven Placement Optimization**

The purpose of statistical admission control is to guarantee that there is enough capacity to find a feasible placement with given probability. Policy-driven placement optimization complements capacity planning and management by improving a given mapping of physical to virtual resources (e.g., VMs).

In the presence of deployment restrictions, efficient capacity planning with guaranteed minimal over-provisioning is still an open research problem. Partially the difficulties lie in hardness of solving multiple knapsacks or its more general version, the generalized assignment problem. Both problems are NP-hard in the strong sense (see discussion in Section 6.4.5). In the RESER-VOIR model, where resource augmentation is possible through cloud partner-ship, solutions that may require doubling of existing local capacity in the worst case are applicable. An interesting line of research is to approximate capacity augmentation introduced by specific constraints, such as bin item and item item. Based on required augmentation, an IaaS provider may either accept or reject the service.

As shown in reference 36, in the presence of placement constraints of type bin item, Bi-criteria Multiple Knapsack with Assignment Restrictions (BMKAR) that maximizes the total profit of placed items (subject to a lower bound) and minimizes the total number of containers (i.e., minimizes utilized capacity) does not admit a polynomial algorithm that satisfies the lower bound exactly unless P 5 NP. Two approximation algorithms with performance ratios (running in pseudo-polynomial time) and (running in polynomial time) were presented. These results are best known today for BMKAR, and the bounds are tight.

In our current prototypical placement solution, we formulated the problem as an Integer Linear Programming problem and used branch-and-bound solver (COIN-CBC) to solve the problem exactly. This serves us as a performance baseline for future research. As was shown by Pisinger, in the absence of constraints, very large problem instances can be solved exactly in a very efficient manner using a branch-and-bound algorithm. Obviously, as the scale

of the problem (in terms of constraints) increases, ILP becomes infeasible. This leads us to focus on developing novel heuristic algorithms extending the state of art, which is discussed in Section 6.4.5.

A number of important aspects should be taken into account in efficient placement optimization.

Penalization for Nonplacement. In BMKAR, as in all classical knapsack problems, no-placement of an item results in 0 profit for that item. In the VM placement with SLA protection problem, nonplacement of an item or a group of items may result in SLA violation and, thus, payment of penalty. The management policy to minimize nonplacements is factored into constraints and an objective function.

Selection Constraints. Selection constraints imply that only when a group of VMs (items) collectively forming a service is placed, this meta-item yields profit. Partial placement may even lead to a penalty, since the SLA of a service may be violated. Thus, partial placement should be prevented. In our formulation, this is factored into constraints.

Repeated Solution. Since the placement problem is solved continuously, it is important to minimize the cost of replacement. In particular, we need to minimize the cost of reassignments of VMs to hosts, because this entails VM migrations. We factor the penalty member on migration in our objective function.

Considering ICT-Level Management Policies. There are three policies that we currently consider: power conservation (by minimizing the number of physical hosts used for placement), load balancing (by spreading load across available physical machines), and migration minimization (by introducing a penalty factor for machines migration). We discuss policies below. In general, RESERVOIR provides an open-ended engine that allows to incorporate different policies. Depending on the policy chosen, the optimization problem is cast into a specific form. Currently, we support two placement policies: “load balancing” and “power con-servation,” with number of migrations minimized in both cases. The first policy is attained through solving GAP with conflicts, and the second one is implemented via bin packing with conflicts.

Inspired by results by Santos et al., who cast infrastructure-level management policies as soft constraints, we factor the load balancing policy into our model using the soft constraints approach.

We exploit the idea that reducing the available capacity at each physical host will force the search for an optimal solution to spread the VEEs over a larger number of knapsacks, thus causing the load to be spread more evenly across the site.

To address power conservation objective as a management policy, we formulate our problem as bin-packing with conflicts.

Since the optimization policy for VEE placement is being continuously solved, it is critical to minimize VEE migrations in order to maintain cost-effectiveness.

Management Policies and Management Goals. Policy-based management is an overused term. Therefore, it is, beneficial to define and differentiate our approach to policy-driven admission control and placement optimization in the more precise terms.

Policy-driven management is a management approach based on “if(con-dition) then(action)” rules defined to deal with the situations that are likely to arise. These policies serve as a basic building blocks for autonomic computing.

The overall optimality criteria of placement, however, are controlled by the management policies, which are defined at a higher level of abstraction than “if (condition) then(action)” rules. To avoid ambiguity, we term these policies management goals. Management goals, such as “conserve power,” “prefer local resources over remote resources,” “balance workload,” “minimize VM migra-tions,” “minimize SLA noncompliance,” and so forth, have complex logical structures. They cannot be trivially expressed by “if(condition) then(action)” rules even though it is possible to create the elementary rules that will strive to satisfy global management preferences in a reactive or proactive manner.

Regarding the management activity involved in VM placement opti-mization, a two-phase approach can be used. In the first phase, a feasible placement—that is, a placement that satisfies the hard constraints imposed by the service manifest—can be obtained without concerns for optimality and, thus, with low effort. In the second phase, either a timer-based or a threshold-based management policy can invoke a site-wide optimization procedure that aligns capacity allocation with the management goals (e.g., with the goal of using minimal capacity, can be triggered).

Management policies and management goals may be defined at different levels of the management architecture—that is, at the different levels of abstraction. At the topmost level, there are business management goals and policies. We briefly discuss them in the next subsection. In the intermediate level there are service-induced goals and policies. Finally, at the infrastructure management level there are ICT management preferences and policies that are our primary focus in this activity. We discuss them in Section 6.4.4.

Business-Level Goals and Policies. Since business goals are defined at such a high level of abstraction, a semantic gap exists between them and the ICT level management goals and policies. Bridging this gap is notoriously difficult. In this work we aim at narrowing this gap and aligning between the high-level business management goals and ICT-level management policies by introducing the notion of acceptable risk level (ARL) of capacity allocation congestion.

Intuitively, we are interested in minimizing the costs of capacity over-provisioning while controlling the risk associated with capacity over-booking.

From minimizing the cost of capacity over-provisioning, we are interested in maximizing yield of the existing capacity. However, at some point, the conflicts (congestions) in capacity allocation may cause excessive SLA penalties that would offset the advantages of yield maximization.

Accounting for benefits from complying with SLAs and for costs of compliance and noncompliance due to congestions, we can compute residual benefit for the site. The target value of residual benefit can be controlled by a high-level business policy. To satisfy this business policy, we need to calculate an appropriate congestion probability, ARL. ARL, in turn, would help us calculate equivalent capacity for the site to take advantage of statistical multiplexing in a safe manner.

To allow calculation of residual benefit, capacity allocation behavior under congestion should deterministic. In particular, a policy under congestion may be a Max Min Fair Share allocation or higher-priority-first (HPF) capacity allocation, where services with lower SLA classes are satisfied only after all services with higher SLA classes are satisfied.

For the sake of discussion, let us assume that the HPF capacity allocation policy is used.5 We use historical data of the capacity demand (in capacity Whether a certain specific policy is being used is of minor importance. It is important, however, that the policy would be deterministic.

units corresponding to different resource types as explained in Section 6.4.2) per service—specifically, the α-percentile of historic capacity demand per application (where α equals the percentile of compliance required in the ser-vice SLA). This is used to compute the expected capacity allocation per service under capacity allocation congestion. Thus, we obtain the set of application services, whose SLAs may be violated.6 Using penalty values defined for each affected SLA, we obtain the residual benefit that would remain after penalties are enforced. Using the management policy that put a lower bound on the expected residual benefit, we compute acceptable risk value, ε, that satisfies this bound.

**Infrastructure-Level Management Goals and Policies**

In general, infrastructure-level management policies are derived from the business-level management goals. For example, consider our sample business level management goal to “reduce energy expenses by 30% in the next quarter.” This broadly defined goal may imply, among other means for achieving it, that we systematically improve consolidation of VMs on physical hosts by putting excessive capacity into a low-power consumption mode. Thus, a site-wide ICT power conservation-level management policy may be formulated as: “minimize number of physical machines while protecting capacity availability SLAs of the application services.”

As another example, consider the business-level management goal: “Improve customer satisfaction by achieving more aggressive performance SLOs.” One possible policy toward satisfying this business-level goal may be formulated as: “Balance load within the site in order to achieve specific average load per physical host.” Another infrastructure-level management policy to imp-rove performance is: “Minimize the number of VM migrations.” The rationale for this policy is that performance degradation necessarily occurs during VM migration.

**State of the Art**

Our approach to capacity management described in Section 6.4.2 is based on the premise that service providers perform sizing of their services. A detailed discussion of the sizing methodologies is out of our scope, and we will only briefly mention results in this area. Capacity planning for Web services was studied by Menasce and Almeida, Doyle et al. considered the problem of how to map requirements of a known media service workload into the corresponding system resource requirements and to accurately size the required system. Based on the past workload history, the capacity planner finds the 95th percentile of the service demand (for various resources and on different usage windows) and asks for the corresponding configuration. Urgaonkar et al. studied model-based sizing of three-tier commercial services. Recently, Chen et al. sudied the similar problem and provided novel performance models for multi-tier services.

Doyle et al. presented new models for automating resource provision-ing for resources that may interact in complex ways. The premise of the model-based resource provisioning is that internal models capturing service workload and behavior can enable prediction of effects on service perfor-mance of the changes to the service workload and resource allotments. For example, the model can answer questions like: “How much memory is needed to reduce this service’s storage access rate by 20%?” The paper introduces simple performance models for Web services and proposes a model-based resource allocator that utilizes them and allocates appropriate resource slices to achieve needed performance versus capacity utilization. A slice may be mapped to a virtual machine or another resource container providing performance isolation.

In cases when exact model-driven service sizing is not available, learning desirable resource allocation from dynamic service behavior may be possible using black box monitoring of the service network activity as was recently shown by Ben-Yehuda et al. [46] for multi-tier services.

Benefits of capacity multiplexing (under the assumption of known resource demands) in shared hosting platforms were quantitatively studied by Chandra et al..

An approach to capacity over-subscribing that is conceptually similar to ours was recently studied by Urgaonkar et al. . In this work, provisioning CPU and network resources with probabilitistic guarantees on a shared hosting platform were considered. The main difference between our methodology and that of Urgaonkar et al. is that we allocate capacity in integral discrete quanta that encapsulate CPU, memory, network bandwidth, and storage rather than allowing independent infinitesimally small resources allocation along each of this capacity dimensions.

An advance of virtualization technologies and increased awareness about management and power costs of running under-utilized servers have spurred interest in consolidating existing applications on a fewer number of servers in the data center. In most practical settings today a static approach to consolidation, where consolidation is performed as a point-in-time optimization activity, is used. With the static approach, the cost of VM migration are usually not accounted for and relatively time-consuming com-putations are tolerated. Gupta et al. demonstrated that static consolidation problem can be modeled as a variant of the bin packing problem where items to be packed are the servers being consolidated and bins are the target servers. The sizes of the servers/items being packed are resource utilizations that are obtained from the performance trace data. The authors present a two-stage heuristic algorithm for handling the “bin item” assignment constraints that inherently restrict any server consolidation problem. The model is able to solve extremely large instances of the problem in a reasonable amount of time.

Autonomic and dynamic optimization of virtual machines placement in a data center received considerable attention (mainly in the research community) recently.

Bobroff et al. introduce empiric dynamic server migration and con-solidation algorithm based on predicting capacity demand of virtual servers using time series analysis.

Mehta and Neogi [49] presented a virtualized servers consolidation planning tool, Recon, that analyzes historical data collected from an existing environ-ment and computes the potential benefits of server consolidation especially in the dynamic setting.

Gmach et al. considered virtualized servers consolidation of multiple servers and their workloads subject to specific quality of service requirements that need to be supported.

Wood et al. presented Sandpiper, a system that automates the task of monitoring and detecting hotspots, determining a new mapping of physical to virtual resources, and initiating the necessary migrations to protect performance. Singh et al. presented a promising approach to the design of an agile data center with integrated server and storage virtualization technologies. Verma et al. studied the design, implementation, and evaluation of a power-aware application placement controller in the context of an environment with heterogeneous virtualized server clusters. Tang et al. presented a performance model-driven approach to applica-tion placement that can be extended to VM placement.

Wang et al. defined a nonlinear constrained optimization model for dynamic resource provisioning and presented a novel analytic solution. Choi et al. proposed machine learning framework that autonomously finds and adjusts utilization thresholds at runtime for different computing requirements. Kelly studied the problem of allocating discrete resources according to utility functions reported by potential recipients with application to resource allocation in a Utility Data Center (UDC).

Knapsack-related optimization has been relentlessly studied over the last 30 years. The scientific literature on the subject is, therefore, abundant.

**ENHANCING CLOUD COMPUTING ENVIRONMENTS USING A CLUSTER AS A SERVICE**

**Amazon Elastic Compute Cloud (EC2)**

An IaaS cloud, EC2 offers “elastic” access to hardware resources that EC2 clients use to create virtual servers. Inside the virtual servers, clients either host the applications they wish to run or host services of their own to access over the Internet. As demand for the services inside the virtual machine rises, it is possible to create a duplicate (instance) of the virtual machine and distribute the load across the instances.

The first problem with EC2 is its low level of abstraction. Tutorials [6 8] show that when using EC2, clients have to create a virtual machine, install software into it, upload the virtual machine to EC2, and then use a command line tool to start it. Even though EC2 has a set of pre-built virtual machines that EC2 clients can use [9], it still falls on the clients to ensure that their own software is installed and then configured correctly.

It was only recently that Amazon announced new scalability features, specifically Auto-Scaling [10] and Elastic Load Balancing [10]. Before the announcement of these services, it fell to EC2 clients to either modify their services running on EC2 or install additional management software into their EC2 virtual servers. While the offering of Auto-Scaling and Elastic Load Balancing reduces the modification needed for services hosted on EC2, both services are difficult to use and require client involvement [11, 12]. In both cases, it is required of the EC2 client to have a reserve of virtual servers and then configure Auto-Scaling and Elastic Load Balancing to make use of the virtual servers based on demand.

Finally, EC2 does not provide any means for publishing services by other providers, nor does it provide the discovery and selection of services within EC2. An analysis of EC2 documentation [13] shows that network multicasting (a vital element to discovery) is not allowed, thus making discovery and selection of services within EC2 difficult. After services are hosted inside the virtual machines on EC2, clients are required to manually publish their services to a discovery service external to EC2.

**Google App Engine**

Google App Engine [5] is a PaaS cloud that provides a complete Web service environment: All required hardware, operating systems, and software are provided to clients. Thus, clients only have to focus on the installation or creation of their own services, while App Engine runs the services on Google’s servers. However, App Engine is very restricted in what language can be used to build services. At the time of writing, App Engine only supports the Java and Python programming languages. If one is not familiar with any of the supported programming languages, the App Engine client has to learn the language before building his or her own services. Furthermore, existing applications cannot simply be placed on App Engine: Only services written completely in Java and Python are supported.

Finally, App Engine does not contain any support to publish services created by other service providers, nor does it provide discovery and selection services. After creating and hosting their services, clients have to publish their services to discovery services external to App Engine. At the time of writing, an examina-tion of the App Engine code pages [24] also found no matches when the keyword “discovery” was used as a search string.

**Microsoft Windows Azure**

Another PaaS cloud, Microsoft’s Azure [4] allows clients to build services using developer libraries which make use of communication, computational, and storage services in Azure and then simply upload the completed services.To ease service-based development, Azure also provides a discovery service within the cloud itself. Called the .NET Service Bus [14], services hosted in Azure are published once and are locatable even if they are frequently moved. When a service is created/started, it publishes itself to the Bus using a URI [15] and then awaits requests from clients.

While it is interesting that the service can move and still be accessible as long as the client uses the URI, how the client gets the URI is not addressed. Furthermore, it appears that no other information such as state or quality of service (QoS) can be published to the Bus, only the URI.

**Salesforce**

Salesforce [16] is a SaaS cloud that offers customer relations management (CRM) software as a service. Instead of maintaining hardware and software licenses, clients use the software hosted on Salesforce servers for a minimal fee. Clients of Salesforce use the software as though it is their own one and do not have to worry about software maintenance costs. This includes the provision of hardware, the installation, and all required software and the routine updates.

However, Salesforce is only applicable for clients who need existing soft-ware. Salesforce only offers CRM software and does not allow the hosting of custom services. So while it is the cloud with the greatest ease of use, Salesforce has the least flexibility.

**Cloud Summary**

While there is much promise with the four major clouds presented in this chapter, all have a problem when it comes to publishing a discovering required services and resources. Put simply, discovery is close to nonexistent and some clouds require significant involvement from their clients. Of all the clouds examined, only Azure offers a discovery service. However, the discovery service in Azure only addresses static attributes. The .NET Service Bus only allows for the publication of unique identifiers.

Furthermore, current cloud providers assume that human users of clouds are experienced programmers. There is no consideration for clients that are specialists in other fields such as business analysis and engineering. Hence, when interface tools are provided, they are primitive and only usable by computing experts. Ease of use needs to be available to both experienced and novice computing users. What is needed is an approach to provide higher layer abstraction and support for users through the provision of simple publication, discovery, selection, and use of resources. In this chapter, the resource focused on is a cluster. Clients should be able to easily place required files and executables on the cluster and get the results back without knowing any cluster specifics. We propose to exploit Web services to provide a higher level of abstraction and offer these services.

**RVWS DESIGN**

While Web services have simplified resource access and management, it is not possible to know if the resource(s) behind the Web service is (are) ready for requests. Clients need to exchange numerous messages with required Web services to learn the current activity of resources and thus face significant overhead loss if most of the Web services prove ineffective. Furthermore, even in ideal circumstances where all resources behind Web services are the best choice, clients still have to locate the services themselves. Finally, the Web services have to be stateful so that they are able to best reflect the current state of their resources.

This was the motivation for creating the RVWS framework. The novelty of RVWS is that it combines dynamic attributes, stateful Web services (aware of their past activity), stateful and dynamic WSDL documents [1], and brokering [17] into a single, effective, service-based framework. Regardless of clients accessing services directly or discovering them via a broker, clients of RVWS-based distributed systems spend less time learning of services.

**Dynamic Attribute Exposure**

There are two categories of dynamic attributes addressed in the RVWS framework: state and characteristic. State attributes cover the current activity of the service and its resources, thus indicating readiness. For example, a Web service that exposes a cluster (itself a complex resource) would most likely have a dynamic state attribute that indicates how many nodes in the cluster are busy and how many are idle.

Characteristic attributes cover the operational features of the service, the resources behind it, the quality of service (QoS), price and provider informa-tion. Again with the cluster Web service example, a possible characteristic is an array of support software within the cluster. This is important information as cluster clients need to know what software libraries exist on the cluster. Figure 7.1 shows the steps on how to make Web services stateful and how the dynamic attributes of resources are presented to clients via the WSDL document.

To keep the stateful Web service current, a Connector [2] is used to detect changes in resources and then inform the Web service. The Connector has three logical modules: Detection, Decision, and Notification. The Detection module routinely queries the resource for attribute information (1 2). Any changes in the attributes are passed to the Decision module (3) that decides if the attribute change is large enough to warrant a notification. This prevents excessive communication with the Web service. Updated attributes are passed on to the Notification module (4), which informs the stateful Web service (5) that updates its internal state. When clients requests the stateful WSDL document (6), the Web service returns the WSDL document with the values of all attributes (7) at the request time.

**Stateful WSDL Document Creation**

When exposing the dynamic attributes of resources, the RVWS framework allows Web services to expose the dynamic attributes through the WSDL documents of Web services. The Web Service Description Language (WSDL) [18] governs a schema that describes a Web service and a document written in the schema. In this chapter, the term WSDL refers to the stateless WSDL document. Stateful WSDL document refers to the WSDL document created by RVWS Web services.

All information of service resources is kept in a new WSDL section called Resources. Figure 7.2 shows the structure of the Resources section with the rest of the WSDL document. For each resource behind the Web service, a ResourceInfo section exists.

Each ResourceInfo section has a resource-id attribute and two child sections: state and characteristic. All resources behind the Web service have unique identifiers. When the Connector learns of the resource for the first time, it publishes the resource to the Web service.

Both the state and characteristics elements contain several description elements, each with a name attribute and (if the provider wishes) one or more attributes of the service. Attributes in RVWS use the {name: op value} notations. An example attribute is {cost: ,5 $5}.

The state of a resource could be very complex and cannot be described in just one attribute. For example, variations in each node in the cluster all contribute significantly to the state of the cluster. Thus the state in RVWS is described via a collection of attributes, all making up the whole state.

The characteristics section describes near-static attributes of resources such as their limitations and data parameters. For example, the type of CPU on a node in a cluster is described in this section.

<definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

**<resources>**

**<resource-info identifier="***resourceID***"> <state>**

**<description name=""** *attribute1***="***value1***" …**

*attributen***="***valuen***">**

*…Other description Elements…*

**</description>**

*…Other description Elements…*

**</state>**

**<characteristics>**

**<description name="" />**

*…Other description Elements…*

**</characteristics>**

**</resource-info>**

*…Other resource-info elements*

**</resources>**

<types>...</types>

message name="MethodSoapIn">...</message> <message name="MethodSoapOut">...</message>

<portType name="CounterServiceSoap">...</portType>

<binding name="CounterServiceSoap" type="tns:CounterServiceSoap">...</wsdl:binding>

<wsdl:service name="CounterService">...</wsdl:service> </wsdl:definitions>

FIGURE 7.2. New WSDL section.

**Publication in RVWS**

While the stateful WSDL document eliminates the overhead incurred from manually learning the attributes of the service and its resource(s), the issues behind discovering services are still unresolved.

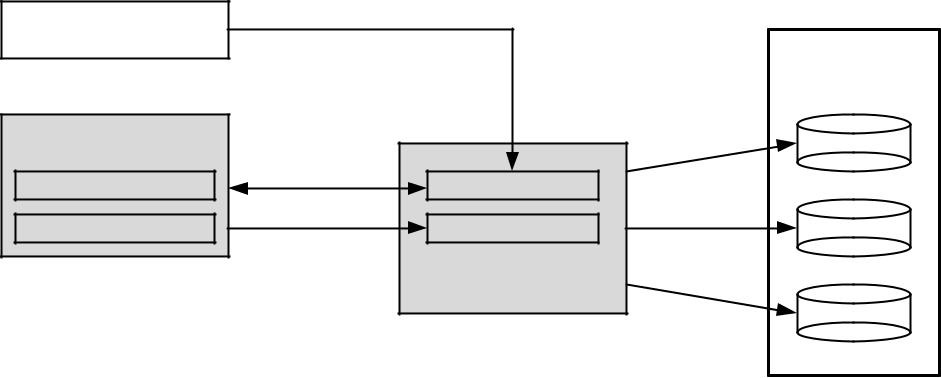
To help ease the publication and discovery of required services with stateful WSDL documents, a Dynamic Broker was proposed (Figure 7.3) [17]. The goal of the Dynamic Broker is to provide an effective publication and discovery service based on service, resource, and provider dynamic attributes.

When publishing to the Broker (1), the provider sends attributes of the Web service to the Dynamic Broker. The dynamic attributes indicate the fun-ctionality, cost, QoS, and any other attributes the provider wishes to have published about the service. Furthermore, the provider is able to publish information about itself, such as the provider’s contact details and reputation.

After publication (1), the Broker gets the stateful WSDL document from the Web service (2). After getting the stateful WSDL document, the Dynamic Broker extracts all resource dynamic attributes from the stateful WSDL documents and stores the resource attributes in the resources store.

The Dynamic Broker then stores the (stateless) WSDL document and service attributes from (1) in the service store. Finally, all attributes about the provider are placed in the providers store.

As the Web service changes, it is able to send a notification to the Broker (3) which then updates the relevant attribute in the relevant store. Had all information about each service been kept in a single stateful WSDL document, the dynamic broker would have spent a lot of time load, thereby editing and saving huge XML documents to the database.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Provider |  | 1. |  |  |
|  |  |  |  |
|  |  |  | Distributed |  |
|  |  |  | Broker Data |  |
| Web Service |  |  | Providers |  |
|  | 2. |  |  |
| State Attrib. | Publication |  |  |
|  |  |  |
| Characteristic Attrib. | 3. | Notication | Services |  |
|  |  |  |
|  |  |  |  |
|  |  | Dynamic Broker |  |  |
|  |  |  | Resources |  |

FIGURE 7.3. Publication.

**Automatic Discovery and Selection**

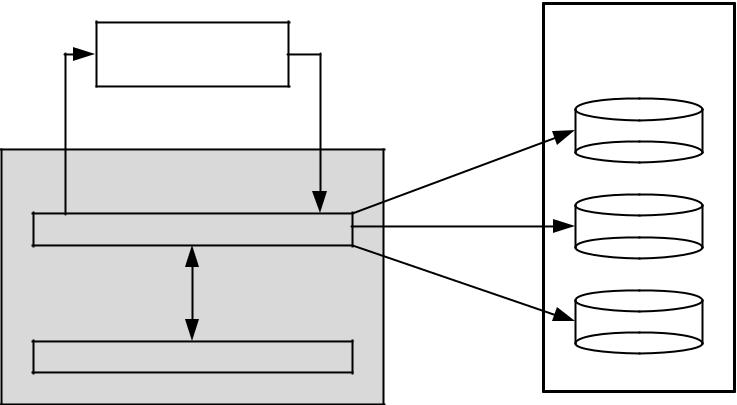
The automatic service discovery that takes into consideration dynamic attri-butes in their WSDL documents allows service (e.g., a cluster) discovery.

When discovering services, the client submits to the Dynamic Broker three groups of requirements (1 in Figure 7.4): service, resource, and provider. The Dynamic Broker compares each requirement group on the related data store (2). Then, after getting matches, the Broker applies filtering (3). As the client using the Broker could vary from human operators to other software units, the resulting matches have to be filtered to suit the client. Finally, the filtered results are returned to the client (4).

The automatic service selection that takes into consideration dynamic attributes in their WSDL documents allows for both a single service (e.g., a cluster) selection and an orchestration of services to satisfy workflow require-ments (Figure 7.5).

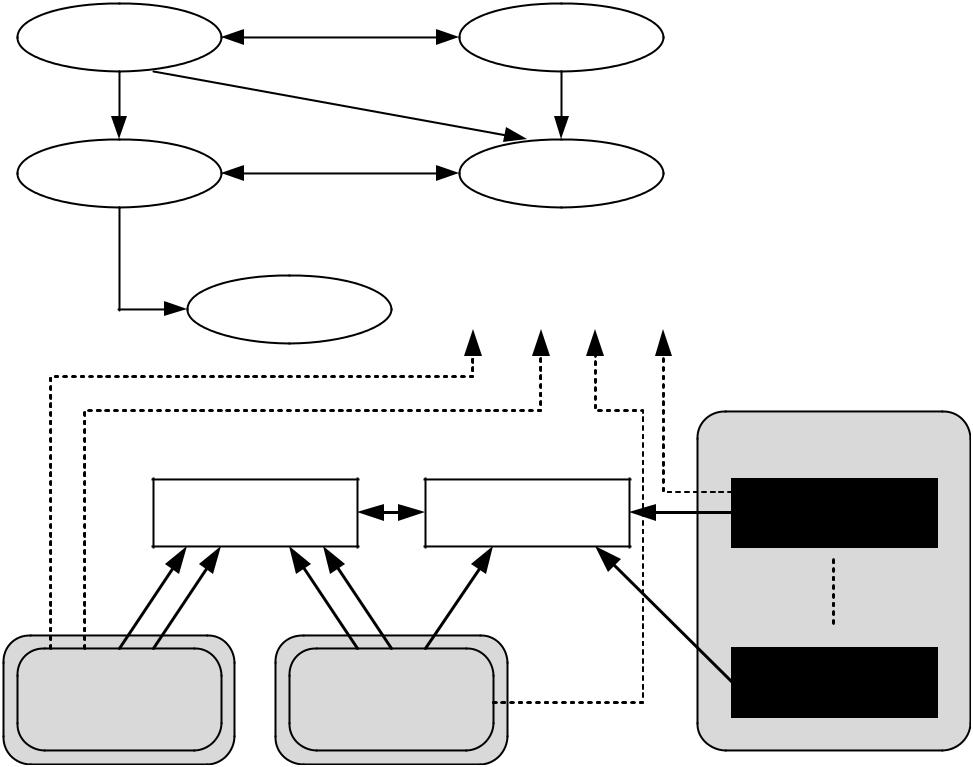
The SLA (service-level agreement) reached by the client and cloud service provider specifies attributes of services that form the client’s request or workflow. This is followed by the process of services’ selection using Brokers. Thus, selection is carried out automatically and transparently. In a system comprising many clouds, the set of attributes is partitioned over many distributed service databases, for autonomy, scalability, and performance.

The automatic selection of services is performed to optimize a function reflecting client requirements. Time-critical and high-throughput tasks benefit by executing a computing intensive application on multiple clusters exposed as services of one or many clouds.



|  |  |  |
| --- | --- | --- |
| Client | Dynamic |  |
| Broker Data |  |
|  |  |
| 4. | 1. |  |
|  | Providers |  |
| Dynamic Broker |  |  |
| Matching | 2. |  |
| Services |  |
| 3. |  |  |
|  | Resources |  |
| Filtering |  |  |

FIGURE 7.4. Matching parameters to attributes.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Client | Negotiation | Cloud |  |  |
|  | Provider |  |  |
|  |  |  |  |
| Selection |  | SLA |  |  |
| Composition | |  |  |  |
|  | Workflow | = <Si1, Si2, Si3 ... Sin> |  |  |
|  |  |  | Public Cloud |  |
|  | Broker | Broker | Service |  |
| Public Cloud | Public Cloud | | Service |  |
|  |  |

FIGURE 7.5. Dynamic discovery and selection.

The dynamic attribute information only relates to clients that are aware of them. Human clients know what the attributes are, owning to the section being clearly named. Software-client-designed pre-RVWS ignore the additional information as they follow the WSDL schema that we have not changed.

**CLUSTER AS A SERVICE: THE LOGICAL DESIGN**

Simplification of the use of clusters could only be achieved through higher layer abstraction that is proposed here to be implemented using the service-based Cluster as a Service (CaaS) Technology. The purpose of the CaaS Technology is to ease the publication, discovery, selection, and use of existing computa-tional clusters.

**CaaS Overview**

The exposure of a cluster via a Web service is intricate and comprises several services running on top of a physical cluster. Figure 7.6 shows the complete CaaS technology.

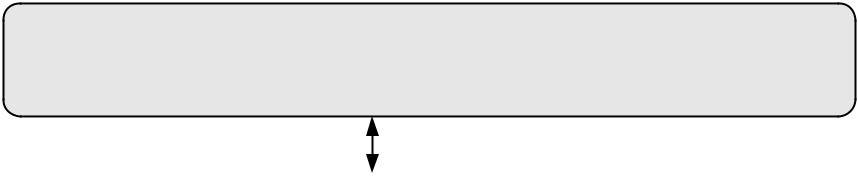
A typical cluster is comprised of three elements: nodes, data storage, and middleware. The middleware virtualizes the cluster into a single system image; thus resources such as the CPU can be used without knowing the organization of the cluster. Of interest to this chapter are the components that manage the allocation of jobs to nodes (scheduler) and that monitor the activity of the cluster (monitor). As time progresses, the amount of free memory, disk space, and CPU usage of each cluster node changes. Information about how quickly the scheduler can take a job and start it on the cluster also is vital in choosing a cluster.

To make information about the cluster publishable, a Publisher Web service and Connector were created using the RVWS framework. The purpose of the publisher Web service was to expose the dynamic attributes of the cluster via the stateful WSDL document. Furthermore, the Publisher service is published to the Dynamic Broker so clients can easily discover the cluster.

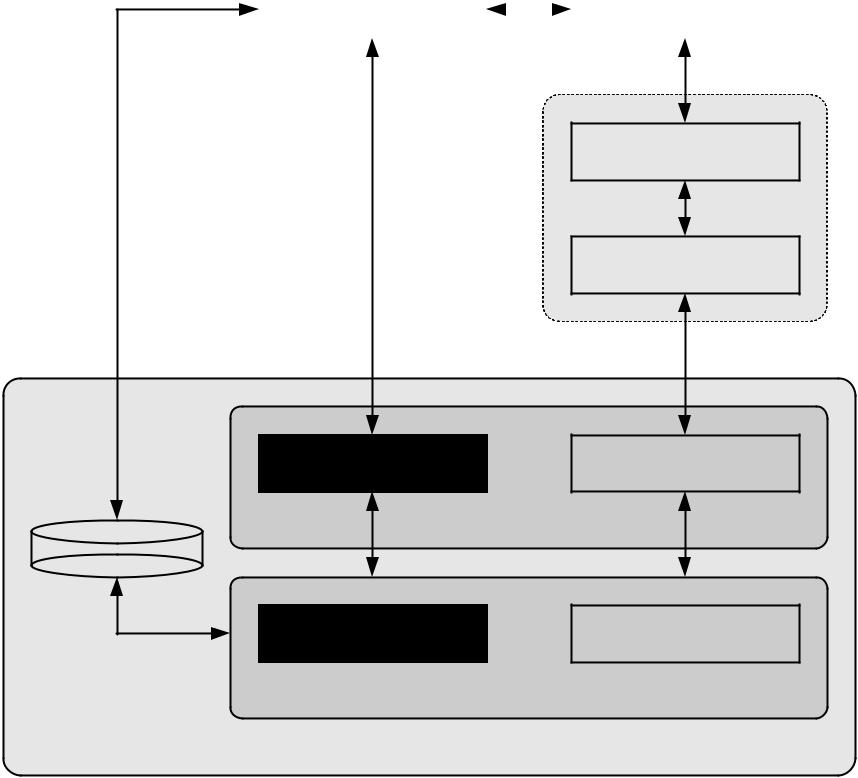
To find clusters, the CaaS Service makes use of the Dynamic Broker. While the Broker is detailed in returning dynamic attributes of matching services, the results from the Dynamic Broker are too detailed for the CaaS Service. Thus another role of the CaaS Service is to “summarize” the result data so that they convey fewer details.

Ordinarily, clients could find required clusters but they still had to manually transfer their files, invoke the scheduler, and get the results back. All three tasks require knowledge of the cluster and are conducted using complex tools. The role of the CaaS Service is to (i) provide easy and intuitive file transfer tools so clients can upload jobs and download results and (ii) offer an easy to use interface for clients to monitor their jobs. The CaaS Service does this by

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | | |  |
|  |  |  |  |  |  |
| Clients | Software Service |  | Human Operator |  |  |
|  |  |  |  |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CaaS Service |  |  |  | Dynamic Broker |  |
|  |  |  |  |
|  |  |  |  |  |  |



Publisher Service

Connector

Scheduler Monitoring

Cluster Middleware

Data Storage

Node 1  Node n

Cluster Nodes

Example Cluster

FIGURE 7.6. Complete CaaS system.

allowing clients to upload files as they would any Web page while carrying out the required data transfer to the cluster transparently.

Because clients to the cluster cannot know how the data storage is managed, the CaaS Service offers a simple transfer interface to clients while addressing the transfer specifics. Finally, the CaaS Service communicates with the cluster’s scheduler, thus freeing the client from needing to know how the scheduler is invoked when submitting and monitoring jobs.

**Cluster Stateful WSDL Document**

As stated in Section 7.4.1, the purpose of the Publisher Web service is to expose the dynamic attributes of a cluster via a stateful WSDL document. Figure 7.7 shows the resources section to be added to the WSDL of the Publisher Web service.

Inside the state and characteristic elements, an XML element for each cluster node was created. The advantage of the XML structuring of our cluster

<definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"> <**resources**>

<**resource-info** resource-identifier="*resourceId*"> <**state** element-identifier="*elementId*">

<**cluster-state** element-identifier="cluster-state-root">

<***cluster-node-name*** free-disk="" free-memory="" native-os-name="" native-os-version="" processes-count="" processes-running="" cpu-usage-percent="" element-identifier="*stateElementId*"

memory-free-percent="" />

*…Other Cluster Node State Elements…*

</**cluster-state**> </**state**>

<**characteristics** element-identifier="*characteristicElementId*">

<**cluster-characteristics** node-count="" element-identifier="cluster-characteristics-root">

<***cluster-node-name*** core-count="" core-speed="" core-speed-unit="" hardware-architecture="" total-disk="" total-memory="" total-disk-unit="" total-memory-unit="" element-identifier="*characteristicElementId*" />

*…Other Cluster Node Characteristic Elements…*

</**cluster-characteristics**> </**characteristics**>

</**resource-info**> </**resources**>

<types>...

<message name="MethodSoapIn">...

<message name="MethodSoapOut">...

<portType name="CounterServiceSoap">...

<binding name="CounterServiceSoap" …>...

<wsdl:service name="CounterService">...

</wsdl:definitions>

FIGURE 7.7. Cluster WSDL.

attributes means that comparing client requirements to resource attributes only requires using XPath queries.

For the CaaS Service to properly support the role of cluster discovery, detailed information about clusters and their nodes needs to be published to the WSDL of the cluster and subsequently to the Broker (Table 7.1).

**CaaS Service Design**

The CaaS service can be described as having four main tasks: cluster discovery and selection, result organization, job management, and file management. Based on these tasks, the CaaS Service has been designed using

TABLE 7.1. Cluster Attributes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | | Attribute Name | Attribute Description | | Source |  |
|  |  |  |  |  |  |  |
| Characteristics | | core-count | Number of cores on a cluster | | Cluster node |  |
|  |  |  | node | |  |  |
|  |  |  |  |  |  |  |
|  |  | core-speed | Speed of each core | |  |  |
|  |  |  |  |  |  |  |
|  |  | core-speed-unit | Unit for the core speed (e.g., | |  |  |
|  |  |  | gigahertz) | |  |  |
|  |  |  |  |  |  |  |
|  |  | hardware- | Hardware architecture of each | |  |  |
|  |  | architecture | cluster node (e.g., 32-bit Intel) | |  |  |
|  |  |  |  |  |  |  |
|  |  | total-disk | Total amount of physical | |  |  |
|  |  |  | storage space | |  |  |
|  |  |  |  |  |  |  |
|  |  | total-disk-unit | Storage amount unit (e.g., | |  |  |
|  |  |  | gigabytes) | |  |  |
|  |  |  |  |  |  |  |
|  |  | total-memory | Total amount of physical | |  |  |
|  |  |  | memory | |  |  |
|  |  |  |  |  |  |  |
|  |  | total-memory-unit | Memory amount measurement | |  |  |
|  |  |  | (e.g., gigabytes) | |  |  |
|  |  |  |  |  |  |  |
|  |  | software-name | Name of an installed piece of | |  |  |
|  |  |  | software. | |  |  |
|  |  |  |  |  |  |  |
|  |  | software-version | Version of a installed piece of | |  |  |
|  |  |  | software | |  |  |
|  |  |  |  |  |  |  |
|  |  | software- | Architecture of a installed piece | |  |  |
|  |  | architecture | of software | |  |  |
|  |  |  |  | |  |  |
|  |  | node-count | Total number of nodes in the | | Generated |  |
|  |  |  | cluster. Node count differs | |  |  |
|  |  |  | from core-count as each node | |  |  |
|  |  |  | in a cluster can have many | |  |  |
|  |  |  | cores. | |  |  |
|  |  |  |  | |  |  |
| State | | free-disk | Amount of free disk space | | Cluster node |  |
|  |  |  |  | |  |  |
|  |  | free-memory | Amount of free memory | |  |  |
|  |  |  |  |  |  |  |
|  |  | os-name | Name of the installed operating | |  |  |
|  |  |  | system | |  |  |
|  |  |  |  |  |  |  |
|  |  | os-version | Version of the running | |  |  |
|  |  |  | operating system | |  |  |
|  |  |  |  |  |  |  |
|  |  | processes-count | Number of processes | |  |  |
|  |  |  |  |  |  |  |
|  |  | processes-running | Number of processes running | |  |  |
|  |  |  |  | |  |  |
|  |  | cpu-usage-percent | Overall percent of CPU used. | | Generated |  |
|  |  |  | As this metric is for the node | |  |  |
|  |  |  | itself, this value becomes | |  |  |
|  |  |  | averaged over cluster core | |  |  |
|  |  |  |  |  |
|  |  |  |  | |  |  |
|  |  | memory-free- | Amount of free memory on the | |  |  |
|  |  | percent | cluster node | |  |  |
|  |  |  |  |  |  |  |



intercommunicating modules. Each module in the CaaS Service encapsulates one of the tasks and is able to communicate with other modules to extend its functionality.

Figure 7.8 presents the modules with the CaaS Service and illustrates the dependencies between them. To improve the description, elements from Figure 7.6 have been included to show what other entities are used by the CaaS service.

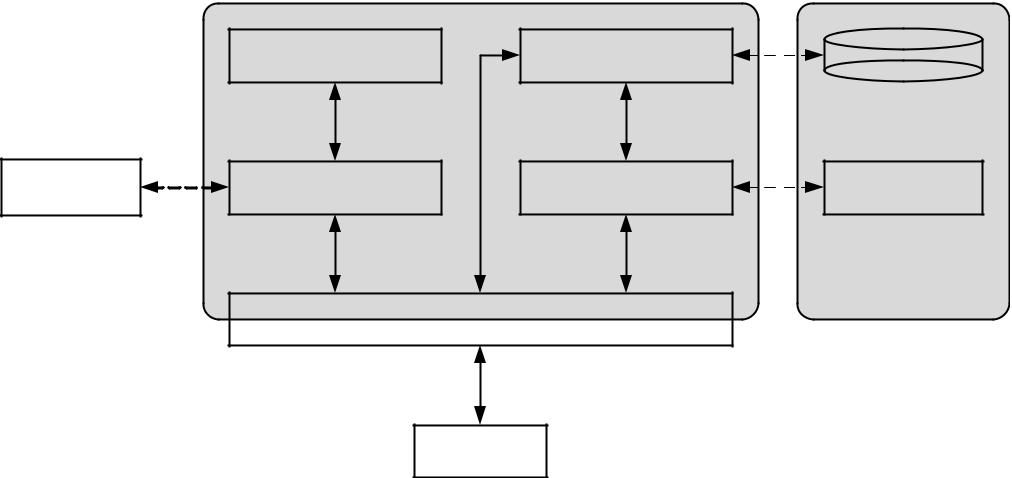
The modules inside the CaaS Web service are only accessed through an interface. The use of the interface means the Web service can be updated over time without requiring clients to be updated nor modified.

Invoking an operation on the CaaS Service Interface (discovery, etc.) invokes operations on various modules. Thus, to best describe the role each module plays, the following sections outline the various tasks that the CaaS Service carries out.

Cluster Discovery. Before a client uses a cluster, a cluster must be discovered and selected first. Figure 7.9 shows the workflow on finding a required cluster. To start, clients submit cluster requirements in the form of attribute values to the CaaS Service Interface (1). The requirements range from the number of nodes in the cluster to the installed software (both operating systems and software APIs). The CaaS Service Interface invokes the Cluster Finder module

(2) that communicates with the Dynamic Broker (3) and returns service matches (if any).

To address the detailed results from the Broker, the Cluster Finder module invokes the Results Organizer module (4) that takes the Broker results and returns an organized version that is returned to the client (5 6). The organized



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Result Organizer | File Manager | Data Storage |  |
|  |  |  |  |
| Dynamic | Cluster Finder | Job Manager | Scheduler |  |
| Broker |  |
|  |  |  |  |
|  |  |  | Example Cluster |  |
|  | CaaS Service Interface | |  |  |
|  |  | Client |  |  |

FIGURE 7.8. CaaS Service design.

results instruct the client what clusters satisfy the specified requirements. After reviewing the results, the client chooses a cluster.

Job Submission. After selecting a required cluster, all executables and data files have to be transferred to the cluster and the job submitted to the scheduler for execution. As clusters vary significantly in the software middleware used to create them, it can be difficult to place jobs on the cluster. To do so requires knowing how jobs are stored and how they are queued for execution on the cluster. Figure 7.10 shows how the CaaS Service simplifies the use of a cluster to the point where the client does not have to know about the underlying middleware.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dynamic Broker | | | | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3. | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 4. | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Cluster Finder | | | | | | |  | Result Organizer | |  |
|  |  |  |  |  |
|  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2. |  |  | | | 5. |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | |  |  | |  |  |  |
|  |  |  |  | CaaS Service Interface | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. | | | | | |  |  | | | | 6. |  |  |
|  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | | | |  | |  |
|  |  |  |  |  |  |  | Client | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

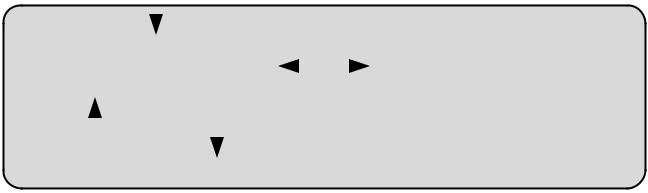
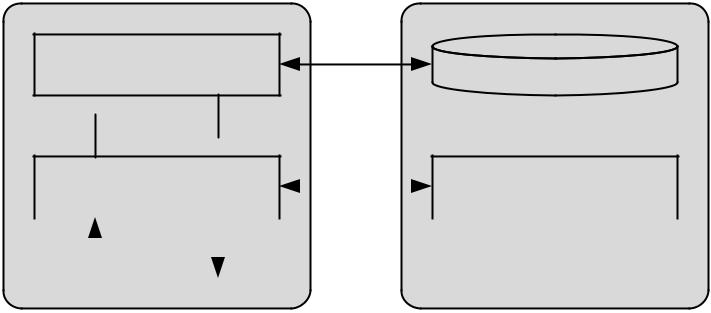


FIGURE 7.9. Cluster discovery.



|  |  |  |
| --- | --- | --- |
| File Manager | 4. |  |
| Data Storage |  |
|  |  |

 5. 3. 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Job Manager | | | 6. | | |  | Scheduler |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 2. |  |  | 7. |  |  |  | Example Cluster |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| CaaS Service | | |  |  |
|  |  |  |  |  |  |
|  | Interface | |  |  |  |  |  |  |
| 1. |  |  | 8. |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | Client | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |



FIGURE 7.10. Job submission.

All required data, parameters, such as estimated runtime, are uploaded to the CaaS Service (1). Once the file upload is complete, the Job Manager is invoked (2). It resolves the transfer of all files to the cluster by invoking the File Manager (3) that makes a connection to the cluster storage and commences the transfer of all files (4).

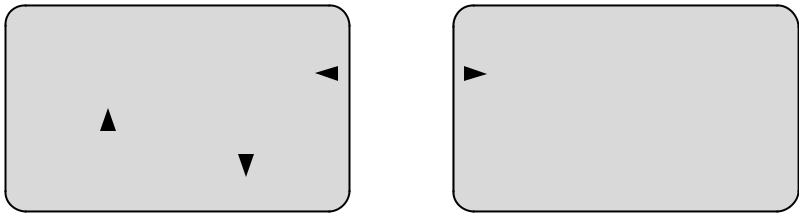
Upon completion of the transfer (4), the outcome is reported back to the Job Manager (5). On failure, a report is sent and the client can decide on the appropriate action to take. If the file transfer was successful, the Job Manager invokes the scheduler on the cluster (6).

The same parameters the client gave to the CaaS Service Interface are submitted to the scheduler; the only difference being that the Job Manager also informs the scheduler where the job is kept so it can be started. If the outcome of the scheduler (6) is successful, the client is then informed (7 8). The outcome includes the response from the scheduler, the job identifier the scheduler gave to the job, and any other information the scheduler provides.

Job Monitoring. During execution, clients should be able to view the execution progress of their jobs. Even though the cluster is not the owned by the client, the job is. Thus, it is the right of the client to see how the job is progressing and (if the client decides) terminate the job and remove it from the cluster. Figure 7.11. outlines the workflow the client takes when querying about job execution.

First, the client contacts the CaaS service interface (1) that invokes the Job Manager module (2). No matter what the operation is (check, pause, or terminate), the Job Manager only has to communicate with the scheduler (3) and reports back a successful outcome to the client (4 5).

Result Collection. The final role of the CaaS Service is addressing jobs that have terminated or completed their execution successfully. In both

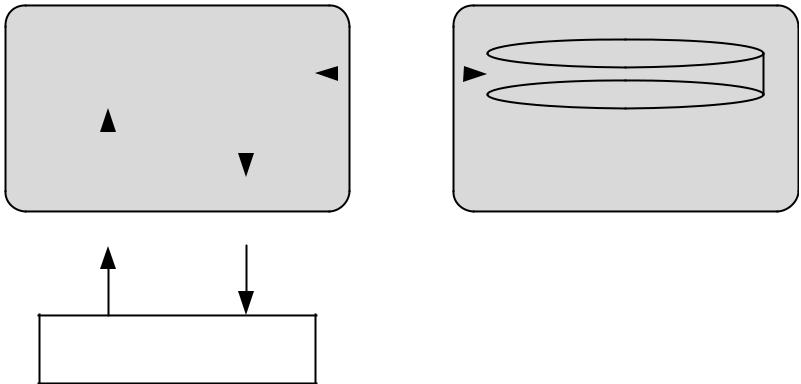


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Job Manager |  | 3. | |  | Scheduler |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 2. |  | 4. |  |  |  | Example Cluster |  |
|  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | CaaS Service |  |  |
|  |  |  |  |  |  |  |
|  | Interface |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1. |  | 5. |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | Client |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |



FIGURE 7.11. Job monitoring.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | |  |  |
|  |  |  | |  |  |  |  |  |  |
|  | File Manager |  | 3. | |  |  | Data Storage |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 2. |  | 4. |  |  |  |  | Example Cluster |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | CaaS Service |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | Interface |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

5.

1.

Client

FIGURE 7.12. Job result collection.

cases, error or data files need to be transferred to the client. Figure 7.12 presents the workflow and CaaS Service modules used to retrieve error or result files from the cluster.

Clients start the error or result file transfer by contacting the CaaS Service Interface (1) that then invokes the File Manager (2) to retrieve the files from the cluster’s data storage (3). If there is a transfer error, the File Manager attempts to resolve the issue first before informing the client. If the transfer of files (3) is successful, the files are returned to the CaaS Service Interface (4) and then the client (5). When returning the files, URL link or a FTP address is provided so the client can retrieve the files.

**User Interface: CaaS Web Pages**

The CaaS Service has to support at least two forms of client: software clients and human operator clients. Software clients could be other software applica-tions or services and thus are able to communicate with the CaaS Service Interface directly.

For human operators to use the CaaS Service, a series of Web pages has been designed. Each page in the series covers a step in the process of discovering, selecting, and using a cluster. Figure 7.13 shows the Cluster Specification Web page where clients can start the discovery of a required cluster.

In Section A the client is able to specify attributes about the required cluster. Section B allows specifying any required software the cluster job needs. Afterwards, the attributes are then given to the CaaS service that performs a search for possible clusters and the results are displayed in a Select Cluster Web page (Figure 7.14).

Next, the client goes to the job specification page, Figure 7.15. Section A allows specifying the job. Section B allows the client to specify and upload all data files and job executables. If the job is complex, Section B also allows specifying a job script. Job scripts are script files that describe and manage

|  |  |  |
| --- | --- | --- |
| Section A: Hardware |  |  |
| Number of Nodes: | 50 |  |
| Amount of Memory: | 50 | GB |
| Free Memory: | 50 | GB |
| Disk Free: | 50 | GB |

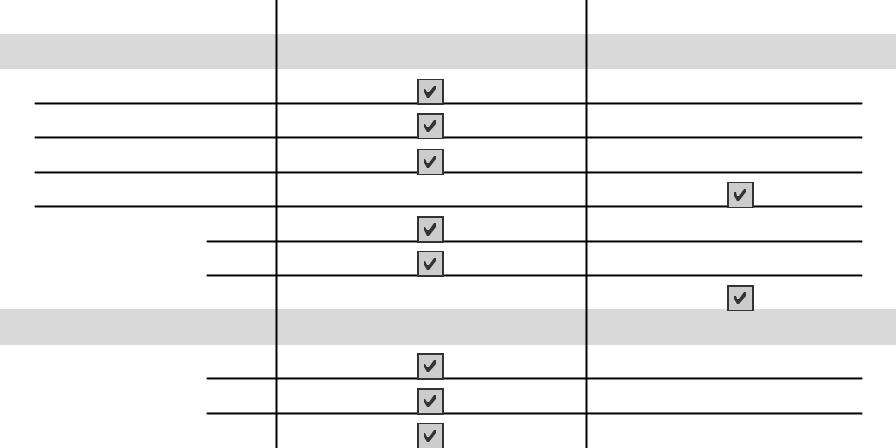


|  |  |  |  |
| --- | --- | --- | --- |
| CPU: Pentium 4 | 64 bit | 3.2 | GHz |
|  |  |  |  |
| Section B: Software |  |  |  |
| Operating System: Windows XP w/Service Pack 2 | |  |  |
|  |  |  |  |
|  |  | Discover | -> |
|  |  |  |  |



FIGURE 7.13. Web page for cluster specification.

|  |  |
| --- | --- |
| Cluster A | Cluster B |
| select | select |

Hardware

Number of Nodes :

Amount of Memory :

Free Memory :

Disk Free :

CPU :

*Architecture :*

*Speed*

Software

Operating System :

*Architecture :*

*Version :*



<- Refine Search

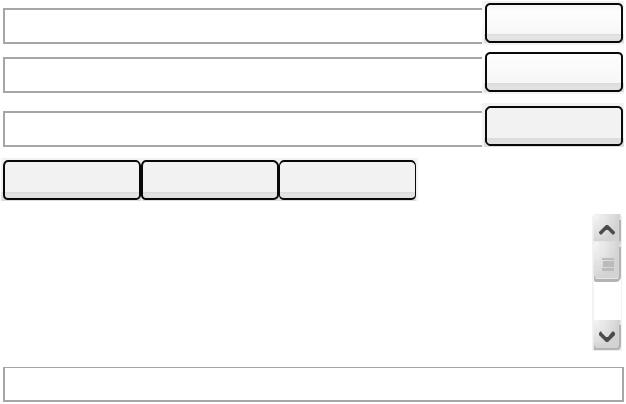
FIGURE 7.14. Web page for showing matching clusters.

various stages of a large cluster job. Section C allows specifying an estimated time the job would take to complete.

Afterword, the CaaS Service attempts to submit the job; the outcome is shown in the Job Monitoring page, Figure 7.16. Section A tells the client whether the job is submitted successfully. Section B offers commands to allow the client to take an appropriate action.

When the job is complete, the client is able to collect the results from the Collect Results page (Figure 7.17). Section A shows the outcome of the job.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | |  |  |
|  | |  |  |  |  |  |
| Section A: Identification | |  |  |  |  |  |
| Job Name: |  |  |  |  |  |  |
| Travelling Sales Man | |  |  |  |  |
| Job Owner |  |  |  |  |  |  |
| Joe Bloggs |  |  |  |  |  |
|  | |  |  |  |  |  |
| Section B: Job File Specification | |  |  |  |  |  |
| Executible |  |  |  | Browse... |  |  |
| My\_exec.exe |  |  |  |  |
| Script: | my\_script.pl |  |  | Browse... |  |  |
| Data files: | custom\_set.dat |  |  | Browse... |  |  |
|  | Add | Remove | Clear | |  |  |
|  |  |  |  |  |  |  |
|  | Proven.dat |  |  |  |  |  |
|  | Control.dat |  |  |  |  |  |
|  | Recent.dat |  |  |  |  |  |
|  |  |  |  |  |  |  |

Output Filename: out.dat

Section C: Execution Specification



Estimated Tme: 3d 14h



<- Change Clusters Submit ->

FIGURE 7.15. Web page for job specification.

Section A: Submission Outcome

|  |  |  |
| --- | --- | --- |
| Outcome: | Submitted Successfully |  |
| Job ID: |  |  |
| cj404 |  |
| Report: | Delegating Submission request…. Request Accepted. |  |
|  | Job has been started. |  |



Section B: Job Control



Refresh Pause Halt



Collect Results ->

FIGURE 7.16. Web page for monitoring job execution.

Section A: Execution Outcome

Outcome: Completed Successfully

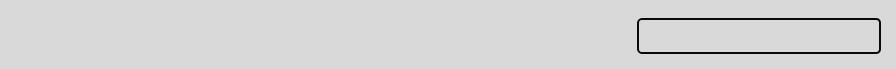
Time Finished: 16:59



Report: After a total of 2 days and 7 hours, your job has completed execution.

Section B : Results Download

HTTP: http://download.clustera.org/cb404/out.dat



Finish

FIGURE 7.17. Web page for collecting result files.

Section B allows the client to easily download the output file generated from the completed/aborted job via HTTP or using an FTP client.

**PROOF OF CONCEPT**

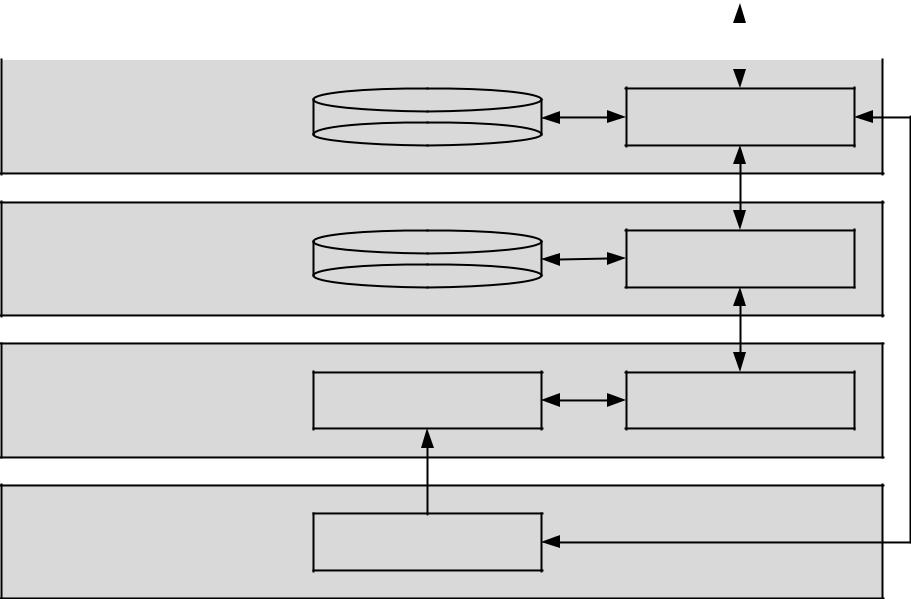
To demonstrate the RVWS framework and CaaS Technology, a proof of concept was performed where an existing cluster was published, discovered, selected, and used. It was expected that the existing cluster could be easily used all through a Web browser and without any knowledge of the underlying middleware.

**CaaS Technology Implementation**

The CaaS Service was implemented using Windows Communication Founda-tions (WCF) of .NET 3.5 that uses Web services. An open source library for building SSH clients in .NET (sharpSSH) [19] was used to build the Job and File Managers. Because schedulers are mostly command driven, the commands and outputs were wrapped into a Web service. Each module outlined in Section 7.4.3 is implemented as its own Web service.

The experiments were carried out on a single cluster exposed via RVWS; communication was carried out only through the CaaS Service. To manage all the services and databases needed to expose and use clusters via Web services, VMware virtual machines were used. Figure 7.18 shows the complete test environment with the contents of each virtual machine. All virtual machines have 512 MB of virtual memory and run the Windows Server 2003. All virtual machines run .NET 2.0; the CaaS virtual machine runs .NET 3.5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | |  |  |
|  |  |  |  |  |  |
| Client System |  |  |  |  |  |
| Web Browser | |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
| CaaS System | Temp File Store | CaaS Service |  |
| {VMware VM} |  |
|  |  |
| Dynamic Broker |  |  |  |
| System | Database | Dynamic Broker |  |
| {VMware VM} |  |  |
|  |  |  |
| Publisher Web |  | Publisher Web |  |
| Service System | Connector |  |
| Service |  |
| {VMware VM} |  |  |
|  |  |  |
| Cluster | Deakin |  |  |

FIGURE 7.18. Complete CaaS environment.

The first virtual machine is the Publisher Web service system. It contains the Connector, Publisher Web service [17], and all required software libraries. The Dynamic Broker virtual machine contains the Broker and its database. The final virtual machine is the CaaS virtual machine; it has the CaaS Service and a temporary data store. To improve reliability, all file transfers between the cluster and the client are cached. The client system is an Asus Notebook with 2 gigabytes of memory and an Intel Centrino Duo processor, and it runs the Windows XP operating system.

**Cluster Behind the CaaS**

The cluster used in the proof of concept consists of 20 nodes plus two head nodes (one running Linux and the other running Windows). Each node in the cluster has two Intel Cloverton Quad Core CPUs running at 1.6 GHz, 8 gigabytes of memory, and 250 gigabytes of data storage, and all nodes are connected via gigabit Ethernet and Infiniband. The head nodes are the same except they have 1.2 terabytes of data storage.

In terms of middleware, the cluster was constructed using Sun GridEngine [20], OpenMPI [21], and Ganglia [22]. GridEngine provided a high level of abstraction where jobs were placed in a queue and then allocated to cluster nodes based on policies. OpenMPI provided a common distribute application API that hid the underlying communication system. Finally, Ganglia provided easy access to current cluster node usage metrics.

Even though there is a rich set of software middleware, the use of the middleware itself is complex and requires invocation from command line tools. In this proof of concept, it is expected that all the list middleware will be abstracted so clients only see the cluster as a large supercomputer and do not have to know about the middleware.

**Experiments and Results**

The first experiment was the publication of the cluster to the publisher Web service and easily discovering the cluster via the Dynamic Broker. For this experiment, a gene folding application from UNAFold [23] was used. The application was used because it had high CPU and memory demands. To keep consistency between results from the publisher Web service and Dynamic Broker, the cluster Connector was instructed to log all its actions to a text file to later examination.

Figure 7.19 shows that after starting the Connector, the Connector was able to learn of cluster node metrics from Ganglia, organize the captured Ganglia metrics into attributes, and forwarded the attributes to the Publisher Web service.

Figure 7.20 shows that the data from the Connector was also being presented in the stateful WSDL document. As the Connector was detecting slight changes in the cluster (created from the management services), the stateful WSDL of the cluster Web service was requested and the same information was found in the stateful WSDL document.

**22/01/2009 1:51:52 PM-Connector[Update]:**

**Passing 23 attribute updates to the web service...**

* **Updating west-03.eit.deakin.edu.au-state in free-memory to 7805776**
* **Updating west-03.eit.deakin.edu.au-state in ready-queue-last-five-minutes to 0.00**

*...* *Other attribute updates from various cluster nodes...*

FIGURE 7.19. Connector output.

<rvwi:state rvwi:element-identifier= "resource-state">

<cluster-state>

**<west-03.eit.deakin.edu.au free-memory="7805776" />**

*...Other Cluster Node Entries...*

</cluster-state>

*...Rest of Stateful WSDL...*

FIGURE 7.20. Updated WSDL element.

In the consistency stage, a computational and memory intense job was started on a randomly selected node and the stateful WSDL of the Publisher Web service requested to see if the correct cluster node was updated. The WSDL document indicated that node 20 was running the job (Figure 7.21). This was confirmed when the output file of the Connector was examined. As the cluster changed, both the Connector and the Publisher Web service were kept current.

After publication, the Dynamic Broker was used to discover the newly published Web service. A functional attribute of {main: 5 monitor} was specified for the discovery. Figure 7.22 shows the Dynamic Broker discovery results with the location of the Publisher Web service and its matching dynamic attribute.

At this point, all the cluster nodes were being shown because no require-ments on the state nor the characteristics of the cluster were specified. The purpose of the selection stage of this experiment is intended to ensure that when given client attribute values, the Dynamic Broker only returned matching attribute.

For this stage, only loaded cluster nodes were required; thus a state attribute value of {cpu\_usage\_percent: >10} was specified. Figure 7.23 shows the Dynamic Broker results only indicating node 20 as a loaded cluster node.

**<west-20.eit.deakin.edu.au**

**cpu-system-usage="1.5"**

**cpu-usage-percent="16.8"**

**free-memory="12104"**

**memory-free-percent="0.001489594" />**

FIGURE 7.21. Loaded cluster node element.

<ArrayOfServiceMatch>

<ServiceMatch>

<Url >http://einstein/rvws/rvwi\_cluster /

ClusterMonitorService.asmx</Url>

<Wsdl>*...Service Stateful WSDL...*</Wsdl>

<Metadata> <service-meta>

<Functionalty **main="monitor"** />

*...Other Provider Attributes...*

</service-meta>

</Metadata>

</ServiceMatch>

</ArrayOfServiceMatch>

FIGURE 7.22. Service match results from dynamic broker.

<west-20.eit.deakin.edu.au

**cpu-usage-percent="64.3"** />

FIGURE 7.23. The only state element returned.

<west-03.eit.deakin.edu.au **cpu-usage-percent="12.5"** />

<west-20.eit.deakin.edu.au **cpu-usage-percent="63"** />

FIGURE 7.24. Cluster nodes returned from the broker.

The final test was to load yet another randomly selected cluster node. This time, the cluster node was to be discovered using only the Dynamic Broker and without looking at the Connector or the Publisher Web service. Once a job was placed on a randomly selected cluster node, the Dynamic Broker was queried with the same attribute values that generated Figure 7.23.

Figure 7.24 shows the Dynamic Broker results indicating node 3 as a loaded cluster node. Figure 7.25 shows an excerpt from the Connector text file that confirmed that node 3 had recently changed state.

Figure 7.26 shows the filled-in Web form from the browser. Figure 7.27 shows the outcome of our cluster discovery. This outcome is formatted like that shown in Figure 7.14. As the cluster was now being successfully published, it was possible to test the rest of the CaaS solution.

Figure 7.26 shows the filled in Web form from the browser. Figure 7.27 shows the outcome of our cluster discovery, formatted like that shown in Figure 7.14. Because only the Deakin cluster was present, that cluster was chosen to run our job. For our example job, we specified the script, data files, and a desired return file.

Figure 7.28 shows the complete form. For this proof of concept, the cluster job was simple: Run UNIX grep over a text file and return another text file with lines that match our required pattern. While small, all the functionality of the CaaS service is used: The script and data file had to be uploaded and then submitted, to the scheduler, and the result file had to be returned.

Once our job was specified, clicking the “Submit” button was expected to upload the files to the CaaS virtual machine and then transfer the files to the cluster. Once the page in Figure 7.29 was presented to us, we examined both the CaaS virtual machine and cluster data store. In both cases, we found our script and data file.

After seeing the output of the Job Monitoring page, we contacted the cluster and queried the scheduler to see if information on the page was correct. The job listed on the page was given the ID of 3888, and we found the same job listed as running with the scheduler.

One final test was seeing if the Job Monitoring Web page was able to check the state of our job and (if finished) allows us to collect our result file. We got confirmation that our job had completed, and we were able to proceed to the Results Collection page.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 22/01/2009 | 2:00:58 PM-Connector[Update]: |  |
| Passing 36 attribute updates to the web service... | |  |
| \* Updating | west-03.eit.deakin.edu.au-state in |  |

**cpu-usage-percent to 12.5**

FIGURE 7.25. Text file entry from the connector.

**Section A: Hardware**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Nodes: | 20 |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Amount of Memory: | 8130000 |  | Gigabyte | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Free Memory: | 7400000 |  | Gigabyte | | | |  |  |  |  |  |
| Disk Free: |  |  |  |  |  |  |  |  |  |  |  |
|  |  | Gigabyte | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| CPU: |  |  | 32-bit |  |  |  |  |  |  | GigaHertz |  |
|  |  |  |  |  |  |  |  |  |  |  |  |



**Section B: Software**



Operating System: Any Linux

FIGURE 7.26. Cluster specification.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Hardware | |  |  |  |  |  | Software | |  |  |  |
|  |  |  |  |  |  | |  |  | |  | |  |  |  |
| Cluster | Nodes | Mem. Amount | Mem. Free | Disk Free | CPU Archi. | CPU Speed |  | OS Name | | OS Ver. | | OS Archi. |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Deakin | 20 | 9 | 3 | – | 9 | – |  | 20 | |  | – | – |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | Use Selected | | |  |
|  |  |  |  |  |  |  |  |  | Deakin |  |  |
|  |  |  | FIGURE 7.27. Cluster selection. | | | |  |  |  |  |  |  |  |  |



**Section B: Job File Submission**



|  |  |  |
| --- | --- | --- |
| Executible: |  | Browse\_ |
|  |  |  |
|  |  |  |
| Script: | C:\collection\execution.s | Browse\_ |
|  |  |  |
|  |  |  |
| Data Files: | C:\collection\data.zip | Browse\_ |
|  |  |  |
|  |  |  |
| Name of Output File: | cats.txt |  |
|  |  |  |

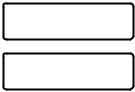


FIGURE 7.28. Job specification.

**Section A: Submission Outcome**

Outcome: Your job 38888 ( execution.sh ) has been submitted

Job ID: 38888



Report: 26/05/2009 10:39:03 AM: You job is still running. 26/05/2009 10:39:55 AM: You job appears to have finished. 26/05/2009 10:39:55 AM: Please collect your result files.

FIGURE 7.29. Job monitoring.

**Section B: Result File Download**

HTTP: cats.txt

FTP:

FIGURE 7.30. Result collection.

The collection of result file(s) starts when the “Collect Results” button (shown in Figure 7.16) is clicked. It was expected that by this time the result file would have been copied to the CaaS virtual machine. Once the collection Web page was displayed (Figure 7.30), we checked the virtual machine and found our results file.

**SECURE DISTRIBUTED DATA STORAGE IN CLOUD COMPUTING**

**CLOUD STORAGE: FROM LANs TO WANs**

Cloud computing has been viewed as the future of the IT industry. It will be a revolutionary change in computing services. Users will be allowed to purchase CPU cycles, memory utilities, and information storage services conveniently just like how we pay our monthly water and electricity bills. However, this image will not become realistic until some challenges have been addressed. In this section, we will briefly introduce the major difference brought by distributed data storage in cloud computing environment. Then, vulnerabilities in today’s cloud computing platforms are analyzed and illustrated.

**Moving From LANs to WANs**

Most designs of distributed storage take the form of either storage area networks (SANs) or network-attached storage (NAS) on the LAN level, such

as the networks of an enterprise, a campus, or an organization. SANs are constructed on top of block-addressed storage units connected through dedicated high-speed networks. In contrast, NAS is implemented by attaching specialized file servers to a TCP/IP network and providing a file-based interface to client machine [6]. For SANs and NAS, the distributed storage nodes are managed by the same authority. The system administrator has control over each node, and essentially the security level of data is under control. The reliability of such systems is often achieved by redundancy, and the storage security is highly dependent on the security of the system against the attacks and intrusion from outsiders. The confidentiality and integrity of data are mostly achieved using robust cryptographic schemes.

However, such a security system would not be robust enough to secure the data in distributed storage applications at the level of wide area net-works, specifically in the cloud computing environment. The recent progress of network technology enables global-scale collaboration over heterogeneous networks under different authorities. For instance, in a peer-to-peer (P2P) file sharing environment, or the distributed storage in a cloud computing environment, the specific data storage strategy is transparent to the user [3]. Furthermore, there is no approach to guarantee that the data host nodes are under robust security protection. In addition, the activity of the medium owner is not controllable to the data owner. Theoretically speaking, an attacker can do whatever she wants to the data stored in a storage node once the node is compromised. Therefore, the confidentiality and the integrity of the data would be violated when an adversary controls a node or the node administrator becomes malicious.

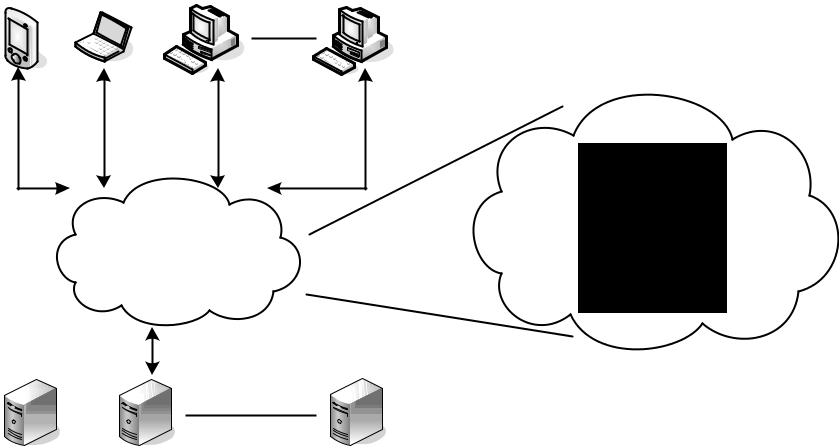
**Existing Commercial Cloud Services**

As shown in Figure 8.1, data storage services on the platform of cloud computing are fundamentally provided by applications/software based on the Internet. Although the definition of cloud computing is not clear yet, several pioneer commercial implementations have been constructed and opened to the public, such as Amazon’s Computer Cloud AWS (Amazon Web service) [7], the Microsoft Azure Service Platform [8], and the Google App Engine (GAE) [9].

In normal network-based applications, user authentication, data confidenti-ality, and data integrity can be solved through IPSec proxy using encryption and digital signature. The key exchanging issues can be solved by SSL proxy. These methods have been applied to today’s cloud computing to secure the data on the cloud and also secure the communication of data to and from the cloud. The service providers claim that their services are secure. This section describes three secure methods used in three commercial cloud services and discusses their vulnerabilities.

Amazon’s Web Service. Amazon provides Infrastructure as a Service (IaaS) with different terms, such as Elastic Compute Cloud (EC2), SimpleDB, Simple

Mobile Laptop Station



|  |  |  |
| --- | --- | --- |
|  | SaaS |  |
|  | PaaS |  |
| Cloud | Cloud |  |
| IaaS |  |
| (Network Fabric) |  |
| Internet |  |
|  |  |

Storage Server Farm

FIGURE 8.1. Illustration of cloud computing principle.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | |  | Service Provider | |
|  |  |  |  |  |
| Create a job | |  | Verify the manifest file | |
| Get the manifest file | |  | with received signature | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sign the manifest file | |  | Operate as the file | |
| Email the manifest file | |  | demand | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |



|  |  |  |
| --- | --- | --- |
| Ship the device with |  | Ship the device, |
| signed file |  | email the log with MD5 |
|  |  |  |



In One Session

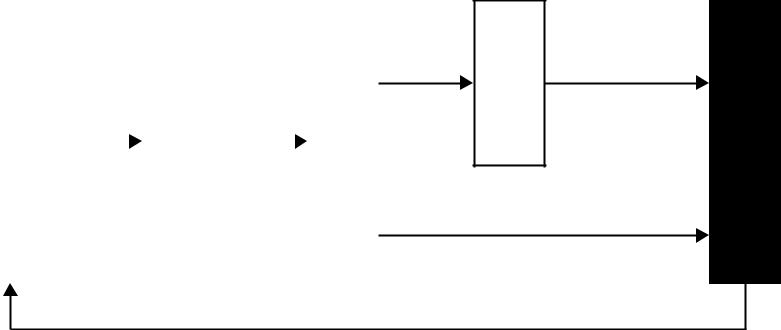
FIGURE 8.2. AWS data processing procedure.

Storage Service (S3), and so on. They are supposed to ensure the confidenti-ality, integrity, and availability of the customers’ applications and data. Figure 8.2 presents one of the data processing methods adopted in Amazon’s AWS [7], which is used to transfer large amounts of data between the AWS cloud and portable storage devices.

When the user wants to upload the data, he/she stores some parameters such as AccessKeyID, DeviceID, Destination, and so on, into an import metadata file called the manifest file and then signs the manifest file and e-mails the signed manifest file to Amazon. Another metadata file named the signature file is used by AWS to describe the cipher algorithm that is adopted to encrypt the job ID and the bytes in the manifest file. The signature file can uniquely identify and authenticate the user request. The signature file is attached with the storage device, which is shipped to Amazon for efficiency. On receiving the stor-age device and the signature file, the service provider will validate the signature in the device with the manifest file sent through the email. Then, Amazon will e-mail management information back to the user including the number of bytes saved, the MD5 of the bytes, the status of the load, and the location on the Amazon S3 of the AWS Import Export Log. This log contains details about the data files that have been uploaded, including the key names, number of bytes, and MD5 checksum values.

The downloading process is similar to the uploading process. The user creates a manifest and signature file, e-mails the manifest file, and ships the storage device attached with signature file. When Amazon receives these two files, it will validate the two files, copy the data into the storage device, ship it back, and e-mail to the user with the status including the MD5 checksum of the data. Amazon claims that the maximum security is obtained via SSL endpoints.

Microsoft Windows Azure. The Windows Azure Platform (Azure) is an Internet-scale cloud services platform hosted in Microsoft data centers, which provides an operating system and a set of developer services that can be used individually or together [8]. The platform also provides scalable storage service. There are three basic data items: blobs (up to 50 GB), tables, and queues ( ,8k). In the Azure Storage, based on the blob, table, and queue structures, Microsoft promises to achieve confidentiality of the users’ data. The procedure shown in Figure 8.3 provides security for data accessing to ensure that the data will not be lost.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a Account |  |  | Secret Key |  |  | Signature |  |
|  |  |  |  |  |
| Create |  |  | Get the |  |  | Create |  |
|  |  |  |  |  |  |  |  |

|  |
| --- |
| ContentMD5 |

|  |
| --- |
| Create |

GET

PUT

|  |
| --- |
| Cloud Storage |

Data with MD5

FIGURE 8.3. Security data access procedure.

To use Windows Azure Storage service, a user needs to create a storage account, which can be obtained from the Windows Azure portal web interface. After creating an account, the user will receive a 256-bit secret key. Each time when the user wants to send the data to or fetch the data from the cloud, the user has to use his secret key to create a HMAC SHA256 signature for each individual request for identification. Then the user uses his signature to authenticate request at server. The signature is passed with each request to authenticate the user requests by verifying the HMAC signature.

The example in Figure 8.4 is a REST request for a PUT/GET block operation [10]. Content-MD5 checksums can be provided to guard against network transfer errors and data integrity. The Content-MD5 checksum in the PUT is the MD5 checksum of the data block in the request. The MD5 checksum is checked on the server. If it does not match, an error is returned. The content length specifies the size of the data block contents. There is also an authorization header inside the HTTP request header as shown above in Figure 8.4.

At the same time, if the Content-MD5 request header was set when the blob has been uploaded, it will be returned in the response header. Therefore, the user can check for message content integrity. Additionally, the secure HTTP connection is used for true data integrity [7].

Google App Engine (GAE). The Google App Engine (GAE) [9] provides a powerful distributed data storage service that features a query engine and transactions. An independent third-party auditor, who claims that GAE can be secure under the SAS70 auditing industry standard, issued Google Apps an unqualified SAS70 Type II certification. However, from its on-line storage technical document of lower API [9], there are only some functions such as GET and PUT. There is no content addressing the issues of securing storage services. The security of data storage is assumed guaranteed using techniques such as by SSL link, based on our knowledge of security method adopted by other services.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | | | | | | |  |  |
|  |  |  |  | Encrypted SDC Tunnel | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Google Apps |  |  | Tunnel Servers | Internet |  | Corporate Firewall |  | Secure Data Connector |  | Optional Firewall |  | WebService, API Server |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

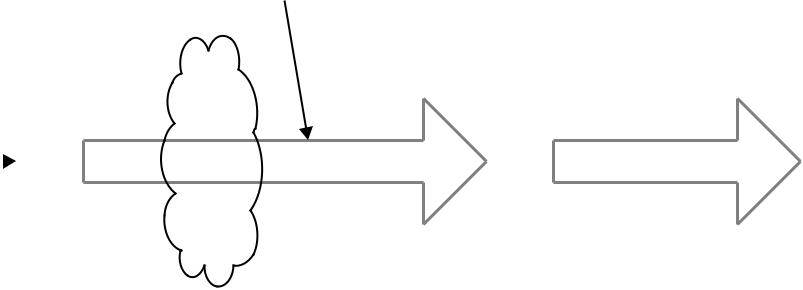


FIGURE 8.5. Illustration of Google SDC working flow.

Figure 8.5 is one of the secure services, called Google Secure Data Connector (SDC), based on GAE [9]. The SDC constructs an encrypted connection between the data source and Google Apps. As long as the data source is in the Google Apps domain to the Google tunnel protocol servers, when the user wants to get the data, he/she will first send an authorized data requests to Google Apps, which forwards the request to the tunnel server. The tunnel servers validate the request identity. If the identity is valid, the tunnel protocol allows the SDC to set up a connection, authenticate, and encrypt the data that flows across the Internet. At the same time, the SDC uses resource rules to validate whether a user is authorized to access a specified resource. When the request is valid, the SDC performs a network request. The server validates the signed request, checks the credentials, and returns the data if the user is authorized.

The SDC and tunnel server are like the proxy to encrypt connectivity between Google Apps and the internal network. Moreover, for more security, the SDC uses signed requests to add authentication information to requests that are made through the SDC. In the signed request, the user has to submit identification information including the owner\_id, viewer\_id, instance\_id, app\_id, public\_key, consumer\_key, nonce, token, and signature within the request [9] to ensure the integrity, security, and privacy of the request.

* Vulnerabilities in Current Cloud Services

Previous subsections describe three different commercial cloud computing secure data storage schemes. Storage services that accept a large amount of data ( .1 TB) normally adopt strategies that help make the shipment more convenient, just as the Amazon AWS does. In contrast, services that only

228 SECURE DISTRIBUTED DATA STORAGE IN CLOUD COMPUTING

accept a smaller data amount ( #50 GB) allow the data to be uploaded or downloaded via the Internet, just as the Azure Storage Service does. To provide data integrity, the Azure Storage Service stores the uploaded data MD5 checksum in the database and returns it to the user when the user wants to retrieve the data. Amazon AWS computes the data MD5 checksum and e-mails it to the user for integrity checking. The SDC is based on GAE’s attempt to strengthen Internet authentication using a signed request. If these services are grouped together, the following scheme can be derived.

As shown in Figure 8.6, when user\_1 stores data in the cloud, she can ship or send the data to the service provider with MD5\_1. If the data are transferred through the Internet, a signed request could be used to ensure the privacy, security, and integrity of the data. When the service provider receives the data and the MD5 checksum, it stores the data with the corresponding checksum (MD5\_1). When the service provider gets a verified request to retrieve the data from another user or the original user, it will send/ship the data with a MD5 checksum to the user. On the Azure platform, the original checksum MD5\_1will be sent, in contrast, a re-computed checksum MD5\_2 is sent on Amazon’s AWS.

The procedure is secure for each individual session. The integrity of the data during the transmission can be guaranteed by the SSL protocol applied. However, from the perspective of cloud storage services, data integrity depends on the security of operations while in storage in addition to the security of the uploading and downloading sessions. The uploading session can only ensure that the data received by the cloud storage is the data that the user uploaded; the downloading session can guarantee the data that the user retrieved is the data cloud storage recorded. Unfortunately, this procedure applied on cloud storage services cannot guarantee data integrity.

To illustrate this, let’s consider the following two scenarios. First, assume that Alice, a company CFO, stores the company financial data at a cloud storage service provided by Eve. And then Bob, the company administration chairman, downloads the data from the cloud. There are three important concerns in this simple procedure:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| USER1 | |  | USER2 | |
|  |  |  |  |  |
|  | MD5\_1 | MD5\_1/2 | |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | Cloud Service | | |
|  |  |  |  |  |



FIGURE 8.6. Illustration of potential integrity problem.

|  |  |
| --- | --- |
| 8.2 CLOUD STORAGE: FROM LANs TO WANs | 229 |

1. Confidentiality. Eve is considered as an untrustworthy third party, Alice and Bob do not want reveal the data to Eve.
2. Integrity. As the administrator of the storage service, Eve has the capability to play with the data in hand. How can Bob be confident that the data he fetched from Eve are the same as what was sent by Alice? Are there any measures to guarantee that the data have not been tampered by Eve?
3. Repudiation. If Bob finds that the data have been tampered with, is there any evidence for him to demonstrate that it is Eve who should be responsible for the fault? Similarly, Eve also needs certain evidence to prove her innocence.

Recently, a potential customer asked a question on a cloud mailing-group regarding data integrity and service reliability. The reply from the developer was “We won’t lose your data—we have a robust backup and recovery strategy — but we’re not responsible for you losing your own data . . . ” [11]. Obviously, it is not persuasive to the potential customer to be confident with the service.

The repudiation issue opens a door for potentially blackmailers when the user is malicious. Let’s assume that Alice wants to blackmail Eve. Eve is a cloud storage service provider who claims that data integrity is one of their key features. For that purpose, Alice stored some data in the cloud, and later she downloaded the data. Then, she reported that her data were incorrect and that it is the fault of the storage provider. Alice claims compensation for her so-called loss. How can the service provider demonstrate her innocence?

Confidentiality can be achieved by adopting robust encryption schemes. However, the integrity and repudiation issues are not handled well on the current cloud service platform. One-way SSL session only guarantees one-way integrity. One critical link is missing between the uploading and downloading sessions: There is no mechanism for the user or service provider to check whether the record has been modified in the cloud storage. This vulnerability leads to the following questions:

Upload-to-Download Integrity. Since the integrity in uploading and down-loading phase are handled separately, how can the user or provider know the data retrieved from the cloud is the same data that the user uploaded previously?

Repudiation Between Users and Service Providers. When data errors happen without transmission errors in the uploading and downloading sessions, how can the user and service provider prove their innocence?

1. Bridge the Missing Link

This section presents several simple ideas to bridge the missing link based on digital signatures and authentication coding schemes. According to whether

230 SECURE DISTRIBUTED DATA STORAGE IN CLOUD COMPUTING

there is a third authority certified (TAC) by the user and provider and whether the user and provider are using the secret key sharing technique (SKS), there are four solutions to bridge the missing link of data integrity between the uploading and downloading procedures. Actually, other digital signature technologies can be adopted to fix this vulnerability with different approaches.

Neither TAC nor SKS.

Uploading Session

1. User: Sends data to service provider with MD5 checksum and MD5 Signature by User (MSU).
2. Service Provider: Verifies the data with MD5 checksum, if it is valid, the service provider sends back the MD5 and MD5 Signature by Provider (MSP) to user.
3. MSU is stored at the user side, and MSP is stored at the service provider side.

Once the uploading operation finished, both sides agreed on the integrity of the uploaded data, and each side owns the MD5 checksum and MD5 signature generated by the opposite site.

Downloading Session

1. User: Sends request to service provider with authentication code.
2. Service Provider: Verifies the request identity, if it is valid, the service provider sends back the data with MD5 checksum and MD5 Signature by Provider (MSP) to user.
3. User verifies the data using the MD5 checksum.

When disputation happens, the user or the service provider can check the MD5 checksum and the signature of MD5 checksum generated by the opposite side to prove its innocence. However, there are some special cases that exist. When the service provider is trustworthy, only MSU is needed; when the user is trustworthy, only MSP is needed; if each of them trusts the other side, neither MSU nor MSP is needed. Actually, that is the current method adopted in cloud computing platforms. Essentially, this approach implies that when the identity is authenticated that trust is established.

With SKS but without TAC.

Uploading Session

1. User: Sends data to service provider with MD checksum 5.
2. Service Provider: Verifies the data with MD5 checksum, if it is valid, the service provider sends back the MD5 checksum.
3. The service provider and the user share the MD5 checksum with SKS.

|  |  |
| --- | --- |
| 8.2 CLOUD STORAGE: FROM LANs TO WANs | 231 |

Then, both sides agree on the integrity of the uploaded data, and they share the agreed MD5 checksum, which is used when disputation happens.

Downloading Session

1. User: Sends request to the service provider with authentication code.
2. Service Provider: Verifies the request identity, if it is valid, the service provider sends back the data with MD5 checksum.
3. User verifies the data through the MD5 checksum.

When disputation happens, the user or the service provider can take the shared MD5 together, recover it, and prove his/her innocence.

With TAC but without SKS.

Uploading Session

1. User: Sends data to the service provider along with MD5 checksum and MD5 Signature by User (MSU).
2. Service Provider: Verifies the data with MD5 checksum, if it is valid, the service provider sends back the MD5 checksum and MD5 Signature by Provider (MSP) to the user.
3. MSU and MSP are sent to TAC.

On finishing the uploading phase, both sides agree on the integrity of the uploaded data, and TAC owns their agreed MD5 signature.

Downloading Session

1. User: Sends request to the service provider with authentication code.
2. Service Provider: Verifies the request with identity, if it is valid, the service provider sends back the data with MD5 checksum.
3. User verifies the data through the MD5 checksum.

When disputation happens, the user or the service provider can prove his innocence by presenting the MSU and MSP stored at the TAC.

Similarly, there are some special cases. When the service provider is trustworthy, only the MSU is needed; when the user is trustworthy, only the MSP is needed; if each of them trusts the other, the TAC is not needed. Again, the last case is the method adopted in the current cloud computing platforms. When the identity is authenticated, trust is established.

With Both TAC and SKS.

Uploading Session

1. User: Sends data to the service provider with MD5 checksum.
2. Service Provider: verifies the data with MD5 checksum.
3. SECURE DISTRIBUTED DATA STORAGE IN CLOUD COMPUTING
   1. Both the user and the service provider send MD5 checksum to TAC.
   2. TAC verifies the two MD5 checksum values. If they match, the TAC distributes MD5 to the user and the service provider by SKS.

Both sides agree on the integrity of the uploaded data and share the same MD5 checksum by SKS, and the TAC own their agreed MD5 signatures.

Downloading Session

* User: Sends request to the service provider with authentication code.
* Service Provider: Verifies the request identity, if it is valid, the service provider sends back the data with MD5 checksum.
* User verifies the data through the MD5 checksum.

When disputation happens, the user or the service provider can prove their innocence by checking the shared MD5 checksum together. If the disputation cannot be resolved, they can seek further help from the TAC for the MD5 checksum.

Here are the special cases. When the service provider is trustworthy, only the user needs the MD5 checksum; when the user is trustworthy, only the service provider needs MD5 checksum; if both of them can be trusted, the TAC is not needed. This is the method used in the current cloud computing platform.

* TECHNOLOGIES FOR DATA SECURITY IN CLOUD COMPUTING

This section presents several technologies for data security and privacy in cloud computing. Focusing on the unique issues of the cloud data storage platform, this section does not repeat the normal approaches that provide confidentiality, integrity, and availability in distributed data storage applications. Instead, we select to illustrate the unique requirements for cloud computing data security from a few different perspectives:

Database Outsourcing and Query Integrity Assurance. Researchers have pointed out that storing data into and fetching data from devices and machines behind a cloud are essentially a novel form of database outsourcing. Section 8.3.1 introduces the technologies of Database Out-sourcing and Query Integrity Assurance on the clouding computing platform.

Data Integrity in Untrustworthy Storage. One of the main challenges that prevent end users from adopting cloud storage services is the fear of losing data or data corruption. It is critical to relieve the users’ fear by providing technologies that enable users to check the integrity of their data. Section 8.3.2 presents two approaches that allow users to detect whether the data has been touched by unauthorized people.

|  |  |
| --- | --- |
|  |  |

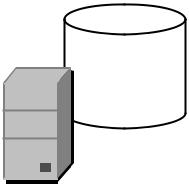
Web-Application-Based Security. Once the dataset is stored remotely, a Web browser is one of the most convenient approaches that end users can use to access their data on remote services. In the era of cloud computing, Web security plays a more important role than ever. Section 8.3.3 discusses the most important concerns in Web security and analyzes a couple of widely used attacks.

Multimedia Data Security. With the development of high-speed network technologies and large bandwidth connections, more and more multi-media data are being stored and shared in cyber space. The security requirements for video, audio, pictures, or images are different from other applications. Section 8.3.4 introduces the requirements for multimedia data security in the cloud.

* Database Outsourcing and Query Integrity Assurance

In recent years, database outsourcing has become an important component of cloud computing. Due to the rapid advancements in network technology, the cost of transmitting a terabyte of data over long distances has decreased significantly in the past decade. In addition, the total cost of data management is five to ten times higher than the initial acquisition costs. As a result, there is a growing interest in outsourcing database management tasks to third parties that can provide these tasks for a much lower cost due to the economy of scale. This new outsourcing model has the benefits of reducing the costs for running Database Management Systems (DBMS) independently and enabling enter-prises to concentrate on their main businesses [12]. Figure 8.7 demonstrates the general architecture of a database outsourcing environment with clients. The database owner outsources its data management tasks, and clients send queries to the untrusted service provider. Let T denote the data to be outsourced. The data T are is preprocessed, encrypted, and stored at the service provider. For evaluating queries, a user rewrites a set of queries Q against T to queries against the encrypted database.

The outsourcing of databases to a third-party service provider was first introduced by Hacigu¨mu¨s et al. [13]. Generally, there are two security concerns



queryRewrite(Q)

DB dataTransform(T)



Clients Query Results

Database Owner

Service Provider

FIGURE 8.7. The system architecture of database outsourcing.

in database outsourcing. These are data privacy and query integrity. The related research is outlined below.

Data Privacy Protection. Hacigu¨mu¨s et al. [37] proposed a method to execute SQL queries over encrypted databases. Their strategy is to process as much of a query as possible by the service providers, without having to decrypt the data. Decryption and the remainder of the query processing are performed at the client side. Agrawal et al. [14] proposed an order-preserving encryption scheme for numeric values that allows any comparison operation to be directly applied on encrypted data. Their technique is able to handle updates, and new values can be added without requiring changes in the encryption of other values. Generally, existing methods enable direct execution of encrypted queries on encrypted datasets and allow users to ask identity queries over data of different encryptions. The ultimate goal of this research direction is to make queries in encrypted databases as efficient as possible while preventing adversaries from learning any useful knowledge about the data. However, researches in this field did not consider the problem of query integrity.

Query Integrity Assurance. In addition to data privacy, an important security concern in the database outsourcing paradigm is query integrity. Query integrity examines the trustworthiness of the hosting environment. When a client receives a query result from the service provider, it wants to be assured that the result is both correct and complete, where correct means that the result must originate in the owner’s data and not has been tampered with, and complete means that the result includes all records satisfying the query. Devanbu et al. [15] authenticate data records using the Merkle hash tree [16], which is based on the idea of using a signature on the root of the Merkle hash tree to generate a proof of correctness. Mykletun et al. [17] studied and compared several signature methods that can be utilized in data authentication, and they identified the problem of completeness but did not provide a solution. Pang et al. [18] utilized an aggregated signature to sign each record with the information from neighboring records by assuming that all the records are sorted with a certain order. The method ensures the completeness of a selection query by checking the aggregated signature. But it has difficulties in handling multipoint selection query of which the result tuples occupy a noncontinuous region of the ordered sequence.

The work in Li et al. [19] utilizes Merkle hash tree-based methods to audit the completeness of query results, but since the Merkle hash tree also applies the signature of the root Merkle tree node, a similar difficulty exists. Besides, the network and CPU overhead on the client side can be prohibitively high for some types of queries. In some extreme cases, the overhead could be as high as processing these queries locally, which can undermine the benefits of database outsourcing. Sion [20] proposed a mechanism called the challenge token and uses it as a probabilistic proof that the server has executed the query over the entire database. It can handle arbitrary types of queries including joins and

|  |  |
| --- | --- |
|  |  |

does not assume that the underlying data is ordered. However, the approach is not applied to the adversary model where an adversary can first compute the complete query result and then delete the tuples specifically corresponding to the challenge tokens [21]. Besides, all the aforementioned methods must modify the DBMS kernel in order to provide proof of integrity.

Recently, Wang et al. [22] proposed a solution named dual encryption to ensure query integrity without requiring the database engine to perform any special function beyond query processing. Dual encryption enables cross-examination of the outsourced data, which consist of (a) the original data stored under a certain encryption scheme and (b) another small percentage of the original data stored under a different encryption scheme. Users generate queries against the additional piece of data and analyze their results to obtain integrity assurance.

For auditing spatial queries, Yang et al [23] proposed the MR-tree, which is an authenticated data structure suitable for verifying queries executed on outsourced spatial databases. The authors also designed a caching technique to reduce the information sent to the client for verification purposes. Four spatial transformation mechanisms are presented in Yiu et al. [24] for protect-ing the privacy of outsourced private spatial data. The data owner selects transformation keys that are shared with trusted clients, and it is infeasible to reconstruct the exact original data points from the transformed points without the key. However, both aforementioned researches did not consider data privacy protection and query integrity auditing jointly in their design. The state-of-the-art technique that can ensure both privacy and integrity for outsourced spatial data is proposed in Ku et al. [12]. In particular, the solution first employs a one-way spatial transformation method based on Hilbert curves, which encrypts the spatial data before outsourcing and hence ensures its privacy. Next, by probabilistically replicating a portion of the data and encrypting it with a different encryption key, the authors devise a mechanism for the client to audit the trustworthiness of the query results.

1. Data Integrity in Untrustworthy Storage

While the transparent cloud provides flexible utility of network-based resources, the fear of loss of control on their data is one of the major concerns that prevent end users from migrating to cloud storage services. Actually it is a potential risk that the storage infrastructure providers become self-interested, untrustworthy, or even malicious. There are different motivations whereby a storage service provider could become untrustworthy—for instance, to cover the consequence of a mistake in operation, or deny the vulnerability in the system after the data have been stolen by an adversary. This section introduces two technologies to enable data owners to verify the data integrity while the files are stored in the remote untrustworthy storage services.

Actually, before the term “cloud computing” appears as an IT term, there are several remote data storage checking protocols that have been suggested [25], [26]. Later research has summarized that in practice a remote data

possession checking protocol has to satisfy the following five requirements [27]. Note that the verifier could be either the data owner or a trusted third party, and the prover could be the storage service provider or storage medium owner or system administrator.

Requirement #1. It should not be a pre-requirement that the verifier has to possess a complete copy of the data to be checked. And in practice, it does not make sense for a verifier to keep a duplicated copy of the content to be verified. As long as it serves the purpose well, storing a more concise contents digest of the data at the verifier should be enough.

Requirement #2. The protocol has to be very robust considering the untrustworthy prover. A malicious prover is motivated to hide the viola-tion of data integrity. The protocol should be robust enough that such a prover ought to fail in convincing the verifier.

Requirement #3. The amount of information exchanged during the verification operation should not lead to high communication overhead.

Requirement #4. The protocol should be computationally efficient.

Requirement #5. It ought to be possible to run the verification an unlimited number of times.

A PDP-Based Integrity Checking Protocol. Ateniese et al. [28] proposed a protocol based on the provable data procession (PDP) technology, which allows users to obtain a probabilistic proof from the storage service providers. Such a proof will be used as evidence that their data have been stored there. One of the advantages of this protocol is that the proof could be generated by the storage service provider by accessing only a small portion of the whole dataset. At the same time, the amount of the metadata that end users are required to store is also small—that is, O(1). Additionally, such a small amount data exchanging procedure lowers the overhead in the communication channels too.

Figure 8.8 presents the flowcharts of the protocol for provable data possession [28]. The data owner, the client in the figure, executes the protocol to verify that a dataset is stored in an outsourced storage machine as a collec-tion of n blocks. Before uploading the data into the remote storage, the data owner pre-processes the dataset and a piece of metadata is generated. The metadata are stored at the data owner’s side, and the dataset will be transmitted to the storage server. The cloud storage service stores the dataset and sends the data to the user in responding to queries from the data owner in the future.

As part of pre-processing procedure, the data owner (client) may conduct operations on the data such as expanding the data or generating additional metadata to be stored at the cloud server side. The data owner could execute the PDP protocol before the local copy is deleted to ensure that the uploaded copy has been stored at the server machines successfully. Actually, the data owner may encrypt a dataset before transferring them to the storage machines. During the time that data are stored in the cloud, the data owner can generate a

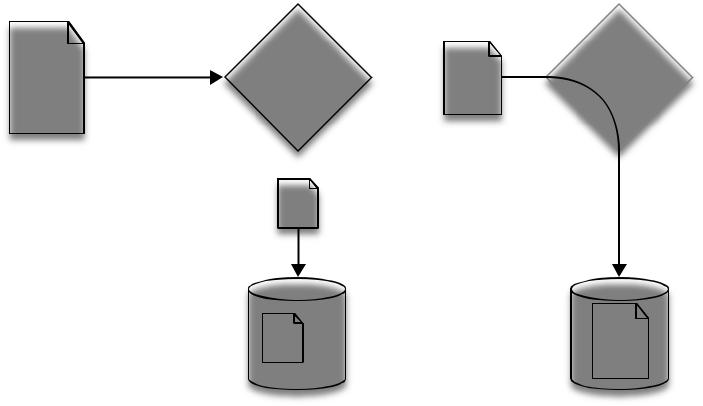
|  |  |
| --- | --- |
|  |  |

Input file

*F*

Client generates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| metadata (*m*) and | | |  | No server |  |
|  | processing |  |
| modified file (*F* ) | | |  |  |
|  |  |  |
| Client | |  | *F* | Server |  |
|  |  |
|  | | |  |  |  |
|  |  | |  |  |  |
| *m* | | |  |  |  |



|  |  |
| --- | --- |
| *m* | *F* |
| Client store | Server store |

1. Pre-process and store
   1. Client generates a random challenge *R*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *R* |  | (2) Server computes |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | proof of possession *P* |  |
|  | 0/1 |  | Client |  | *P* |  | Server |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| (3) Client verifies | | | |  |  |  |  |  |
| server’s proof | | |  |  |  |  |  |  |
|  |  |  | *m* |  |  |  | *F* |  |
|  |  |  | *m* |  |  |  | *F* |  |
|  |  |  | Client store |  |  |  | Server store |  |

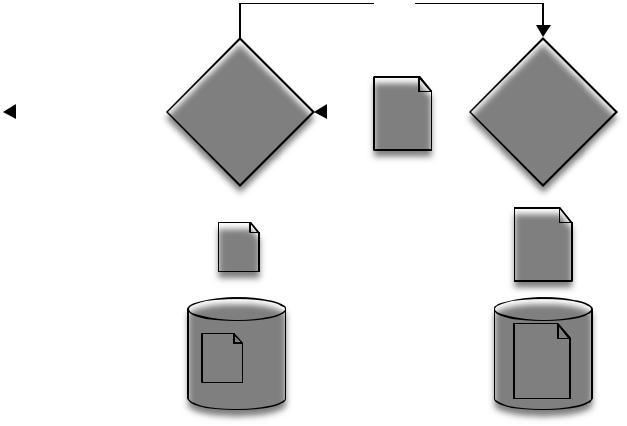
(b) Verify server possession

FIGURE 8.8. Protocol for provable data possession [28].

“challenge” and send it to the service provider to ensure that the storage server has stored the dataset. The data owner requests that the storage server generate a metadata based on the stored data and then send it back. Using the previously stored local metadata, the owner verifies the response.

On the behalf of the cloud service provider’s side, the server may receive multiple challenges from different users at the same time. For the sake of availability, it is highly desired to minimize not only the computational

overhead of each individual calculation, but also the number of data blocks to be accessed. In addition, considering the pressure on the communication networks, minimal bandwidth consumption also implies that there are a limited amount of metadata included in the response generated by the server. In the protocol shown in Figure 8.8, the PDP scheme only randomly accesses one subdata block when the sample the stored dataset [28]. Hence, the PDP scheme probabilistically guarantees the data integrity. It is mandatory to access the whole dataset if a deterministic guarantee is required by the user.

An Enhanced Data Possession Checking Protocol. Sebe et al. [27] pointed out that the above PDP-based protocol does not satisfy Requirement #2 with 100% probability. An enhanced protocol has been proposed based on the idea of the Diffie Hellman scheme. It is claimed that this protocol satisfies all five requirements and is computationally more efficient than the PDP-based protocol [27]. The verification time has been shortened at the setup stage by taking advantage of the trade-offs between the computation times required by the prover and the storage required at the verifier. The setup stage sets the following parameters:

p and q : two primary factors chosen by the verifier;

N 5 pq: a public RSA modulus created by the verifier;

Φ(N) 5 (p 2 1)(q 2 1): the private key of the verifier, which is the secret only known by the verifier;

l: an integer that is chosen depending on the trade-offs between the computation time required at the prover and the storage required at the verifier;

t: a security parameter;

PRNG: a pseudorandom number generator, which generates t-bit integer values.

The protocol is presented as follows:

At first, the verifier generates the digest of data m:

|  |  |  |
| --- | --- | --- |
| 1. | Break the data m into n pieces, each is | l-bit. Let m1, m2, . . . , mn |
|  | (n ¼ djmj=le) be the integer values corresponding to fragments of m. | |
| 2. | For each fragment mi, compute and store Mi | 5 mi mod Φ(N). |

The challenge response verification protocol is as follows:

1. The verifier
   1. generates a random seed S and a random element α A ZN \{1, N 1} and
   2. sends the challenge (α, S) to the prover.

|  |  |
| --- | --- |
|  |  |

1. Upon receiving the challenge, the prover:
   1. generates n pseudorandom values ci A [1,2t], for i 5 1 to n, using PRNG seeded by S,
   2. calculates r ¼ Pni¼1 cimi and R 5 αr mod N, and
   3. sends R to the verifier.
2. The verifier:
   1. regenerates the n pseudorandom values ci A [1,2t], for i 5 1 to n, using PRNG seeded by S,
   2. calculates r0 ¼ Pni¼1 cimi mod Φ(N) and R ’ 5 αr’ mod N, and
   3. checks whether R 5 R’.

Due to the space constraints, this section only introduces the basic princi-ples and the working flows of the protocols for data integrity checking in untrustworthy storages. The proof of the correctness, security analysis, and the performance analysis of the protocols are left for the interested readers to explore deeper in the cited research papers [25, 26 28].

1. Web-Application-Based Security

In cloud computing environments, resources are provided as a service over the Internet in a dynamic, virtualized, and scalable way [29, 30]. Through cloud computing services, users access business applications on-line from a Web browser, while the software and data are stored on the servers. Therefore, in the era of cloud computing, Web security plays a more important role than ever. The Web site server is the first gate that guards the vast cloud resources. Since the cloud may operate continuously to process millions of dollars’ worth of daily on-line transactions, the impact of any Web security vulnerability will be amplified at the level of the whole cloud.

Web attack techniques are often referred as the class of attack. When any Web security vulnerability is identified, attacker will employ those techniques to take advantage of the security vulnerability. The types of attack can be categorized in Authentication, Authorization, Client-Side Attacks, Comm-and Execution, Information Disclosure, and Logical Attacks [31]. Due to the limited space, this section introduces each of them briefly. Interested read-ers are encouraged to explore for more detailed information from the materials cited.

Authentication. Authentication is the process of verifying a claim that a subject made to act on behalf of a given principal. Authentication attacks target a Web site’s method of validating the identity of a user, service, or application, including Brute Force, Insufficient Authentication, and Weak Password Recovery Validation. Brute Force attack employs an automated process to guess a person’s username and password by trial and error. In the Insufficient Authentication case, some sensitive content or functionality are protected by

“hiding” the specific location in obscure string but still remains accessible directly through a specific URL. The attacker could discover those URLs through a Brute Force probing of files and directories. Many Web sites provide password recovery service. This service will automatically recover the user name or password to the user if she or he can answer some questions defined as part of the user registration process. If the recovery questions are either easily guessed or can be skipped, this Web site is considered to be Weak Password Recovery Validation.

Authorization. Authorization is used to verify if an authenticated subject can perform a certain operation. Authentication must precede authorization. For example, only certain users are allowed to access specific content or functionality.

Authorization attacks use various techniques to gain access to protected areas beyond their privileges. One typical authorization attack is caused by Insufficient Authorization. When a user is authenticated to a Web site, it does not necessarily mean that she should have access to certain content that has been granted arbitrarily. Insufficient authorization occurs when a Web site does not protect sensitive content or functionality with proper access control restrictions. Other authorization attacks are involved with session. Those attacks include Credential/Session Prediction, Insufficient Session Expiration, and Session Fixation.

In many Web sites, after a user successfully authenticates with the Web site for the first time, the Web site creates a session and generate a unique “session ID” to identify this session. This session ID is attached to subsequent requests to the Web site as “Proof” of the authenticated session.

Credential/Session Prediction attack deduces or guesses the unique value of a session to hijack or impersonate a user.

Insufficient Session Expiration occurs when an attacker is allowed to reuse old session credentials or session IDs for authorization. For example, in a shared computer, after a user accesses a Web site and then leaves, with Insufficient Session Expiration, an attacker can use the browser’s back button to access Web pages previously accessed by the victim.

Session Fixation forces a user’s session ID to an arbitrary value via Cross-Site Scripting or peppering the Web site with previously made HTTP requests. Once the victim logs in, the attacker uses the predefined session ID value to impersonate the victim’s identity.

Client-Side Attacks. The Client-Side Attacks lure victims to click a link in a malicious Web page and then leverage the trust relationship expectations of the victim for the real Web site. In Content Spoofing, the malicious Web page can trick a user into typing user name and password and will then use this information to impersonate the user.

Cross-Site Scripting (XSS) launches attacker-supplied executable code in the victim’s browser. The code is usually written in browser-supported scripting

|  |  |
| --- | --- |
|  |  |

languages such as JavaScript, VBScript, ActiveX, Java, or Flash. Since the code will run within the security context of the hosting Web site, the code has the ability to read, modify, and transmit any sensitive data, such as cookies, accessible by the browser.

Cross-Site Request Forgery (CSRF) is a serve security attack to a vulnerable site that does not take the checking of CSRF for the HTTP/HTTPS request. Assuming that the attacker knows the URLs of the vulnerable site which are not protected by CSRF checking and the victim’s browser stores credentials such as cookies of the vulnerable site, after luring the victim to click a link in a malicious Web page, the attacker can forge the victim’s identity and access the vulnerable Web site on victim’s behalf.

Command Execution. The Command Execution attacks exploit server-side vulnerabilities to execute remote commands on the Web site. Usually, users supply inputs to the Web-site to request services. If a Web application does not properly sanitize user-supplied input before using it within application code, an attacker could alter command execution on the server. For example, if the length of input is not checked before use, buffer overflow could happen and result in denial of service. Or if the Web application uses user input to construct statements such as SQL, XPath, C/C11 Format String, OS system command, LDAP, or dynamic HTML, an attacker may inject arbitrary executable code into the server if the user input is not properly filtered.

Information Disclosure. The Information Disclosure attacks acquire sensi-tive information about a web site revealed by developer comments, error messages, or well-know file name conventions. For example, a Web server may return a list of files within a requested directory if the default file is not present. This will supply an attacker with necessary information to launch further attacks against the system. Other types of Information Disclosure includes using special paths such as “.” and “..” for Path Traversal, or uncovering hidden URLs via Predictable Resource Location.

Logical Attacks. Logical Attacks involve the exploitation of a Web applica-tion’s logic flow. Usually, a user’s action is completed in a multi-step process. The procedural workflow of the process is called application logic. A common Logical Attack is Denial of Service (DoS). DoS attacks will attempt to consume all available resources in the Web server such as CPU, memory, disk space, and so on, by abusing the functionality provided by the Web site. When any one of any system resource reaches some utilization threshold, the Web site will no long be responsive to normal users. DoS attacks are often caused by Insufficient Anti-automation where an attacker is permitted to automate a process repeatedly. An automated script could be executed thousands of times a minute, causing potential loss of performance or service.

1. Multimedia Data Security Storage

With the rapid developments of multimedia technologies, more and more multimedia contents are being stored and delivered over many kinds of devices, databases, and networks. Multimedia Data Security plays an important role in the data storage to protect multimedia data. Recently, how storage multimedia contents are delivered by both different providers and users has attracted much attentions and many applications. This section briefly goes through the most critical topics in this area.

Protection from Unauthorized Replication. Contents replication is requi-red to generate and keep multiple copies of certain multimedia contents. For example, content distribution networks (CDNs) have been used to manage content distribution to large numbers of users, by keeping the replicas of the same contents on a group of geographically distributed surrogates [32, 33]. Although the replication can improve the system performance, the unauthor-ized replication causes some problems such as contents copyright, waste of replication cost, and extra control overheads.

Protection from Unauthorized Replacement. As the storage capacity is limited, a replacement process must be carried out when the capacity exceeds its limit. It means the situation that a currently stored content [34] must be removed from the storage space in order to make space for the new coming content. However, how to decide which content should be removed is very important. If an unauthorized replacement happens, the content which the user doesn’t want to delete will be removed resulting in an accident of the data loss. Furthermore, if the important content such as system data is removed by unauthorized replacement, the result will be more serious.

Protection from Unauthorized Pre-fetching. The Pre-fetching is widely deployed in Multimedia Storage Network Systems between server databases and end users’ storage disks [35]. That is to say, If a content can be predicted to be requested by the user in future requests, this content will be fetched from the server database to the end user before this user requests it, in order to decrease user response time. Although the Pre-fetching shows its efficiency, the un-authorized pre-fetching should be avoided to make the system to fetch the necessary content.

1. OPEN QUESTIONS AND CHALLENGES

Almost all the current commercial cloud service providers claim that their platforms are secure and robust. On one hand, they adopt robust cipher algorithms for confidentiality of stored data; on the other hand, they depend on network communication security protocols such as SSL, IPSec, or others to protect data in transmission in the network. For the service availability and high performance, they choose virtualization technologies and apply strong authentication and authorization schemes in their cloud domains. However, as a new infrastructure/platform leading to new application/service models of the future’s IT industry, the requirement for a security cloud computing is different from the traditional security problems. As pointed out by Dr. K. M. Khan [4]:

Encryption, digital signatures, network security, firewalls, and the isolation of virtual environments all are important for cloud computing security, but these alone won’t make cloud computing reliable for consumers.

1. Concerns at Different Levels

The cloud computing environment consists of three levels of abstractions [4]:

The cloud infrastructure providers, which is at the back end, own and manage the network infrastructure and resources including hardware devices and system software.

The cloud service providers, which offer services such as on-demand computing, utility computing, data processing, software services, and platforms for developing application software.

The cloud consumers, which is at the front end of the cloud computing environment and consists of two major categories of users: (a) application developers, who take advantage of the hardware infrastructure and the software platforms to construct application software for ultimate end users; and (b) end users, who carry out their daily works using the on-demand computing, software services, and utility services.

Regarding data/information security, the users at different levels have variant expectations and concerns due to the roles they play in the data’s life cycle.

From the perspective of cloud consumers, normally who are the data owners, the concerns are essentially raised from the loss of control when the data are in a cloud. As the dataset is stored in unknown third-party infrastructure, the owner loses not only the advantages of endpoint restrictions and management, but also the fine-grained credential quality control. The uncertainty about the privacy and the doubt about the vulnerability are also resulted from the disappearing physical and logical network boundaries [36]. The main security concerns of the end users include confidentiality, loss of control of data, and the undisclosed security profiles of the cloud service and infrastructure providers. The users’ data are transmitted between the local machine and cloud service provider for variant operations, and they are also persistently stored in the cloud infrastructure provider’s facilities. During this procedure, data might not be adequately protected while they are being moved within the systems or across multiple sites owned by these providers. The data owner also cannot check the security assurances before using the service from the cloud, because the actual security capabilities associated with the providers are transparent to the user/owner.

The problem becomes more complicated when the service and infrastructure providers are not the same, and this implies additional communication links in the chain. Involving a third party in the services also introduces an additional vector of attack. Actually, in practice there are more challenging scenarios. For instance, consider that multiple end users have different sets of security requirements while using the same service offered by an individual cloud service provider. To handle such kind of complexity, one single set of security provisions does not fit all in cloud computing. The scenarios also imply that the back-end infrastructure and/or service providers must be capable of supporting multiple levels requirements of security similar to those guaranteed by front-end service provider.

From the perspective of the cloud service providers, the main concern with regard to protecting users’ data is the transfer of data from devices and servers within the control of the users to its own devices and subsequently to those of the cloud infrastructure, where the data is stored. The data are stored in cloud service provider’s devices on multiple machines across the entire virtual layer. The data are also hosted on devices that belong to infrastructure provider. The cloud service provider needs to ensure users that the security of their data is being adequately addressed between the partners, that their virtual environ-ments are isolated with sufficient protection, and that the cleanup of outdated images is being suitably managed at its site and cloud infrastructure provider’s storage machines.

Undoubtedly, the cloud infrastructure providers’ security concerns are not less than those of end users or cloud service providers. The infrastructure provider knows that a single point of failure in its infrastructure security mechanisms would allow hackers to take out thousands of data bytes owned by the clients, and most likely data owned by other enterprises. The cloud infrastructure providers need to ask the following questions:

How are the data stored in its physical devices protected?

How does the cloud infrastructure manage the backup of data, and the destruction of outdated data, at its site?

How can the cloud infrastructure control access to its physical devices and the images stored on those devices?