Force.com, Salesforce.com's developer platform for its SaaS offerings, described in the next section. Force.com is an example of an add-on development environment.

A developer might write an application in a programming language like Python using the Google API. The vendor of the PaaS solution is in most cases the developer, who is offering a complete solution to the customer. Google itself also serves as a PaaS vendor within this system, because it offers many of its Web service applications to customers as part of this service model. You can think of Google Maps, Google Earth, Gmail, and the myriad of other PaaS offerings as conforming to the PaaS service model, although these applications themselves are offered to customers under what is more aptly described as the Software as a Service (SaaS) model that is described below.

The difficulty with PaaS is that it locks the developer (and the customer) into a solution that is dependent upon the platform vendor. An application written in Python against Google's API using the Google App Engine is likely to work only in that environment. There is considerable vendor lock-in associated with a PaaS solution.

Defining Software as a Service (SaaS)

The most complete cloud computing service model is one in which the computing hardware and software, as well as the solution itself, are provided by a vendor as a complete service offering. It is referred to as the Software as a Service (SaaS) model. SaaS provides the complete infrastructure, software, and solution stack as the service offering. A good way to think about SaaS is that it is the cloud-based equivalent of shrink-wrapped software.

Software as a Service (SaaS) may be succinctly described as software that is deployed on a hosted service and can be accessed globally over the Internet, most often in a browser. With the exception of the user interaction with the software, all other aspects of the service are abstracted away.

Every computer user is familiar with SaaS systems, which are either replacements or substitutes for locally installed software. Examples of SaaS software for end-users are Google Gmail and Calendar, QuickBooks online, Zoho Office Suite, and others that are equally well known. SaaS applications come in all shapes and sizes, and include custom software such as billing and invoicing systems, Customer Relationship Management (CRM) applications, Help Desk applications, Human Resource (HR) solutions, as well as myriad online versions of familiar applications.

Many people believe that SaaS software is not customizable, and in many SaaS applications this is indeed the case. For user-centric applications such as an office suite, that is mostly true; those suites allow you to set only options or preferences. However, many other SaaS solutions expose Application Programming Interfaces (API) to developers to allow them to create custom composite applications. These APIs may alter the security model used, the data schema, workflow characteristics, and other fundamental features of the service's expression as experienced by the user. Examples of an SaaS platformwith an exposed API are Salesforce.com and Quicken.com. So SaaS does not necessarily mean that the software is static or monolithic.

SaaS characteristics

All Software as a Service (SaaS) applications share the following characteristics:

1. The software is available over the Internet globally through a browser on demand.
2. The typical license is subscription-based or usage-based and is billed on a recurring basis.

   In a small number of cases a flat fee may be changed, often coupled with a maintenance fee. Table 4.1 shows how different licensing models compare.
3. The software and the service are monitored and maintained by the vendor, regardless of where all the different software components are running.

   There may be executable client-side code, but the user isn't responsible for maintaining that code or its interaction with the service.
4. Reduced distribution and maintenance costs and minimal end-user system costs generally make SaaS applications cheaper to use than their shrink-wrapped versions.
5. Such applications feature automated upgrades, updates, and patch management and much faster rollout of changes.
6. SaaS applications often have a much lower barrier to entry than their locally installed competitors, a known recurring cost, and they scale on demand (a property of cloud computing in general).
7. All users have the same version of the software so each user's software is compatible with another's.
8. SaaS supports multiple users and provides a shared data model through a single-instance, multi-tenancy model.

   The alternative of software virtualization of individual instances also exists, but is less common.

TABLE 4.1

Shrink-Wrapped versus SaaS Licensing

	Shrink-Wrapped Software	Hybrid Model	SaaS
Licensing	Owned	Subscription (flat fee)	Metered subscription
Location	Locally installed	Available through an application	Cloud based
Management	Local IT staff	Application Service Provider (ASP)	Cloud vendor through a Service Level Agreement (SLA)

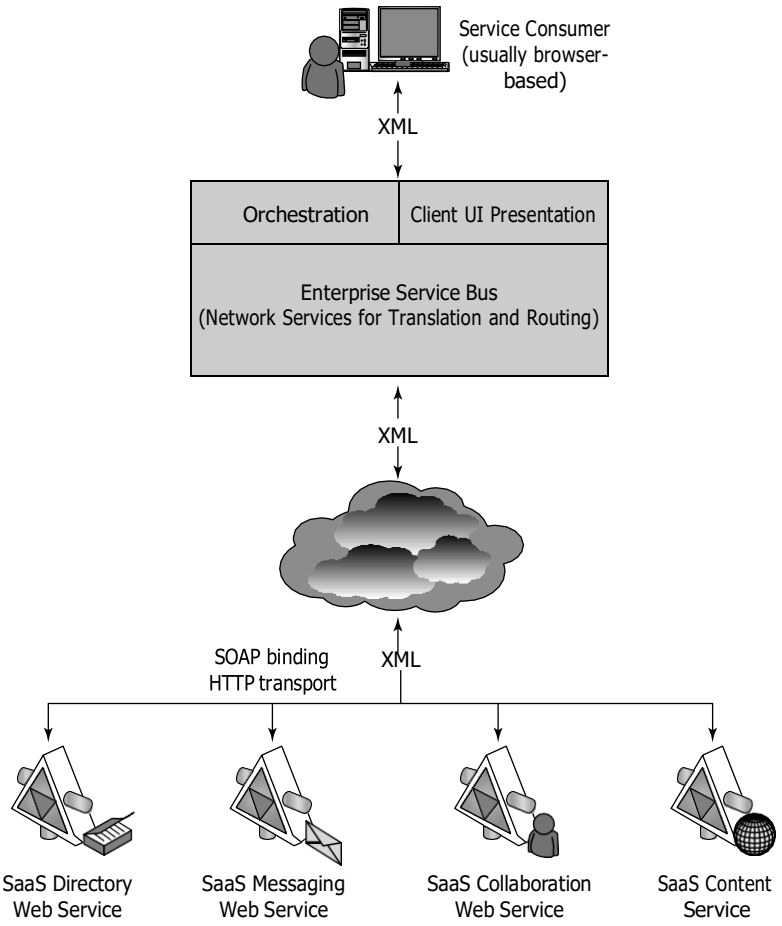
Open SaaS and SOA

A considerable amount of SaaS software is based on open source software. When open source software is used in a SaaS, you may hear it referred to as *Open SaaS*. The advantages of using open source software are that systems are much cheaper to deploy because you don't have to purchase the operating system or software, there is less vendor lock-in, and applications are more portable. The popularity of open source software, from Linux to APACHE, MySQL, and Perl (the LAMP platform) on the Internet, and the number of people who are trained in open source software make Open SaaS an attractive proposition. The impact of Open SaaS will likely translate into better profitability for the companies that deploy open source software in the cloud, resulting in lower development costs and more robust solutions.

A mature SaaS implementation based on SOA is shown in Figure 4.4.

FIGURE 4.4

A modern implement of SaaS using an Enterprise Service Bus and architected with SOA components



The componentized nature of SaaS solutions enables many of these solutions to support a feature called mashups. A mashup is an application that can display a Web page that shows data and supports features from two or more sources. Annotating a map such as Google maps is an example of a mashup. Mashups are considered one of the premier examples of Web 2.0, and that is technology's ability to support social network systems.

A mashup requires three separate components:

- An interactive user interface, which is usually created with HTML/XHTML, Ajax, JavaScript, or CSS.
- Web services that can be accessed using an API, and whose data can be bound and transported by Web service protocols such as SOAP, REST, XML/HTTP, XML/RPC, and JSON/RPC.
- Data transfer in the form of XML, KML (Keyhole Markup Language), JSON (JavaScript Object Notation), or the like.

Mashups are an incredibly useful hybrid Web application, one that SaaS is a great enabler for. The Open Mashup Alliance (OMA; see `http://www.openmashup.org/`) is a non-profit industry group dedicated to supporting technologies that implement enterprise mashups. This group supports the developing standard, the Enterprise Mashup Markup Language (EMML), which is a Domain Specific Language (DSL). This group predicts that the use of mashups will grow by a factor of 10 within just a few years.

Gartner Group predicts that approximately 25 percent of all software sold by 2011 will use the SAAS model, offered either by vendors or an intermediary party, sometimes referred to as an *aggregator*. An aggregator bundles SaaS applications from different vendors and presents them as part of a unified platform or solution.

Note

There's a notion that SaaS will eventually replace all locally installed software. However, owning software has several advantages that greatly inhibit this trend: You aren't exposed to the risk of an SaaS company going out of business; you aren't dependent upon the vagaries of an Internet connection; you aren't subject to the often much slower processing speed of a distributed computing model (compared to your local system).

I believe the introduction of SaaS applications will eventually drive down the price of major applications such as Microsoft Office and Adobe Photoshop because the most important functionality contained in commercial packages becomes available at equal quality online for a much lower cost. ■

SaaS examples abound, and while many SaaS offerings are based on proprietary software, a cloud computing service is required to be highly interoperable with other services and therefore easily replaced by a newer or better version of that software's function. When you think about this difference, visualize the situation involved for large enterprise applications such as CRM and ERP, which involve large application suites containing multiple interacting data stores. SaaS software based on the Service Oriented Architecture described in Chapter 13 has the potential of decoupling this integration, with all the attendant benefits.

Salesforce.com and CRM SaaS

Perhaps the best-known example of Software as a Service (SaaS) is the Customer Relationship Management software offered by Salesforce.com whose solution offers sales, service, support, marketing, content, analytical analysis, and even collaboration through a platform called Chatter. Salesforce.com was founded in 1999 by a group of Oracle executives and early adopters of many of the technologies that are becoming cloud computing staples.

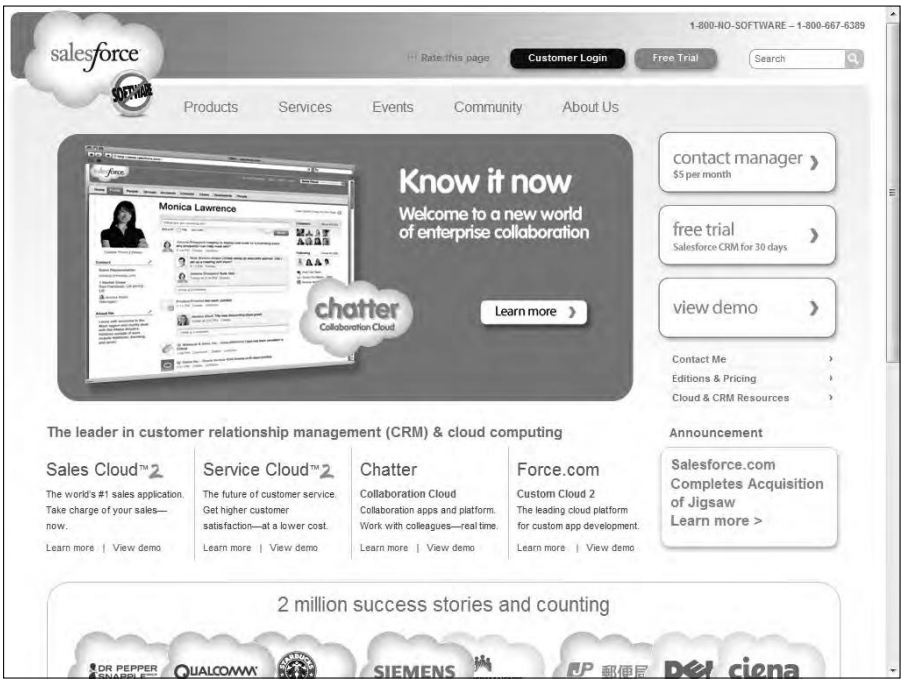
Note

Sometimes people refer to this type of software as CaaS or CRaaS for Customer Relationship as a Service software. In cases where many different kinds of relationship data are maintained as a service, you might also see those types of services referred to as XaaS. ■

Salesforce.com extended its SaaS offering to allow developers to create add-on applications, essentially turning the SaaS service into a Platform as a Service (PaaS) offering called the Force.com Platform. Applications built on Force.com are in the form of the Java variant called Apex using an XML syntax for creating user interfaces in HTML, Ajax, and Flex. Nearly a thousand applications now exist for this platform from hundreds of vendors. Figure 4.5 shows Salesforce.com's home page.

FIGURE 4.5

Salesforce.com is the largest SaaS provider of CRM software and a pioneer in this type of cloud computing software. This is the company's home page.



Defining Identity as a Service (IDaaS)

The establishment and proof of an identity is a central network function. An identity service is one that stores the information associated with a digital entity in a form that can be queried and managed for use in electronic transactions. Identity services have as their core functions: a data store, a query engine, and a policy engine that maintains data integrity.

Distributed transaction systems such as internetworks or cloud computing systems magnify the difficulties faced by identity management systems by exposing a much larger attack surface to an intruder than a private network does. Whether it is network traffic protection, privileged resource access, or some other defined right or privilege, the validated authorization of an object based on its identity is the central tenet of secure network design. In this regard, establishing identity may be seen as the key to obtaining trust and to anything that an object or entity wants to claim ownership of.

Services that provide digital identity management as a service have been part of internetworked systems from Day One. Like so many concepts in cloud computing, IDentity as a Service is a FLAVor (Four Letter Acronym) of the month, applied to services that already exist. The Domain Name Service can run on a private network, but is at the heart of the Internet as a service that provides identity authorization and lookup. The name servers that run the various Internet domains (.COM, .ORG, .EDU, .MIL, .TV, .RU, and so on) *are* IDaaS servers. DNS establishes the identity of a domain as belonging to a set of assigned addresses, associated with an owner and that owner's information, and so forth. If the identification is the assigned IP number, the other properties are its metadata.

You can categorize a myriad of services as IDaaS that run in the cloud. However, when most experts in the area of IDaaS define an identity service, they narrow the definition so the service must operate as a component according to the rules of a Service Oriented Architecture, as is defined in Chapter 13. This narrower definition restricts IDaaS to newer software services, services that interoperate, and therefore services that are standards based. It's best to keep this narrower definition in mind when you discuss IDaaS in a modern context.

What is an identity?

An identity is a set of characteristics or traits that make something recognizable or known. In computer network systems, it is one's digital identity that most concerns us. A digital identity is those attributes and metadata of an object along with a set of relationships with other objects that makes an object identifiable. Not all objects are unique, but by definition a digital identity must be unique, if only trivially so, through the assignment of a unique identification attribute. An identity must therefore have a context in which it exists.

This description of an identity as an object with attributes and relationships is one that programmer's would recognize. Databases store information and relationships in tables, rows, and columns, and the identity of information stored in this way conforms to the notion of an entity and a relationship—or alternatively under the notion of an object role model (ORM)—and database architects are always wrestling with the best way of reducing their data set to a basic set of identities. You can extend this notion to the idea of an identity having a profile and profiling services such as Facebook as being an extension of the notion of Identity as a Service in cloud computing.

An identity can belong to a person and may include the following:

- **Things you are:** Biological characteristics such as age, race, gender, appearance, and so forth
- **Things you know:** Biography, personal data such as social security numbers, PINs, where you went to school, and so on
- **Things you have:** A pattern of blood vessels in your eye, your fingerprints, a bank account you can access, a security key you were given, objects and possessions, and more
- **Things you relate to:** Your family and friends, a software license, beliefs and values, activities and endeavors, personal selections and choices, habits and practices, an iGoogle account, and more

To establish your identity on a network, you might be asked to provide a name and password, which is called a single-factor authentication method. More secure authentication requires the use of at least two-factor authentication; for example, not only name and password (things you know) but also a transient token number provided by a hardware key (something you have). To get to multifactor authentication, you might have a system that examines a biometric factor such as a fingerprint or retinal blood vessel pattern—both of which are essentially unique things you are. Multifactor authentication requires the outside use of a network security or trust service, and it is in the deployment of trust services that our first and most common IDaaS applications are employed in the cloud.

Of course, many things have digital identities. User and machine accounts, devices, and other objects establish their identities in a number of ways. For user and machine accounts, identities are created and stored in domain security databases that are the basis for any network domain, in directory services, and in data stores in federated systems. Network interfaces are identified uniquely by Media Access Control (MAC) addresses, which alternatively are referred to as Ethernet Hardware Addresses (EHAs). It is the assignment of a network identity to a specific MAC address that allows systems to be found on networks.

The manner in which Microsoft validates your installation of Windows and Office is called Windows Product Activation and creates an identification index or profile of your system, which is instructive. During activation, the following unique data items are retrieved:

- A 25-character software product key and product ID
- The uniquely assigned Global Unique Identifier or GUID
- PC manufacturer
- CPU type and serial number
- BIOS checksum
- Network adapter and its MAC address
- Display adapter
- SCSCI and IDE adapters
- RAM amount
- Hard drive and volume serial number

- Optical drive
- Region and language settings and user locale

From this information, a code is calculated, checked, and entered into the registration database. Each of these uniquely identified hardware attributes is assigned a weighting factor such that an overall sum may be calculated. If you change enough factors—NIC and CPU, display adapter, RAM amount, and hard drive—you trigger a request for a reactivation based on system changes. This activation profile is also required when you register for the Windows Genuine Advantage program. Windows Product Activation and Windows Genuine Advantage are cloud computing applications, albeit proprietary ones. Whether people consider these applications to be services is a point of contention.

Networked identity service classes

To validate Web sites, transactions, transaction participants, clients, and network services—various forms of identity services—have been deployed on networks. Ticket or token providing services, certificate servers, and other trust mechanisms all provide identity services that can be pushed out of private networks and into the cloud.

Identity protection is one of the more expensive and complex areas of network computing. If you think about it, requests for information on identity by personnel such as HR, managers, and others; by systems and resources for access requests; as identification for network traffic; and the myriad other requirements mean that a significant percentage of all network traffic is supporting an identification service. Literally hundreds of messages on a network every minute are checking identity, and every Ethernet packet contains header fields that are used to identify the information it contains.

As systems become even more specialized, it has become increasingly difficult to find the security experts needed to run an ID service. So Identity as a Service or the related hosted (managed) identity services may be the most valuable and cost effective distributed service types you can subscribe to.

Identity as a Service (IDaaS) may include any of the following:

- Authentication services (identity verification)
- Directory services
- Federated identity
- Identity governance
- Identity and profile management
- Policies, roles, and enforcement
- Provisioning (external policy administration)
- Registration
- Risk and event monitoring, including audits
- Single sign-on services (pass-through authentication)

The sharing of any or all of these attributes over a network may be the subject of different government regulations and in many cases must be protected so that only justifiable parties may have access to the minimal amount that may be disclosed. This level of access defines what may be called an identity relationship.

Note

The Burton Group (`http://www.burtongroup.com/`), a well-known computer industry analyst firm located in Midvale, Utah, has a trademark on the term IaaS as defined as Identity as a Service for use in the publication of their research in this area. The Burton Group is a well-known authority in the field of network infrastructure, particularly directory services and more recently in cloud computing. In this book, I use the term IaaS as applied to Infrastructure as a Service and use IDaaS to identify Identity as a Service. ■

Identity system codes of conduct

Certain codes of conduct must be observed legally, and if not legally at the moment, then certainly on a moral basis. Cloud computing services that don't observe these codes do so at their peril. In working with IDaaS software, evaluate IDaaS applications on the following basis:

- **User control for consent:** Users control their identity and must consent to the use of their information.
- **Minimal Disclosure:** The minimal amount of information should be disclosed for an intended use.
- **Justifiable access:** Only parties who have a justified use of the information contained in a digital identity and have a trusted identity relationship with the owner of the information may be given access to that information.
- **Directional Exposure:** An ID system must support bidirectional identification for a public entity so that it is discoverable and a unidirectional identifier for private entities, thus protecting the private ID.
- **Interoperability:** A cloud computing ID system must interoperate with other identity services from other identity providers.
- **Unambiguous human identification:** An IDaaS application must provide an unambiguous mechanism for allowing a human to interact with a system while protecting that user against an identity attack.
- **Consistency of Service:** An IDaaS service must be simple to use, consistent across all its uses, and able to operate in different contexts using different technologies.

IDaaS interoperability

Identity as a Service provides an easy mechanism for integrating identity services into individual applications with minimal development effort, by allowing the identification logic and storage of an identity's attributes to be maintained externally. IDaaS applications may be separated from other distributed security systems by their compliance with SOA standards (as described in Chapter 13, "Understanding Service Oriented Architecture"), particularly if you want to have these services interoperate and be federated.

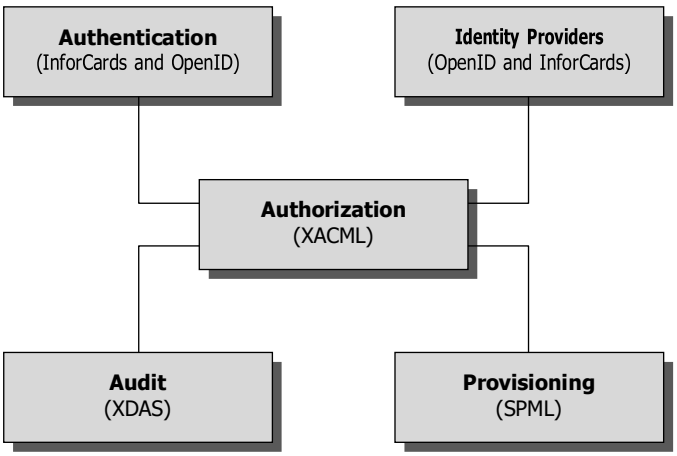
Therefore, cloud computing IDaaS applications must rely on a set of developing industry standards to provide interoperability. The following are among the more important of these services:

- **User centric authentication (usually in the form of information cards):** The OpenID and CardSpace specifications support this type of data object.
- **The XACML Policy Language:** This is a general-purpose authorization policy language that allows a distributed ID system to write and enforce custom policy expressions. XACML can work with SAML; when SAML presents a request for ID authorization, XACML checks the ID request against its policies and either allows or denies the request.
- **The SPML Provisioning Language:** This is an XML request/response language that is used to integrate and interoperate service provisioning requests. SPML is a standard of OASIS's Provision Services Technical Committee (PSTC) that conforms to the SOA architecture.
- **The XDAS Audit System:** The Distributed Audit Service provides accountability for users accessing a system, and the detection of security policy violations when attempts are made to access the system by unauthorized users or by users accessing the system in an unauthorized way.

Figure 4.6 shows how these different standards form an identity service framework.

FIGURE 4.6

Open standards that support an IDaaS infrastructure for cloud computing



The Identity Governance Framework (IGF) is a standards initiative of the Liberty Alliance (`http://www.projectliberty.org/`) that is concerned with the exchange and control of identity information using standards such as WS-Trust, ID-WSF, SAML, and LDAP directory services. The Liberty Alliance was established by an industry group in 2001 with the purpose of

promoting open identity interchanges through policy standards that applications can use to enforce privacy as well as to allow privacy auditing. In 2009, this group released its Client Attribute Requirements Markup Language (CARML) and a set of IGF Privacy Constraints that forms the basis of the open source project called Aristotle (`http://www.openliberty.org/wiki/index.php/ProjectAris`), which has as its goal the creation of an API for identity interchange.

User authentication

OpenID is a developing industry standard for authenticating "end users" by storing their digital identity in a common format. When an identity is created in an OpenID system, that information is stored in the system of any OpenID service provider and translated into a unique identifier. Identifiers take the form of a Uniform Resource Locator (URL) or as an Extensible Resource Identifier (XRI) that is authenticated by that OpenID service provider. Any software application that complies with the standard accepts an OpenID that is authenticated by a trusted provider. A very impressive group of cloud computing vendors serve as identity providers (or OpenID providers), including AOL, Facebook, Google, IBM, Microsoft, MySpace, Orange, PayPal, VeriSign, LiveJournal, Ustream, Yahoo!, and others.

The OpenID standard applies to the unique identity of the URL; it is up to the service provider to store the information and specify the forms of authentication required to successfully log onto the system. Thus an OpenID authorization can include not only passwords, but smart cards, hardware keys, tokens, and biometrics as well. OpenID is supported by the OpenID Foundation (`http://openid.net/foundation/`), a not-for-profit organization that promotes the technology.

These are samples of trusted providers and their URL formats:

- **Blogger:** <*username*>`.blogger.com` or <*blogid*>`.blogspot.com`
- **MySpace:** `myspace.com/`<*username*>
- **Google:** `https://www.google.com/accounts/o8/id`
- **Google Profile:** `google.com/profiles/`<*username*>
- **Microsoft:** `accountservices.passport.net/`
- **MyOpenID:** <*username*>`.myopenid.com`
- **Orange:** `openid.orange.fr/username` or simply `orange.fr/`
- **Verisign:** <*username*>`.pip.verisinglabs.com`
- **WordPress:** <*username*>`.wordpress.com`
- **Yahoo!:** `openid.yahoo.com`

After you have logged onto a trusted provider, that logon may provide you access to other Web sites that support OpenID. When you request access to a site through your browser (or another application that is referred to as a user-agent), that site serves as the "relying party" and requests of the server or server-agent that it verify the end-user's identifier. You won't need to log onto these other Web sites, if your OpenID is provided. Most trusted providers require that you indicate which Web sites you want to share your OpenID identifier with and the information is submitted automatically to the next site.

CardSpace is a Microsoft software client that is part of the company's Identity Metasystem and built into the Web Services Protocol Stack. This stack is built on the OASIS standards (WS-Trust, WS-Security, WS-SecurityPolicy, and WS-MetadataExchange), so any application that conforms with the OASIS WS- standards can interoperate with CardSpace. CardSpace was introduced with .NET Frameworks 3.0 and can be installed on Windows XP, Server 2003, and later. It is installed by default on Windows Vista and Windows 7.

CardSpace offers another way of authenticating users in the cloud. An Information Card may be requested with an HTML <OBJECT> tag, and the trusted Identity Provider then creates an encrypted and digitally signed token using the Security Token Service (STS) that is part of a WS-Trust request/reply mechanism. CardSpace may be seen as an alternative mechanism to the use of OpenID and SAML and is used to sign into those services as well as Windows Live ID accounts.

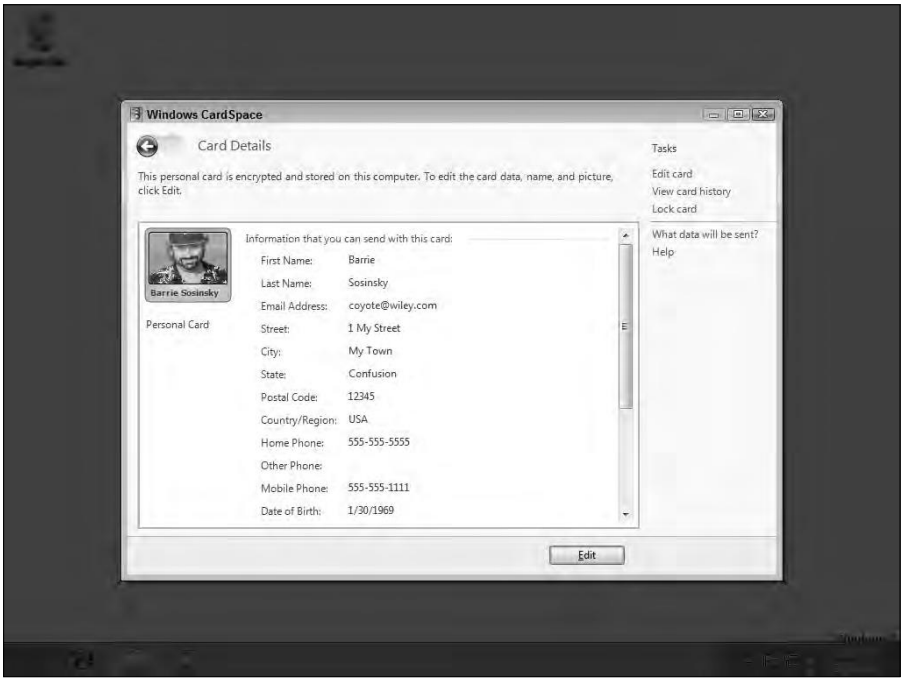
Cross-Ref

See Chapter 10 for more information on Windows Live. ■

Figure 4.7 shows a personal Identification Card that is stored locally on a Windows 7 system. Managed Information Cards are stored on a network service and can be made available to other cloud services upon demand.

FIGURE 4.7

This is a private CardSpace Identification Card. Managed Identification Cards that store similar information are stored on a network service and can be shared to the cloud.



Note

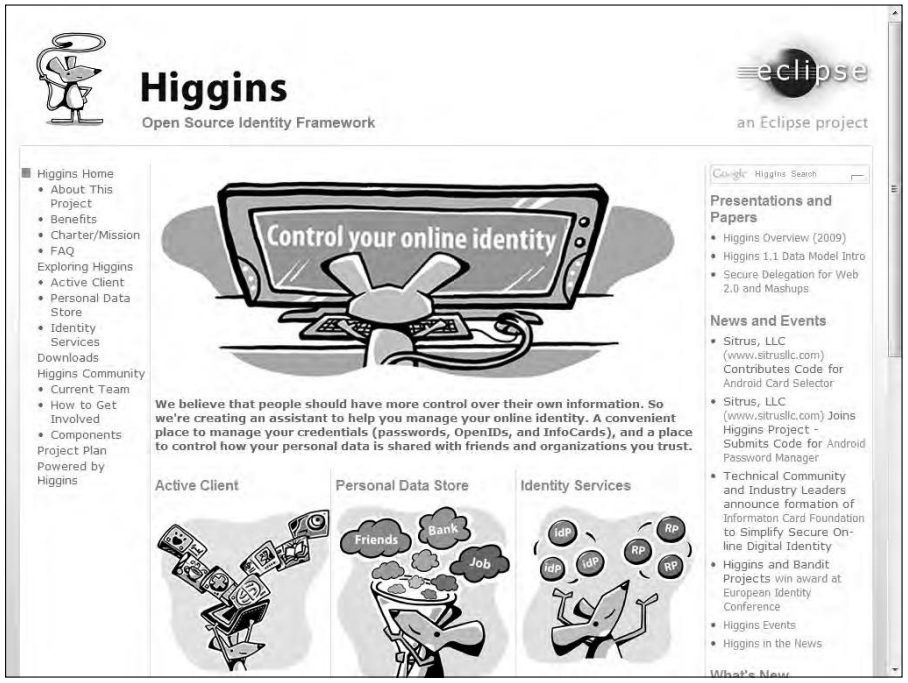
IBM and Novell both support Information Cards under the Web Service Protocol Stack, as well as SAML and OpenID through an industry initiative known at the Higgins Open Source Identity Framework (`http://eclipse.org/higgins/`), which uses the interface metaphor of an i-Card. The Higgins's home page is shown in Figure 4.8. Higgins develops a browser add-on that logs you into Web sites and manages digital identities. ■

A CardSpace object called an Identity Selector stores a digital identity, making it available to Windows applications in the form of a visual Information Card that can be accepted by complying applications and Web sites. When a user selects an Information Card, the CardSpace software issues a request to validate the information from a Web service that returns a digitally signed XML token that contains the identity and its metadata. A personal or self-issued Information Card also can be created into which you can store biographical information. Personal Information Cards may or may not be managed by a trusted third party.

Before leaving the subject of digital identity services and formats, it is appropriate to mention some file format standards such as vCard, which stores identity data in the form of an electronic business card. Less frequently encountered is the vCard derivative hCard. vCard was an industry initiative in the mid-1990s as a means for identifying parties in e-mail exchanges.

FIGURE 4.8

The Higgins Open Source Identity Framework uses an i-Card metaphor and interoperable identity service APIs to create a vendor-neutral cloud-based authentication service.



As e-mail platforms such as Lotus Notes and MS Exchange broadened the notion of e-mail to include calendars and appointments, the vCalendar format was created, which eventually was developed into iCalendar. The current version 3.0 of vCard is part of the vCardDAV working group within the IETF and through RFC 2425 and RFC 2426 is undergoing the process of standardization. vCard is more of an interchange file format for an identity than it is an identity trust-authorization service, which is why it is mentioned in passing here. To read more about vCard and vCalendar, go to `http://www.imc.org/pdi/`.

Authorization markup languages

Information requests and replies in cloud computing are nearly always in the form of XML replies or requests. XML files are text files and are self-describing. That is, XML files contain a schema that describes the data it contains or contains a point to another text file with its schema. A variety of specialized XML files are in the identity framework, the ones of note being XACML and SAML, shown in Figure 4.9.

The eXtensible Access Control Markup Language (XACML) is an OASIS standard (see `http://xml.coverpages.org/xacml.html`) for a set of policy statements written in XML that support an authentication process. A policy in XACML describes a subject element that requests an action from a resource. These three elements operate within an environment that also can be described in terms of an Action element. Subject and Action elements (which are terms of art in XACML) are elements that can have one or more attributes. Resources (which are services, system components, or data) have a single attribute, which is usually its URL.

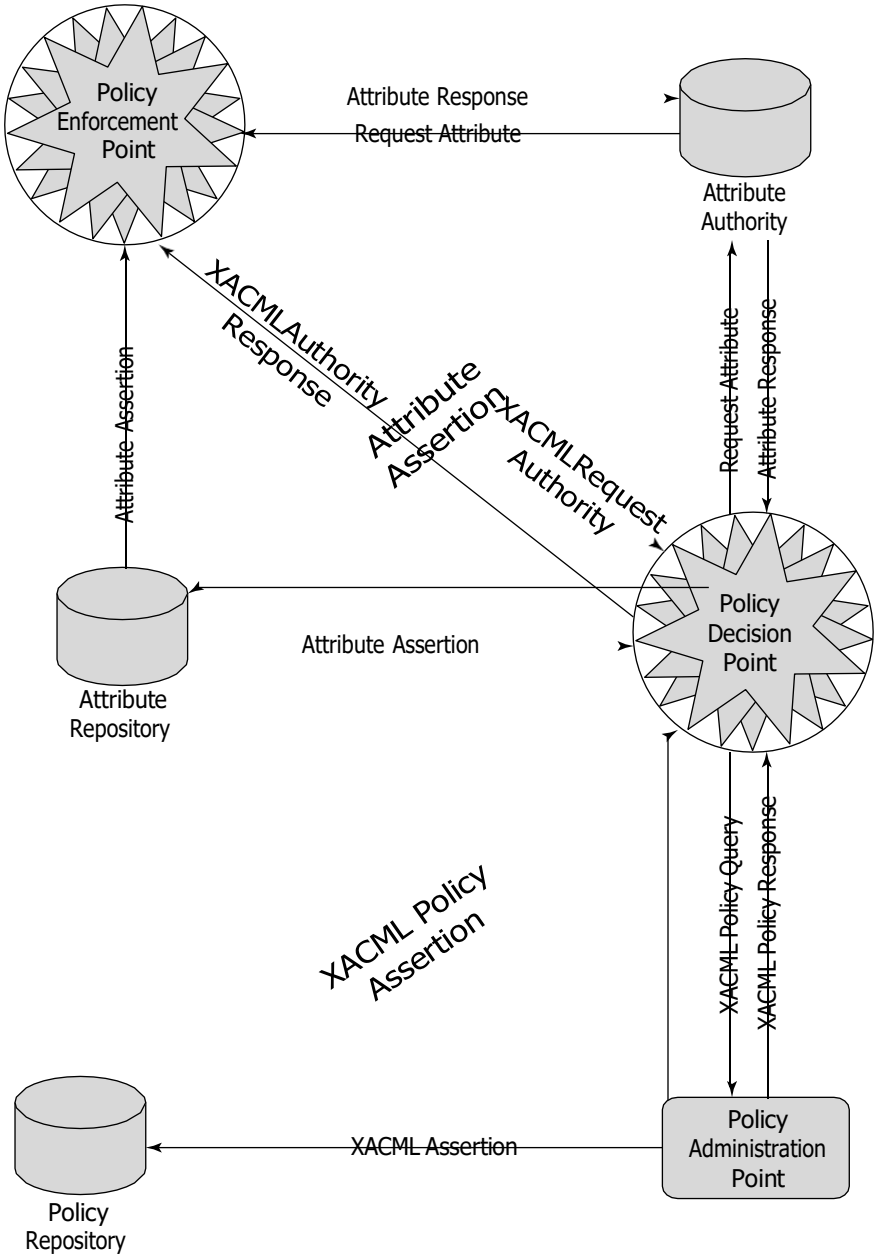
The location at which policy is managed is referred to as the Policy Administration Point (PAP). Policy requests are passed through to the location where the policy logic can be executed, referred to as the Policy Decision Point (PDP). The result of the policy is transmitted through the PAP to the resource that acts on and enforces the PDP policy decision, which is referred to as the Policy Enforcement Point (PEP). An XACML engine also may access a resource that provides additional information that can be used to determine policy logic, called a Policy Information Point (PIP). Examples of PIP services are LDAP or Active Directory servers. A request for identification goes to the XACML engine, where it becomes a directive from the Policy Decision Point to the Policy Enforcement Point called an obligation.

The main method for exchanging information between an authentication and authorization service in a Service Oriented Architecture is the Security Assertion Markup Language (SAML). In SAML, a service provider passes a statement or set of statements (called assertions) to an identity provider that must be evaluated. The identity provider must determine if the principle or user requesting access is registered with the identity service and is who he claims to be.

SAML in most instances operates as a reply/request mechanism; it makes no demands on the identification service as to how the identity is authenticated. Should the identity provider validate the identity of the principle in the assertion, it passes a local authorization to the service provider, which then enforces an access-control judgment based on the authorization.

FIGURE 4.9

SAML integrates with XACML to implement a policy engine in a Service Oriented Architecture to support identity services authorization.



A SAML assertion is a security statement in the SAML file that makes a claim regarding authentication, attributes, or authorization. The statement is of this form:

> Assertion X created at Time T by User U about Subject S is true when Conditions C are TRUE.

It is up to the identity provider to parse this statement and determine its validity. The SAML protocol request is often referred to as a query; the three different supported query types are an authentication query, an attribute query, and an authorization decision query. SAML requests use a SOAP binding; that is, the SAML request or response is embedded in a SOAP wrapper within an HTTP message.

SAML is used to provide a mechanism for a Web Browser Single Sign On (SSO). In this instance, a Web browser is the user agent, which requests access to a resource that is authorized by a SAML service provider. The service provider takes a request from a user for access to the resource and sends an authentication request to the SAML identity provider directly from the initiating user agent (Web browser). Figure 4.10 shows the SAML Single Sign On Request/Response mechanism.

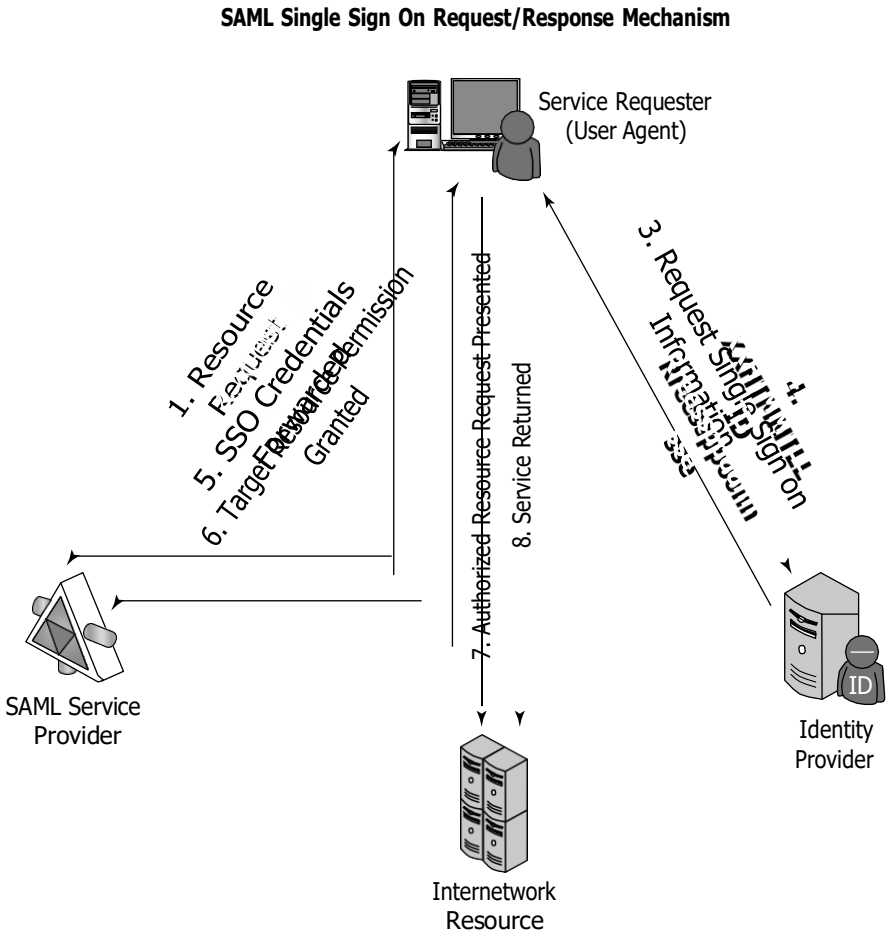
The Service Provisioning Markup Language (SPML) is another of the OASIS open standards developed to provide for service provisioning. Provisioning is the process by which a resource is prepared for use, reserved, accessed, used, and then released when the transaction is completed. A classic example of provisioning a resource is the reservation and use of a phone line or a Virtual Private Network.

A provisioning system has three types of components: A Requesting Authority (RA) is the client, the Provisioning Service Point (PSP) is the cloud component that receives the request and returns a response to the RA, and a Provisioning Service Targets (PST) is the software application upon which the provisioning action is performed. The SPML provisioning system (which can be thought of as an architectural layer) means that identity information need only be entered into these three components once.

SPML is used to prepare Web services and applications for use, signal that the resource is available for use and waiting for instructions, and signal when the use or transaction has been completed. With SPML, a system can provide automated user and system access, enforce access rights, and make cloud computing services available across network systems. Without a provisioning system, a cloud computing system can be very inefficient and potentially unreliable.

FIGURE 4.10

SAML provides a mechanism by which a service requester can use a Single Sign On logon to access Web services securely.



Defining Compliance as a Service (CaaS)

Cloud computing by its very nature spans different jurisdictions. The laws of the country of a request's origin may not match the laws of the country where the request is processed, and it's possible that neither location's laws match the laws of the country where the service is provided. Compliance is much more than simply providing an anonymous service token to an identity so they can obtain access to a resource. Compliance is a complex issue that requires considerable expertise.

While Compliance as a Service (CaaS) appears in discussions, few examples of this kind of service exist as a general product for a cloud computing architecture. A Compliance as a Service application would need to serve as a trusted third party, because this is a man-in-the-middle type of service. CaaS may need to be architected as its own layer of a SOA architecture in order to be trusted. A CaaS would need to be able to manage cloud relationships, understand security policies and procedures, know how to handle information and administer privacy, be aware of geography, provide an incidence response, archive, and allow for the system to be queried, all to a level that can be captured in a Service Level Agreement. That's a tall order, but CaaS has the potential to be a great value-added service.

In order to implement CaaS, some companies are organizing what might be referred to as "vertical clouds," clouds that specialize in a vertical market. Examples of vertical clouds that advertise CaaS capabilities include the following:

- **athenahealth** (`http://www.athenahealth.com/`) for the medical industry
- **bankserv** (`http://www.bankserv.com/`) for the banking industry
- **ClearPoint PCI** Compliance-as-a-Service for merchant transactions under the Payment Card Industry Data Security Standard
- **FedCloud** (`http://www.fedcloud.com/`) for government
- **Rackserve PCI** Compliant Cloud (`http://www.rackspace.com/`; another PCI CaaS service)

It's much easier to envisage a CaaS system built inside a private cloud where the data is under the control of a single entity, thus ensuring that the data is under that entity's secure control and that transactions can be audited. Indeed, most of the cloud computing compliance systems to date have been built using private clouds.

It is easy to see how CaaS could be an incredibly valuable service. A well-implemented CaaS service could measure the risks involved in servicing compliance and ensure or indemnify customers against that risk. CaaS could be brought to bear as a mechanism to guarantee that an e-mail conformed to certain standards, something that could be a new electronic service of a network of national postal systems—and something that could help bring an end to the scourge of spam.

Summary

In this chapter, you learned about cloud computing service types. Service types are models upon which distributed applications are created and hosted. The main service models in cloud computing are Infrastructure, Platform, and Software. Vendors offer services based on these different service models called Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS), which are helpful in understanding how products and services are organized.

Infrastructure as a Service is a hardware model in which you can create virtual computing systems or networks and deploy your applications on those servers. Software as a Service is a hosted application that is the cloud equivalent of a traditional desktop application. Platform as a Service is a development environment that builds upon an existing cloud computing application infrastructure. Other service types are possible. You learned about Identity as a Service (IDaaS), which enables secure transactions on a distributed network. In time many vertical services for cloud computing will appear.

Chapter 5, “Understanding Abstraction and Virtualization,” considers some of the more important characteristics of cloud computing networks and applications.

Part II

Using Platforms

IN THIS PART

CHAPTER 5

Understanding Abstraction and Virtualization

In this chapter, I discuss different technologies that create shared pools of resources. The key to creating a pool is to provide an abstraction mechanism so that a logical address can be mapped to a physical resource. Computers use this technique for placing files on disk drives, and cloud computing networks use a set of techniques to create virtual servers, virtual storage, virtual networks, and perhaps one day virtual applications. Abstraction enables the key benefit of cloud computing: shared, ubiquitous access.

In this chapter, you learn about how load balancing can be used to create high performance cloud-based solutions. Google.com's network is an example of this approach. Google uses commodity servers to direct traffic appropriately.

Another technology involves creating virtual hardware systems. An example of this type of approach is hypervisors that create virtual machine technologies. Several important cloud computing approaches use a strictly hardware-based approach to abstraction. I describe VMware's vSphere infrastructure in some detail, along with some of the unique features and technologies that VMware has developed to support this type of cloud.

Finally, I describe some approaches to making applications portable. Application portability is a difficult proposition, and work to make applications portable is in its infancy. Two approaches are described, the Simple API and AppZero's Virtual Application Appliance (VAA). VAAs are containers that abstract an application from the operating system, and they offer the potential to make an application portable from one platform to another.

IN THIS CHAPTER

- Understanding how abstraction makes cloud computing possible
- Understanding how virtualization creates shared resource pools
- Using load balancing to enable large cloud computing applications
- Using hypervisors to make virtual machines possible
- Discussing system imaging and application portability for the cloud

Using Virtualization Technologies

The dictionary includes many definitions for the word "cloud." A cloud can be a mass of water droplets, gloom, an obscure area, or a mass of similar particles such as dust or smoke. When it comes to cloud computing, the definition that best fits the context is "a collection of objects that are grouped together." It is that act of grouping or creating a resource pool that is what succinctly differentiates cloud computing from all other types of networked systems.

Not all cloud computing applications combine their resources into pools that can be assigned on demand to users, but the vast majority of cloud-based systems do. The benefits of pooling resources to allocate them on demand are so compelling as to make the adoption of these technologies a priority. Without resource pooling, it is impossible to attain efficient utilization, provide reasonable costs to users, and proactively react to demand. In this chapter, you learn about the technologies that abstract physical resources such as processors, memory, disk, and network capacity into virtual resources.

When you use cloud computing, you are accessing pooled resources using a technique called virtualization. Virtualization assigns a logical name for a physical resource and then provides a pointer to that physical resource when a request is made. Virtualization provides a means to manage resources efficiently because the mapping of virtual resources to physical resources can be both dynamic and facile. Virtualization is dynamic in that the mapping can be assigned based on rapidly changing conditions, and it is facile because changes to a mapping assignment can be nearly instantaneous.

These are among the different types of virtualization that are characteristic of cloud computing:

- **Access:** A client can request access to a cloud service from any location.
- **Application:** A cloud has multiple application instances and directs requests to an instance based on conditions.
- **CPU:** Computers can be partitioned into a set of virtual machines with each machine being assigned a workload. Alternatively, systems can be virtualized through load-balancing technologies.
- **Storage:** Data is stored across storage devices and often replicated for redundancy.

To enable these characteristics, resources must be highly configurable and flexible. You can define the features in software and hardware that enable this flexibility as conforming to one or more of the following mobility patterns:

- **P2V:** Physical to Virtual
- **V2V:** Virtual to Virtual
- **V2P:** Virtual to Physical
- **P2P:** Physical to Physical
- **D2C:** Datacenter to Cloud

- **C2C:** Cloud to Cloud
- **C2D:** Cloud to Datacenter
- **D2D:** Datacenter to Datacenter

The techniques used to achieve these different types of virtualization are the subject of this chapter. According to Gartner ("Server Virtualization: One Path that Leads to Cloud Computing," by Thomas J. Bittman, 10/29/2009, Research Note G00171730), virtualization is a key enabler of the first four of five key attributes of cloud computing:

- **Service-based:** A service-based architecture is where clients are abstracted from service providers through service interfaces.
- **Scalable and elastic:** Services can be altered to affect capacity and performance on demand.
- **Shared services:** Resources are pooled in order to create greater efficiencies.
- **Metered usage:** Services are billed on a usage basis.
- **Internet delivery:** The services provided by cloud computing are based on Internet protocols and formats.

Load Balancing and Virtualization

One characteristic of cloud computing is virtualized network access to a service. No matter where you access the service, you are directed to the available resources. The technology used to distribute service requests to resources is referred to as *load balancing*. Load balancing can be implemented in hardware, as is the case with F5's BigIP servers, or in software, such as the Apache `mod_proxy_balancer` extension, the Pound load balancer and reverse proxy software, and the Squid proxy and cache daemon. Load balancing is an optimization technique; it can be used to increase utilization and throughput, lower latency, reduce response time, and avoid system overload.

The following network resources can be load balanced:

- Network interfaces and services such as DNS, FTP, and HTTP
- Connections through intelligent switches
- Processing through computer system assignment
- Storage resources
- Access to application instances

Without load balancing, cloud computing would very difficult to manage. Load balancing provides the necessary redundancy to make an intrinsically unreliable system reliable through managed redirection. It also provides fault tolerance when coupled with a failover mechanism. Load balancing is nearly always a feature of server farms and computer clusters and for high availability applications.

A load-balancing system can use different mechanisms to assign service direction. In the simplest load-balancing mechanisms, the load balancer listens to a network port for service requests. When a request from a client or service requester arrives, the load balancer uses a scheduling algorithm to assign where the request is sent. Typical scheduling algorithms in use today are round robin and weighted round robin, fastest response time, least connections and weighted least connections, and custom assignments based on other factors.

A session ticket is created by the load balancer so that subsequent related traffic from the client that is part of that session can be properly routed to the same resource. Without this session record or persistence, a load balancer would not be able to correctly failover a request from one resource to another. Persistence can be enforced using session data stored in a database and replicated across multiple load balancers. Other methods can use the client's browser to store a client-side cookie or through the use of a rewrite engine that modifies the URL. Of all these methods, a session cookie stored on the client has the least amount of overhead for a load balancer because it allows the load balancer an independent selection of resources.

The algorithm can be based on a simple round robin system where the next system in a list of systems gets the request. Round robin DNS is a common application, where IP addresses are assigned out of a pool of available IP addresses. Google uses round robin DNS, as described in the next section.

Advanced load balancing

The more sophisticated load balancers are workload managers. They determine the current utilization of the resources in their pool, the response time, the work queue length, connection latency and capacity, and other factors in order to assign tasks to each resource. Among the features you find in load balancers are polling resources for their health, the ability to bring standby servers online (priority activation), workload weighting based on a resource's capacity (asymmetric loading), HTTP traffic compression, TCP offload and buffering, security and authentication, and packet shaping using content filtering and priority queuing.

An Application Delivery Controller (ADC) is a combination load balancer and application server that is a server placed between a firewall or router and a server farm providing Web services. An Application Delivery Controller is assigned a virtual IP address (VIP) that it maps to a pool of servers based on application specific criteria. An ADC is a combination network and application layer device. You also may come across ADCs referred to as a content switch, multilayer switch, or Web switch.

These vendors, among others, sell ADC systems:

- A10 Networks (`http://www.a10networks.com/`)
- Barracuda Networks (`http://www.barracudanetworks.com/`)
- Brocade Communication Systems (`http://www.brocade.com/`)
- Cisco Systems (`http://www.cisco.com/`)
- Citrix Systems (`http://www.citrix.com/`)

- F5 Networks (http://www.f5.com/)
- Nortel Networks (http://www.nortel.com/)
- Coyote Point Systems (http://www.coyotepoint.com/)
- Radware (http://www.radware.com/)

An ADC is considered to be an advanced version of a load balancer as it not only can provide the features described in the previous paragraph, but it conditions content in order to lower the workload of the Web servers. Services provided by an ADC include data compression, content caching, server health monitoring, security, SSL offload and advanced routing based on current conditions. An ADC is considered to be an application accelerator, and the current products in this area are usually focused on two areas of technology: network optimization, and an application or framework optimization. For example, you may find ADC's that are tuned to accelerate ASP.NET or AJAX applications.

An architectural layer containing ADCs is described as an Application Delivery Network (ADN), and is considered to provide WAN optimization services. Often an ADN is comprised of a pair of redundant ADCs. The purpose of an ADN is to distribute content to resources based on application specific criteria. ADN provide a caching mechanism to reduce traffic, traffic prioritization and optimization, and other techniques. ADN began to be deployed on Content Delivery Networks (CDN) in the late 1990s, where it added the ability to optimize applications (application fluency) to those networks. Most of the ADC vendors offer commercial ADN solutions.

In addition to the ADC vendors in the list above, these are additional ADN vendors, among others:

- Akamai Technologies (http://www.akamai.com/)
- Blue Coat Systems (http://www.bluecoat.com/)l

CDNetworks (http://www.cdnetworks.com/)

- Crescendo Networks (http://www.crescendonetworks.com/)
- Expand Networks (http://www.expand.com/)
- Juniper Networks (http://www.juniper.net/)

Google's cloud is a good example of the use of load balancing, so in the next section let's consider how Google handles the many requests that they get on a daily basis.

The Google cloud

According to the Web site tracking firm Alexa (http://www.alexa.com/topsites), Google is the single most heavily visited site on the Internet; that is, Google gets the most hits. The investment Google has made in infrastructure is enormous, and the Google cloud is one of the largest in use today. It is estimated that Google runs over a million servers worldwide, processes a billion search requests, and generates twenty petabytes of data per day.

Google is understandably reticent to disclose much about its network, because it believes that its infrastructure, system response, and low latency are key to the company's success. Google never gives datacenter tours to journalists, doesn't disclose where its datacenters are located, and obfuscates the locations of its datacenters by wrapping them in a corporate veil. Thus, the discretely named Tetra LLC (limited liability company) owns the land for the Council Bluffs, Iowa, site, and Lapis LLC owns the land for the Lenoir, North Carolina, site. This makes Google infrastructure watching something akin to a sport to many people.

So what follows is what we think we know about Google's infrastructure and the basic idea behind how Google distributes its traffic by pooling IP addresses and performing several layers of load balancing.

Google has many datacenters around the world. As of March 2008, Rich Miller of DataCenterKnowledge.com wrote that Google had at least 12 major installations in the United States and many more around the world. Google supports over 30 country specific versions of the Google index, and each localization is supported by one or more datacenters. For example, Paris, London, Moscow, Sao Paolo, Tokyo, Toronto, Hong Kong, Beijing and others support their countries' locale. Germany has three centers in Berlin, Frankfurt, and Munich; the Netherlands has two at Groningen and Eemshaven. The countries with multiple datacenters store index replicas and support network peering relationships. Network peering helps Google have low latency connections to large Internet hubs run by different network providers.

You can find a list of sites as of 2008 from Miller's FAQ at `http://www.datacenterknowledge.com/archives/2008/03/27/google-data-center-faq/`.

Based on current locations and the company's statements, Google's datacenters are sited based on the following factors (roughly in order of importance):

1. Availability of cheap and, if possible, renewable energy
2. The relative locations of other Google datacenters such that the site provides the lowest latency response between sites
3. Location of nearby Internet hubs and peering sites
4. A source of cooling water
5. The ability to purchase a large area of land surrounding the site

   Speculation on why Google purchases large parcels of land ranges from creating a buffer zone between the datacenter and surrounding roads and towns or possibly to allow for building wind farms when practical.
6. Tax concessions from municipalities that lower Google's overhead

Google maintains a pool of hundreds of IP addresses, all of which eventually resolve to its Mountain View, California, headquarters. When you initiate a Google search, your query is sent to a DNS server, which then queries Google's DNS servers. The Google DNS servers examine the pool of addresses to determine which addresses are geographically closest to the query origin and uses a round robin policy to assign an IP address to that request. The request usually goes to the nearest

datacenter, and that IP address is for a cluster of Google servers. This DNS assignment acts as a first level of IP virtualization, a pool of network addresses have been load balanced based on geography.

A Google cluster can contain thousands of servers. Google servers are racks of commodity (low cost) 1U or 2U servers containing 40 to 80 servers per rack with one switch per rack. Each switch is connected to a core gigabit switch. Google servers run a customized version of Linux with applications of several types.

When the query request arrives at its destination, a Google cluster is sent to a load balancer, which forwards that request to a Squid proxy server and Web cache dameon. This is the second level of IP distribution, based on a measure of the current system loading on proxy servers in the cluster. The Squid server checks its cache, and if it finds a match to the query, that match is returned and the query has been satisfied. If there is no match in the Squid cache, the query is sent to an individual Google Web Server based on current Web server utilizations, which is the third level of network load balancing, again based on utilization rates.

It is the Google Web Servers that perform the query against the Google index and then format the results into an HTML page that is returned to the requester. This procedure then performs two more levels of load balancing based on utilization rates.

Google's secret sauce is its in-memory inverted index and page rank algorithm. Google's GoogleBot (a spider or robot) crawls the Web and collects document information. Some details of the search and store algorithm are known. Google looks at the title and first few hundred words and builds a word index from the result. Indexes are stored on an index server.

Some documents are stored as snapshots (PDF, DOC, XLS, and so on), but lots of information is not addressed in the index. Each document is given a unique ID ("docid"), and the content of the document is disassembled into segments called shards, subjected to a data compression scheme and stored on a document server. The entire index is maintained in system memory partitioned over each instance of the index's replicas. A page rank is created based on the significant links to that page.

Queries are divided into word lists, and the Google algorithm examines the words and the relationships of one word to another. Those word relationships are mapped against the main index to create a list of documents, a feature called an inverted index. In an inverted index, words are mapped to documents, which can be done very quickly when the index is fully kept in memory.

The Web server takes the result of a query and composes the Web page from that result. Ads included on the page are from ad servers, which provide Google's AdSense and AdWords services. The query also is presented to a spelling server to provide suggestions for alternative spellings to include in the search result. Certain keywords, data input patterns, and other strings are recognized as having special operational significance. For example entering "2 plus 2" initiates Google's calculator program, and a ten-digit number returns a reverse phone lookup using the phonebook program. These programs are supported by special application servers.

Google doesn't use hardware virtualization; it performs server load balancing to distribute the processing load and to get high utilization rates. The workload management software transfers the workload from a failed server over to a redundant server, and the failed server is taken offline. Multiple instances of various Google applications are running on different hosts, and data is stored on redundant storage systems.

Understanding Hypervisors

Load balancing virtualizes systems and resources by mapping a logical address to a physical address. Another fundamental technology for abstraction creates virtual systems out of physical systems. If load balancing is like playing a game of hot potato, then virtual machine technologies is akin to playing slice and dice with the potato.

Given a computer system with a certain set of resources, you can set aside portions of those resources to create a virtual machine. From the standpoint of applications or users, a virtual machine has all the attributes and characteristics of a physical system but is strictly software that emulates a physical machine. A system virtual machine (or a hardware virtual machine) has its own address space in memory, its own processor resource allocation, and its own device I/O using its own virtual device drivers. Some virtual machines are designed to run only a single application or process and are referred to as process virtual machines.

A virtual machine is a computer that is walled off from the physical computer that the virtual machine is running on. This makes virtual machine technology very useful for running old versions of operating systems, testing applications in what amounts to a sandbox, or in the case of cloud computing, creating virtual machine instances that can be assigned a workload. Virtual machines provide the capability of running multiple machine instances, each with their own operating system.

From the standpoint of cloud computing, these features enable VMMs to manage application provisioning, provide for machine instance cloning and replication, allow for graceful system failover, and provide several other desirable features. The downside of virtual machine technologies is that having resources indirectly addressed means there is some level of overhead.

Virtual machine types

A low-level program is required to provide system resource access to virtual machines, and this program is referred to as the hypervisor or Virtual Machine Monitor (VMM). A hypervisor running on bare metal is a Type 1 VM or native VM. Examples of Type 1 Virtual Machine Monitors are LynxSecure, RTS Hypervisor, Oracle VM, Sun xVM Server, VirtualLogix VLX, VMware ESX and ESXi, and Wind River VxWorks, among others. The operating system loaded into a virtual machine is referred to as the guest operating system, and there is no constraint on running the same guest on multiple VMs on a physical system. Type 1 VMs have no host operating system because they are installed on a bare system.

An operating system running on a Type 1 VM is a full virtualization because it is a complete simulation of the hardware that it is running on.

Note

Not all CPUs support virtual machines, and many that do require that you enable this support in the BIOS. For example, AMD-V processors (code named Pacifica) and Intel VT-x (code named Vanderpool) were the first of these vendor's 64-bit offerings that added this type of support. ■

Some hypervisors are installed over an operating system and are referred to as Type 2 or hosted VM. Examples of Type 2 Virtual Machine Monitors are Containers, KVM, Microsoft Hyper V, Parallels Desktop for Mac, Wind River Simics, VMWare Fusion, Virtual Server 2005 R2, Xen, Windows Virtual PC, and VMware Workstation 6.0 and Server, among others. This is a very rich product category. Type 2 virtual machines are installed over a host operating system; for Microsoft Hyper-V, that operating system would be Windows Server. In the section that follows, the Xen hypervisor (which runs on top of a Linux host OS) is more fully described. Xen is used by Amazon Web Services to provide Amazon Machine Instances (AMIs).

Figure 5.1 shows a diagram of Type 1 and Type 2 hypervisors.

On a Type 2 VM, a software interface is created that emulates the devices with which a system would normally interact. This abstraction is meant to place many I/O operations outside the virtual environment, which makes it both programmatically easier and more efficient to execute device I/O than it would be inside a virtual environment. This type of virtualization is sometimes referred to as *paravirtualization*, and it is found in hypervisors such as Microsoft's Hyper-V and Xen. It is the host operating system that is performing the I/O through a para-API.

FIGURE 5.1

VMware's vSphere cloud computing infrastructure model

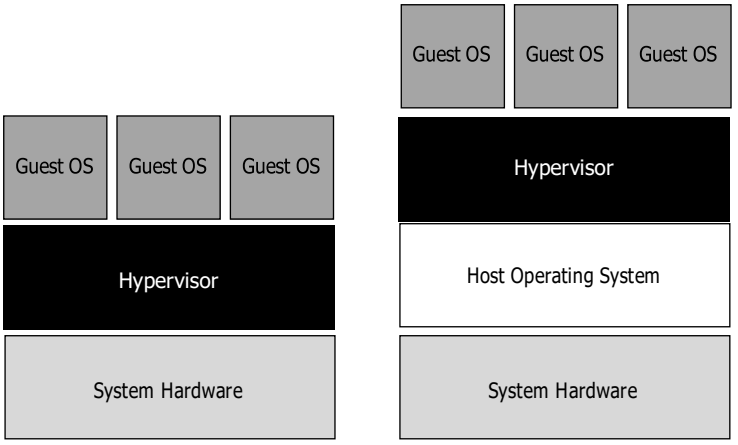


Figure 5.2 shows the difference between emulation, paravirtualization, and full virtualization. In emulation, the virtual machine simulates hardware, so it can be independent of the underlying system hardware. A guest operating system using emulation does not need to be modified in any way. Paravirtualization requires that the host operating system provide a virtual machine interface for the guest operating system and that the guest access hardware through that host VM. An operating system running as a guest on a paravirtualization system must be ported to work with the host interface. Finally, in a full virtualization scheme, the VM is installed as a Type 1 Hypervisor directly onto the hardware. All operating systems in full virtualization communicate directly with the VM hypervisor, so guest operating systems do not require any modification. Guest operating systems in full virtualization systems are generally faster than other virtualization schemes.

The Virtual Machine Interface (VMI) open standard (`http://vmi.ncsa.uiuc.edu/`) that VMware has proposed is an example of a paravirtualization API. The latest version of VMI is 2.1, and it ships as a default installation with many versions of the Linux operating system.

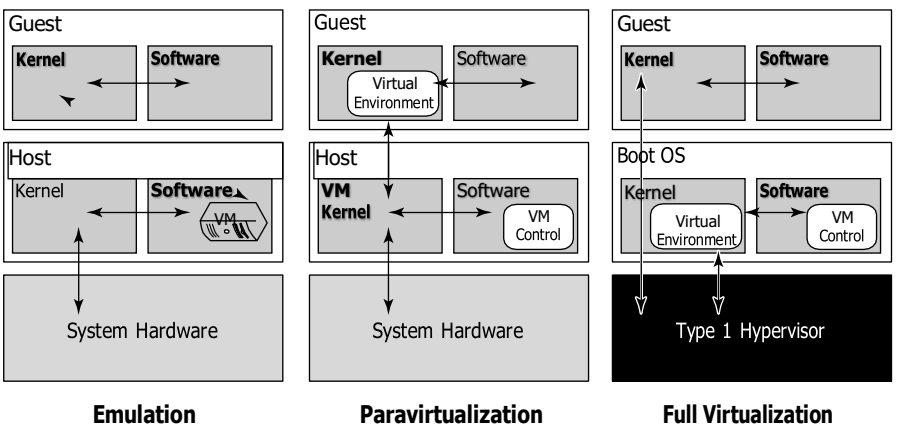
Note

Wikipedia maintains a page called "Comparison of platform virtual machines" at `http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines`. The page contains a table of features of the most common Virtual Machine Managers. ■

You are probably familiar with process or application virtual machines. Most folks run the Java Virtual Machine or Microsoft's .NET Framework VM (called the Common Language Runtime or CLR) on their computers. A process virtual machine instantiates when a command begins a process, the VM is created by an interpreter, the VM then executes the process, and finally the VM exits the system and is destroyed. During the time the VM exists, it runs as a high-level abstraction.

FIGURE 5.2

Emulation, paravirtualization, and full virtualization types



Applications running inside an application virtual machine are generally slow, but these programs are very popular because they provide portability, offer rich programming languages, come with many advanced features, and allow platform independence for their programs. Although many cloud computing applications provide process virtual machine applications, this type of abstraction isn't really suitable for building a large or high-performing cloud network, with one exception.

The exception is the process VMs that enable a class of parallel cluster computing applications. These applications are high-performance systems where the virtual machine is operating one process per cluster node, and the system maintains the necessary intra-application communications over the network interconnect. Examples of this type of system are the Parallel Virtual Machine (PVM; see `http://www.csm.ornl.gov/pvm/pvm_home.html`) and the Message Passing Interface (MPI; see `http://www.mpi-forum.org/`). Some people do not consider these application VMs to be true virtual machines, noting that these applications can still access the host operating system services on the specific system on which they are running. The emphasis on using these process VMs is in creating a high-performance networked supercomputer often out of heterogeneous systems, rather than on creating a ubiquitous utility resource that characterizes a cloud network.

Some operating systems such as Sun Solaris and IBM AIX 6.1 support a feature known as *operating system virtualization.* This type of virtualization creates virtual servers at the operating system or kernel level. Each virtual server is running in its own virtual environment (VE) as a virtual private server (VPS). Different operating systems use different names to describe these machine instances, each of which can support its own guest OS. However, unlike true virtual machines, VPS must all be running the same OS and the same version of that OS. Sun Solaris 10 uses VPS to create what is called Solaris Zones. With IBM AIX, the VPS is called a System Workload Partition (WPAR). This type of virtualization allows for a dense collection of virtual machines with relatively low overhead. Operating system virtualization provides many of the benefits of virtualization previously noted in this section.

VMware vSphere

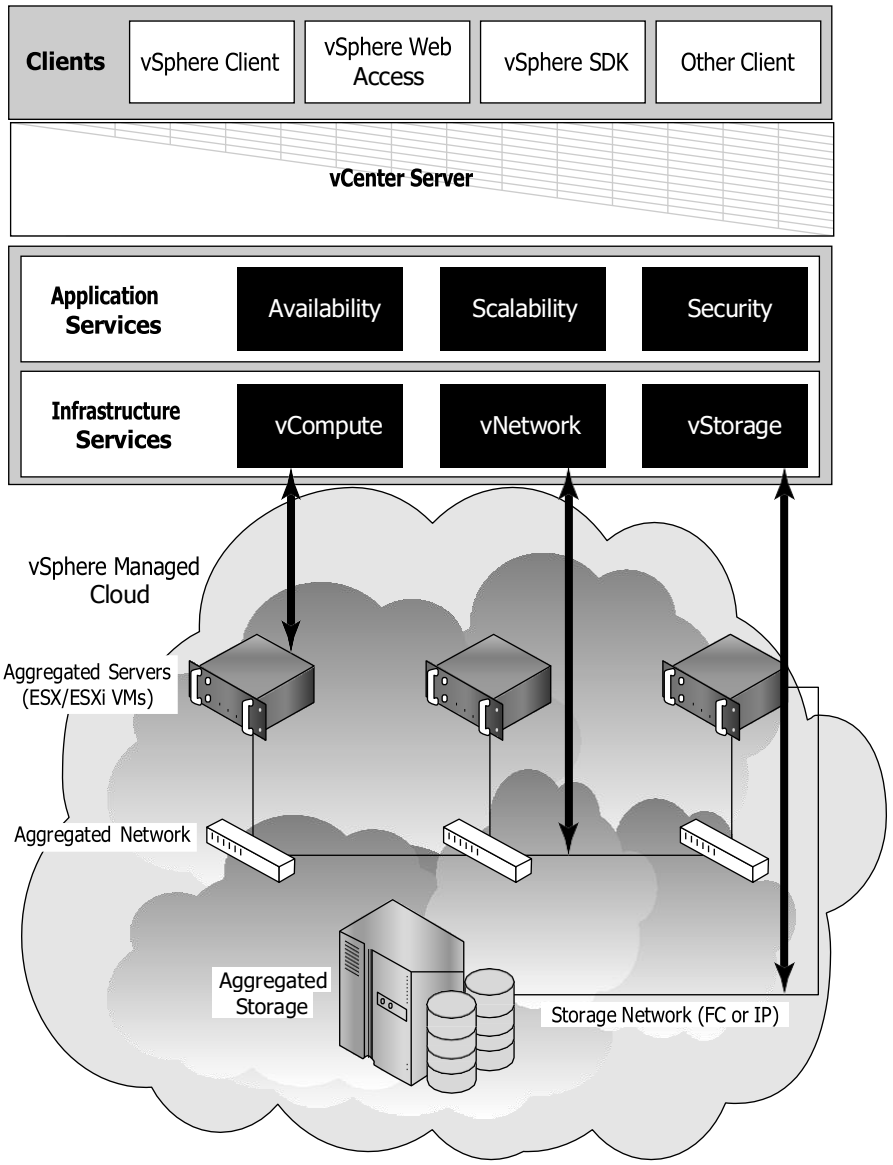
VMware vSphere is a management infrastructure framework that virtualizes system, storage, and networking hardware to create cloud computing infrastructures. vSphere is the branding for a set of management tools and a set of products previously labeled VMware Infrastructure. vSphere provides a set of services that applications can use to access cloud resources, including these:

- **VMware vCompute:** A service that aggregates servers into an assignable pool
- **VMware vStorage:** A service that aggregates storage resources into an assignable pool
- **VMware vNetwork:** A service that creates and manages virtual network interfaces
- **Application services:** Such as HA (High Availability) and Fault Tolerance
- **vCenter Server:** A provisioning, management, and monitoring console for VMware cloud infrastructures

Figure 5.3 shows an architectural diagram of a vSphere cloud infrastructure.

FIGURE 5.3

VMware's vSphere cloud computing infrastructure model



A vSphere cloud is a pure infrastructure play. The virtualization layer that abstracts processing, memory, and storage uses the VMware ESX or ESXi virtualization server. ESX is a Type 1 hypervisor; it installs over bare metal (a clean system) using a Linux kernel to boot and installs the vmkernel hypervisor (virtualization kernel and support files). When the system is rebooted, the vmkernel loads first, and then the Linux kernel becomes the first guest operating system to run as a virtual machine on the system and contains the service console.

VMware is a very highly developed infrastructure and the current leader in this industry. A number of important add-on products are available for cloud computing applications. These are among the more notable products:

- **Virtual Machine File System (VMFS):** A high-performance cluster file system for an ESX/ESXi cluster.
- **VMotion:** A service that allows for the migration of a virtual machine from one physical server to another physical server while the virtual server runs continuously and without any interruption of ongoing transactions.

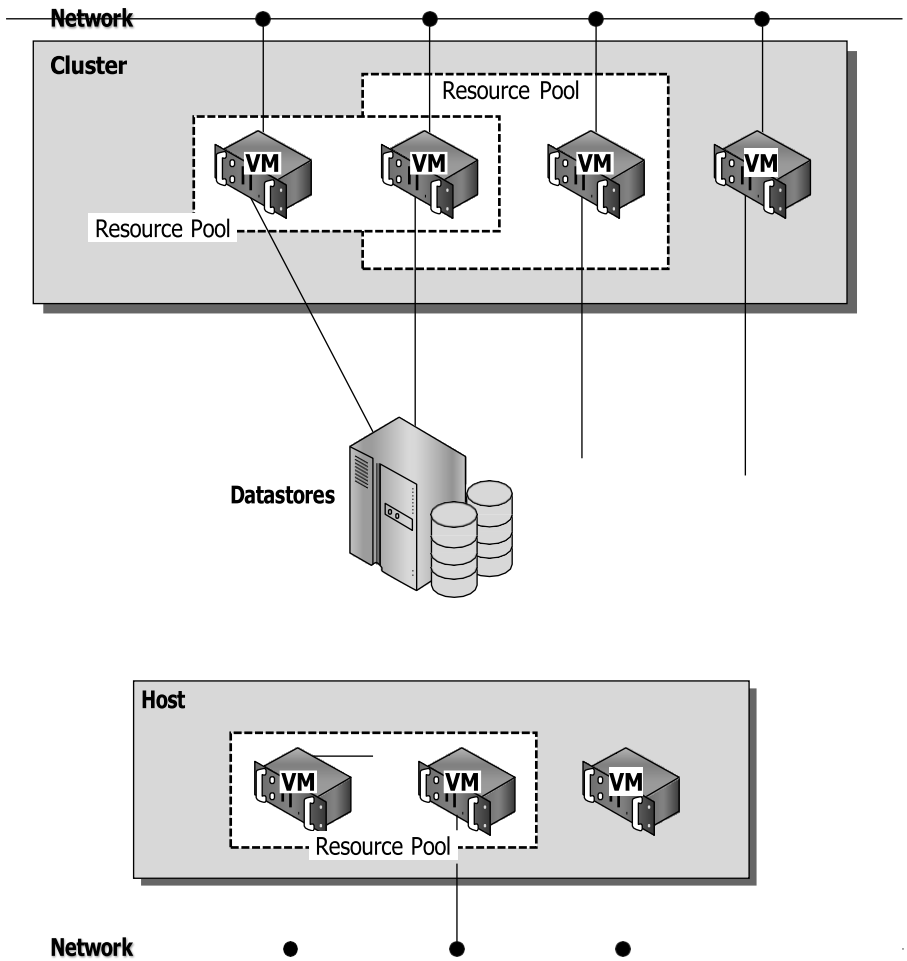
  The ability to live migrate virtual machines is considered to be a technological tour de force and a differentiator from other virtual machine system vendors.
- **Storage VMotion:** A product that can migrate files from one datastore to another datastore while the virtual machine that uses the datastore continues to run.
- **Virtual SMP:** A feature that allows a virtual machine to run on two or more physical processors at the same time.
- **Distributed Resource Scheduler (DRS):** A system for provisioning virtual machines and load balancing processing resources dynamically across the different physical systems that are in use. A part of the DRS called the distributed power management (DPM) module can manage the power consumption of systems.
- **vNetwork Distributed Switch (DVS):** A capability to maintain a network runtime state for virtual machines as they are migrated from one physical system to another. DVS also monitors network connections, provides firewall services, and enables the use of third-party switches such as the Cisco Nexus 1000V to manage virtual networks.

You can get a better sense of how the different resources are allocated by vSphere into a virtual set of components by examining Figure 5.4. Physical computers can be standalone hosts or a set of clustered systems. In either case, a set of virtual machines can be created that is part of a single physical system or spans two or more physical systems.

You can define a group of VMs as a Resource Pool (RP) and, by doing so, manage those virtual machines as a single object with a single policy. Resource Pools can be placed into a hierarchy or nested and can inherit properties of their parent RP. As more hosts or cluster nodes are added or removed, vSphere can dynamically adjust the provisioning of VMs to accommodate the policy in place. This fine tuning of pooled resources is required to accommodate the needs of cloud computing networks.

FIGURE 5.4

Virtual infrastructure elements



The datastore shown at the center of Figure 5.4 is a shared storage resource. These storage resources can be either Direct Attached Storage (DAS) of a server using SCSI, SAS, or SATA connections, Fibre Channel disk arrays/SANs, iSCSI disk arrays/SANs, or Network Attached Storage (NAS) disk arrays. Although the lines drawn between the datastore and different VMs indicate a direct connection, with the exception of DAS, the other storage types are shared storage solutions.

Storage virtualization is most commonly achieved through a mapping mechanism where a logical storage address is translated into a physical storage address. Block-based storage such as those used

in SANs use a feature called a Logical Unit Identifier (LUN**)** with specific addresses stored in the form of an offset called the Logical Block Address (LBA). The address space mapping then maps the address of the logical or virtual disk (vdisk) to the logical unit on a storage controller. Storage virtualization may be done in software or in hardware, and it allows requests for virtualized storage to be redirected as needed.

Similarly, network virtualization abstracts networking hardware and software into a virtual network that can be managed. A virtual network can create virtual network interfaces (VNICs) or virtual LANs (VLANS) and can be managed by a hypervisor, operating system, or external management console. In a virtualized infrastructure such as the one presented in this section, internal network virtualization is occurring and the hypervisor interacts with networking hardware to create a pseudo-network interface. External network virtualization can be done using network switches and VLAN software.

The key feature that makes virtual infrastructure so appealing for organizations implementing a cloud computing solution is flexibility. Instantiating a virtual machine is a very fast process, typically only a few seconds in length. You can make machine images of systems in the configuration that you want to deploy or take snapshots of working virtual machines. These images can be brought on-line as needed.

Understanding Machine Imaging

In the preceding sections, you have seen how the abstractions that cloud computing needs can be achieved through redirection and virtualization. A third mechanism is commonly used to provide system portability, instantiate applications, and provision and deploy systems in the cloud. This third mechanism is through storing the state of a systems using a system image.

Cross-Ref

Backing up to the cloud often involves imaging or snapshot applications; this process is described in Chapter 15, "Working with Cloud-Based Storage." ■

A system image makes a copy or a clone of the entire computer system inside a single container such as a file. The system imaging program is used to make this image and can be used later to restore a system image. Some imaging programs can take snapshots of systems, and most allow you to view the files contained in the image and do partial restores.

Note

The one open standard for storing a system image is the Open Virtualization Format (OVF; see `http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf`**) that is published by the Distributed Task Format (DMTF;** `http://www.dmtf.org/`**). Some notable virtualization vendors, such as VMware, Microsoft, Citrix, and Oracle (Sun), are supporting this effort. ■**

A prominent example of a system image and how it can be used in cloud computing architectures is the Amazon Machine Image (AMI) used by Amazon Web Services to store copies of a virtual machine. Because this is a key feature of Amazon's Elastic Compute Cloud and is discussed in detail in Chapter 9, I briefly mention it here. An AMI is a file system image that contains an operating system, all appropriate device drivers, and any applications and state information that the working virtual machine would have.

When you subscribe to AWS, you can choose to use one of its hundreds of canned AMIs or to create a custom system and capture that system's image to an AMI. An AMI can be for public use under a free distribution license, for pay-per-use with operating systems such as Windows, or shared by an EC2 user with other users who are given the privilege of access.

Cross-Ref

Refer to Chapter 9, "Using Amazon Web Services," for more information about AMIs and their uses in the EC2 service. ■

The AMI file system is not a standard bit-for-bit image of a system that is common to many disk imaging programs. AMI omits the kernel image and stores a pointer to a particular kernel that is part of the AWS kernel library. Among the choices are Red Hat Linux, Ubuntu, Microsoft Windows, Solaris, and others. Files in AMI are compressed and encrypted, and an XML file is written that describes the AMI archive. AMIs are typically stored in your Amazon S3 (Simple Storage System) buckets as a set of 10MB chunks.

Machine images are sometimes referred to as "virtual appliances"—systems that are meant to run on virtualization platforms. AWS EC2 runs on the Xen hypervisor, for example. The term *virtual appliance* is meant to differentiate the software image from an operating virtual machine. The system image contains the operating system and applications that create an environment. Most virtual appliances are used to run a single application and are configurable from a Web page. Virtual appliances are a relatively new paradigm for application deployment, and cloud computing is the major reason for the interest in them and for their adoption. This area of WAN application portability and deployment, and of WAN optimization of an application based on demand, is one with many new participants. Certeon (`http://www.certeon.com/`), Expand Networks (`http://www.expand.com/`), and Replify (`http://www.replify.com/`) are three vendors offering optimization appliances for VMware's infrastructure.

Porting Applications

Cloud computing applications have the ability to run on virtual systems and for these systems to be moved as needed to respond to demand. Systems (VMs running applications), storage, and network assets can all be virtualized and have sufficient flexibility to give acceptable distributed WAN application performance. Developers who write software to run in the cloud will undoubtedly want the ability to port their applications from one cloud vendor to another, but that is a much more difficult proposition. Cloud computing is a relatively new area of technology, and the major vendors have technologies that don't interoperate with one another.

The Simple Cloud API

If you build an application on a platform such as Microsoft Azure, porting that application to Amazon Web Services or GoogleApps may be difficult, if not impossible. In an effort to create an interoperability standard, Zend Technologies has started an open source initiative to create a common application program interface that will allow applications to be portable. The initiative is called the Simple API for Cloud Application Services (`http://www.simplecloud.org/`), and the effort has drawn interest from several major cloud computing companies. Among the founding supporters are IBM, Microsoft, Nivanix, Rackspace, and GoGrid.

Simple Cloud API has as its goal a set of common interfaces for:

- **File Storage Services:** Currently Amazon S3, Windows Azure Blob Storage, Nirvanix, and Local storage is supported by the Storage API. There are plans to extend this API to Rackspace Cloud Files and GoGrid Cloud Storage.
- **Document Storage Services:** Amazon SimpleDB and Windows Azure Table Storage are currently supported. Local document storage is planned.
- **Simple Queue Services:** Amazon SQS, Windows Azure Queue Storage, and Local queue services are supported.

Zend intends to add the interface to their open source PHP Framework (`http://www.framework.zend.com`) as the Zend_Cloud framework component. Vendors such as Microsoft and IBM are supplying adapters that will use part of the Simple Cloud API for their cloud application services.

AppZero Virtual Application Appliance

Applications that run in datacenters are captive to the operating systems and hardware platforms that they run on. Many datacenters are a veritable Noah's Ark of computing. So moving an application from one platform to another isn't nearly as simple as moving a machine image from one system to another.

The situation is further complicated by the fact that applications are tightly coupled with the operating systems on which they run. An application running on Windows, for example, isn't isolated from other applications. When the application loads, it often loads or uses different Dynamic Link Libraries (DLL), and it is through the sharing or modification of DLLs that Windows applications get themselves in trouble. Further modifications include modifying the registry during installation. These factors make it difficult to port applications from one platform to another without lots of careful work. If you are a Platform as a Service (PaaS) application developer, you are packaging a complete software stack that includes not only your application, but the operating system and application logic and rules as well. Vendor lock-in for you application is assured.

The ability to run an application from whatever platform you want is not one of the characteristics of cloud computing, but you can imagine that it is a very attractive proposition. While the Simple Cloud API is useful for applications written in PHP, other methods may be needed to make applications easily portable. One company working on this problem is AppZero (`http://www.appzero.com/`), and its solution is called the Virtual Application Appliance (VAA).

The AppZero solution creates a virtual application appliance as an architectural layer between the Windows or the UNIX operating system and applications. The virtualization layer serves as the mediator for file I/O, memory I/O, and application calls and response to DLLs, which has the effect of sandboxing the application. The running application in AppZero changes none of the registry entries or any of the files on the Windows Server.

VAA creates a container that encapsulates the application and all the application's dependencies within a set of files; it is essentially an Application Image for a specific OS. Dependencies include DLL, service settings, necessary configuration files, registry entries, and machine and network settings. This container forms an installable server-side application stack that can be run after installation, but has no impact on the underlying operating system. VAAs are created using the AppZero Creator wizard, managed with the AppZero Admin tool, and may be installed using the AppZero Director, which creates a VAA runtime application. If desired, an application called AppZero Dissolve removes the VAA virtualization layer from the encapsulated application and installs that application directly into the operating system.

Note

Microsoft App-V (`http://www.microsoft.com/windows/enterprise/products/mdop/app-v.aspx`**) and VMware ThinApp (**`http://www.vmware.com/products/thinapp/`**) are two application delivery platforms, but their main focus is on desktop installations and not on server deployment in the cloud.** ■

Installations can be done over the network after the AppZero application appliance is installed. Therefore, with this system, you could run applications on the same Windows Server and eliminate one application from interfering with another; applications would be much more easily ported from one Windows system to another. AppZero's approach provides the necessary abstraction layer that frees an application from its platform dependence.

An interesting use of VAAs involves segmenting an application into several VAAs, some of which are read-only runtime components, while others can be modified. When backing up or replicating VAAs in a cloud, you would need to synchronize only those VAAs that are modified. In many instances, the portion of an application that changes is only a very small component of large applications, which means that this technique can greatly reduce the amount of data required to replicate a VM in the cloud.

AppZero envisages using VAAs to create what it calls a *stateless cloud.* In a stateless cloud, the application's state information is stored on a network share where it is available to run on different cloud systems as needed. This approach allows the cloud system to run with a VM containing a clean operating system (like AWS does now) and provisioned by the VAA. This approach should greatly reduce the number of complete system images that cloud vendors and cloud users should need to store to support their work; it also should make the running of applications on secure, well-performing VEs easier to achieve.

Summary

In this chapter, you learned about some of the more important characteristics of cloud computing networks and applications, including ubiquitousness and on-demand service. To enable a cloud service, you need to create a pool of resources you can call on. The key techniques for enabling this are abstraction and virtualization. Abstraction maps a logical identity or address to a physical identity or address. Changes to the underlying systems, therefore, do not affect the client requesting a service.

Several different methods for abstraction have been considered. A widely used technique is load balancing. With load balancing, system requests are directed to appropriate systems on demand. All large cloud networks use some form of load balancing. You learned about some of the details of Google's load balancing for queries.

Another technology virtualizes hardware. You learned about the different types of hypervisors—software that can serve as a virtualization layer for operating systems accessing the underlying hardware. As an example of hardware virtualization VMware's vSphere infrastructure was considered. vSphere can create virtual machines, virtual datastores, and virtual networks, and move these resources about while the system is active. vSphere is a potent cloud-building technology.

System imaging also can be useful in creating and instantiating machine instances. A brief explanation of Amazon Machine Instances was given.

Finally, the topic of application portability was considered. Applications are hard to move from platform to platform, because they are bound up with the operating system on which they run. Eventually, applications will be as portable as virtual machines. A cloud programming interface was described, as was an application delivery appliance.

In Chapter 6, "Capacity Planning," the idea of system workloads is described. Understanding this concept allows you to scale your systems correctly, choose the right type of infrastructure, and do availability planning. Some of the key performance metrics for cloud computing "right sizing" are described.

CHAPTER

6

Capacity Planning

Capacity planning seeks to match demand to available resources. Capacity planning examines what systems are in place, measures their performance, and determines patterns in usage that enables the planner to predict demand. Resources are provisioned and allocated to meet demand.

Although capacity planning measures performance and in some cases adds to the expertise needed to improve or optimize performance, the goal of capacity planning is to accommodate the workload and not to improve efficiency. Performance tuning and optimization is not a primary goal of capacity planners.

To successfully adjust a system's capacity, you need to first understand the workload that is being satisfied and characterize that workload. A system uses resources to satisfy cloud computing demands that include processor, memory, storage, and network capacity. Each of these resources has a utilization rate, and one or more of these resources reaches a ceiling that limits performance when demand increases.

It is the goal of a capacity planner to identify the critical resource that has this resource ceiling and add more resources to move the bottleneck to higher levels of demand.

Scaling a system can be done by scaling up vertically to more powerful systems or by scaling out horizontally to more but less powerful systems. This is a fundamental architectural decision that is affected by the types of workloads that cloud computing systems are being asked to perform. This chapter presents some of these tradeoffs.

IN THIS CHAPTER

- Learning about capacity planning for the cloud
- Capturing baselines and metrics
- Determining resources and their ceilings
- Scaling your systems appropriately

Network capacity is one of the hardest factors to determine. Network performance is affected by network I/O at the server, network traffic from the cloud to Internet service providers, and over the last mile from ISPs to homes and offices. These factors are also considered.

Capacity Planning

Capacity planning for cloud computing sounds like an oxymoron. Why bother doing capacity planning for a resource that is both ubiquitous and limitless? The reality of cloud computing is rather different than the ideal might suggest; cloud computing is neither ubiquitous, nor is it limitless. Often, performance can be highly variable, and you pay for what you use. That said, capacity planning for a cloud computing system offers you many enhanced capabilities and some new challenges over a purely physical system.

A capacity planner seeks to meet the future demands on a system by providing the additional capacity to fulfill those demands. Many people equate capacity planning with system optimization (or performance tuning, if you like), but they are not the same. System optimization aims to get more production from the system components you have. Capacity planning measures the maximum amount of work that can be done using the current technology and then adds resources to do more work as needed. If system optimization occurs during capacity planning, that is all to the good; but capacity planning efforts focus on meeting demand. If that means the capacity planner must accept the inherent inefficiencies in any system, so be it.

Note

Capacity and performance are two different system attributes. With capacity, you are concerned about how much work a system can do, whereas with performance, you are concerned with the rate at which work gets done. ■

Capacity planning is an iterative process with the following steps:

1. Determine the characteristics of the present system.
2. Measure the workload for the different resources in the system: CPU, RAM, disk, network, and so forth.
3. Load the system until it is overloaded, determine when it breaks, and specify what is required to maintain acceptable performance.

   Knowing when systems fail under load and what factor(s) is responsible for the failure is the critical step in capacity planning.
4. Predict the future based on historical trends and other factors.
5. Deploy or tear down resources to meet your predictions.
6. Iterate Steps 1 through 5 repeatedly.

Defining Baseline and Metrics

The first item of business is to determine the current system capacity or workload as a measurable quantity over time. Because many developers create cloud-based applications and Web sites based on a LAMP solution stack, let's use those applications for our example in this chapter.

LAMP stands for:

- **L**inux, the operating system
- **A**pache HTTP Server (`http://httpd.apache.org/`), the Web server based on the work of the Apache Software Foundation
- **M**ySQL (`http://www.mysql.com`), the database server developed by the Swedish company MySQL AB, owned by Oracle Corporation through its acquisition of Sun Microsystems
- **P**HP (`http://www.php.net/`), the Hypertext Preprocessor scripting language developed by The PHP Group

Note

Either Perl or Python is often substituted for PHP as the scripting language used. ■

These four technologies are open source products, although the distributions used may vary from cloud to cloud and from machine instance to machine instance. On Amazon Web Services, machine instances are offered for both Red Hat Linux and for Ubuntu. LAMP stacks based on Red Hat Linux are more common than the other distributions, but SUSE Linux and Debian GNU/Linux are also common. Variants of LAMP are available that use other operating systems such as the Macintosh, OpenBSD (OpAMP), Solaris (SAMP), and Windows (WAMP).

Although many other common applications are in use, LAMP is good to use as an example because it offers a system with two applications (APACHE and MySQL) that can be combined or run separately on servers. LAMP is one of the major categories of Amazon Machine Instances that you can create on the Amazon Web Service (see Chapter 9).

Baseline measurements

Let's assume that a capacity planner is working with a system that has a Web site based on APACHE, and let's assume the site is processing database transactions using MySQL. There are two important overall workload metrics in this LAMP system:

- **Page views** or hits on the Web site, as measured in hits per second
- **Transactions** completed on the database server, as measured by transactions per second or perhaps by queries per second

In Figure 6.1, the historical record for the Web server page views over a hypothetical day, week, and year are graphed. These graphs are created by summing the data from the different servers

contained in the performance Web logs of the site or measuring throughput from the system based on an intervening service (such as a proxy server). What Figure 6.1 shows is a measure of the *overall* workload of the Web site involved.

Tip

Your performance logs are a primary source of data that everyone should have available for planning purposes, but they are not the only source of performance measurements. A number of companies offer services that can monitor your Web service's performance. For example, sites such as Alertra (`http://www.alertra.com`), Keynote (`http://www.keynote.com`), Gomez (`http://www.gomez.com`), PingDom (`http://www.pingdom.com`), and SiteUpTime (`http://www.siteuptime.com`) can monitor your Web pages to determine their response, latency, uptime, and other characteristics. Gomez and Keynote expose your site data visually as a set of dashboard widgets. Because these are third-party services, many Web services use these sites as the compliance monitor for Service Level Agreements (SLAs). ■

Notice several things about the graphs in Figure 6.1. A typical daily log (shown on top) shows two spikes in demand, one that is centered around 10 AM EST when users on the east coast of the United States use the site heavily and another at around 1 PM EST (three hours later; 10 AM PST) when users on the west coast of the United States use the site heavily. The three-hour time difference is also the difference between Eastern Standard Time and Pacific Standard Time.

These daily performance spikes occur on workdays, but the spikes aren't always of equal demand, as you can see from the weekly graph, shown in the middle of Figure 6.1. On weekends (Saturday and Sunday), the demand rises through the middle of the day and then ebbs later. Some days show a performance spike, as shown on the Tuesday morning section of the weekly graph. A goal of capacity planning (and an important business goal) is to correlate these performance spikes and dips with particular events and causations. Also, at some point your traffic patterns may change, so you definitely want to evaluate these statistics on an ongoing basis.

The yearly graph, at the bottom of Figure 6.1, shows that the daily averages and the daily peaks rise steadily over the year and, in fact, double from January 1st at the start of the year to December 31st at year end. Knowing this, a capacity planner would need to plan to serve twice the traffic while balancing the demands of peak versus average loads. However, it may not mean that twice the resources need to be deployed. The amount of resources to be deployed depends upon the characterization of the Web servers involved, their potential utilization rates, and other factors.

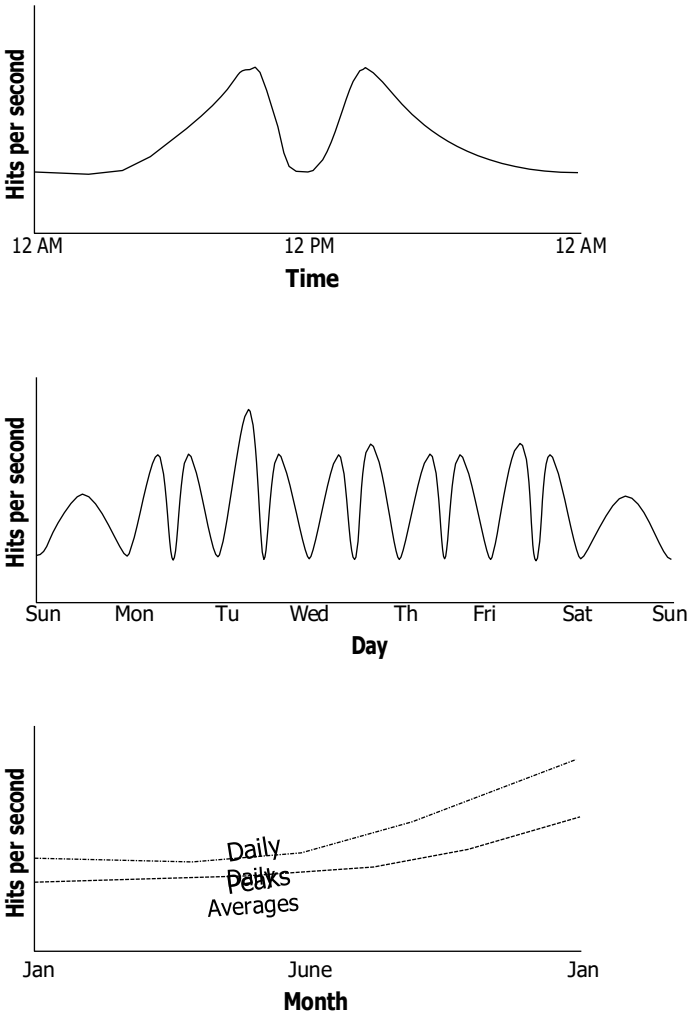
Caution

In predicting trends, it is important that the time scale be selected appropriately. A steady sure rise in demand over a year is much more accurate in predicting future capacity requirements than a sharp spike over a day or a week. Forecasting based on small datasets is notoriously inaccurate. You can't accurately predict the future (no one can), but you can make good guesses based on your intuition. ■

The total workload might be served by a single server instance in the cloud, a number of virtual server instances, or some combination of physical and virtual servers.

FIGURE 6.1

A Web servers' workload measured on a day, a week, and over the course of a year



A number of important characteristics are determined by these baseline studies:

- W_T, the total workload for the system per unit time

  To obtain W_T, you need to integrate the area under the curve for the time period of interest.

- WAVG, the average workload over multiple units of time

  To obtain W_{AVG}, you need to sum various W_Ts and divide by the number of unit times involved. You may also want to draw a curve that represents the mean work done.

- W_{MAX}, the highest amount of work recorded by the system

  This is the highest recorded system utilization. In the middle graph of Figure 6.1, it would be the maximum number recorded on Tuesday morning.

- W_{TOT}, the total amount of work done by the system, which is determined by the sum of W_T (ΣW_T)

A similar set of graphs would be collected to characterize the database servers, with the workload for those servers measured in transactions per second. As part of the capacity planning exercise, the workload for the Web servers would be correlated with the workload of the database servers to determine patterns of usage.

The goal of a capacity planning exercise is to accommodate spikes in demand as well as the overall growth of demand over time. Of these two factors, the growth in demand over time is the most important consideration because it represents the ability of a business to grow. A spike in demand may or may not be important enough to an activity to attempt to capture the full demand that the spike represents.

System metrics

Notice that in the previous section, you determined what amounts to application-level statistics for your site. These measurements are an aggregate of one or more Web servers in your infrastructure. Capacity planning must measure system-level statistics, determining what each system is capable of, and how resources of a system affect system-level performance.

In some instances, the capacity planner may have some influence on system architecture, and the impact of system architecture on application- and system-level statistics may be examined and altered. Because this is a rare case, let's focus on the next step, which defines system performance.

A machine instance (physical or virtual) is primarily defined by four essential resources:

- CPU
- Memory (RAM)
- Disk
- Network connectivity

Each of these resources can be measured by tools that are operating-system-specific, but for which tools that are their counterparts exist for all operating systems. Indeed, many of these tools were written specifically to build the operating systems themselves, but that's another story. In Linux/UNIX, you might use the `sar` command to display the level of CPU activity. `sar` is installed in Linux as part of the `sysstat` package. In Windows, a similar measurement may be made using the Task Manager, the data from which can be dumped to a performance log and/or graphed.

Some tools give you a historical record of performance, which is particularly useful in capacity planning. For example, the popular Linux performance measurement tool `RRDTool` (the Round Robin Database tool; `http://oss.oetiker.ch/rrdtool/`) is valuable in this regard.

RRDTool is a utility that can capture time-dependent performance data from resources such as a CPU load, network utilization (bandwidth), and so on and store the data in a circular buffer. It is commonly used in performance analysis work.

A time interval in RRDTool is called a step, with the value in a step referred to as a primary data point (PDP). When a function is applied to a data point (average, minimum, maximum, and so on), the function is referred to as a Consolidation Function (CF), and the value obtained is a Consolidated Data Point (CDP). An interval is stored in the Round Robin Archive (RRA), and when that interval is filled, it is replaced by new step data. RRDTool includes a graphical front end for displaying performance results visually. RRDTool is widely used for a number of different purposes. Figure 6.2 shows some of the examples from a gallery of RRDTool graphs found at http://oss.oetiker.ch/rrdtool/gallery/.

FIGURE 6.2

RRDTool lets you create historical graphs of a wide variety of performance data. Some samples are shown in the gallery at http://oss.oetiker.ch/rrdtool/gallery/.

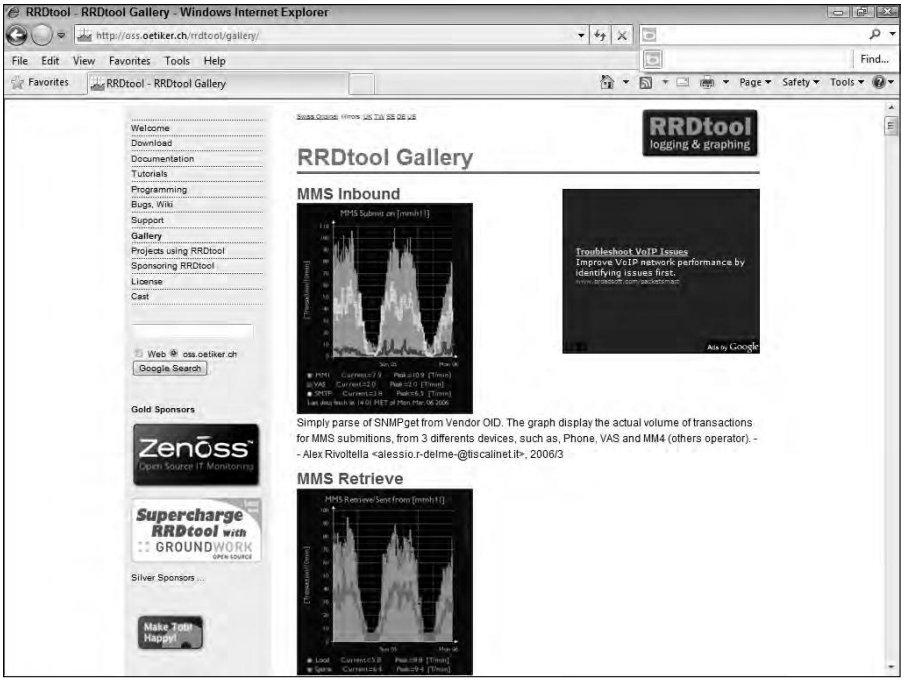


Table 6.1 lists some LAMP performance testing tools.

TABLE 6.1

LAMP Performance Monitoring Tools

Tool Name	Web Site	Developer	Description
Alertra	http://www.alertra.com	Alertra	Web site monitoring service
Cacti	http://www.cacti.net	Cacti	Open source RRDTool graphing module
Collectd	http://www.collectd.org/	collectd	System statistics collection daemon
Dstat	http://dag.wieers.com/home-made/dstat/	DAG	System statistics utility; replaces `vmstat`, `iostat`, `netstat`, and `ifstat`
Ganglia	http://www.ganglia.info	Ganglia	Open source distributed monitoring system
Gomez	http://www.gomez.com	Gomez	Commercial third-party Web site performance monitor
GraphClick	http://www.arizona-software.ch/graphclick/	Arizona	A digitizer that can create a graph from an image
GroundWork	http://www.groundworkopensource.com/	Groundwork's Open Source	Network monitoring solution
Hyperic HQ	http://www.hyperic.com	Spring Source	Monitoring and alert package for virtualized environments
Keynote	http://www.keynote.com	Keynote	Commercial third-party Web site performance monitor
Monit	http://www.tildeslash.com/monit	Monit	Open source process manager
Munin	http://munin.projects.linpro.no/	Munin	Open source network resource monitoring tool
Nagios	http://www.nagios.org	Nagios	Metrics collection and event notification tool
OpenNMS	http://www.opennms.org	OpenNMS	Open source network management platform
Pingdom	http://www.pingdom.com	Pingdom	Uptime and performance monitor
RRDTool	http://www.RRDTool.org/	Oetiker+Partner AG	Graphing and performance metrics storage utility
SiteUpTime	http://www.siteuptime.com	SiteUpTime	Web site monitoring service
Zabbix	http://www.zabbix.com	Zabbix	Performance monitor
ZenOSS	http://www.zenoss.com/	Zenoss	Operations monitor, both open source and commercial versions