MODULE 1

INTRODUCTION

Computing is being transformed into a model consisting of services that are commoditized and delivered in a manner similar to utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements, regardless of where the services are hosted. *Cloud computing* is the most recent emerging paradigm promising to turn the vision of "computing utilities" into a reality.

Cloud computing is a technological advancement it is based on the concept of *dynamic provisioning*, which is applied not only to services but also to compute capability, storage, networking, and information technology (IT) infrastructure in general. Resources are made available through the Internet and offered on a *pay-per-use* basis from cloud computing vendors.

Cloud computing at a glance

In 1969, Leonard Kleinrock, one of the chief scientists of the original Advanced Research Projects Agency Network (ARPANET), which seeded the Internet, said:

> *As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of* 'computer utilities' *which, like present electric and telephone utilities, will service individual homes and offices across the country.*

Cloud computing allows renting infrastructure, runtime environments, and services on a pay-per-use basis. End users leveraging cloud computing services can access their documents and data anytime, anywhere, and from any device connected to the Internet. One of the most diffuse views of cloud computing can be summarized as follows:

I don't care where my servers are, who manages them, where my documents are stored, or where my applications are hosted. I just want them always available and access them from any device connected through Internet. And I am willing to pay for this service for as a long as I need it.

Web 2.0 technologies play a central role in making cloud computing an attractive opportunity for building computing systems. *Service orientation* allows cloud computing to deliver its capabilities with familiar abstractions, while *virtualization* confers on cloud computing the necessary degree of customization, control, and flexibility for building production and enterprise systems.

The vision of cloud computing

Cloud computing allows anyone with a credit card to provision virtual hardware, runtime environments, and services. These are used for as long as needed, with no up-front commitments required. The entire stack of a computing system is transformed into a collection of utilities, which can be provisioned and composed together to deploy systems in hours rather than days and with virtually no maintenance costs.

Figure 1.1 Cloud computing vision

Previously, the lack of effective standardization efforts made it difficult to move hosted services from one vendor to another. The long-term vision of cloud computing is that IT services are traded as utilities in an open market, without technological and legal barriers. In this cloud

marketplace, cloud service providers and consumers, trading cloud services as utilities, play a central role.

Many of the technological elements contributing to this vision already exist. The capability for Web- based access to documents and their processing using sophisticated applications is one of the appealing factors for end users.

Vision of cloud computing is that in the near future it will be possible to find the solution that matches the customer needs by simply entering our request in a global digital market that trades cloud computing services.

Defining a cloud

Different notions of cloud computing is as shown in figure 1.2. The Internet plays a fundamental role in cloud computing, since it represents either the medium or the platform through which many cloud computing services are delivered and made accessible. This aspect is also reflected in the definition given by Armbrust:

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the datacenters that provide those services.

This definition describes cloud computing as a phenomenon touching on the entire stack: from the underlying hardware to the high-level software services and applications. It introduces the concept of *everything as a service*, mostly referred as *XaaS*, where the different components of a system—IT infrastructure, development platforms, databases, and so on—can be delivered, measured, and consequently priced as a service. The approach fostered by cloud computing is global.

This notion of multiple parties using a shared cloud computing environment is highlighted in a definition proposed by the U.S. National Institute of Standards and Technology (NIST):

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

According to Reese, we can define three criteria to discriminate whether a service is delivered in the cloud computing style:

- **The service is accessible via a Web browser (nonproprietary) or a Web services application programming interface (API).**
- **Zero capital expenditure is necessary to get started.**
- **You pay only for what you use as you use it.**

The utility-oriented nature of cloud computing is clearly expressed by Buyya:
A cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.

A closer look

Cloud computing is helping enterprises, governments, public and private institutions, and research organizations shape more effective and demand-driven computing systems. Practical examples of such systems exist across all market segments:

- ***Large enterprises can offload some of their activities to cloud-based systems.*** Recently, the *New York Times* has converted its digital library of past editions into a Web-friendly format. This required a considerable amount of computing power for a short period of time. By renting Amazon EC2 and S3 Cloud resources, the *Times* performed this task in 36 hours and relinquished these resources, with no additional costs.

- ***Small enterprises and start-ups can afford to translate their ideas into business results more quickly, without excessive up-front costs***. Animoto is a company that creates videos out of images, music, and video fragments submitted by users. The process involves a considerable amount of storage and backend processing required for producing the video, which is finally made available to the user. Animoto does not own a single server and bases its computing infrastructure entirely on Amazon Web Services.

- ***System developers can concentrate on the business logic rather than dealing with the complexity of infrastructure management and scalability***. Little Fluffy Toys is a

company in London that has developed a widget providing users with information about nearby bicycle rental services. The company has managed to back the widget's computing needs on Google AppEngine and be on the market in only one week.

- ***End users can have their documents accessible from everywhere and any device.*** Apple iCloud is a service that allows users to have their documents stored in the Cloud and access them from any device users connect to it.

Cloud computing does not only contribute with the opportunity of easily accessing IT services on demand, it also introduces a new way of thinking about IT services and resources: as utilities. A bird's-eye view of a cloud computing environment is shown in Figure 1.3.

The three major models for deploying and accessing cloud computing environments are public clouds, private/enterprise clouds, and hybrid clouds. Figure 1.4.
Public clouds are the most common deployment models in which necessary IT infrastructure (e.g., virtualized datacenters) is established by a third-party service provider that makes it available to any consumer on a subscription basis. Such clouds are appealing to users because they allow users to quickly leverage compute, storage, and application services. In this environment, users' data and applications are deployed on cloud datacenters on the vendor's premises.

Large organizations that own massive computing infrastructures can still benefit from cloud computing by replicating the cloud IT service delivery model in-house. This idea has given birth to the concept of *private clouds* as opposed to public clouds. In 2010, for example, the U.S. federal government, one of the world's largest consumers of IT spending started a cloud computing initiative aimed at providing government agencies with a more efficient use of their computing facilities. The use of cloud-based in-house solutions is also driven by the need to keep confidential information within an organization's premises. Institutions such as governments and banks that have high security, privacy, and regulatory concerns prefer to build and use their own private or enterprise clouds.

Whenever private cloud resources are unable to meet users' quality-of-service requirements, hybrid computing systems, partially composed of public cloud resources and privately owned infra- structures, are created to serve the organization's needs. These are often referred as *hybrid*

clouds, which are becoming a common way for many stakeholders to start exploring the possibilities offered by cloud computing.

The cloud computing reference model

A fundamental characteristic of cloud computing is the capability to deliver, on demand, a variety of IT services that are quite diverse from each other. This variety creates different perceptions of what cloud computing is among users. Despite this lack of uniformity, it is possible to classify cloud computing services offerings into three major categories: *Infrastructure-as-a-Service (IaaS)*,
Platform-as-a-Service (PaaS), and
Software-as-a-Service (SaaS).
These categories are related to each other as described in Figure 1.5
The model organizes the wide range of cloud computing services into a layered view that walks the computing stack from bottom to top.

At the base of the stack, *Infrastructure-as-a-Service* solutions deliver infrastructure on demand in the form of virtual *hardware*, *storage*, and *networking*. Virtual hardware is utilized to provide compute on demand in the form of virtual machine instances. These are created at users' request on the provider's infrastructure, and users are given tools and interfaces to configure the software stack installed in the virtual machine. The pricing model is usually defined in terms of dollars per hour. Virtual storage is delivered in the form of raw disk space or object store. Virtual networking identifies the collection of services that manage the networking among virtual instances and their connectivity to the Internet or private networks.

Platform-as-a-Service solutions are the next step in the stack. They deliver scalable and elastic runtime environments on demand and host the execution of applications. These services are backed by a core middleware platform that is responsible for creating the abstract environment where applications are deployed and executed. It is the responsibility of the service provider to provide scalability and to manage fault tolerance, while users are requested to focus on the logic of the application developed by leveraging the provider's APIs and libraries. This approach increases the level of abstraction at which cloud computing is leveraged but also constrains the user in a more controlled environment.

At the top of the stack, *Software-as-a-Service* solutions provide applications and services on demand. Most of the common functionalities of desktop applications—such as office automation, document management, photo editing, and customer relationship management (CRM) software—are replicated on the provider's infrastructure and made more scalable and accessible through a browser on demand. These applications are shared across multiple users whose interaction is isolated from the other users. The SaaS layer is also the area of social networking Websites, which leverage cloud-based infrastructures to sustain the load generated by their popularity.

Each layer provides a different service to users. IaaS solutions are sought by users who want to leverage cloud computing from building dynamically scalable computing systems requiring a specific software stack. IaaS services are therefore used to develop scalable Websites or for back- ground processing.

PaaS solutions provide scalable programming platforms for developing applications and are more appropriate when new systems have to be developed.

SaaS solutions target mostly end users who want to benefit from the elastic scalability of the cloud without doing any software development, installation, configuration, and maintenance.

Characteristics and benefits

Cloud computing has some interesting characteristics that bring benefits to both cloud service consumers (CSCs) and cloud service providers (CSPs). These characteristics are:

- No up-front commitments
- On-demand access
- Nice pricing
- Simplified application acceleration and scalability
- Efficient resource allocation
- Energy efficiency
- Seamless creation and use of third-party services

Challenges ahead

Challenges' concerning the dynamic provisioning of cloud computing services and resources

arises. For example, in the Infrastructure-as-a-Service domain, how many resources need to be provisioned, and for how long should they be used, in order to maximize the benefit? Technical challenges also arise for cloud service providers for the management of large computing infrastructures and the use of virtualization technologies on top of them.

Security in terms of confidentiality, secrecy, and protection of data in a cloud environment is another important challenge. Organizations do not own the infrastructure they use to process data and store information. This condition poses challenges for confidential data, which organizations cannot afford to reveal.

Legal issues may also arise. These are specifically tied to the ubiquitous nature of cloud computing, which spreads computing infrastructure across diverse geographical locations. Different legislation about privacy in different countries may potentially create disputes as to the rights that third parties (including government agencies) have to your data.

Historical developments

The idea of renting computing services by leveraging large distributed computing facilities has been around for long time. It dates back to the days of the mainframes in the early 1950s. From there on, technology has evolved and been refined. This process has created a series of favorable conditions for the realization of cloud computing.

Figure 1.6 provides an overview of the evolution of the distributed computing technologies that have influenced cloud computing. In tracking the historical evolution, we briefly review five core technologies that played an important role in the realization of cloud computing. These technologies are distributed systems, virtualization, Web 2.0, service orientation, and utility computing.

Distributed systems: Clouds are essentially large distributed computing facilities that make available their services to third parties on demand. As a reference, we consider the characterization of a distributed system proposed by Tanenbaum et al. [1]:

A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Three major milestones have led to cloud computing: mainframe computing, cluster computing, and grid computing.

Mainframes. These were the first examples of large computational facilities leveraging multiple processing units. Mainframes were powerful, highly reliable computers specialized for large data movement and massive input/output (I/O) operations. They were mostly used by large organizations for bulk data processing tasks such as online transactions, enterprise resource planning, and other operations involving the processing of significant amounts of data. One of the most attractive features of mainframes was the ability to be highly reliable computers that were "always on" and capable of tolerating failures transparently. No system shutdown was required to replace failed components, and the system could work without interruption. Now their popularity and deployments have reduced, but evolved versions of such systems are still in use for transaction processing (such as online banking, airline ticket booking, supermarket and telcos, and government services).

Clusters. Cluster computing started as a low-cost alternative to the use of mainframes and supercomputers. The technology advancement that created faster and more powerful mainframes and supercomputers eventually generated an increased availability of cheap commodity machines as a side effect. These machines could then be connected by a high-bandwidth network and controlled by specific software tools that manage them as a single system. Built by commodity machines, they were cheaper than mainframes and made high-performance computing available to a large number of groups, including universities and small research labs. Moreover, clusters could be easily extended if more computational power was required.

Grids. Grid computing appeared in the early 1990s as an evolution of cluster computing. In an analogy to the power grid, grid computing proposed a new approach to access large computational power, huge storage facilities, and a variety of services. Users can "consume" resources in the same way as they use other utilities such as power, gas, and water. Grids initially developed as aggregations of geographically dispersed clusters by means of Internet connections. These clusters belonged to different organizations, and arrangements were made among them to share the computational power. Several developments made possible the diffusion of computing grids: (a) clusters became quite common resources; (b) they were often underutilized; (c) new problems were requiring computational power that went beyond the capability of single clusters;

and (d) the improvements in networking and the diffusion of the Internet made possible long-distance, high-bandwidth connectivity. All these elements led to the development of grids, which now serve a multitude of users across the world.

Virtualization

Virtualization is another core technology for cloud computing. It encompasses a collection of solutions allowing the abstraction of some of the fundamental elements for computing, such as hardware, runtime environments, storage, and networking. Today virtualization has become a fundamental element of cloud computing. Virtualization confers that degree of customization and control that makes cloud computing appealing for users and, at the same time, sustainable for cloud services providers.

Virtualization is essentially a technology that allows creation of different computing environments. These environments are called *virtual* because they simulate the interface that is expected by a guest. The most common example of virtualization is *hardware virtualization*. This technology allows simulating the hardware interface expected by an operating system. Hardware virtualization allows the coexistence of different software stacks on top of the same hardware. These stacks are contained inside *virtual machine instances*, which operate in complete isolation from each other. High-performance servers can host several virtual machine instances, thus creating the opportunity to have a customized software stack on demand. This is the base technology that enables cloud computing solutions to deliver virtual servers on demand, such as Amazon EC2, RightScale, VMware vCloud, and others. Together with hardware virtualization, *storage* and *network virtualization* complete the range of technologies for the emulation of IT infrastructure. Virtualization technologies are also used to replicate runtime environments for programs.

Web 2.0

The Web is the primary interface through which cloud computing delivers its services. At present, the Web encompasses a set of technologies and services that facilitate interactive information sharing, collaboration, user-centered design, and application composition. This evolution has transformed the Web into a rich platform for application development and is known as *Web 2.0.* This term captures a new way in which developers architect applications and

deliver services through the Internet and provides new experience for users of these applications and services.

Web 2.0 brings *interactivity* and *flexibility* into Web pages, providing enhanced user experience by gaining Web-based access to all the functions that are normally found in desktop applications. These capabilities are obtained by integrating a collection of standards and technologies such as *XML, Asynchronous JavaScript and XML (AJAX), Web Services,* and others.

Service-oriented computing

Service orientation is the core reference model for cloud computing systems. This approach adopts the concept of services as the main building blocks of application and system development. *Service-oriented computing (SOC)* supports the development of rapid, low-cost, flexible, interoperable, and evolvable applications and systems.

A *service* is an abstraction representing a self-describing and platform-agnostic component that can perform any function anything from a simple function to a complex business process. Virtually any piece of code that performs a task can be turned into a service and expose its functionalities through a network-accessible protocol. A service is supposed to be *loosely coupled, reusable, programming language independent,* and *location transparent*. Services are composed and aggregated into a *service-oriented architecture (SOA).* Service-oriented computing introduces and diffuses two important concepts, which are also fundamental to cloud computing: *quality of service (QoS)* and *Software-as-a-Service (SaaS).*

Quality of service (QoS) identifies a set of functional and nonfunctional attributes that can be used to evaluate the behavior of a service from different perspectives. These could be performance metrics such as response time, or security attributes, transactional integrity, reliability, scalability, and availability. QoS requirements are established between the client and the provider via an SLA that identifies the minimum values (or an acceptable range) for the QoS attributes that need to be satisfied upon the service call.

The concept of Software-as-a-Service introduces a new delivery model for applications. The term has been inherited from the world of application service providers (ASPs), which deliver

software services-based solutions across the wide area network from a central datacenter and make them available on a subscription or rental basis.

Utility-oriented computing

Utility computing is a vision of computing that defines a service-provisioning model for compute services in which resources such as storage, compute power, applications, and infrastructure are packaged and offered on a pay-per-use basis. The idea of providing computing as a *utility* like natural gas, water, power, and telephone connection has a long history but has become a reality today with the advent of cloud computing.

Building cloud computing environments

The creation of cloud computing environments encompasses both the development of applications and systems that leverage cloud computing solutions and the creation of frameworks, platforms, and infrastructures delivering cloud computing services.

Application development

Applications that leverage cloud computing benefit from its capability to dynamically scale on demand. One class of applications that takes the biggest advantage of this feature is that of *Web applications*. These applications are characterized by complex processes that are triggered by the interaction with users and develop through the interaction between several tiers behind the Web front end.

Another class of applications that can potentially gain considerable advantage by leveraging cloud computing is represented by *resource-intensive applications*. These can be either data-intensive or compute-intensive applications. In both cases, considerable amounts of resources are required to complete execution in a reasonable timeframe. It is worth noticing that these large amounts of resources are not needed constantly or for a long duration.

Infrastructure and system development

Distributed computing, virtualization, service orientation, and Web 2.0 form the core technologies enabling the provisioning of cloud services from anywhere on the globe. Developing applications and systems that leverage the cloud requires knowledge across all these technologies.

Infrastructure-as-a-Service solutions provide the capabilities to add and remove resources, but it

is up to those who deploy systems on this scalable infrastructure to make use of such opportunities with wisdom and effectiveness. Platform-as-a-Service solutions embed into their core offering algorithms and rules that control the provisioning process and the lease of resources. These can be either completely transparent to developers or subject to fine control. Web 2.0 technologies constitute the interface through which cloud computing services are delivered, managed, and provisioned. Besides the interaction with rich interfaces through the Web browser, Web services have become the primary access point to cloud computing systems from a programmatic standpoint. Service orientation is the underlying paradigm that defines the architecture of a cloud computing system.

Virtualization is another element that plays a fundamental role in cloud computing. This technology is a core feature of the infrastructure used by cloud providers.

Computing platforms and technologies

Amazon web services (AWS)

AWS offers comprehensive cloud IaaS services ranging from virtual compute, storage, and networking to complete computing stacks. AWS is mostly known for its compute and storage-on- demand services, namely *Elastic Compute Cloud (EC2)* and *Simple Storage Service (S3)*. EC2 provides users with customizable virtual hardware that can be used as the base infrastructure for deploying computing systems on the cloud. It is possible to choose from a large variety of virtual hardware configurations, including GPU and cluster instances. S3 is organized into buckets; these are containers of objects that are stored in binary form. Users can store objects of any size, from simple files to entire disk images, and have them accessible from everywhere.

Google AppEngine

Google AppEngine is a scalable runtime environment mostly devoted to executing Web applications. AppEngine provides both a secure execution environment and a collection of services that simplify the development of scalable and high-performance Web applications. These services include in-memory caching, scalable data store, job queues, messaging, and cron tasks. Developers can build and test applications on their own machines using the AppEngine software development kit (SDK), which replicates the production runtime environment and helps test and profile applications. Once development is complete, developers can easily migrate their application to AppEngine, and make the application available to the

world. The languages currently supported are Python, Java.

Microsoft Azure

Microsoft Azure is a cloud operating system and a platform for developing applications in the cloud. It provides a scalable runtime environment for Web applications and distributed applications in general. Applications in Azure are organized around the concept of roles. Currently, there are three types of role: *Web role*, *worker role*, and *virtual machine role*. The Web role is designed to host a Web application, the worker role is a more generic container of applications and can be used to perform workload processing, and the virtual machine role provides a virtual environment in which the computing stack can be fully customized, including the operating systems.

Hadoop

Apache Hadoop is an open-source framework that is suited for processing large data sets on commodity hardware. Yahoo!, the sponsor of the Apache Hadoop project, has put considerable effort into transforming the project into an enterprise-ready cloud computing platform for data processing. Hadoop is an integral part of the Yahoo! cloud infrastructure and supports several business processes of the company. Currently, Yahoo! manages the largest Hadoop cluster in the world.

Force.com and Salesforce.com

Force.com is a cloud computing platform for developing social enterprise applications. Force.com allows developers to create applications by composing ready-to-use blocks; a complete set of components supporting all the activities of an enterprise are available. The Force.com platform is completely hosted on the cloud and provides complete access to its functionalities and those implemented in the hosted applications through Web services technologies.

Manjrasoft Aneka

Manjrasoft Aneka is a cloud application platform for rapid creation of scalable applications and their deployment on various types of clouds in a seamless and elastic manner. It supports a collection of programming abstractions for developing applications and a distributed runtime environment that can be deployed on heterogeneous hardware (clusters, networked desktop

computers, and cloud resources).

These platforms are key examples of technologies available for cloud computing. They mostly fall into the three major market segments identified in the reference model: *Infrastructure-as-a-Service*, *Platform-as-a-Service*, and *Software-as-a-Service*.

VIRTUALIZATION

Virtualization is a large umbrella of technologies and concepts that are meant to provide an abstract environment—whether virtual hardware or an operating system—to run applications. The term *virtualization* is often synonymous with *hardware virtualization*, which plays a fundamental role in efficiently delivering *Infrastructure-as-a-Service* (IaaS) solutions for cloud computing. In fact, virtualization technologies available in many flavors by providing virtual environments at the operating system level, the programming language level, and the application level. Moreover, virtualization technologies provide a virtual environment for not only executing applications but also for storage, memory, and networking.

Virtualization technologies have gained renewed interested recently due to the confluence of several phenomena:

- *Increased performance and computing capacity.* Nowadays, the average end-user desktop PC is powerful enough to meet almost all the needs of everyday computing, with extra capacity that is rarely used. Almost all these PCs have resources enough to host a virtual machine manager and execute a virtual machine with by far acceptable performance. The same consideration applies to the high-end side of the PC market, where supercomputers can provide immense compute power that can accommodate the execution of hundreds or thousands of virtual machines.
- *Underutilized hardware and software resources.* Hardware and software underutilization is occurring due to (1) increased performance and computing capacity, and (2) the effect of limited or sporadic use of resources. Computers today are so powerful that in most cases only a fraction of their capacity is used by an application or the system. Moreover, if we consider the IT infrastructure of an enterprise, many computers are only

partially utilized whereas they could be used without interruption on a 24/7/365 basis. For example, desktop PCs mostly devoted to office automation tasks and used by administrative staff are only used during work hours, remaining completely unused overnight. Using these resources for other purposes after hours could improve the efficiency of the IT infrastructure. To transparently provide such a service, it would be necessary to deploy a completely separate environment, which can be achieved through virtualization.

- *Lack of space.* The continuous need for additional capacity, whether storage or compute power, makes data centers grow quickly. Companies such as Google and Microsoft expand their infrastructures by building data centers as large as football fields that are able to host thousands of nodes. Although this is viable for IT giants, in most cases enterprises cannot afford to build another data center to accommodate additional resource capacity. This condition, along with hardware underutilization, has led to the diffusion of a technique called *server consolidation*, for which virtualization technologies are fundamental.

- *Greening initiatives.* Recently, companies are increasingly looking for ways to reduce the amount of energy they consume and to reduce their carbon footprint. Data centers are one of the major power consumers; Maintaining a data center operation not only involves keeping servers on, but a great deal of energy is also consumed in keeping them cool. Infrastructures for cooling have a significant impact on the carbon footprint of a data center. Hence, reducing the number of servers through server consolidation will definitely reduce the impact of cooling and power consumption of a data center. Virtualization technologies can provide an efficient way of consolidating servers.

- *Rise of administrative costs. The* increased demand for additional capacity, which translates into more servers in a data center, is also responsible for a significant increment in administrative costs. Computers—in particular, servers—do not operate all on their own, but they require care and feeding from system administrators. Common system administration tasks include hardware monitoring, defective hardware replacement, server setup and updates, server resources monitoring, and backups. These are labor-intensive operations, and the higher the number of servers that have to be managed, the higher the

administrative costs. Virtualization can help reduce the number of required servers for a given workload, thus reducing the cost of the administrative personnel.

Characteristics of virtualized environments

Virtualization is a broad concept that refers to the creation of a virtual version of something, whether hardware, a software environment, storage, or a network. In a virtualized environment there are three major components: *guest*, *host*, and *virtualization layer*. The *guest* represents the system component that interacts with the virtualization layer rather than with the host, as would normally happen. The *host* represents the original environment where the guest is supposed to be managed. The *virtualization layer* is responsible for recreating the same or a different environment where the guest will operate (see Figure 2.1).

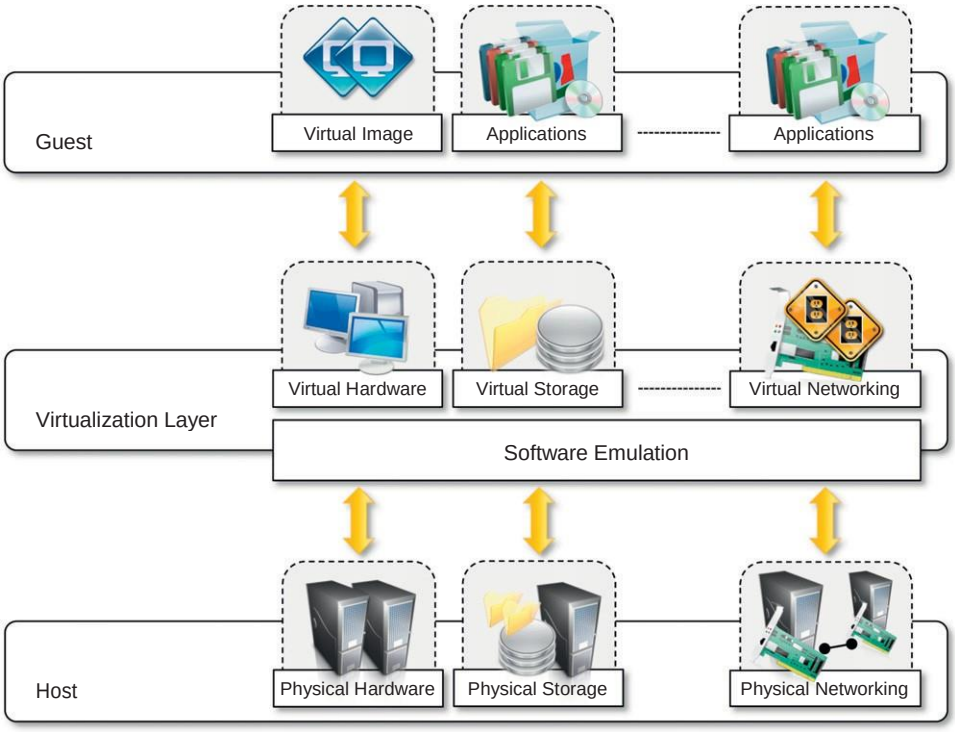


Figure 2.1: The virtualization reference model.

The Characteristics of Virtualization is as follows

1. **Increased security: The** ability to control the execution of a guest in a completely transparent manner opens new possibilities for delivering a secure, controlled execution environment. The virtual machine represents an emulated environment in which the guest is executed. All the operations of the guest are generally performed against the virtual machine, which then translates and

applies them to the host. Resources exposed by the host can then be hidden or simply protected from the guest. Sensitive information that is contained in the host can be naturally hidden without the need to install complex security policies.

2. **Managed execution:** Virtualization of the execution environment not only allows increased security, but a wider range of features also can be implemented. In particular, *sharing*, *aggregation*, *emulation*, and *isolation* are the most relevant features

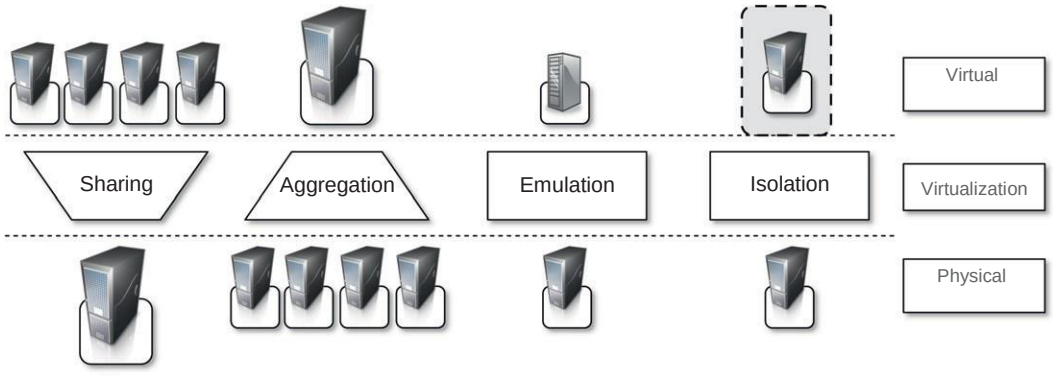


Figure 2.2 : Functions enabled by managed execution

- *Aggregation.* Not only is it possible to share physical resource among several guests, but virtualization also allows aggregation, which is the opposite process. A group of separate hosts can be tied together and represented to guests as a single virtual host. This function is naturally implemented in middleware for distributed computing, with a classical example represented by cluster management software, which harnesses the physical resources of a homogeneous group of machines and represents them as a single resource.

- *Emulation.* Guest programs are executed within an environment that is controlled by the virtualization layer, which ultimately is a program. This allows for controlling and tuning the environment that is exposed to guests. For instance, a completely different environment with respect to the host can be emulated, thus

allowing the execution of guest programs requiring specific characteristics that are not present in the physical host. Hardware virtualization solutions are able to provide virtual hardware and emulate a particular kind of device such as *Small Computer System Interface (SCSI)* devices for file I/O, without the hosting machine having such hardware installed.

- *Isolation.* Virtualization allows providing guests—whether they are operating systems, applications, or other entities—with a completely separate environment, in which they are executed. The guest program performs its activity by interacting with an abstraction layer, which provides access to the underlying resources. The virtual machine can filter the activity of the guest and prevent harmful operations against the host.

3. **Portability:** The concept of *portability* applies in different ways according to the specific type of virtualization considered. In the case of a hardware virtualization solution, the guest is packaged into a virtual image that, in most cases, can be safely moved and executed on top of different virtual machines. Except for the file size, this happens with the same simplicity with which we can display a picture image in different computers. Virtual images are generally proprietary formats that require a specific virtual machine manager to be executed.

Taxonomy of virtualization techniques

Virtualization covers a wide range of emulation techniques that are applied to different areas of computing. A classification of these techniques helps us better understand their characteristics and use (see Figure 2.3).

The first classification discriminates against the service or entity that is being emulated. Virtualization is mainly used to emulate *execution environments*, *storage*, and *networks*. Among these categories, *execution virtualization* constitutes the oldest, most popular, and most developed area. In particular we can divide these execution virtualization techniques into two major categories by considering the type of host they require. *Process-level*

techniques are implemented on top of an existing operating system, which has full control of the hardware. *System-level* techniques are implemented directly on hardware and do not require—or require a minimum of support from—an existing operating system. Within these two categories we can list various techniques that offer the guest a different type of virtual computation environment: bare hardware, operating system resources, low-level programming language, and application libraries.

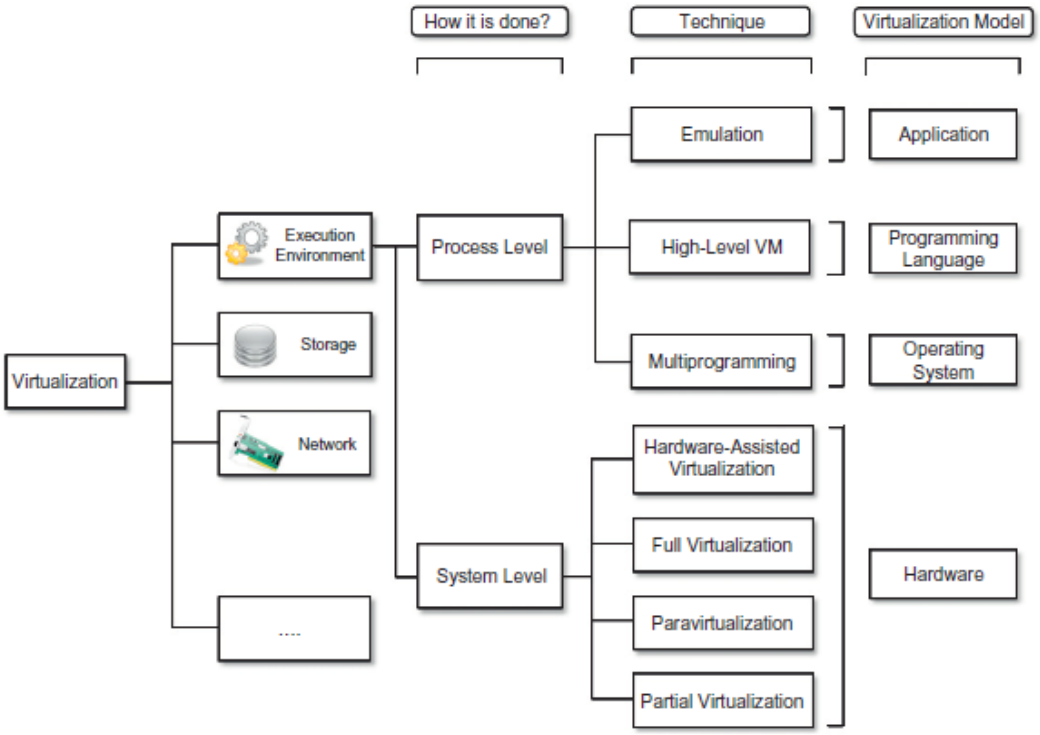


Figure 2.3 A taxonomy of virtualization techniques

Execution virtualization

Execution virtualization includes all techniques that aim to emulate an execution environment that is separate from the one hosting the virtualization layer. All these techniques concentrate their interest on providing support for the execution of programs, whether these are the operating system, a binary specification of a program compiled against an abstract machine model, or an application. Therefore, execution virtualization can be implemented directly on top of the hardware by the operating system, an application, or libraries dynamically or statically linked to an application image.

Machine reference model

Virtualzing an execution environment at different levels of the computing stack requires a reference model that defines the interfaces between the levels of abstractions, which hide implementation details. From this perspective, virtualization techniques actually replace one of the layers and intercept the calls that are directed toward it.

Modern computing systems can be expressed in terms of the reference model described in Figure 2.4. At the bottom layer, the model for the hardware is expressed in terms of the *Instruction Set Architecture (ISA)*, which defines the instruction set for the processor, registers, memory, and interrupts management. ISA is the interface between hardware and software, and it is important to the operating system (OS) developer (*System ISA*) and developers of applications that directly manage the underlying hardware (*User ISA*).

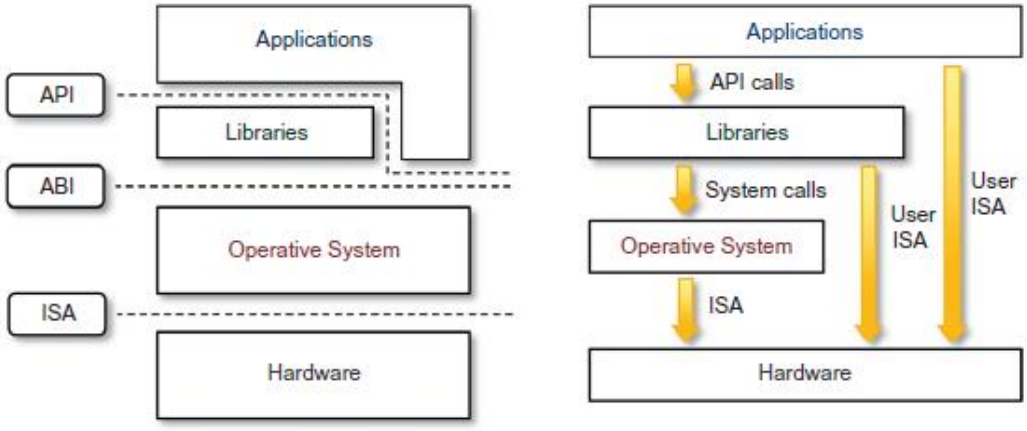


Figure 2.4 A Machine Reference Model

The *application binary interface (ABI)* separates the operating system layer from the applications and libraries, which are managed by the OS. ABI covers details such as low-level data types, alignment, and call conventions and defines a format for executable programs. System calls are defined at this level. This interface allows portability of applications and libraries across operating systems that implement the same ABI. The highest level of abstraction is represented by the *application programming interface (API)*, which interfaces applications to libraries and/or the underlying operating system.

The instruction set exposed by the hardware has been divided into different security classes that define who can operate with them. The first distinction can be made between *privileged* and *nonprivileged* instructions. Nonprivileged instructions are those instructions that can be used without interfering with other tasks because they do not access shared resources. This category contains, for example, all the floating, fixed-point, and arithmetic instructions. Privileged instructions are those that are executed under specific restrictions and are mostly used for sensitive operations, which expose (*behavior-sensitive*) or modify (*control-sensitive*) the privileged state. For instance, behavior-sensitive instructions are those that operate on the I/O, whereas control-sensitive instructions alter the state of the CPU registers.

A possible implementation features a hierarchy of privileges (see Figure 2.5) in the form of ring-based security: *Ring 0*, *Ring 1*, *Ring 2*, and *Ring 3*; Ring 0 is in the most privileged level and Ring 3 in the least privileged level. Ring 0 is used by the kernel of the OS, rings 1 and 2 are used by the OS-level services, and Ring 3 is used by the user. Recent systems support only two levels, with Ring 0 for supervisor mode and Ring 3 for user mode.

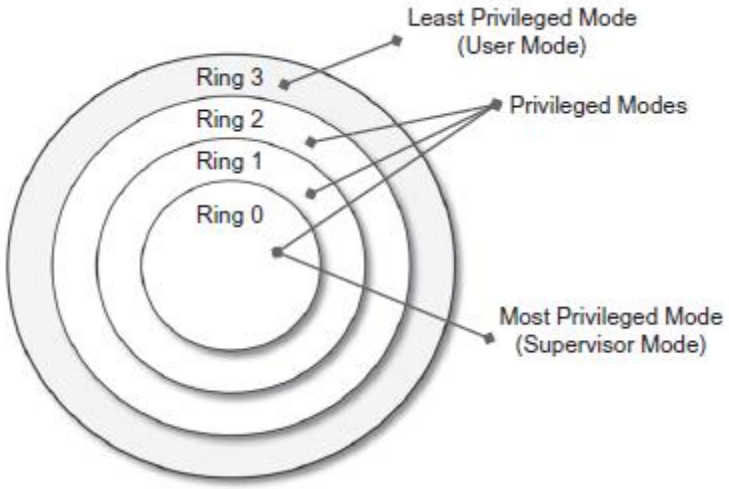


Figure 2.5 Security Rings and Privileged mode

All the current systems support at least two different execution modes: *supervisor mode* and *user mode*. The first mode denotes an execution mode in which all the instructions (privileged and nonprivileged) can be executed without any restriction. This mode, also called *master mode* or *kernel mode*, is generally used by the operating system (or the hypervisor) to perform sensitive operations on hardware level resources. In user mode, there are restrictions to control the

machine-level resources. If code running in user mode invokes the privileged instructions, hardware interrupts occur and trap the potentially harmful execution of the instruction. Conceptually, the hypervisor runs above the supervisor mode.

Hardware-level virtualization

Hardware-level virtualization is a virtualization technique that provides an abstract execution environment in terms of computer hardware on top of which a guest operating system can be run. In this model, the guest is represented by the operating system, the host by the physical computer hardware, the virtual machine by its emulation, and the virtual machine manager by the hypervisor (see Figure 2.6). The hypervisor is generally a program or a combination of software and hardware that allows the abstraction of the underlying physical hardware.

Hardware-level virtualization is also called *system virtualization*, since it provides ISA to virtual machines, which is the representation of the hardware interface of a system.

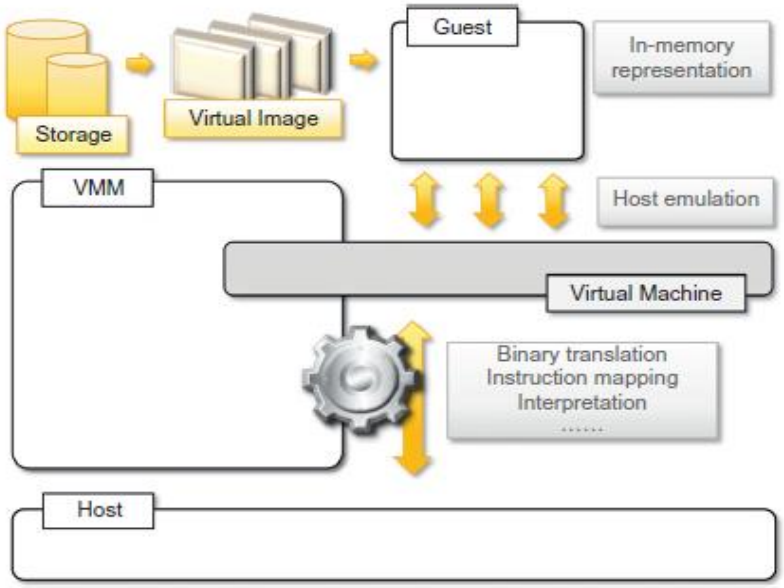


Figure 2.6 A hardware virtualization reference model.

Hypervisors: A fundamental element of hardware virtualization is the hypervisor, or virtual machine manager (VMM).

It recreates a hardware environment in which guest operating systems are installed. There are two major types of hypervisor: *Type I* and *Type II* (see Figure 2.7).

- *Type I* hypervisors run directly on top of the hardware. Therefore, they take the place of the operating systems and interact directly with the ISA interface exposed by the underlying hardware, and they emulate this interface in order to allow the management of guest operating systems. This type of hypervisor is also called a *native virtual machine* since it runs natively on hardware.

- *Type II* hypervisors require the support of an operating system to provide virtualization services. This means that they are programs managed by the operating system, which interact with it through the ABI and emulate the ISA of virtual hardware for guest operating systems. This type of hypervisor is also called a *hosted virtual machine* since it is hosted within an operating system.

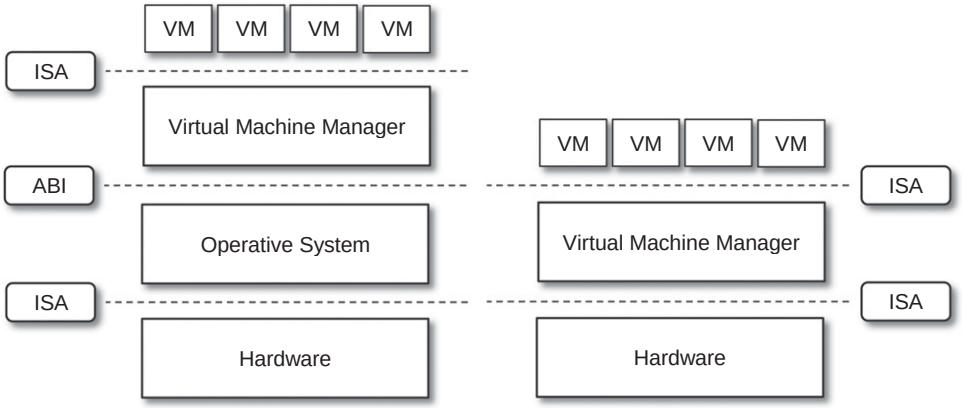


Figure 2.7 Hosted (left) and native (right) virtual machines.

A virtual machine manager is internally organized as described in Figure 2.8. Three main modules, *dispatcher*, *allocator*, and *interpreter*, coordinate their activity in order to emulate the underlying hardware. The dispatcher constitutes the entry point of the monitor and reroutes the instructions issued by the virtual machine instance to one of the two other modules. The allocator is responsible for deciding the system resources to be provided to the VM: whenever a virtual machine tries to execute an instruction that results in changing the machine resources associated with that VM, the allocator is invoked by the dispatcher. The interpreter module consists of interpreter routines. These are executed whenever a virtual machine executes a privileged instruction: a trap is triggered and the corresponding routine is executed.

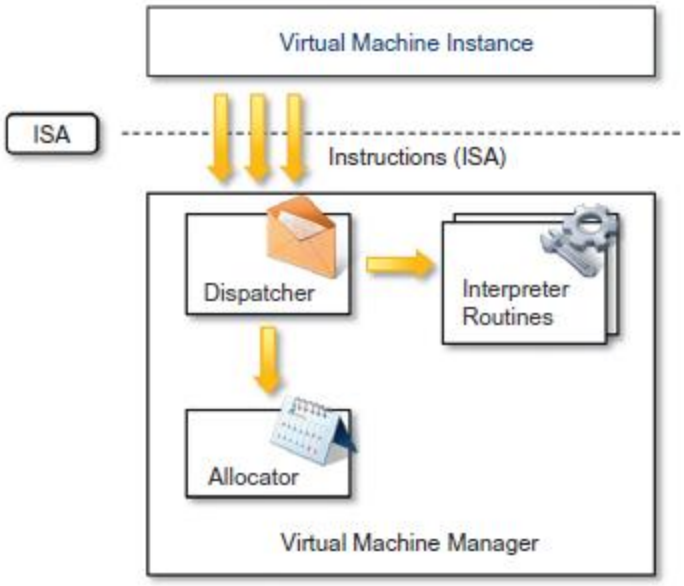


Figure 2.8 A hypervisor reference architecture.

Three properties of Virtual Machine Manager that have to be satisfied:

- *Equivalence.* A guest running under the control of a virtual machine manager should exhibit the same behavior as when it is executed directly on the physical host.
- *Resource control.* The virtual machine manager should be in complete control of virtualized resources.
- *Efficiency.* A statistically dominant fraction of the machine instructions should be executed without intervention from the virtual machine manager.

Popek and Goldberg provided a classification of the instruction set and proposed three theorems that define the properties that hardware instructions need to satisfy in order to efficiently support virtualization

THEOREM 1

For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

This theorem establishes that all the instructions that change the configuration of the system resources should generate a trap in user mode and be executed under the control of the virtual machine manager. This allows hypervisors to efficiently control only those instructions that would reveal the presence of an abstraction layer while executing all the rest of the instructions without considerable performance loss.

THEOREM 2

A conventional third-generation computer is recursively virtualizable if:

- **It is virtualizable and**
- **A VMM without any timing dependencies can be constructed for it.**

Recursive virtualization is the ability to run a virtual machine manager on top of another virtual machine manager. This allows nesting hypervisors as long as the capacity of the underlying resources can accommodate that. Virtualizable hardware is a prerequisite to recursive virtualization.

THEOREM 3

A hybrid VMM may be constructed for any conventional third-generation machine in which the set of user-sensitive instructions is a subset of the set of privileged instructions.

There is another term, *hybrid virtual machine (HVM)*, which is less efficient than the virtual machine system. In the case of an HVM, more instructions are interpreted rather than being executed directly. All instructions in virtual supervisor mode are interpreted. Whenever there is an attempt to execute a behavior-sensitive or control-sensitive instruction, HVM controls the execution directly or gains the control via a trap.

Hardware virtualization techniques

Hardware-assisted virtualization. This term refers to a scenario in which the hardware provides architectural support for building a virtual machine manager able to run a guest operating system in complete isolation. This technique was originally introduced in the IBM System/370. At present, examples of hardware-assisted virtualization are the extensions to the x86-64 bit architecture introduced with *Intel VT* (formerly known as *Vanderpool*) and *AMD V* (formerly known as *Pacifica*).. Products such as VMware Virtual Platform, introduced in 1999 by VMware, which pioneered the field of x86 virtualization, were based on this technique. After 2006, Intel and AMD introduced processor extensions, and a wide range of virtualiza- tion

solutions took advantage of them*:* Kernel-based Virtual Machine (KVM), VirtualBox, Xen, VMware, Hyper-V, Sun xVM, Parallels, and others.

Full virtualization. *Full virtualization* refers to the ability to run a program, most likely an operating system, directly on top of a virtual machine and without any modification, as though it were run on the raw hardware. To make this possible, virtual machine managers are required to provide a complete emulation of the entire underlying hardware. The principal advantage of full virtualization is complete isolation, which leads to enhanced security, ease of emulation of different architectures, and coexistence of different systems on the same platform. A simple solution to achieve full virtualization is to provide a virtual environment for all the instructions, thus posing some limits on performance.

Paravirtualization. This is a not-transparent virtualization solution that allows implementing thin virtual machine managers. Paravirtualization techniques expose a software interface to the virtual machine that is slightly modified from the host and, as a consequence, guests need to be modified. The aim of paravirtualization is to provide the capability to demand the execution of performance-critical operations directly on the host, thus preventing performance losses that would otherwise be experienced in managed execution. This technique has been successfully used by Xen for providing virtualization solutions for Linux-based operating systems specifically ported to run on Xen hypervisors.

Partial virtualization. Partial virtualization provides a partial emulation of the underlying hardware, thus not allowing the complete execution of the guest operating system in complete isolation. Partial virtualization allows many applications to run transparently, but not all the features of the operating system can be supported, as happens with full virtualization. Partial virtualization was implemented on the experimental IBM M44/44X. Address space virtualization is a common feature of contemporary operating systems.

Operating system-level virtualization

Operating system-level virtualization offers the opportunity to create different and separated execution environments for applications that are managed concurrently. Differently from hardware virtualization, there is no virtual machine manager or hypervisor, and the virtualization is done within a single operating system, where the OS kernel allows for multiple isolated user space instances. The kernel is also responsible for sharing the system resources among instances and for limiting the impact of instances on each other. A user space instance in

general contains a proper view of the file system, which is completely isolated, and separate IP addresses, software configurations, and access to devices. Operating system-level virtualization aims to provide separated and multiple execution containers for running applications. Compared to hardware virtualization, this strategy imposes little or no overhead because applications directly use OS system calls and there is no need for emulation.

Examples of operating system-level virtualizations are FreeBSD Jails, IBM Logical Partition (LPAR), SolarisZones and Containers, Parallels Virtuozzo Containers, OpenVZ, iCore Virtual Accounts, Free Virtual Private Server (FreeVPS).

Programming language-level virtualization

Programming language-level virtualization is mostly used to achieve ease of deployment of applications, managed execution, and portability across different platforms and operating systems. It consists of a virtual machine executing the byte code of a program, which is the result of the compilation process. Compilers implemented and used this technology to produce a binary format representing the machine code for an abstract architecture. The characteristics of this architecture vary from implementation to implementation.

Programming language-level virtualization has a long trail in computer science history and originally was used in 1966 for the implementation of Basic Combined Programming Language (BCPL), a language for writing compilers and one of the ancestors of the C programming language. Other important examples of the use of this technology have been the UCSD Pascal and Smalltalk. Virtual machine programming languages become popular again with Sun's introduction of the Java platform in 1996.

Currently, the Java platform and .NET Framework represent the most popular technologies for enterprise application development. The main advantage of programming-level virtual machines, also called *process virtual machines*, is the ability to provide a uniform execution environment across different platforms. Programs compiled into byte code can be executed on any operating system and platform for which a virtual machine able to execute that code has been provided.

Application-level virtualization

Application-level virtualization is a technique allowing applications to be run in runtime environments that do not natively support all the features required by such applications. In this scenario, applications are not installed in the expected runtime environment but are run as

though they were. In general, these techniques are mostly concerned with partial file systems, libraries, and operating system component emulation. Such emulation is performed by a thin layer—a program or an operating system component—that is in charge of executing the application. Emulation can also be used to execute program binaries compiled for different hardware architectures. In this case, one of the following strategies can be implemented:

- **Interpretation.** In this technique every source instruction is interpreted by an emulator for executing native ISA instructions, leading to poor performance. Interpretation has a minimal startup cost but a huge overhead, since each instruction is emulated.
- **Binary translation.** In this technique every source instruction is converted to native instructions with equivalent functions. After a block of instructions is translated, it is cached and reused. Binary translation has a large initial overhead cost, but over time it is subject to better performance, since previously translated instruction blocks are directly executed.

Application virtualization is a good solution in the case of missing libraries in the host operating system. One of the most popular solutions implementing application virtualization is Wine, which is a software application allowing Unix-like operating systems to execute programs written for the Microsoft Windows platform

Other types of virtualization

1. Storage virtualization

Storage virtualization is a system administration practice that allows decoupling the physical organization of the hardware from its logical representation. Using this technique, users do not have to be worried about the specific location of their data, which can be identified using a logical path. There are different techniques for storage virtualization, one of the most popular being network-based virtualization by means of *storage area networks (SANs)*.

2. Network virtualization

Network virtualization combines hardware appliances and specific software for the creation and management of a virtual network. Network virtualization can aggregate different physical networks into a single logical network (*external* network virtualization) or provide network-like functionality to an operating system partition (*internal* network virtualization). The result of external network virtualization is generally a *virtual LAN (VLAN)*. A VLAN is an aggregation of hosts that communicate with each other as though they were located under the same

broadcasting domain. There are several options for implementing internal network virtualization: The guest can share the same net- work interface of the host and use Network Address Translation (NAT) to access the network; the virtual machine manager can emulate, and install on the host, an additional network device, together with the driver; or the guest can have a private network only with the guest.

3. Desktop virtualization

Desktop virtualization abstracts the desktop environment available on a personal computer in order to provide access to it using a client/server approach. Desktop virtualization provides the same out- come of hardware virtualization but serves a different purpose. Similarly to hardware virtualization, desktop virtualization makes accessible a different system as though it were natively installed on the host, but this system is remotely stored on a different host and accessed through a network connection. Moreover, desktop virtualization addresses the problem of making the same desktop environment accessible from everywhere. The advantages of desktop virtualization are high availability, persistence, accessibility, and ease of management. Infrastructures for desktop virtua- lization based on cloud computing solutions include Sun Virtual Desktop Infrastructure (VDI), Parallels Virtual Desktop Infrastructure (VDI), Citrix XenDesktop, and others.

4. **Application server virtualization**

Application server virtualization abstracts a collection of application servers that provide the same services as a single virtual application server by using load-balancing strategies and providing a high-availability infrastructure for the services hosted in the application server. This is a particular form of virtualization and serves the same purpose of storage virtualization: providing a better quality of service rather than emulating a different environment.

Virtualization and cloud computing

Virtualization plays an important role in cloud computing since it allows for the appropriate degree of customization, security, isolation, and manageability that are fundamental for delivering IT services on demand. Virtualization technologies are primarily used to offer configurable computing environments and storage.

Particularly important is the role of virtual computing environment and execution virtualization techniques. Among these, hardware and programming language virtualization are the techniques

adopted in cloud computing systems. Hardware virtualization is an enabling factor for solutions in the Infrastructure-as-a-Service (IaaS) market segment, while programming language virtualization is a technology leveraged in Platform-as-a-Service (PaaS) offerings. In both cases, the capability of offering a customizable and sandboxed environment constituted an attractive business opportunity for companies featuring a large computing infrastructure that was able to sustain and process huge workloads. Moreover, virtualization also allows isolation and a finer control, thus simplifying the leasing of services and their accountability on the vendor side.

Virtualization allows us to create isolated and controllable environments, it is possible to serve these environments with the same resource without them interfering with each other. If the underlying resources are capable enough, there will be no evidence of such sharing.It allows reducing the number of active resources by aggregating virtual machines over a smaller number of resources that become fully utilized. This practice is also known as server consolidation, while the movement of virtual machine instances is called virtual machine migration (see Figure 3.10). Because virtual machine instances are controllable environments, consolidation can be applied with a minimum impact, either by temporarily stopping its execution and moving its data to the new resources or by performing a finer control and moving the instance while it is running. This second techniques is known as live migration and in general is more complex to implement but more efficient since there is no disruption of the activity of the virtual machine instance

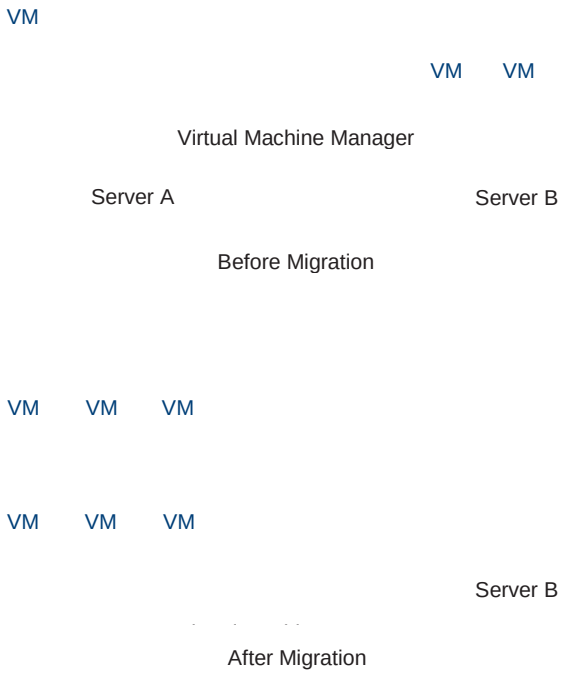


Figure 2.10 Live migration and server consolidation.

Pros and cons of virtualization

Advantages of virtualization

Managed execution and isolation are perhaps the most important advantages of virtualization. In the case of techniques supporting the creation of virtualized execution environments, these two characteristics allow building secure and controllable computing environments. A virtual execution environment can be configured as a sandbox, thus preventing any harmful operation to cross the borders of the virtual host. Moreover, allocation of resources and their partitioning among different guests is simplified, being the virtual host controlled by a program. This enables fine-tuning of resources, which is very important in a server consolidation scenario and is also a requirement for effective quality of service

Portability and self-containment also contribute to reducing the costs of maintenance, since the number of hosts is expected to be lower than the number of virtual machine instances. By means of virtualization it is possible to achieve a more efficient use of resources. Multiple systems can securely coexist and share the resources of the underlying host, without interfering with each other.

Performance degradation

Performance is definitely one of the major concerns in using virtualization technology. Since virtualization interposes an abstraction layer between the guest and the host, the guest can experience increased latencies.

For instance, in the case of hardware virtualization, where the intermediate emulates a bare machine on top of which an entire system can be installed, the causes of performance degradation can be traced back to the overhead introduced by the following activities:

- Maintaining the status of virtual processors
- Support of privileged instructions (trap and simulate privileged instructions)
- Support of paging within VM
- Console functions

Furthermore, when hardware virtualization is realized through a program that is installed or executed on top of the host operating systems, a major source of performance degradation is represented by the fact that the virtual machine manager is executed and scheduled together with other applications, thus sharing with them the resources of the host.

Similar consideration can be made in the case of virtualization technologies at higher levels, such as in the case of programming language virtual machines (Java, .NET, and others). Binary translation and interpretation can slow down the execution of managed applications. Moreover, because their execution is filtered by the runtime environment, access to memory and other physi- cal resources can represent sources of performance degradation.

These concerns are becoming less and less important thanks to technology advancements and the ever-increasing computational power available today. For example, specific techniques for hard- ware virtualization such as paravirtualization can increase the performance of the guest program by offloading most of its execution to the host without any change. In programming-level virtual machines such as the JVM or .NET, compilation to native code is offered as an option when perfor- mance is a serious concern.

Disadvantages:

Inefficiency and degraded user experience

Virtualization can sometime lead to an inefficient use of the host. In particular, some of the specific features of the host cannot be exposed by the abstraction layer and then become inaccessible. In the case of hardware virtualization, this could happen for device drivers: The virtual machine can sometime simply provide a default graphic card that maps only a subset of the features available in the host. In the case of programming-level virtual machines, some of the features of the underlying operating systems may become inaccessible unless specific libraries are used.

Security holes and new threats

Virtualization opens the door to a new and unexpected form of *phishing.*The capability of emulating a host in a completely transparent manner led the way to malicious programs that are designed to extract sensitive information from the guest.

In the case of hardware virtualization, malicious programs can preload themselves before the operating system and act as a thin virtual machine manager toward it. The operating system is then controlled and can be manipulated to extract sensitive information of interest to third parties.

Technology examples

Xen: paravirtualization

Xen is an open-source initiative implementing a virtualization platform based on paravirtualization. Initially developed by a group of researchers at the University of Cambridge in the United Kingdom, Xen now has a large open-source community backing it. . Xen-based technology is used for either desktop virtualization or server virtualization, and recently it has also been used to provide cloud computing solutions by means of Xen Cloud Platform (XCP).

Figure 2.11 describes the architecture of Xen and its mapping onto a classic x86 privilege model. A Xen-based system is managed by the Xen hypervisor, which runs in the highest

privileged mode and controls the access of guest operating system to the underlying hardware.

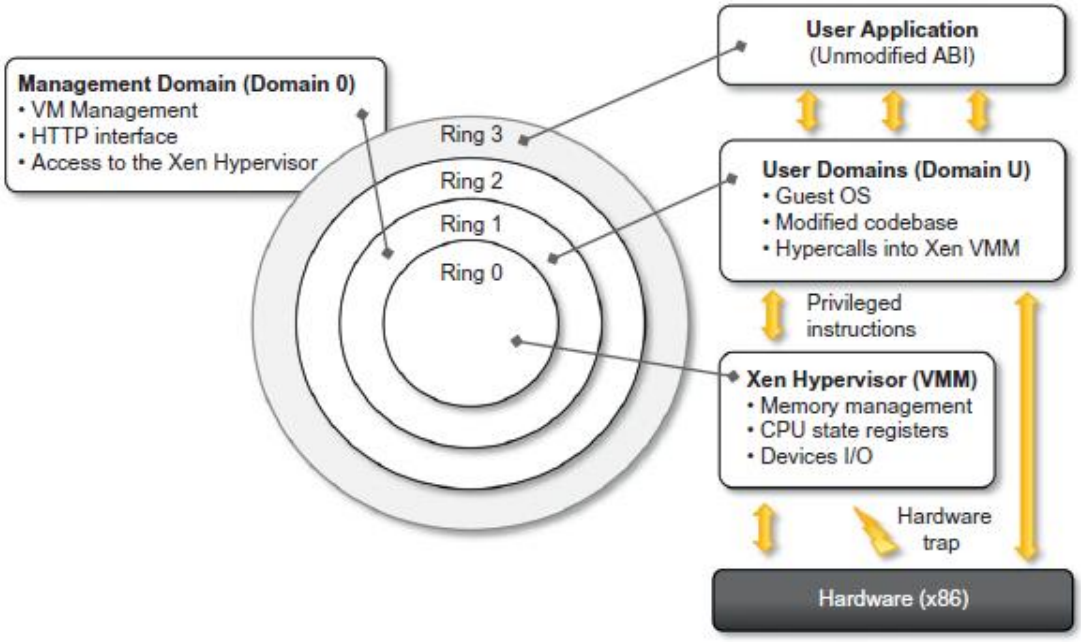


Figure 2.11 Xen architecture and guest OS management

Guest operating systems are executed within domains, which represent virtual machine instances. Moreover, specific control software, which has privileged access to the host and controls all the other guest operating systems, is executed in a special domain called Domain 0. This is the first one that is loaded once the virtual machine manager has completely booted, and it hosts a HyperText Transfer Protocol (HTTP) server that serves requests for virtual machine creation, con- figuration, and termination. This component constitutes the embryonic version of a distributed virtual machine manager, which is an essential component of cloud computing systems providing Infrastructure-as-a-Service (IaaS) solutions.

Many of the x86 implementations support four different security levels, called rings, where Ring 0 represent the level with the highest privileges and Ring 3 the level with the lowest ones.

Because of the structure of the x86 instruction set, some instructions allow code executing in Ring 3 to jump into Ring 0 (kernel mode). Such operation is performed at the hardware level and therefore within a virtualized environment will result in a trap or silent fault, thus preventing

the normal operations of the guest operating system, since this is now running in Ring 1.This condition is generally triggered by a subset of the system calls. To avoid this situation, operating systems need to be changed in their implementation, and the sensitive system calls need to be reimplemented with hypercalls, which are specific calls exposed by the virtual machine interface of Xen. With the use of hypercalls, the Xen hypervisor is able to catch the execution of all the sensitive instructions, manage them, and return the control to the guest operating system by means of a supplied handler.

Paravirtualization needs the operating system codebase to be modified, and hence not all operating systems can be used as guests in a Xen-based environment. Open-source operating systems such as Linux can be easily modified, since their code is publicly available and Xen provides full support for their virtualization, whereas components of the Windows family are generally not supported by Xen unless hardware-assisted virtualization is available.

VMware: full virtualization: VMware's technology is based on the concept of *full virtualization*, where the underlying hardware is replicated and made available to the guest operating system, which runs unaware of such abstrac- tion layers and does not need to be modified. VMware implements full virtualization either in the desktop environment, by means of *Type II* hypervisors, or in the server environment, by means of *Type I* hypervisors. In both cases, full virtualization is made possible by means of *direct execution* (for nonsensitive instructions) and *binary translation* (for sensitive instructions), thus allowing the virtualization of architecture such as x86.

Module 2

Cloud Computing Architecture

Cloud computing is a utility-oriented and Internet-centric way of delivering IT services on demand. These services cover the entire computing stack: from the hardware infrastructure pack- aged as a set of virtual machines to software services such as development platforms and distributed applications.

The cloud reference model

It is possible to organize all the concrete realizations of cloud computing into a layered view covering the entire stack (see Figure 3.1), from hardware appliances to software systems. Cloud resources are harnessed to offer "computing horsepower" required for providing services. Cloud infrastructure can be heterogeneous in nature because a variety of resources, such as clusters and even networked PCs, can be used to build it. Moreover, database systems and other storage services can also be part of the infrastructure.

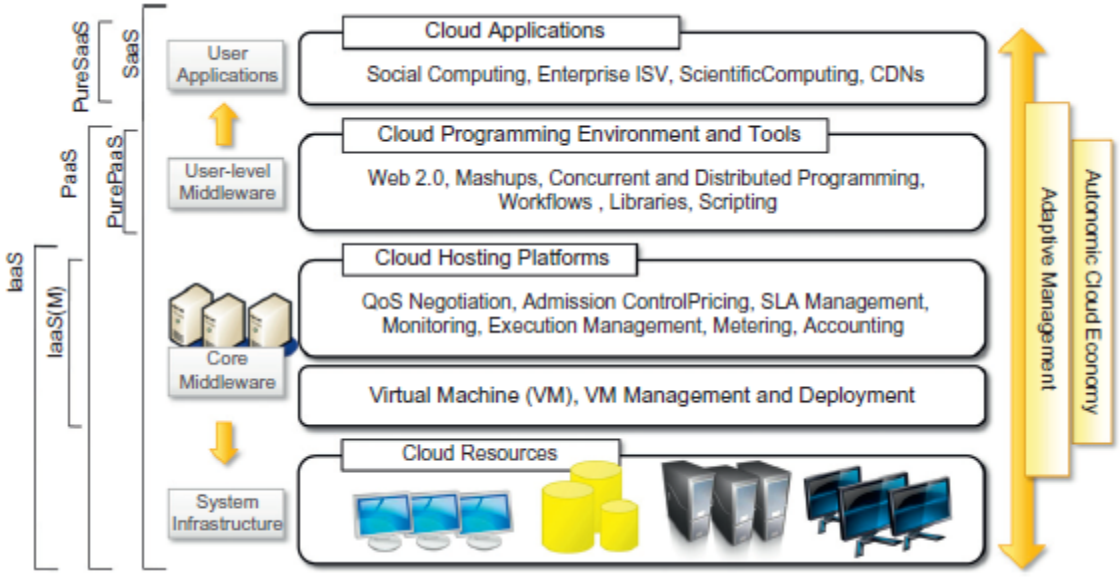


Figure 3.1 The cloud computing architecture.

The physical infrastructure is managed by the core middleware, the objectives of which are to provide an appropriate runtime environment for applications and to best utilize resources. At the bottom of the stack, virtualization technologies are used to guarantee runtime environment customization, application isolation, sandboxing, and quality of service. Hardware virtualization

is most commonly used at this level. Hypervisors manage the pool of resources and expose the distributed infrastructure as a collection of virtual machines. Infrastructure management is the key function of core middleware, which supports capabilities such as negotiation of the quality of service, admission control, execution management and monitoring, accounting, and billing.

The combination of cloud hosting platforms and resources is generally classified as a *Infrastructure-as-a-Service (IaaS)* solution. The IaaS has two categories: Some of them provide both the management layer and the physical infrastructure; others provide only the management layer (*IaaS (M)*). In this second case, the management layer is often integrated with other IaaS solutions that provide physical infrastructure and adds value to them. IaaS solutions are suitable for designing the system infrastructure but provide limited services to build applications.

PaaS solutions generally include the infrastructure as well, which is bundled as part of the service provided to users. In the case of *Pure PaaS*, only the user-level middleware is offered, and it has to be complemented with a virtual or physical infrastructure. The top layer of the reference model depicted in Figure 3.1 contains services delivered at the application level. These are mostly referred to as Software-as-a-Service (SaaS).

Table 3.1 Cloud Computing Services Classification

Category	Characteristics	Product Type	Vendors and Products
SaaS	Customers are provided with applications that are accessible anytime and from anywhere.	Web applications and services (Web 2.0)	SalesForce.com (CRM) Clarizen.com (project management) Google Apps
PaaS	Customers are provided with a platform for developing applications hosted in the cloud.	Programming APIs and frameworks Deployment systems	Google AppEngine Microsoft Azure Manjrasoft Aneka Data Synapse
IaaS/HaaS	Customers are provided with virtualized hardware and storage on top of which they can build their infrastructure.	Virtual machine management infrastructure Storage management Network management	Amazon EC2 and S3 GoGrid Nirvanix

Figure 3.1 also introduces the concept of everything as a Service (XaaS). Table 3.1 summarizes the characteristics of the three major categories used to classify cloud computing solutions.

Infrastructure and hardware-as-a-service

Infrastructure and Hardware-as-a-Service (IaaS/HaaS) solutions are the most popular and developed market segment of cloud computing. They deliver customizable infrastructure on

demand. The available options within the IaaS offering umbrella range from single servers to entire infrastructures, including network devices, load balancers, and database and Web servers.

The main technology used to deliver and implement these solutions is hardware virtualization: one or more virtual machines opportunely configured and interconnected define the distributed system on top of which applications are installed and deployed. IaaS/HaaS solutions bring all the benefits of hardware virtualization: workload partitioning, application isolation, sandboxing, and hard- ware tuning. From the perspective of the service provider, IaaS/HaaS allows better exploiting the IT infrastructure and provides a more secure environment where executing third party applications. From the perspective of the customer it reduces the administration and maintenance cost as well as the capital costs allocated to purchase hardware.

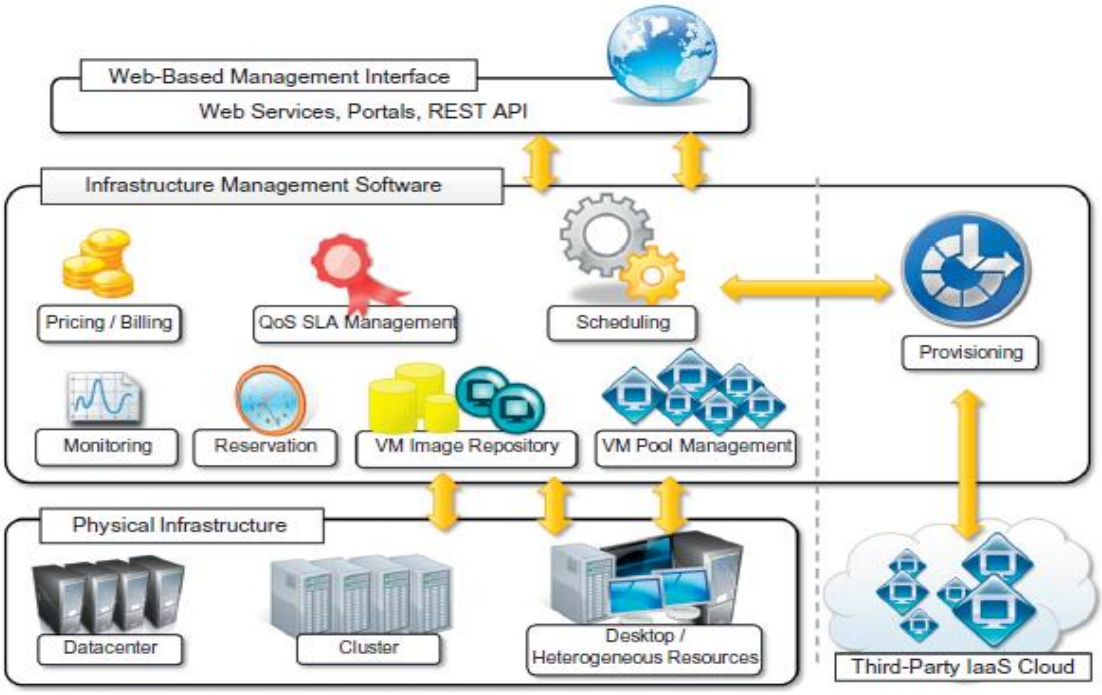


Figure 3.2 IaaS Reference implementation

Figure 3.2 provides an overall view of the components forming an Infrastructure-as-a-Service solution. It is possible to distinguish three principal layers: the physical infrastructure, the software management infrastructure, and the user interface. At the top layer the user interface provides access to the services exposed by the software management infrastructure.

The core features of an IaaS solution are implemented in the infrastructure management software layer. In particular, management of the virtual machines is the most important function performed by this layer. A central role is played by the **scheduler**, which is in charge of

allocating the execution of virtual machine instances. The scheduler interacts with the other components that perform a variety of tasks:

- **The *pricing and billing*** component takes care of the cost of executing each virtual machine instance and maintains data that will be used to charge the user.
- The ***monitoring*** component tracks the execution of each virtual machine instance and maintains data required for reporting and analyzing the performance of the system.
- The ***reservation*** component stores the information of all the virtual machine instances that have been executed or that will be executed in the future.
- If support for QoS-based execution is provided, a ***QoS/SLA management*** component will maintain a repository of all the SLAs made with the users; together with the monitoring component, this component is used to ensure that a given virtual machine instance is executed with the desired quality of service.
- The ***VM repository*** component provides a catalog of virtual machine images that users can use to create virtual instances. Some implementations also allow users to upload their specific virtual machine images.
- A ***VM pool manager*** component is responsible for keeping track of all the live instances.
- *A* ***provisioning*** component interacts with the scheduler to provide a virtual machine instance that is external to the local physical infrastructure directly managed by the pool.

The bottom layer is composed of the physical infrastructure, on top of which the management layer operates. From an architectural point of view, the physical layer also includes the virtual resources that are rented from external IaaS providers. In the case of complete IaaS solutions, all three levels are offered as service.

The reference architecture applies to IaaS implementations that provide computing resources, especially for the scheduling component. The role of infrastructure management software is not to keep track and manage the execution of virtual machines but to provide access to large infrastructures and implement storage virtualization solutions on top of the physical layer.

Platform as a Service

Platform-as-a-Service (PaaS) solutions provide a development and deployment platform for running applications in the cloud. They constitute the middleware on top of which

applications are built. A general overview of the features characterizing the PaaS approach is given in Figure 3.3.

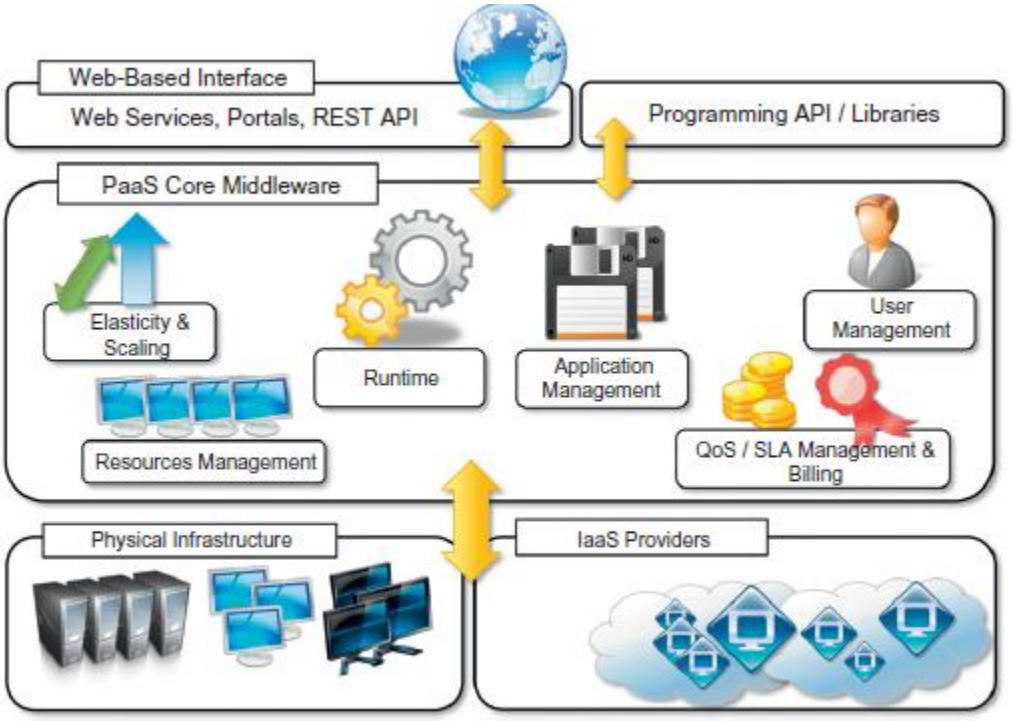


Figure 3.3 The Platform-as-a-Service reference model

Application management is the core functionality of the middleware. PaaS implementations provide applications with a runtime environment and do not expose any service for managing the underlying infrastructure. They automate the process of deploying applications to the infrastructure, configuring application components, provisioning and configuring supporting technologies such as load balancers and databases, and managing system change based on policies set by the user. Developers design their systems in terms of applications and are not concerned with hardware (physical or virtual), operating systems, and other low-level services.

The core middleware is in charge of managing the resources and scaling applications on demand or automatically, according to the commitments made with users. From a user point of view, the core middleware exposes interfaces that allow programming and deploying applications on the cloud. These can be in the form of a Web-based interface or in the form of programming APIs and libraries.

Some implementations provide a completely Web-based interface hosted in the cloud and offering a variety of services. Other implementations of the PaaS model provide a complete object model for representing an application and provide a programming language-based approach.

Table 3.1 PaaS offering classification

Category	Description	Product Type	Vendors and Products
PaaS-I	Runtime environment with Web-hosted application development platform. Rapid application prototyping.	Middleware + Infrastructure Middleware + Infrastructure	Force.com Longjump
PaaS-II	Runtime environment for scaling Web applications. The runtime could be enhanced by additional components that provide scaling capabilities.	Middleware + Infrastructure Middleware Middleware + Infrastructure Middleware + Infrastructure Middleware + Infrastructure Middleware	Google AppEngine AppScale Heroku Engine Yard Joyent Smart Platform GigaSpaces XAP
PaaS-III	Middleware and programming model for developing distributed applications in the cloud.	Middleware + Infrastructure Middleware Middleware Middleware Middleware Middleware	Microsoft Azure DataSynapse Cloud IQ Manjrasof Aneka Apprenda SaaSGrid GigaSpaces DataGrid

PaaS solutions can offer middleware for developing applications together with the infrastructure or simply provide users with the software that is installed on the user premises. Table 3.2 provides a classification of the most popular PaaS implementations. It is possible to organize the various solutions into three wide categories: *PaaS-I*, *PaaS-II*, and *PaaS-III*. The first category identifies PaaS implementations that completely follow the cloud computing style for application development and deployment. They offer an integrated development environment hosted within the Web browser where applications are designed, developed, composed, and deployed. The second class gives solutions that are focused on providing a scalable infrastructure for Web application, mostly websites. In this case, developers generally use the providers' APIs, which are built on top of industrial runtimes, to develop applications. The third category consists of all those solutions that provide a cloud programming platform for any kind of application, not only Web applications.

The essential characteristics that identify a PaaS solution:

- ***Runtime framework.*** The runtime framework executes end-user code according to the policies set by the user and the provider.
- ***Abstraction.*** PaaS solutions are distinguished by the higher level of abstraction that they provide. This means that PaaS solutions offer a way to deploy and manage applications on the cloud.
- ***Automation.*** PaaS environments automate the process of deploying applications to the infrastructure, scaling them by provisioning additional resources when needed. This process is performed automatically and according to the SLA made between the customers and the provider.
- ***Cloud services.*** PaaS offerings provide developers and architects with services and APIs, helping them to simplify the creation and delivery of elastic and highly available cloud applications.

Another essential component for a PaaS-based approach is the ability to integrate third-party cloud services offered from other vendors by leveraging service-oriented architecture. One of the major concerns of leveraging PaaS solutions for implementing applications is *vendor lock-in*. Even though a platform-based approach strongly simplifies the development and deployment cycle of applications, it poses the risk of making these applications completely dependent on the provider. Such dependency can become a significant obstacle in retargeting the application to another environment and runtime if the commitments made with the provider cease.

PaaS solutions can cut the cost across development, deployment, and management of applications. It helps management reduce the risk of ever-changing technologies by offloading the cost of upgrading the technology to the PaaS provider. The PaaS approach, when bundled with underlying IaaS solutions, helps even small start-up companies quickly offer customers integrated solutions on a hosted platform at a very minimal cost.

Software as a service

Software-as-a-Service (SaaS) is a software delivery model that provides access to applications through the Internet as a Web-based service. It provides a means to free users

from complex hardware and software management by offloading such tasks to third parties, which build applications accessible to multiple users through a Web browser. In this scenario, customers neither need install anything on their premises nor have to pay considerable up-front costs to purchase the software and the required licenses. On the provider side, the specific details and features of each customer's application are maintained in the infrastructure and made available on demand.

SaaS applications are naturally multitenant. *Multitenancy*, which is a feature of SaaS compared to traditional packaged software, allows providers to centralize and sustain the effort of managing large hardware infrastructures, maintaining and upgrading applications transparently to the users, and optimizing resources by sharing the costs among the large user base. On the customer side, such costs constitute a minimal fraction of the usage fee paid for the software.

The acronym SaaS was then coined in 2001 by the *Software Information & Industry Association (SIIA)* with the following connotation:

In the software as a service model, the application, or service, is deployed from a centralized datacenter across a network—Internet, Intranet, LAN, or VPN—providing access and use on a recurring fee basis. Users "rent," "subscribe to," "are assigned," or "are granted access to" the applications from a central provider. Business models vary according to the level to which the software is streamlined, to lower price and increase efficiency, or value-added through cus- tomization to further improve digitized business processes.

The analysis carried out by SIIA was mainly oriented to cover application service providers (ASPs) and all their variations, which capture the concept of software applications consumed as a service in a broader sense. ASPs already had some of the core characteristics of SaaS:

- The product sold to customer is *application access*.
- The application is centrally managed.
- The service delivered is *one-to-many*.
- The service delivered is an integrated solution *delivered on the contract*, which means provided as promised.

The SaaS approach introduces a more flexible way of delivering application services that are fully customizable by the user by integrating new services, injecting their own components, and designing the application and information workflows. Such a new approach has also been possible with the support of Web 2.0 technologies, which allowed turning the Web browser into a full-featured interface, able even to support application composition and development.

Initially the SaaS model was of interest only for lead users and early adopters. The benefits delivered at that stage were the following:

- Software cost reduction and total cost of ownership (TCO) were paramount Service-level improvements
- Rapid implementation
- Standalone and configurable applications
- Rudimentary application and data integration
- Subscription and pay-as-you-go (PAYG) pricing

With the advent of cloud computing there has been an increasing acceptance of SaaS as a viable software delivery model. This led to transition into SaaS 2.0, which does not introduce a new technology but transforms the way in which SaaS is used. In particular, SaaS 2.0 is focused on providing a more robust infrastructure and application platforms driven by SLAs. It is important to note the role of SaaS solution enablers, which provide an environment in which to integrate third-party services and share information with others.

Types of clouds

Clouds constitute the primary outcome of cloud computing. They are a type of parallel and distributed system harnessing physical and virtual computers presented as a unified computing resource. Clouds build the infrastructure on top of which services are implemented and delivered to customers. A more useful classification is given according to the administrative domain of a cloud: It identifies the boundaries within which cloud computing services are implemented, provides hints on the underlying infrastructure adopted to support such services, and qualifies them. It is then possible to differentiate four different types of cloud:

- *Public clouds*. The cloud is open to the wider public.

- *Private clouds*. The cloud is implemented within the private premises of an institution and generally made accessible to the members of the institution or a subset of them.
- *Hybrid or heterogeneous clouds*. The cloud is a combination of the two previous solutions and most likely identifies a private cloud that has been augmented with resources or services hosted in a public cloud.
- *Community clouds*. The cloud is characterized by a multi-administrative domain involving different deployment models (public, private, and hybrid), and it is specifically designed to address the needs of a specific industry.

Public clouds

Public clouds constitute the first expression of cloud computing. They are a realization of the canonical view of cloud computing in which the services offered are made available to anyone, from anywhere, and at any time through the Internet. From a structural point of view they are a distributed system, most likely composed of one or more datacenters connected together, on top of which the specific services offered by the cloud are implemented. Any customer can easily sign in with the cloud provider, enter their credential and billing details, and use the services offered. Public clouds are used both to completely replace the IT infrastructure of enterprises and to extend it when it is required.

A fundamental characteristic of public clouds is multitenancy. A public cloud is meant to serve a multitude of users, not a single customer. A public cloud is meant to serve a multitude of users, not a single customer. Any customer requires a virtual computing environment that is separated, and most likely isolated, from other users.

QoS management is a very important aspect of public clouds. significant portion of the software infrastructure is devoted to monitoring the cloud resources, to bill them according to the contract made with the user, and to keep a complete history of cloud usage for each customer.

A public cloud can offer any kind of service: infrastructure, platform, or applications. For example, Amazon EC2 is a public cloud that provides infrastructure as a service; Google AppEngine is a public cloud that provides an application development platform as a service; and SalesForce.com is a public cloud that provides software as a service.

From an architectural point of view there is no restriction concerning the type of distributed

system implemented to support public clouds. Public clouds can be composed of geographically dispersed datacenters to share the load of users and better serve them according to their locations.

According to the specific class of services delivered by the cloud, a different software stack is installed to manage the infrastructure: virtual machine managers, distributed middleware, or distributed applications**.**

Private clouds

Private clouds are virtual distributed systems that rely on a private infrastructure and provide internal users with dynamic provisioning of computing resources. Instead of a pay-as-you-go model as in public clouds, there could be other schemes in place, taking into account the usage of the cloud and proportionally billing the different departments or sections of an enterprise. Private clouds have the advantage of keeping the core business operations in-house by relying on the existing IT infrastructure and reducing the burden of maintaining it once the cloud has been set up.

In this scenario, security concerns are less critical, since sensitive information does not flow out of the private infrastructure. Existing IT resources can be better utilized because the private cloud can provide services to a different range of users. Another interesting opportunity that comes with private clouds is the possibility of testing applications and systems at a comparatively lower price rather than public clouds before deploying them on the public virtual infrastructure.

The key advantages of using a private cloud computing infrastructure:

- *Customer information protection*. Despite assurances by the public cloud leaders about security, few provide satisfactory disclosure or have long enough histories with their cloud offerings to provide warranties about the specific level of security put in place on their systems. In-house security is easier to maintain and rely on.
- *Infrastructure ensuring SLAs*. Quality of service implies specific operations such as appropriate clustering and failover, data replication, system monitoring and maintenance, and disaster recovery, and other uptime services can be commensurate to the application needs.
- *Compliance with standard procedures and operations*. If organizations are subject to third-party compliance standards, specific procedures have to be put in place when deploying

and executing applications.

From an architectural point of view, private clouds can be implemented on more heterogeneous hardware: They generally rely on the existing IT infrastructure already deployed on the private premises. This could be a datacenter, a cluster, an enterprise desktop grid, or a combination of them.

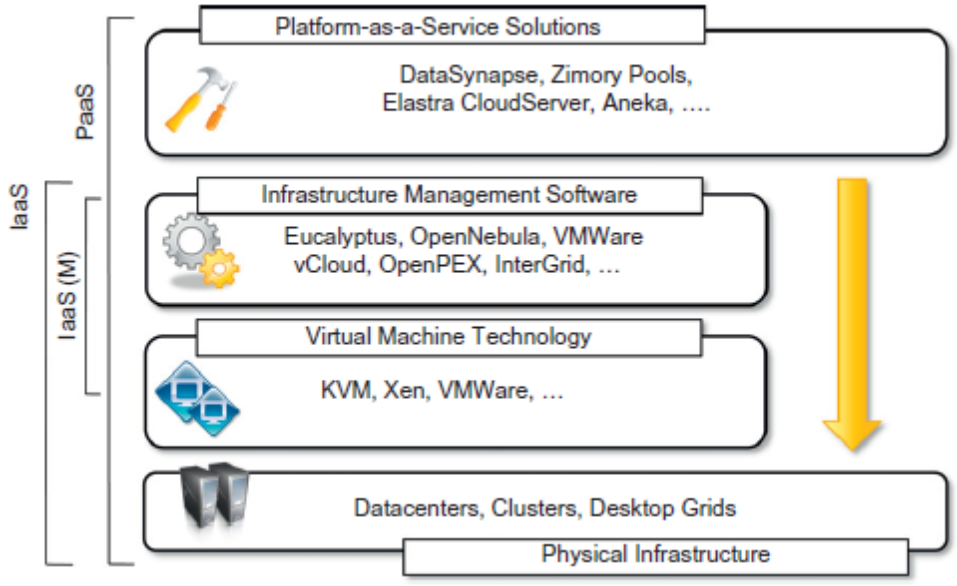


Figure 3.4 Private cloud software and hardware stack

Figure 3.4 provides a comprehensive view of the solutions together with some reference to the most popular software used to deploy private clouds. Private clouds can provide in-house solutions for cloud computing, but if compared to public clouds they exhibit more limited capability to scale elastically on demand.

Hybrid Cloud

Hybrid clouds allow enterprises to exploit existing IT infrastructures, maintain sensitive information within the premises, and naturally grow and shrink by provisioning external resources and releasing them when they're no longer needed. Security concerns are then only limited to the public portion of the cloud that can be used to perform operations with less stringent constraints but that are still part of the system workload. Figure 3.5 provides a general overview of a hybrid cloud: It is a heterogeneous distributed system resulting from a private

cloud that integrates additional services or resources from one or more public clouds. For this reason they are also called heterogeneous clouds.

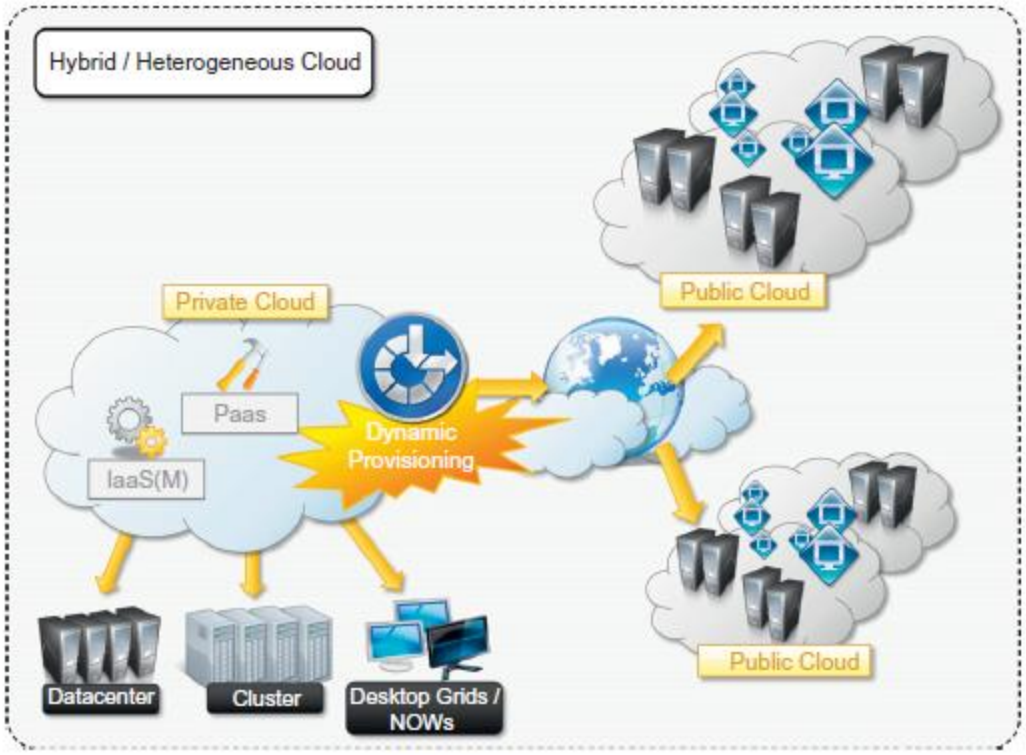


Figure 3.5 Hybrid/heterogeneous cloud overview.

As depicted in the diagram, dynamic provisioning is a fundamental component in this scenario. Hybrid clouds address scalability issues by leveraging external resources for exceeding capacity demand. These resources or services are temporarily leased for the time required and then released. This practice is also known as *cloud bursting.*

Whereas the concept of hybrid cloud is general, it mostly applies to IT infrastructure rather than software services. Service-oriented computing already introduces the concept of integration of paid software services with existing application deployed in the private premises. In an IaaS scenario, dynamic provisioning refers to the ability to acquire on demand virtual machines in order to increase the capability of the resulting distributed system and then release them. Infrastructure management software and PaaS solutions are the building blocks for deploying and managing hybrid clouds. In particular, with respect to private clouds, dynamic provisioning

introduces a more complex scheduling algorithm and policies, the goal of which is also to optimize the budget spent to rent public resources.

Community clouds

Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector. The National Institute of Standards and Technologies (NIST) characterize community clouds as follows:

The infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Figure 3.6 provides a general view of the usage scenario of community clouds, together with reference architecture. The users of a specific community cloud fall into a well-identified community, sharing the same concerns or needs; they can be government bodies, industries, or even simple users, but all of them focus on the same issues for their interaction with the cloud. This is a different scenario than public clouds, which serve a multitude of users with different needs. Community clouds are also different from private clouds, where the services are generally delivered within the institution that owns the cloud.

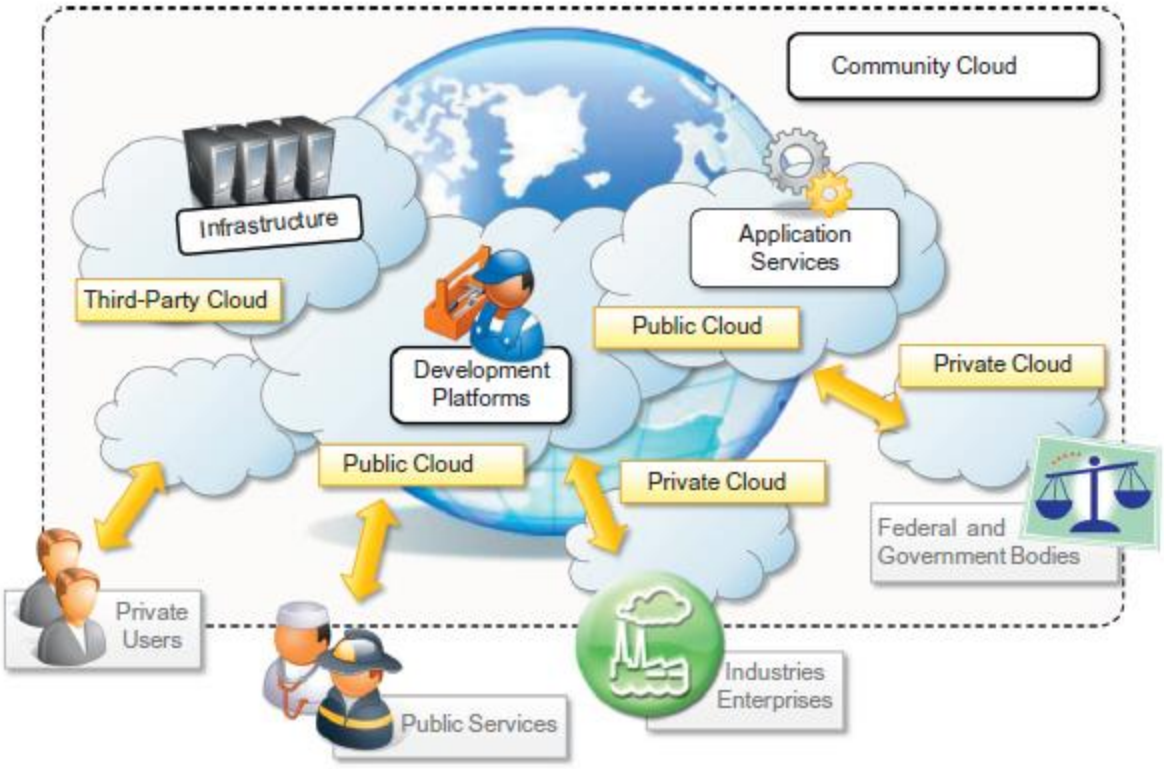


Figure 3.5 A Community Cloud

From an architectural point of view, a community cloud is most likely implemented over multiple administrative domains.

This means that different organizations such as government bodies, private enterprises, research organizations, and even public virtual infrastructure providers contribute with their resources to build the cloud infrastructure.

Candidate sectors for community clouds are as follows:

- *Media industry*. In the media industry, companies are looking for low-cost, agile, and simple solutions to improve the efficiency of content production. Most media productions involve an extended ecosystem of partners. In particular, the creation of digital content is the outcome of a collaborative process that includes movement of large data, massive compute-intensive rendering tasks, and complex workflow executions.
- *Healthcare industry*. In the healthcare industry, there are different scenarios in which community clouds could be of use. In particular, community clouds can provide a global platform on which

to share information and knowledge without revealing sensitive data maintained within the private infrastructure.

- *Energy and other core industries*. In these sectors, community clouds can bundle the comprehensive set of solutions that together vertically address management, deployment, and orchestration of services and operations.
- *Public sector*. Legal and political restrictions in the public sector can limit the adoption of public cloud offerings. Moreover, governmental processes involve several institutions and agencies and are aimed at providing strategic solutions at local, national, and international administrative levels. They involve business-to-administration, citizen-to-administration, and possibly business-to-business processes.
- *Scientific research*. Science clouds are an interesting example of community clouds. In this case, the common interest driving different organizations sharing a large distributed infrastructure is scientific computing.

The benefits of these community clouds are the following:

- *Openness*. By removing the dependency on cloud vendors, community clouds are open systems in which fair competition between different solutions can happen.
- *Community*. Being based on a collective that provides resources and services, the infrastructure turns out to be more scalable because the system can grow simply by expanding its user base.
- *Graceful failures*. Since there is no single provider or vendor in control of the infrastructure, there is no single point of failure.
- *Convenience and control*. Within a community cloud there is no conflict between convenience and control because the cloud is shared and owned by the community, which makes all the decisions through a collective democratic process.
- *Environmental sustainability*. The community cloud is supposed to have a smaller carbon footprint because it harnesses underutilized resources. Moreover, these clouds tend to be more organic by growing and shrinking in a symbiotic relationship to support the demand of the community, which in turn sustains it.

Economics of the cloud

The main drivers of cloud computing are economy of scale and simplicity of software delivery and its operation. In fact, the biggest benefit of this phenomenon is financial: the *pay-as-you-go* model offered by cloud providers. In particular, cloud computing allows:

- Reducing the capital costs associated to the IT infrastructure
- Eliminating the depreciation or lifetime costs associated with IT capital assets
- Replacing software licensing with subscriptions
- Cutting the maintenance and administrative costs of IT resources

A *capital cost* is the cost occurred in purchasing an asset that is useful in the production of goods or the rendering of services. Capital costs are one-time expenses that are generally paid up front and that will contribute over the long term to generate profit. IT resources constitute a capital cost for any kind of enterprise. It is good practice to try to keep capital costs low because they introduce expenses that will generate profit over time; more than that, since they are associated with material things they are subject to depreciation over time, which in the end reduces the profit of the enterprise because such costs are directly subtracted from the enterprise revenues. In the case of IT capital costs, the depreciation costs are represented by the loss of value of the hardware over time and the aging of software products that need to be replaced because new features are required. One of the advantages introduced by the cloud computing model is that it shifts the capital costs previously allocated to the purchase of hardware and software into operational costs inducted by renting the infrastructure and paying subscriptions for the use of software. These costs can be better controlled according to the business needs and prosperity of the enterprise. Cloud computing also introduces reductions in administrative and maintenance costs. That is, there is no or limited need for having administrative staff take care of the management of the cloud infrastructure. At the same time, the cost of IT support staff is also reduced. When it comes to depreciation costs, they simply disappear for the enterprise, since in a scenario where all the IT needs are served by the cloud there are no IT capital assets that depreciate over time.

The amount of cost savings that cloud computing can introduce within an enterprise is related to the specific scenario in which cloud services are used and how they contribute to generate a profit for the enterprise. In the case of a small startup, it is possible to completely leverage the cloud for many aspects, such as:

- IT infrastructure
- Software development
- CRM and ERP

Another important aspect is the elimination of some indirect costs that are generated by IT assets, such as software licensing and support and carbon footprint emissions. With cloud computing, an enterprise uses software applications on a subscription basis, and there is no need for any licensing fee because the software providing the service remains the property of the provider. Leveraging IaaS solutions allows room for datacenter consolidation that in the end could result in a smaller carbon footprint. In some countries such as Australia, the carbon footprint emissions are taxable, so by reducing or completely eliminating such emissions, enterprises can pay less tax.

In terms of the pricing models introduced by cloud computing, we can distinguish three different strategies that are adopted by the providers:

- *Tiered pricing*. In this model, cloud services are offered in several tiers, each of which offers a fixed computing specification and SLA at a specific price per unit of time.
- *Per-unit pricing*. This model is more suitable to cases where the principal source of revenue for the cloud provider is determined in terms of units of specific services, such as data transfer and memory allocation.
- *Subscription-based pricing*. This is the model used mostly by SaaS providers in which users pay a periodic subscription fee for use of the software or the specific component services that are integrated in their applications.

Open challenges

Still in its infancy, cloud computing presents many challenges for industry and academia.

Cloud definition

There have been several attempts made to define cloud computing and to pro- vide a classification of all the services and technologies identified as such. One of the most comprehensive formalizations is noted in the NIST working definition of cloud computing. Despite the general agreement on the NIST definition, there are alternative taxonomies for cloud services. David Linthicum, founder of BlueMountains Labs, provides a more detailed classification, which comprehends 10 different classes and better suits the vision of cloud computing within the enterprise. A different approach has been taken at the University of California, Santa Barbara which departs from the XaaS concept and tries to define an ontology for cloud computing.

Cloud interoperability and standards

To fully realize this goal, introducing standards and allowing interoperability between solutions offered by different vendors are objectives of fundamental importance. Vendor lock-in constitutes one of the major strategic barriers against the seam- less adoption of cloud computing at all stages. Vendor lock-in can prevent a customer from switching to another competitor's solution, or when this is possible, it happens at considerable conversion cost and requires significant amounts of time. The standardization efforts are mostly concerned with the lower level of the cloud computing architecture, which is the most popular and developed.

Scalability and fault tolerance

Clouds allow scaling beyond the limits of the existing in-house IT resources. To implement such a capability, the cloud middleware has to be designed with the principle of scalability along different dimensions in mind—for example, performance, size, and load. the ability to tolerate failure becomes fundamental, sometimes even more important than providing an extremely efficient and optimized system. Hence, the challenge in this case is designing highly scalable and fault-tolerant systems.

Security, trust, and privacy

Security, trust, and privacy issues are major obstacles for massive adoption of cloud computing. The massive use of virtualization technologies exposes the existing system to new threats, which previously were not considered applicable. For example, it might be possible that applications hosted in the cloud can process sensitive information; such information can be stored within a cloud storage facility using the most advanced technology in cryptography to protect data and then be considered safe from any attempt to access it without the required permissions. Although these data are processed in memory, they must necessarily be decrypted by the legitimate application, but since the application is hosted in a managed virtual environment it becomes accessible to the virtual machine manager that by program is designed to access the memory pages of such an application.

Organizational aspects

Cloud computing introduces a significant change in the way IT services are consumed and man-aged. In particular, a wide acceptance of cloud computing will require a significant change to business processes and organizational boundaries. From an organizational point of view, the lack of control over the management of data and processes poses not only security threats but also new problems that previously did not exist. The existing IT staff is required to have a different kind of competency and, in general, fewer skills, thus reducing their value. These are the challenges from an organizational point of view that must be faced

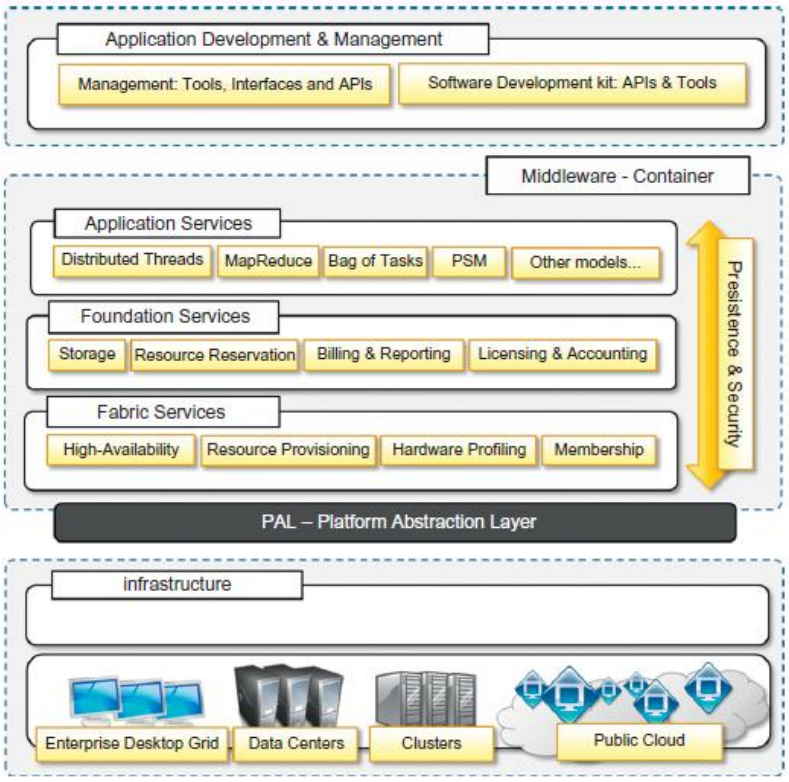
Module-2 (Section-2)

Aneka Cloud Application Platform

What Aneka is?

- Aneka is a software platform for developing cloud computing applications.
- Aneka is a pure PaaS solution for cloud computing.
- Aneka is a cloud middleware product that can be deployed on a heterogeneous set of resources: Like: a network of computers, a multi core server, data centers, virtual cloud infrastructures, or a mixture of all
- The framework provides both middleware for managing and scaling distributed applications and an extensible set of APIs for developing them

Aneka Framework with Diagram



- A collection of interconnected containers constitute the Aneka Cloud: a single domain in which services are made available to users and developers.
- The container features three different classes of services:
  - Fabric Services,
  - Foundation Services,
  - Execution Services.
- Fabric services take care of infrastructure management for the Aneka Cloud
- Foundation Services take care of supporting services for the Aneka Cloud
- Application Services take care of application management and execution respectively

Various Services offered by Aneka Cloud Platform are:
Elasticity and scaling: By means of the dynamic provisioning service, Aneka supports dynamically upsizing and downsizing of the infrastructure available for applications.

Runtime management: The run time machinery is responsible for keeping the infrastructure up and running and serves as a hosting environment for services.

Resource management: Aneka is an elastic infrastructure in which resources are added and removed dynamically according to application needs and user requirements

Application management: A specific subset of services is devoted to managing applications. These services include scheduling, execution, monitoring, and storage management.

User management: Aneka is a multi tenant distributed environment in which multiple applications, potentially belonging to different users, are executed. The framework provides an extensible user system via which it is possible to define users, groups, and permissions.

QoS/SLA management and billing: Within a cloud environment, application execution is metered and billed. Aneka provides a collection of services that coordinate together to take into account the usage of resources by each application and to bill the owning user accordingly

Anatomy of the Aneka container

- The Aneka container constitutes the building blocks of Aneka Clouds and represents the runtime machinery available to services and applications
- The container is the unit of deployment in Aneka Clouds, and it is a lightweight software layer designed to host services and interact with the underlying operating system and hardware

The Aneka container can be classified into three major categories:

• Fabric Services
• Foundation Services
• Application Services

- These services stack resides on top of the Platform Abstraction Layer (PAL) (Refer Diagram-5.2) it represents the interface to the underlying operating system and hardware.
- PAL provides a uniform view of the software and hardware environment in which the container is running

Here is the functionality of each components of Aneka Framework given in diagram 5.2

PAL –Platform Abstraction Layer

- In a cloud environment each operating system has a different file system organization and stores that information differently.
- It is The Platform Abstraction Layer (PAL) that addresses this heterogeneity problem with Operating systems and provides the container with a uniform interface for accessing the relevant information, thus the rest of the container can be operated without modification on any supported platform

The PAL provides the following features:

• Uniform and platform-independent implementation interface for accessing the hosting platform
• Uniform access to extended and additional properties of the hosting platform
• Uniform and platform-independent access to remote nodes
• Uniform and platform-independent management interfaces

Also The PAL is a small layer of software that comprises a detection engine, which automatically configures the container at boot time, with the platform-specific component to access the above information and an implementation of the abstraction layer for the Windows, Linux, and Mac OS X operating systems.

Following are the collectible data that are exposed by the PAL:
- Number of cores, frequency, and CPU usage
- Memory size and usage
- Aggregate available disk space
- Network addresses and devices attached to the node

Fabric services

- **FabricServices** define the lowest level of the software stack representing the Aneka Container.
- They provide access to the Resource-provisioning subsystem and to the Monitoring facilities implemented in Aneka.
- Resource-provisioning services are in charge of dynamically providing new nodes on demand by relying on virtualization technologies
- Monitoring services allow for hardware profiling and implement a basic monitoring infrastructure that can be used by all the services installed in the container

The two services of Fabric class are:

- Profiling and monitoring
- Resource management

Profiling and monitoring

Profiling and monitoring services are mostly exposed through following services

- Heartbeat,
- Monitoring,
- Reporting
- The Heart Beat makes the information that is collected through the PAL available
- The Monitoring and Reporting implement a generic infrastructure for monitoring the activity of any service in the Aneka Cloud;

Heartbeat Functions in detail

- The Heartbeat Service periodically collects the dynamic performance information about the node and publishes this information to the membership service in the Aneka Cloud
- It collects basic information about memory, disk space, CPU, and operating system
- These data are collected by the index node of the Cloud, and makes them available for reservations and scheduling services that optimizes them for heterogeneous infrastructure

- Heartbeat works with a specific component, called Node Resolver, which is in charge of collecting these data

Reporting & Monitoring Functions in detail

- The Reporting Service manages the store for monitored data and makes them accessible to other services for analysis purposes.
- On each node, an instance of the Monitoring Service acts as a gateway to the Reporting Service and forwards to it all the monitored data that has been collected on the node

Many Built-in services use this channel to provide information, important built-in services are:

- The Membership Catalogue service tracks the performance information of nodes.
- The Execution Service monitors several time intervals for the execution of jobs.
- The Scheduling Service tracks the state transitions of jobs.
- The Storage Service monitors and obtains information about data transfer such as upload and download times, file names, and sizes.
- The Resource Provisioning Service tracks the provisioning and life time information of virtual nodes.

Resource management

Aneka provides a collection of services that are in charge of managing resources. These are

- Index Service (or Membership Catalogue)
- Resource Provisioning Service

Membership Catalogue features

- The MembershipCatalogue is Aneka's fundamental component for resource management
- It keeps track of the basic node information for all the nodes that are connected or disconnected.
- It implements the basic services of a directory service, whereservices can be searched using attributes such as names and nodes

Resource Provisioning Service Features

- The resource provisioning infrastructure built into Aneka is mainly concentrated in the Resource Provisioning Service,
- It includes all the operations that are needed for provisioning virtual instances (Providing virtual instances as needed by users).
- The implementation of the service is based on the idea of resource pools.
- A resource pool abstracts the interaction with a specific IaaS provider by exposing a common interface so that all the pools can be managed uniformly.

Foundation services

Foundation Services are related to the logical management of the distributed system built on top of the infrastructure and provide supporting services for the execution of distributed applications. These services cover:

- Storage management for applications
- Accounting, billing and resource pricing
- Resource reservation

Storage management

Aneka offers two different facilities for storage management:

- A centralized file storage, which is mostly used for the execution of compute- intensive applications,
- A distributed file system, which is more suitable for the execution of data-intensive applications.
- As the requirements for the two types of applications are rather different. Compute-intensive applications mostly require powerful processors and do not have high demands in terms of storage, which in many cases is used to store small files that are easily transferred from one node to another. Here, a centralized storage node is sufficient to store data
- Centralized storage is implemented through and managed by Aneka's Storage Service.
- The protocols for implementing centralized storage management are supported by a concept of File channel. It consists of a File Channel controller and File channel handler. File channel controller is a server component whereas File channel handler is a client component (that allows browsing, downloading and uploading of files).
- In contrast, data-intensive applications are characterized by large data files (gigabytes or terabytes, peta bytes) and here processing power required by tasks is not more.
- Here instead of centralized data storage a distributed file system is used for storing data by using all the nodes belonging to the cloud.
- Data intensive applications are implemented by means of a distributed file system. Google File system is best example for distributed file systems.
- Typical characteristics of Google File system are:
    - Files are huge by traditional standards (multi-gigabytes).
    - Files are modified by appending new data rather than rewriting existing data.
    - There are two kinds of major workloads: large streaming reads and small random reads.
    - It is more important to have a sustained bandwidth than a low latency.

Accounting, billing, and resource pricing

- Accounting Services keep track of the status of applications in the Aneka Cloud
- The information collected for accounting is primarily related to infrastructure usage and application execution
- Billing is another important feature of accounting.
- Billing is important since Aneka is a multitenant cloud programming platform in which the execution of applications can involve provisioning additional resources from commercial providers.

- Aneka Billing Service provides detailed information about each user's usage of resources, with the associated costs.
- Resource pricing is associated with the price fixed for different types of resources/nodes that are provided for the subscribers. Powerful resources are priced high and less featured resources are priced low
- Two internal services used by accounting and billing are Accounting service and Reporting Service

Resource reservation

Resource Reservation supports the execution of distributed applications and allows for reserving resources for exclusive use by specific applications.

Two types of services are used to build resource reservation:

- The Resource Reservation
- The Allocation Service
- Resource Reservation keeps track of all the reserved time slots in the Aneka Cloud and provides a unified view of the system. (provides overview of the system)
- The Allocation Service is installed on each node that features execution services and manages the database of information regarding the allocated slots on the local node.

Different Reservation Service Implementations supported by Aneka Cloud are:

Basic Reservation: Features the basic capability to reserve execution slots on nodes and implements the alternate offers protocol, which provides alternative options in case the initial reservation requests cannot be satisfied.

Libra Reservation: Represents a variation of the previous implementation that features the ability to price nodes differently according to their hardware capabilities.

Relay Reservation: This implementation is useful in integration scenarios in which Aneka operates in an inter cloud environment.

Application services

Application Services manage the execution of applications and constitute a layer that differentiates according to the specific programming model used for developing distributed applications on top of Aneka

Two types of services are:

1. The Scheduling Service :

Scheduling Services are in charge of planning the execution of distributed applications on top of Aneka and governing the allocation of jobs composing an application to nodes. Common tasks that are performed by the scheduling component are the following:

- Job to node mapping

- Rescheduling of failed jobs
- Job status monitoring
- Application status monitoring

2. The Execution Service

Execution Services control the execution of single jobs that compose applications. They are in charge of setting up the runtime environment hosting the execution of jobs.

Some of the common operations that apply across all the range of supported models are:

• Unpacking the jobs received from the scheduler
• Retrieval of input files required for job execution
• Sandboxed execution of jobs
• Submission of output files at the end of execution
• Execution failure management (i.e., capturing sufficient contextual information useful to identify the nature of the failure)
• Performance monitoring
• Packing jobs and sending them back to the scheduler

The various Programming Models Supported by Execution Services of Aneka Cloud are:

1. Task Model. This model provides the support for the independent "bag of tasks" applications and many computing tasks. In this model application is modelled as a collection of tasks that are independent from each other and whose execution can be sequenced in any order
2. Thread Model. This model provides an extension to the classical multithreaded programming to a distributed infrastructure and uses the abstraction of Thread to wrap a method that is executed remotely.
3. Map Reduce Model. This is an implementation of Map Reduce as proposed by Google on top of Aneka.
4. Parameter Sweep Model. This model is a specialized form of Task Model for applications that can be described by a template task whose instances are created by generating different combinations of parameters.

Building Aneka clouds

Aneka Cloud can be realized by two methods:

1. Infrastructure Organization
2. Logical Organization

Infrastructure based organization of Aneka Cloud is given in the following figure-5.3:

The working mechanism of this model:

It contains **Aneka Repository, Administrative Console, Aneka Containers & Node Managers as major components.**

- The Management Console manages multiple repositories and select the one that best suits the specific deployment

- A **Repository** provides storage for all the libraries required to layout and install the basic Aneka platform, by installing images of the required software in particular Aneka Container through node managers by using various protocols like FTP, HTTP etc.
- A number of node managers and Aneka containers are deployed across the cloud platform to provision necessary services, The Aneka node manager are also known as AnekaDaemon
- The collection of resulting containers identifies the final AnekaCloud

FIGURE 5.3
Aneka cloud infrastructure overview.

Logical organization

The logical organization of Aneka Clouds can be very diverse, since it strongly depends on the configuration selected for each of the container instances belonging to the Cloud.

Here is a scenario that has master-worker configuration with separate nodes for storage, the Figure 5.4. Portray

The master node comprises of following services:

- Index Service (master copy)
- Heartbeat Service
- Logging Service
- Reservation Service
- Resource Provisioning Service
- Accounting Service
- Reporting and Monitoring Service
- Scheduling Services for the supported programming models

Here Logging service and Heartbeat service and Monitoring service are considered as Mandatory services in all the block diagrams whereas other services are shown ditto.

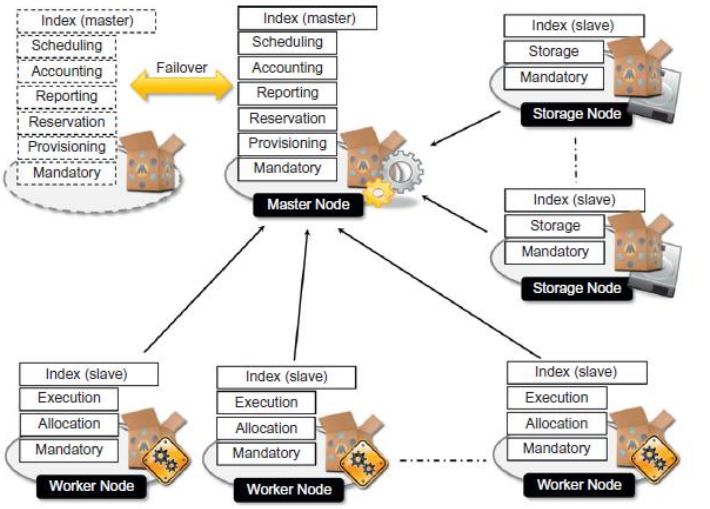


FIGURE 5.4
Logical organization of an Aneka cloud.

Similarly the Worker Node comprises of following services:

- Index Service
- Execution service
- Allocation service
- And mandatory (Logging, Heartbeat and monitoring services)

The Storage Node comprises of :

- Index service
- Storage Service
- And mandatory (Logging, Heartbeat and monitoring services)

In addition all nodes are registered with the master node and transparently refer to any failover partner in the case of a high-availability configuration

Aneka Cloud Deployment Models

All the general cloud deployment models like Private cloud deployment mode, Public cloud deployment mode and Hybrid Cloud deployment mode are applicable to Aneka Clouds also.

Private cloud deployment mode

A private deployment mode is mostly constituted by local physical resources and infrastructure management software providing access to a local pool of nodes, which might be virtualized.

Figure 5.5 shows a common deployment for a private Aneka Cloud. This deployment is acceptable for a scenario in which the workload of the system is predictable and a local virtual machine manager can easily address excess capacity demand. Most of the Aneka nodes are constituted of physical nodes with a long lifetime and a static configuration and generally do not need to be reconfigured often. The different nature of the machines harnessed in a private environment allows for specific policies on resource management and usage that can be

accomplished by means of the Reservation Service. For example, desktop machines that are used during the day for office automation can be exploited outside the standard working hours to execute distributed applications. Workstation clusters might have some specific legacy software that is required for supporting the execution of applications and should be executed with special requirements.

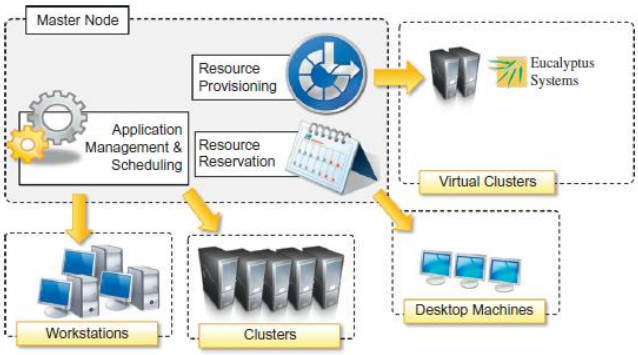


FIGURE 5.5
Private cloud deployment.

Note: In the master node: Resource Provisioning, Application Management & Scheduling and Resource Reservation are the primary services.

Public cloud deployment mode

Public Cloud deployment mode features the installation of Aneka master and worker nodes over a completely virtualized infrastructure that is hosted on the infrastructure of one or more resource providers such as Amazon EC2 or GoGrid.

Figure 5.6 provides an overview of this scenario. The deployment is generally contained within the infrastructure boundaries of a single IaaS pro- vider. The reasons for this are to minimize the data transfer between different providers, which is generally priced at a higher cost, and to have better network performance. In this scenario it is pos- sible to deploy an Aneka Cloud composed of only one node and to completely leverage dynamic provisioning to elastically scale the infrastructure on demand. A fundamental role is played by the Resource Provisioning Service, which can be configured with different images and templates to instantiate. Other important services that have to be included in the master node are the Accounting and Reporting Services. These provide details about resource utilization by users and applications and are fundamental in a multitenant Cloud where users are billed according to their consumption of Cloud capabilities.

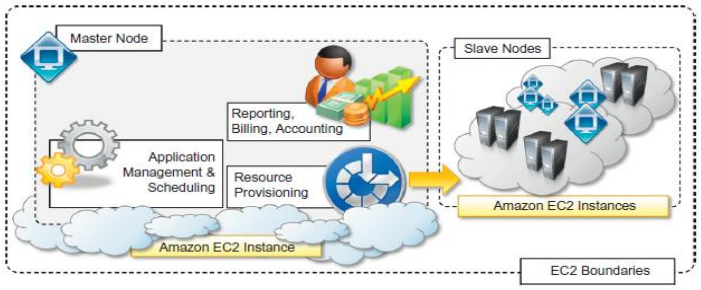


FIGURE 5.6
Public Aneka cloud deployment.

Note: Reporting, Billing, Accounting, Resource Provisioning and Application Management & Scheduling are the primary services in master node

Hybrid cloud deployment mode

The hybrid deployment model constitutes the most common deployment of Aneka. In many cases, there is an existing computing infrastructure that can be leveraged to address the computing needs of applications. This infrastructure will constitute the static deployment of Aneka that can be elastically scaled on demand when additional resources are required

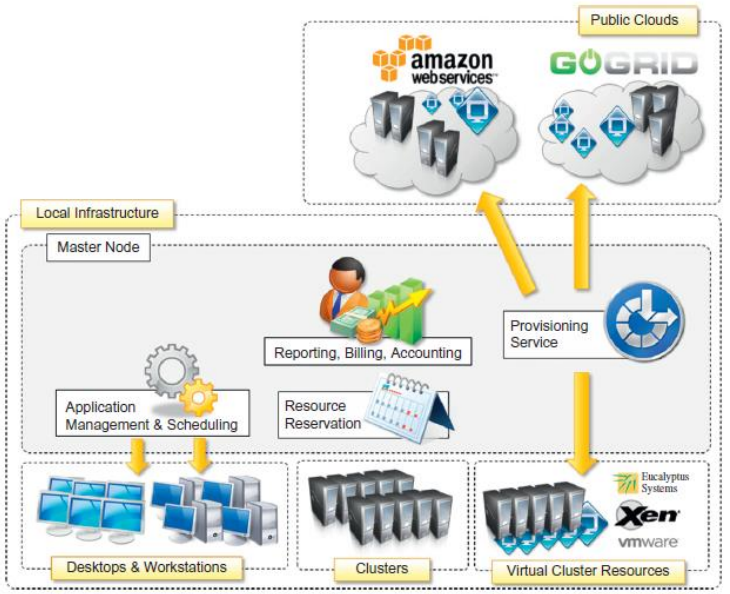


FIGURE 5.7
Hybrid cloud deployment.

An overview of this deployment is presented in Figure 5.7. This scenario constitutes the most complete deployment for Aneka that is able to leverage all the capabilities of the framework:

- Dynamic Resource Provisioning
- Resource Reservation
- Workload Partitioning (Scheduling)
- Accounting, Monitoring, and Reporting

In a hybrid scenario, heterogeneous resources can be used for different purposes. As we discussed in the case of a private cloud deployment, desktop machines can be reserved for low priority work- load outside the common working hours. The majority of the applications will be executed on work- stations and clusters, which are the nodes that are constantly connected to the Aneka Cloud. Any additional computing capability demand can be primarily addressed by the local virtualization facili- ties, and if more computing power is required, it is possible to leverage external IaaS providers.

Cloud programming and management

- Aneka's primary purpose is to provide a scalable middleware product in which to execute distributed applications.
- Application development and management constitute the two major features that are exposed to developers and system administrators.
- Aneka provides developers with a comprehensive and extensible set of APIs and administrators with powerful and intuitive management tools.
- The APIs for development are mostly concentrated in the Aneka SDK; management tools are exposed through the Management Console

Aneka SDK

Aneka provides APIs for developing applications on top of existing programming models, implementing new programming models, and developing new services to integrate into the Aneka Cloud.

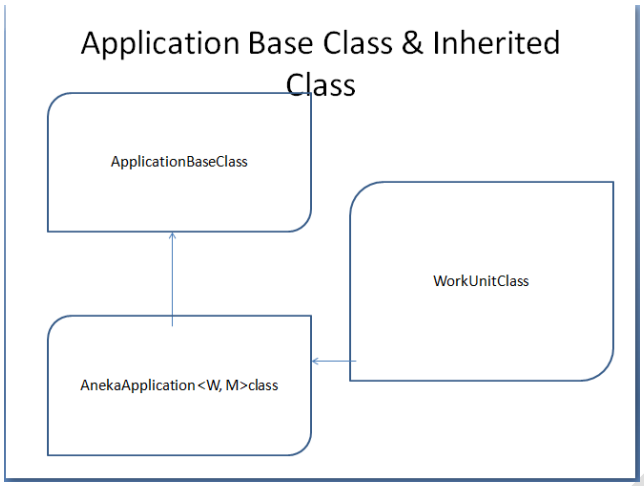
The SDK provides support for both programming models and services by

- The Application Model
- The Service Model.

Application Model

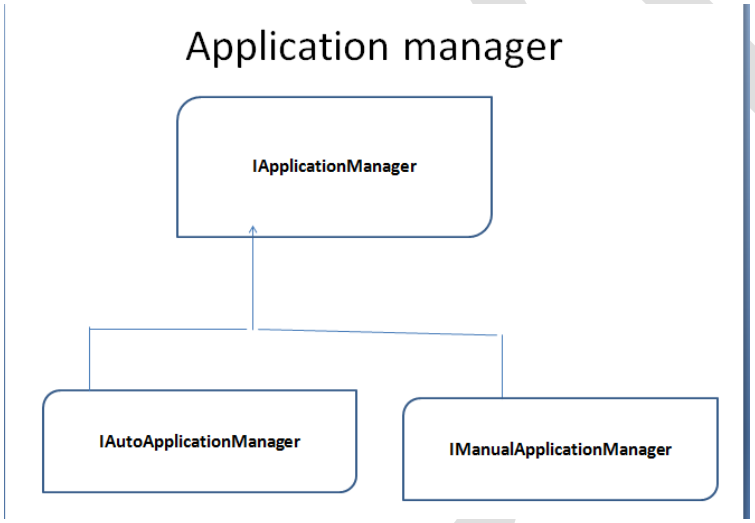
- The Application Model covers the development of applications and new programming models
- It Consists of Application Class & Application Manager
- Application Class – Provide user/developer view about distributed applications of the Aneka cloud
- Application Manager – Are Internal components that control and monitor the execution of Aneka clouds

The Application Class can be represented by following class diagram



Note: All the Aneka Application<W,M> class where W stands for Worker and M stands for Manager is inherited from base class and all Manual services are represented by WorkUnitClass. In addition there are two other classes in Application Class representation viz: Configuration Class and Application Data Class

The Application manager is represented with following class diagram:



Also the table given below summarizes Application Class, The programming models supported and work units assigned to them.

Table 5.1 Aneka's Application Model Features

Category	Description	Base Application Type	Work Units?	Programming Models
Manual	Units of work are generated by the user and submitted through the application.	*AnekaApplication<W,M>* *IManualApplicationManager<W>* *ManualApplicationManager<W>*	Yes	Task Model Thread Model Parameter Sweep Model
Auto	Units of work are generated by the runtime infrastructure and managed internally.	*ApplicationBase<M>* *IAutoApplicationManager*	No	*MapReduce* Model

The Service Model defines the general infrastructure for service development.

The Aneka Service Model defines the basic requirements to implement a service that can be hosted in an Aneka Cloud. The container defines the runtime environment in which services are hosted. Each service that is hosted in the container must use IService interface, which exposes the following methods and properties:

• Name and status

• Control operations such as Start, Stop, Pause, and Continue methods

• Message handling by means of the HandleMessage method

Figure 5.9 describes the reference life cycle of each service instance in the Aneka container.

A service instance can initially be in the Unknown or Initialized state, a condition that refers to the creation of the service instance by invoking its constructor during the configuration of the container. Once the container is started, it will iteratively call the Start method on each service method. As a result the service instance is expected to be in a Starting state until the startup process is completed, after which it will exhibit the Running state. This is the condition in which the service will last as long as the container is active and running. This is the only state in which the service is able to process messages. If an exception occurs while starting the service, it is expected that the service will fall back to the Unknown state, thus signalling an error. When a service is running it is possible to pause its activity by calling the Pause method and resume it by calling Continue.

As described in the figure, the service moves first into the Pausing state, thus reaching the Paused state. From this state, it moves into the Resuming state while restoring its activity to return to the Running state. Not all the services need to support the pause/continue operations, and the current implementation of the framework does not feature any service with these capabilities. When the container shutdown, the Stop method is iteratively called on each service running, and services move first into the transient Stopping state to reach the final Stopped state, whereall resources that were initially allocated have been released.

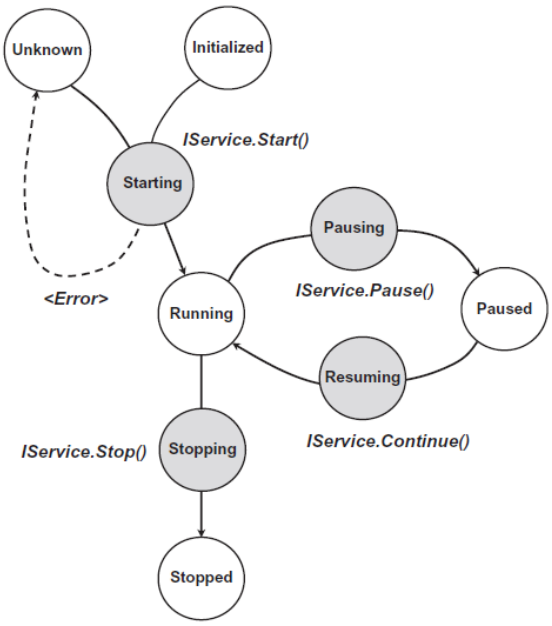


FIGURE 5.9
Service life cycle.

Note: Here all Unfilled Circles: Running, Unknown, Initialize, Paused and Stopped are Steady states. The filled Circles: Starting, Pausing, Resuming and Stopping are Transient States.

MANAGEMENT TOOLS

Aneka is a pure PaaS implementation and requires virtual or physical hardware to be deployed. Aneka's management layer, also includes capabilities for managing services and applications running in the Aneka Cloud.

Infrastructure management

Aneka leverages virtual and physical hardware in order to deploy Aneka Clouds. Virtual hardware is generally managed by means of the Resource Provisioning Service, which acquires resources on demand according to the need of applications, while physical hardware is directly managed by the Administrative Console by leveraging the Aneka management API of the PAL.

Platform management

The creation of Clouds is orchestrated by deploying a collection of services on the physical infra-structure that allows the installation and the management of containers. A collection of connected containers defines the platform on top of which applications are executed. The features available for platform management are mostly concerned with the logical organization and structure of Aneka Clouds.

Application management

Applications identify the user contribution to the Cloud. This is an important feature in a cloud computing scenario in which users are billed for their resource usage. Aneka exposes capabilities for giving summary and detailed information about application execution and resource utilization.

Module 3

Concurrent Computing

Throughput computing focuses on delivering high volumes of computation in the form of transactions. Advances in hardware technologies led to the creation of multi core systems, which have made possible the delivery of high-throughput computations

Multiprocessing is the execution of multiple programs in a single machine, whereas multithreading relates to the possibility of multiple instruction streams within the same program.

This chapter presents the concept of multithreading and describes how it supports the development of high-throughput computing applications.

6.1 Introducing parallelism for single-machine computation

Parallelism has been a technique for improving the performance of computers since the early 1960s.

In particular, multiprocessing, which is the use of multiple processing units within a single machine, has gained a good deal of interest and gave birth to several parallel architectures.

Asymmetric multiprocessing involves the concurrent use of different processing units that are specialized to perform different functions. **Symmetric multiprocessing** features the use of similar or identical

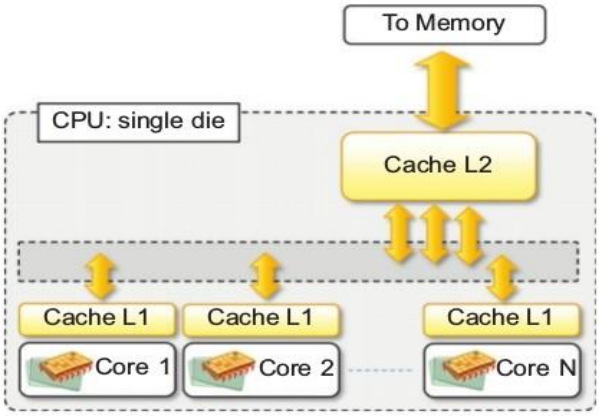


FIGURE 6.1

Multicore processor.

processing units to share the computation load.

Multicore systems are composed of a single processor that features multiple processing cores that share the memory. Each core has generally its own L1 cache, and the L2 cache is common to all the cores, which connect to it by means of a shared bus, as depicted in Figure 6.1. Dual- and quad-core configurations are quite popular nowadays and constitute the standard hardware configuration for commodity computers. Architectures with multiple cores are also available but are not designed for the commodity market. Multicore technology has been used not only as a support for processor design but also in other devices, such as GPUs and network devices, thus becoming a standard practice for improving performance.

Multiprocessing is just one technique that can be used to achieve parallelism, and it does that by leveraging parallel hardware architectures.

In particular, an important role is played by the operating system, which defines the runtime structure of applications by means of the abstraction of process and thread. A process is the runtime image of an application, or better, a program that is running, while a thread identifies a single flow of the execution within a process. A system that allows the execution of multiple processes at the same time supports multitasking. It supports multithreading when it provides structures for explicitly defining multiple threads within a process.

6.2 Programming applications with threads

Modern applications perform multiple operations at the same time.The use of threads might be implicit or explicit.

Implicit threading happens when the underlying APIs use internal threads to perform specific tasks supporting the execution of applications such as graphical user interface (GUI) rendering, or garbage collection in the case of virtual machine-based languages.

Explicit threading is characterized by the use of threads within a program by application developers, who use this abstraction to introduce parallelism. Common cases in which threads are explicitly used are I/O from devices and network connections, long computations, or the execution of background operations.

6.2.1 What is a thread?

A thread identifies a single control flow, which is a logical sequence of instructions, within a process. By logical sequence of instructions, we mean a sequence of instructions that have been designed to be executed one after the other one.

Operating systems that support multithreading identify threads as the minimal building blocks for expressing running code.

Each process contains at least one thread but, in several cases, is composed of many threads having variable lifetimes. Threads within the same process share the memory space and the execution context.

In a **multitasking** environment the operating system assigns different time slices to each process and interleaves their execution. The process of temporarily stopping the execution of one process, saving all the information in the registers, and replacing it with the information related to another process is known as a **context switch**.

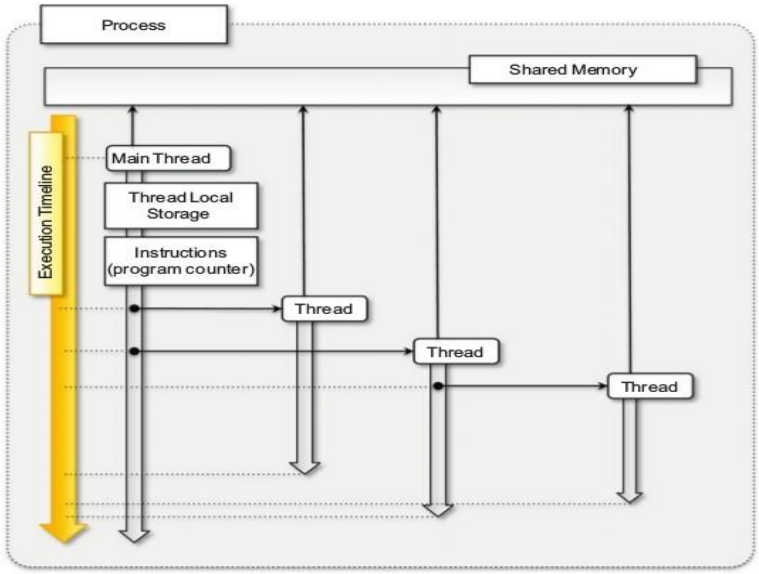


FIGURE 6.2
The relationship between processes and threads.

Figure 6.2 provides an overview of the relation between threads and processes and a simplified representation of the runtime execution of a multithreaded application. A running program is identified by a process, which contains at least one thread, also called the main thread. Such a thread is implicitly created by the compiler or the runtime environment executing the program. This thread is likely to last for the entire lifetime of the process and be the origin of other threads, which in general exhibit a shorter duration. As main threads, these threads can spawn other threads. There is no difference between the main thread and other threads created during the process lifetime. Each of them has its own local storage and a sequence of instructions to execute, and they all share the memory space allocated for the entire process. The execution of the process is considered terminated when all the threads are completed.

Thread APIs

1 POSIX Threads

2 Threading support in java and .NET

1 POSIX Threads

Portable Operating System Interface for Unix (POSIX) is a set of standards related to the application programming interfaces for a portable development of applications over the Unix operating system flavors. Standard POSIX 1.c (IEEE Std 1003.1c-1995) addresses the implementation of threads and the functionalities that should be available for application programmers to develop portable multithreaded applications.

Important to remember from a programming point of view is the following:

- A thread identifies a logical sequence of instructions.
- A thread is mapped to a function that contains the sequence of instructions to execute.
- A thread can be created, terminated, or joined.

- A thread has a state that determines its current condition, whether it is executing, stopped, terminated, waiting for I/O, etc.
- The sequence of states that the thread undergoes is partly determined by the operating system scheduler and partly by the application developers.
- Threads share the memory of the process, and since they are executed concurrently, they need synchronization structures.
- Different synchronization abstractions are provided to solve different synchronization problems.

2 Threading support in java and .NET

Languages such as Java and C# provide a rich set of functionalities for multithreaded programming by using an object-oriented approach. both Java and .NET execute code on top of a virtual machine, the APIs exposed by the libraries refer to managed or logical threads. These are mapped to physical threads.

- Both Java and .NET provide class Thread with the common operations on threads: start, stop, suspend, resume, abort, sleep, join, and interrupt.
- Start and stop/abort are used to control the lifetime of the thread instance.
- Suspend and resume are used to programmatically pause and then continue the execution of a thread. Sleep operation allows pausing the execution of a thread for a predefined period of time.
- Join operation that makes one thread wait until another thread is completed. Waiting states can be interrupted by using the interrupt operation.

6.2.2 Techniques for parallel computation with threads

Developing parallel applications requires an understanding of the problem and its logical structure. Decomposition is a useful technique that aids in understanding whether a problem is divided into components (or tasks) that can be executed concurrently. it allows the breaking down into independent units of work that can be executed concurrently with the support provided by threads.

1 Domain decomposition

2 Functional decomposition

3 Computation vs. Communication

1 Domain decomposition

Domain decomposition is the process of identifying patterns of functionally repetitive, but independent, computation on data. This is the most common type of decomposition in the case of throughput computing, and it relates to the identification of repetitive calculations required for solving a problem. The master-slave model is a quite common organization for these scenarios:

- The system is divided into two major code segments.
- One code segment contains the decomposition and coordination logic.

- Another code segment contains the repetitive computation to perform.
- A master thread executes the first code segment.
- As a result of the master thread execution, as many slave threads as needed are created to execute the repetitive computation.
- The collection of the results from each of the slave threads and an eventual composition of the final result are performed by the master thread.

Embarrassingly parallel problems constitute the easiest case for parallelization because there is no need to synchronize different threads that do not share any data. Embarrassingly parallel problems are quite common, they are based on the strong assumption that at each of the iterations of the decomposition method, it is possible to isolate an independent unit of work. This is what makes it possible to obtain a high computing throughput. If the values of all the iterations are dependent on some of the values obtained in the previous iterations, the problem is said to be **inherently sequential**. **Figure 6.3** provides a schematic representation of the decomposition of embarrassingly parallel and inherently sequential problems.

The matrix product computes each element of the resulting matrix as a linear combination of the corresponding row and column of the first and second input matrices, respectively. The formula that applies

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

for each of the resulting matrix elements is the following:

Two conditions hold in order to perform a matrix product:

- Input matrices must contain values of a comparable nature for which the scalar product is defined.
- The number of columns in the first matrix must match the number of rows of the second matrix

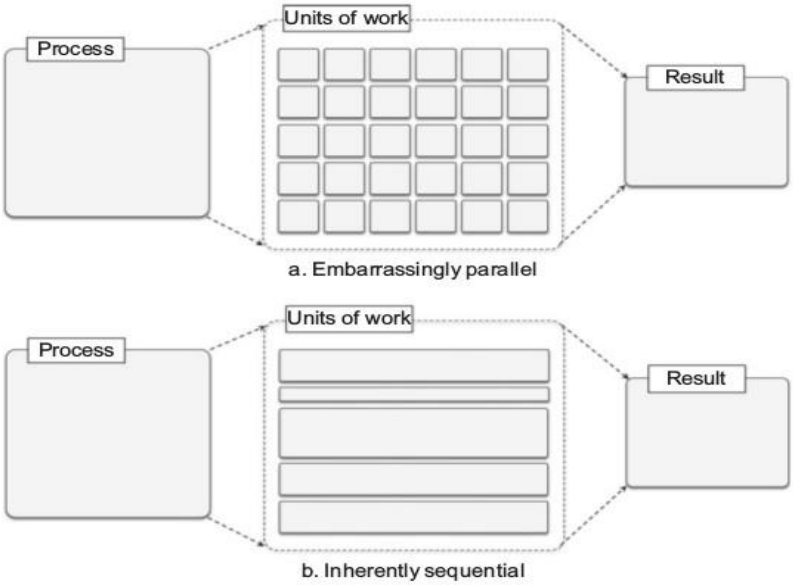


FIGURE 6.3
Domain decomposition techniques.

Figure 6.4 provides an overview of how a matrix product can be performed.

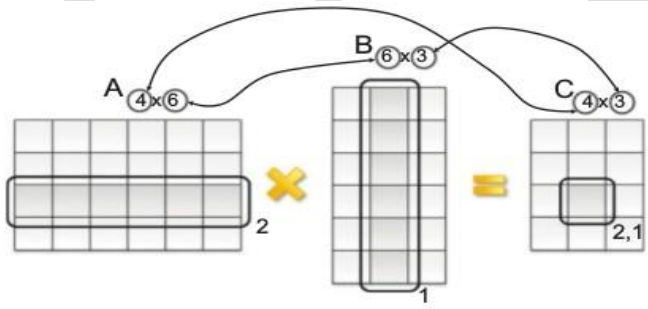


FIGURE 6.4
A matrix product.

The problem is embarrassingly parallel, and we can logically organize the multithreaded program in the following steps:

- Define a function that performs the computation of the single element of the resulting matrix by implementing the previous equation.
- Create a double for loop (the first index iterates over the rows of the first matrix and the second over the columns of the second matrix) that spawns a thread to compute the elements of the resulting matrix.
- Join all the threads for completion, and compose the resulting matrix.

The .NET framework provides the System.Threading.Thread class that can be configured with a function pointer, also known as a delegate, to execute asynchronously. This class will also define the method for performing the actual computation.

Functional decomposition

Functional decomposition is the process of identifying functionally distinct but independent computations. The focus here is on the type of computation rather than on the data manipulated by the computation.

This kind of decomposition is less common and does not lead to the creation of a large number of threads, since the different computations that are performed by a single program are limited.

Functional decomposition leads to a natural decomposition of the problem in separate units of work.

Figure 6.5 provides a pictorial view of how decomposition operates and allows parallelization.

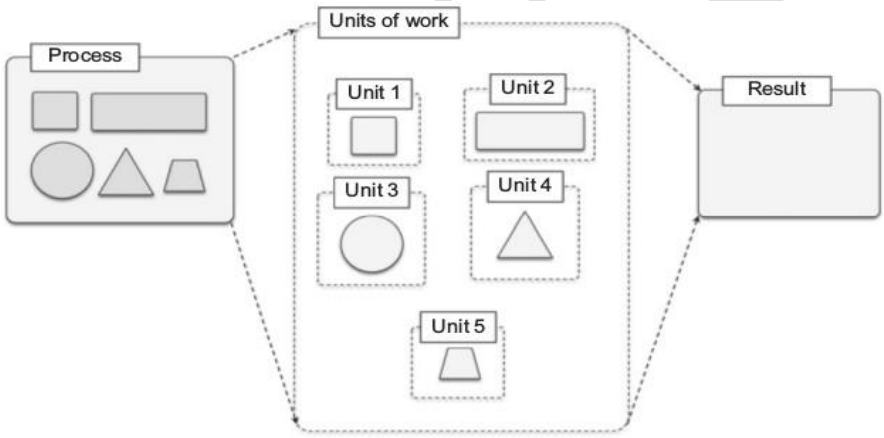


FIGURE 6.5

Functional decomposition.

The problems that are subject to functional decomposition can also require a composition phase in which the outcomes of each of the independent units of work are composed together.

In the following, we show a very simple example of how a mathematical problem can be parallelized using functional decomposition. Suppose, for example, that we need to calculate the value of the following function for a given value of x:

$$f(x) = \sin(x) + \cos(x) + \tan(x)$$

Once the value of x has been set, the three different operations can be performed independently of each other. This is an example of functional decomposition because the entire problem can be separated into three distinct operations.

Computation vs. Communication

It is very important to carefully evaluate the communication patterns among the components that have been identified during problem decomposition. The two decomposition methods presented in this section and the corresponding sample applications are based on the assumption that the computations are independent. This

means that:

- The input values required by one computation do not depend on the output values generated by another computation.
- The different units of work generated as a result of the decomposition do not need to interact (i.e., exchange data) with each other.

These two assumptions strongly simplify the implementation and allow achieving a high degree of parallelism and a high throughput.

6.3 Multithreading with Aneka

As applications become increasingly complex, there is greater demand for computational power that can be delivered by a single multicore machine.

Often this demand cannot be addressed with the computing capacity of a single machine. It is then necessary to leverage distributed infrastructures such as clouds.

Decomposition techniques can be applied to partition a given application into several units of work, submitted for execution by leveraging clouds.

Aneka, as middleware for managing clusters, grids, and clouds, provides developers with advanced capabilities for implementing distributed applications. In particular, it takes traditional thread programming a step further. It lets you write multithreaded applications the traditional way, with the added twist that each of these threads can now be executed outside the parent process and on a separate machine. In reality, these "threads" are independent processes executing on different nodes and do not share memory or other resources, but they allow you to write applications using the same thread constructs for concurrency and synchronization as with traditional threads. Aneka threads, as they are called, let you easily port existing multithreaded compute-intensive applications to distributed versions that can run faster by utilizing multiple machines simultaneously, with minimum conversion effort.

6.3.1 Introducing the thread programming model

Aneka offers the capability of implementing multithreaded applications over the cloud by means of the **Thread Programming Model**.

This model introduces the abstraction of distributed thread, also called **Aneka thread**, which mimics the behavior of local threads but executes over a distributed infrastructure.

This model provides the best advantage in the case of **embarrassingly parallel applications**.

The Thread Programming Model exhibits APIs that mimic the ones exposed by .NET base class libraries for threading. In this way developers do not have to completely rewrite applications in order to leverage Aneka by replacing the **System.Threading.Thread** class and introducing the **AnekaApplication** class.

There are three major elements that constitute the object model of applications based on the Thread

Programming Model:

Application. This class represents the interface to the Aneka middleware and constitutes a local view of a distributed application. In the Thread Programming Model the single units of work are created by the programmer. Therefore, the specific class used will be Aneka.Entity. AnekaApplication<T,M>, with T and M properly selected.

Threads. Threads represent the main abstractions of the model and constitute the building blocks of the distributed application. Aneka provides the Aneka.Threading.AnekaThread class, which represents a distributed thread.

Thread Manager. This is an internal component that is used to keep track of the execution of distributed threads and provide feedback to the application. Aneka provides a specific version of the manager for this model, which is implemented in the Aneka.Threading.ThreadManager class.

6.3.2 Aneka thread vs. common threads

To efficiently run on a distributed infrastructure, Aneka threads have certain limitations compared to local threads.

These limitations relate to the communication and synchronization strategies.

1 Interface compatibility

2 Thread life cycle

3 Thread synchronization

4 Thread priorities

5 Type serialization

1 Interface compatibility

The Aneka.Threading.AnekaThread class exposes almost the same interface as the System.Threading.Thread class with the exception of a few operations that are not supported. Table 6.1 compares the operations that are exposed by the two classes.

Table 6.1 Thread API Comparison

.Net Threading API	Aneka Threading API
System.Threading	*Aneka.Threading*
Thread	*AnekaThread*
Thread.ManagedThreadId (int)	*AnekaThread.Id (string)*
Thread.Name	*AnekaThread.Name*
Thread.ThreadState (ThreadState)	*AnekaThread.State*
Thread.IsAlive	*AnekaThread.IsAlive*
Thread.IsRunning	*AnekaThread.IsRunning*
Thread.IsBackground	*AnekaThread.IsBackground[false]*
Thread.Priority	*AnekaThread.Priority[ThreadPriority.Normal]*
Thread.IsThreadPoolThread	*AnekaThread.IsThreadPoolThread [false]*
Thread.Start	*AnekaThread.Start*
Thread.Abort	*AnekaThread.Abort*
Thread.Sleep	[Not provided]
Thread.Interrupt	[Not provided]
Thread.Suspend	[Not provided]
Thread.Resume	[Not provided]
Thread.Join	*AnekaThread.Join*

The basic control operations for local threads such as Start and Abort have a direct mapping, whereas operations that involve the temporary interruption of the thread execution have not been supported.

The reasons for such a design decision are twofold. **Firs**t, the use of the Suspend/Resume operations is generally a deprecated practice, even for local threads, since Suspend abruptly interrupts the execution state of the thread. **Second**, thread suspension in a distributed environment leads to an ineffective use of the infrastructure, where resources are shared among different tenants and applications.

Sleep operation is not supported. Therefore, there is no need to support the Interrupt operation, which forcibly resumes the thread from a waiting or a sleeping state.

2 Thread life cycle

Aneka Thread life cycle is different from the life cycle of local threads. It is not possible to directly map the state values of a local thread to Aneka threads. **Figure 6.6** provides a comparative view of the two life cycles. The white balloons in the figure indicate states that do not have a corresponding mapping on the other life cycle; the shaded balloons indicate the common states.

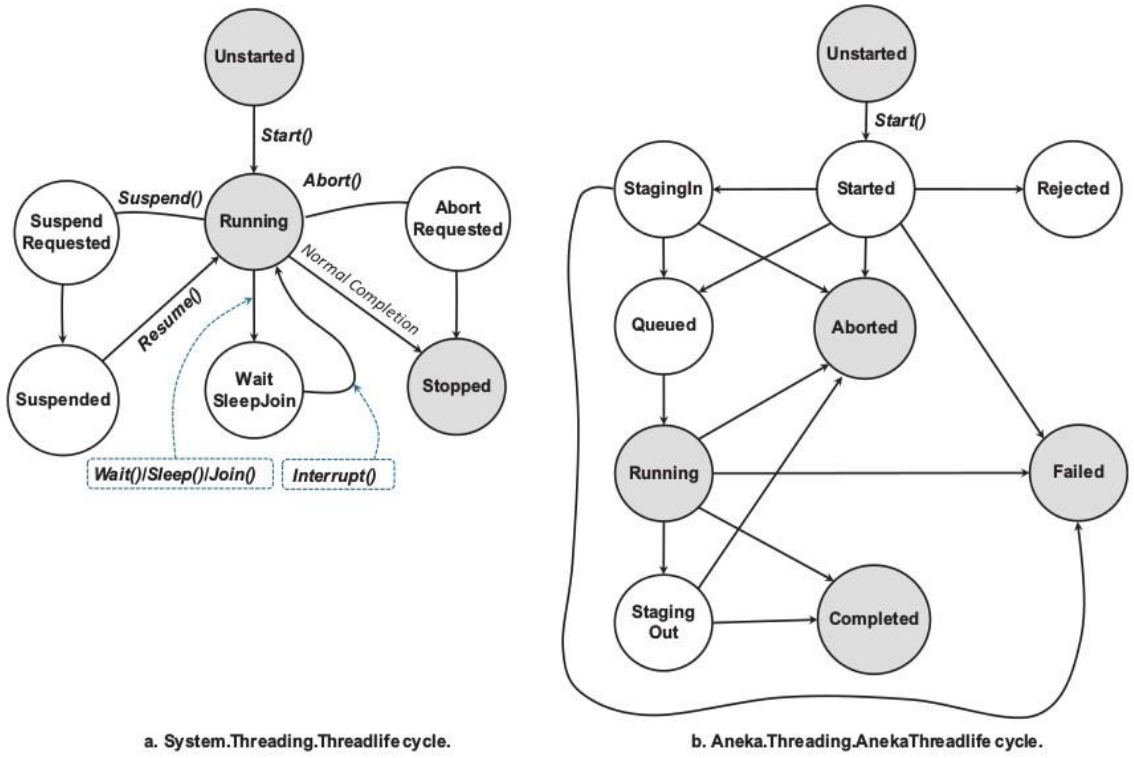


FIGURE 6.6

Thread life-cycle comparison.

In local threads most of the state transitions are controlled by the developer, who actually triggers the state transition by invoking methods. Whereas in Aneka threads, many of the state transitions are controlled by the middleware.

Aneka threads support file staging and they are scheduled by the middleware, which can queue them for a considerable amount of time.

An Aneka thread is initially found in the Unstarted state. Once the Start() method is called, the thread transits to the Started state, from which it is possible to move to the StagingIn state if there are files to upload for its execution or directly to the Queued state. If there is any error while uploading files, the thread fails and it ends its execution with the Failed state, which can also be reached for any exception that occurred while invoking Start().

Another outcome might be the Rejected state that occurs if the thread is started with an invalid reservation token. This is a final state and implies execution failure due to lack of rights.

Once the thread is in the queue, if there is a free node where to execute it, the middleware moves all the object data and depending files to the remote node and starts its execution, thus changing the state into Running.

If the thread generates an exception or does not produce the expected output files, the execution is considered failed and the final state of the thread is set to Failed. If the execution is successful, the final state is set to Completed. If there are output files to retrieve, the thread state is set to StagingOut while files are

collected and sent to their final destination.

At any point, if the developer stops the execution of the application or directly calls the Abort() method, the thread is aborted and its final state is set to Aborted.

3 Thread synchronization

The .NET base class libraries provide advanced facilities to support thread synchronization by the means of monitors, semaphores, reader-writer locks, and basic synchronization constructs at the language level. Aneka provides minimal support for thread synchronization that is limited to the implementation of the join operation for thread abstraction.

This requirement is less stringent in a distributed environment, where there is no shared memory among the thread instances and therefore it is not necessary.

Providing coordination facilities that introduce a locking strategy in such an environment might lead to distributed deadlocks that are hard to detect. Therefore, by design Aneka threads do not feature any synchronization facility except join operation.

4 Thread priorities

The System.Threading.Thread class supports thread priorities, where the scheduling priority can be one selected from one of the values of the ThreadPriority enumeration: Highest, AboveNormal, Normal, BelowNormal, or Lowest.

Aneka does not support thread priorities, the Aneka.Threading.Thread class exhibits a Priority property whose type is ThreadPriority, but its value is always set to Normal, and changes to it do not produce any effect on thread scheduling by the Aneka middleware.

5 Type serialization

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Main purpose is to save the state of an object to recreate it when needed. The reverse process is called deserialization.

Local threads execute all within the same address space and share memory; therefore, they do not need objects to be copied or transferred into a different address space. Aneka threads are distributed and execute on remote computing nodes, and this implies that the object code related to the method to be executed within a thread needs to be transferred over the network.

A .NET type is considered serializable if it is possible to convert an instance of the type into a binary array containing all the information required to revert it to its original form or into a possibly different execution context. This property is generally given for several types defined in the .NET framework by simply tagging the class definition with the Serializable attribute.

Aneka threads execute methods defined in serializable types, since it is necessary to move the enclosing

instance to remote execution method. In most cases, providing serialization is as easy as tagging the class definition with the Serializable attribute; in other cases it might be necessary to implement the ISerializable interface and provide appropriate constructors for the type.

6.4 Programming applications with Aneka threads

To show how it is possible to quickly port multithreaded application to Aneka threads, we provide a distributed implementation of the previously discussed examples for local threads.

6.4.1 Aneka threads application model

TheThread Programming Model is a programming model in which the programmer creates the units of work as Aneka threads. Therefore, it is necessary to utilize the AnekaApplicatioN<W,M> class, which is the application reference class for all the programming models.

The Aneka APIs support different programming models through template specialization. Hence, to develop distributed applications with Aneka threads, it is necessary to specialize the template type as follows:

AnekaApplication<AnekaThread, ThreadManager>

These two types are defined in the **Aneka.Threading** namespace noted in the Aneka. **Threading.dll** library of the Aneka SDK.

Another important component of the application model is the **Configuration** class, which is defined in the **Aneka.Entity** namespace (**Aneka.dll**). This class contains a set of properties that allow the application class to configure its interaction with the middleware, such as the address of the Aneka index service, which constitutes the main entry point of Aneka Clouds.

Data Intensive Computing: Map-Reduce Programming

•Focuses on class of applications that deal with a large amount of data.

- Cmputational science to social networking, produce large volumes of data that need to be efficiently stored, made accessible, indexed, and analyzed.
- Distributed computing is definitely of help in addressing these challenges .
- This chapter characterizes nature of data-intensive computing and presents an overview of the challenges of large volumes of data and how they are handled.
- MapReduce.

8.1 What is data-intensive computing?

•Concerned with production, manipulation, and analysis of large-scale data in MB to PB or beyond.

•**Dataset** is used to identify a collection of information that is relevant to one or more applications.

•Datasets are maintained in repositories.

•**Metadata** are attached to datasets.

• Data-intensive computations occur in many application domains.

•**Computational science** - People conducting scientific simulations and experiments produce, analyze, and process huge volumes of data.

•**Bioinformatics** applications mine databases that may end up containing terabytes of data.

•**Earthquake simulators** process a massive amount of data, which is produced as a result of recording the vibrations of the Earth across the entire globe.

8.1 What is data-intensive computing?

8.1.1 Characterizing data-intensive computations

8.1.2 Challenges ahead

8.1.3 Historical perspective

1 The early age: high-speed wide-area networking

2 Data grids

3 Data clouds and “Big Data”

4 Databases and data-intensive computing

8.1 What is data-intensive computing?

8.1.1 Characterizing data-intensive computations

- Deals with huge volumes of data, also exhibit compute-intensive properties.
- Handle datasets on the scale of multiple terabytes and petabytes.
- Applications process data in multistep analytical pipelines, including transformation and fusion stages.

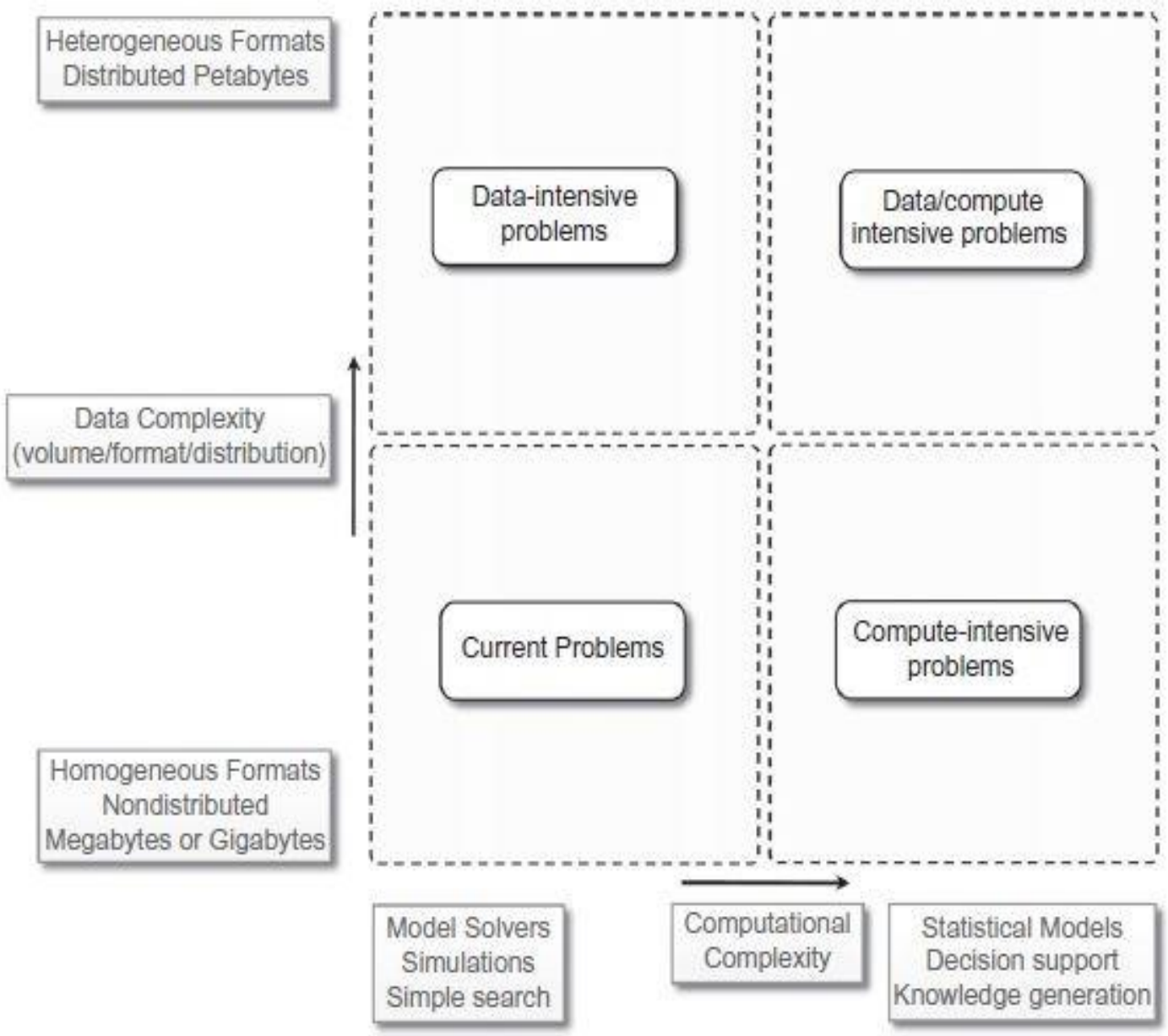


FIGURE 8.1

Data-intensive research issues.

8.1 What is data-intensive computing?

8.1.2 Challenges ahead

- Huge amount of data produced, analyzed, or stored imposes requirements on infrastructures and middleware that are least found in the traditional solutions.
- Moving terabytes of data.
- **Data partitioning, content replication** and **scalable algorithms** help in improving performance.

8.1 What is data-intensive computing?

8.1.2 Challenges ahead

Open challenges in data-intensive computing given by Ian Gorton et al.

1. Scalable algorithms that can search and process massive datasets.
2. New metadata management technologies that can handle complex, heterogeneous, and distributed data sources.
3. Advances in high-performance computing platforms for accessing in-memory multi terabyte data structures.
4. High-performance, highly reliable, petascale distributed file systems.
5. Data signature-generation techniques for data reduction and rapid processing.
6. Software mobility to move computation where data is present.
7. Interconnected architectures that provide support for filtering multi gigabyte datastreams.
8. Software integration techniques that facilitate the combination of software modules running on different platforms.

8.1 What is data-intensive computing?

8.1.3 Historical perspective

1 The early age: high-speed wide-area networking

- In 1989, the first experiments in high-speed networking as a support for remote visualization of scientific data.
- Two years later, TCP/IP-based distributed applications was demonstrated at Supercomputing 1991 (SC91).
- **Wide Area Large Data Object (WALDO)** system:
  - auto- matic generation of metadata;
  - automatic cataloguing of data and metadata;
  - facilitation of cooperative research by providing local and remote users access to data;
- mechanisms to incorporate data into databases and other documents.
- **Distributed Parallel Storage System (DPSS)** to support TerraVision, a terrain visualization application that lets users explore and navigate a tridimensional real landscape.
- **Clipper project:** a collection of independent, architecturally consistent service components to support data-intensive computing.
- Clipper's main focus to develop collection of services for applications to build **on-demand, large-scale, high-performance, wide-area problem-solving**

8.1 What is data-intensive computing?

environments.

8.1 What is data-intensive computing?

8.1.3 Historical perspective

2 Data grids

A data grid provides services that help users discover, transfer, and manipulate large datasets stored in distributed repositories as well as create and manage copies of them.

two main functionalities: high-performance and reliable file transfer.

Data grids mostly provide storage and dataset management facilities as support for scientific experiments

Heterogeneity and security.

Data grids have their own characteristics and introduce new challenges:

1. **Massive datasets.**
2. **Shared data collections.**
3. **Unified namespace.**
4. **Access restrictions.**

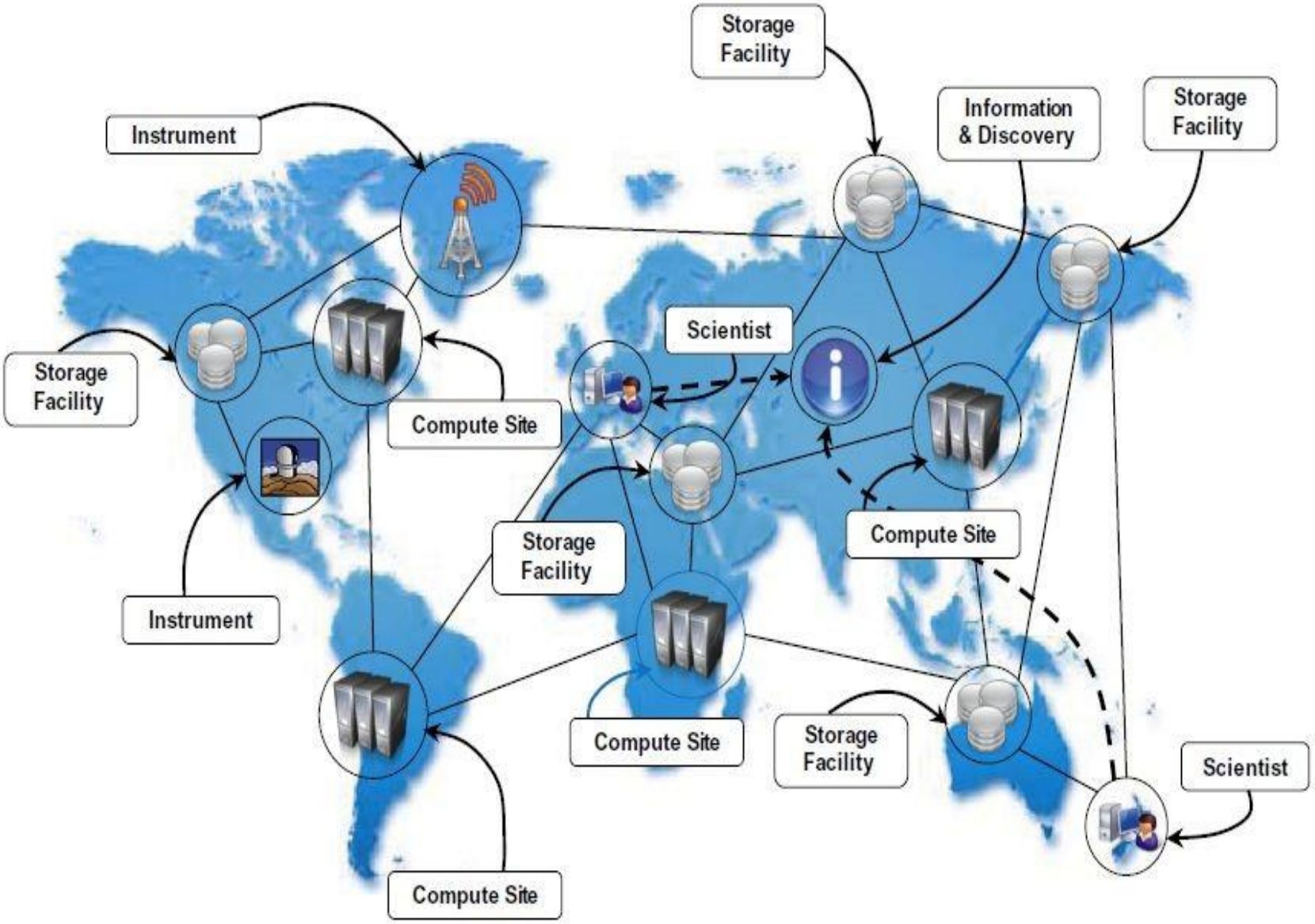


FIGURE 8.2

Data grid reference scenario.

8.1 What is data-intensive computing?

8.1.3 Historical perspective

3 Data clouds and "Big Data"

- Big Data - characterizes the nature of data- intensive computations and identifies datasets that grow large they become complex to work using DBMS tools.
- The term Big Data applies to datasets of which the size is beyond the ability of commonly used software tools to capture, manage, and process within a tolerable elapsed time.

Cloud technologies support data-intensive computing in several ways:

1. By providing a large amount of compute instances on demand, which can be used to process and analyze large datasets in parallel.
2. By providing a storage system optimized for keeping large blobs of data and other distributed data store architectures.
3. By providing frameworks and programming APIs optimized for the processing and management of large amounts of data.

8.1 What is data-intensive computing?

8.1.3 Historical perspective

3 Data clouds and "Big Data"

A datacloud is a combination of above 3 components.

Ex 1: MapReduce framework, which provides the best performance for leveraging Google File System on top of Google's large computing infrastructure.

Ex 2: Hadoop system, the most mature, large, and open-source datacloud. It consists of Hadoop Distributed File System (HDFS) and Hadoop's implementation of MapReduce.

Ex 3: Sector, which consists of Sector Distributed File System (SDFS) and compute service called Sphere, that allows users to execute arbitrary user-defined functions (UDFs) over the data managed by SDFS.

8.1 What is data-intensive computing?

8.1.3 Historical perspective

4. Databases and data-intensive computing

- Traditional distributed database using DBMS less computation power.
- Distributed databases are a collection of data stored at different sites of a computer network.
- A distributed database can be created by splitting and scattering the data of an existing database over different sites.
- These systems are very robust and provide distributed transaction processing, distributed query optimization, and efficient management of resources.

8.2 Technologies for data-intensive computing

Concerns the development of applications that are mainly focused on processing large quantities of data.

8.2.1 Storage systems

1. High-performance distributed file systems and storage clouds
2. NoSQL systems

8.2.2 Programming platforms

1. The MapReduce programming model.
2. Variations and extensions of MapReduce.
3. Alternatives to MapReduce.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

- DBMS constituted **de facto** storage support.
- Due to the explosion of unstructured data - blogs, Web pages, software logs, and sensor readings, **the relational model** is not preferred solution.

Some factors contributing to change in data are:

1. Growing of popularity of Big Data.
2. Growing importance of data analytics in the business chain.
3. Presence of data in several forms, not only structured.
4. New approaches and technologies for computing.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

1. **High-performance distributed file systems and storage clouds.**

- Distributed file systems constitute primary support for data management.
- They provide interface to store information in the form of files
- later access them for read and write operations.

a. Lustre.

- massively parallel distributed file system
- covers small workgroup of clusters to a large-scale computing cluster.
- The file system is used by several of the Top 500 supercomputing systems.
- Designed to provide access to petabytes (PBs) of storage to serve thousands of clients with an I/O throughput of hundreds of gigabytes per second (GB/s).
- The system is composed of **metadata server** that contains the metadata about the file system and collection of **object storage servers** that are in charge of providing storage.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

1. **High-performance distributed file systems and storage clouds.**

b. IBM General Parallel File System (GPFS).

- GPFS is high-performance distributed file system.
- provides support for RS/6000 supercomputer and Linux computing clusters.
- Multiplatform distributed file system.
- Concept of shared disks, in which a collection of disks is attached to the file system nodes by switching fabric.
- The file system makes this infrastructure transparent to users and stripes large files over the disk array by replicating portions of the file to ensure high availability.
- Eliminates a single point of failure.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

1. **High-performance distributed file systems and storage clouds.**

c. Google File System (GFS)

storage infrastructure supports execution of distributed applications in Google's computing cloud.

GFS is designed with the following assumptions:

1. System is built on top of commodity hardware that often fails.
2. System stores large files; multi-GB files are common and should be treated efficiently, and small files must be supported.
3. Workloads has two kinds of reads: large streaming reads and small random reads.
4. Workloads also have large, sequential writes that append data to files.
5. High-sustained bandwidth is more important than low latency.

Architecture organized into a **single master**, which contains the metadata of file system, and collection of **chunk servers**, which provide storage space..

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

1. **High-performance distributed file systems and storage clouds.**

d. Sector.

Storage cloud that supports execution of data-intensive applications defined according to **Sphere framework.**

Architecture is composed of four nodes:

- security server,
- master nodes,
- slave nodes, and
- client machines.

Security server maintains all the information about access control policies for user and files,
Master servers coordinate and serve the I/O requests of **clients**,
which interact with **Slave nodes** to access files.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

1. **High-performance distributed file systems and storage clouds.**

e. Amazon Simple Storage Service (S3).

- Amazon S3 is the online storage service provided by Amazon.
- System offers flat storage space organized into buckets, which are attached to an Amazon Web Services (AWS) account.
- Each bucket can store multiple objects, identified by a unique key.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

- **Not Only SQL (NoSQL)** identify set of UNIX shell scripts and commands to operate on text files containing the actual data.
- NoSQL cannot be considered a relational database, it is a collection of scripts that allow users to manage database tasks by using text files as information stores.

Some prominent implementations that support data-intensive applications:

a. Apache CouchDB and MongoDB.

b. Amazon Dynamo.

c. Google Bigtable.

d. Apache Cassandra.

e. Hadoop HBase.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

a. Apache CouchDB and MongoDB.

- Both provide a schema-less store whereby the primary objects are documents organized into a collection of key-value fields.
- The value of each field can be of type string, integer, float, date, or an array of values.
- RESTful interface and represent data in JSON format.
- querying and indexing data by using the MapReduce programming model.
- JavaScript as a base language.
- support large files as documents.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

b. Amazon Dynamo.

- provide an incrementally scalable and highly available storage system.
- provides a simplified interface based on get/put semantics, where objects are stored and retrieved with a unique identifier (key).

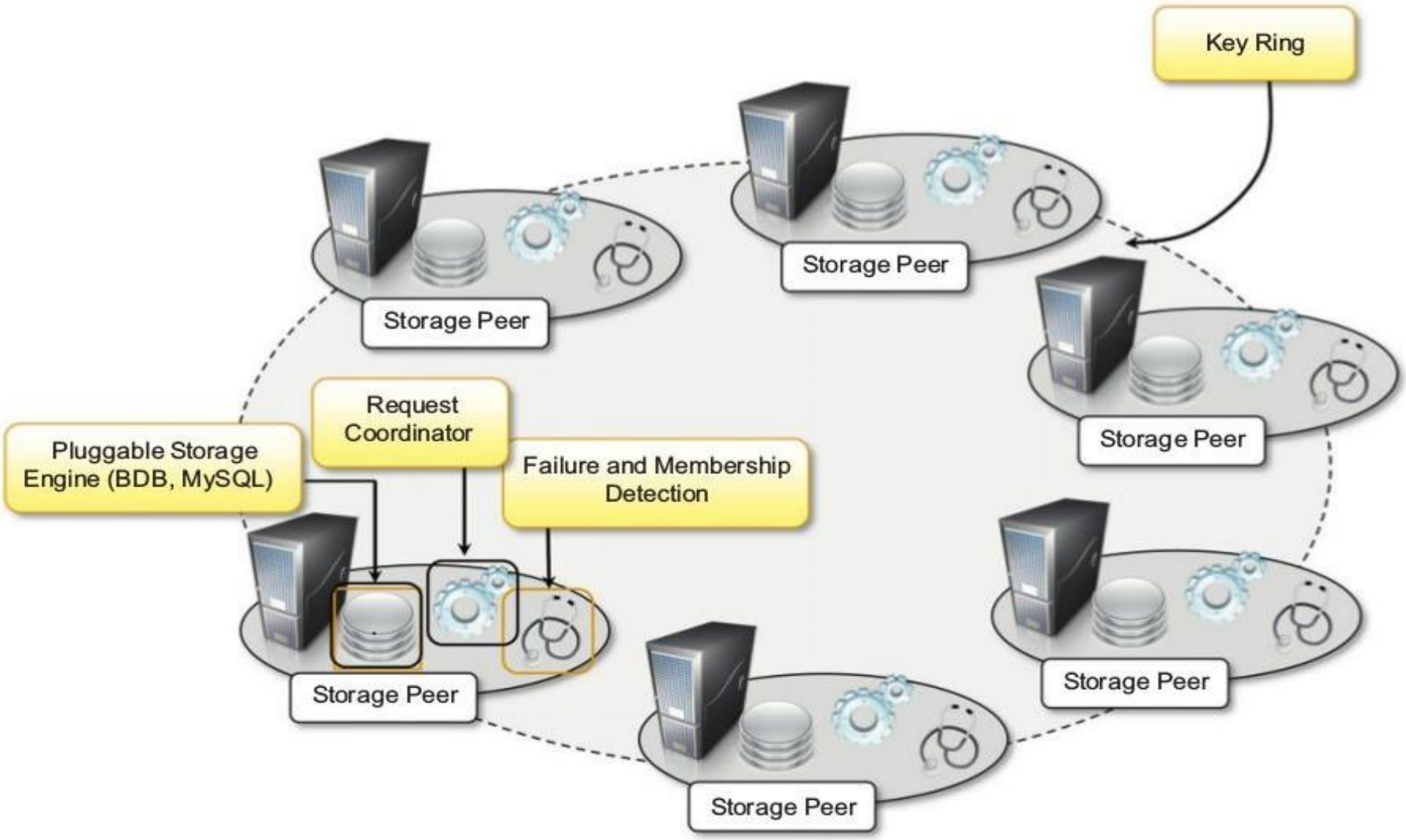


FIGURE 8.3

Amazon Dynamo architecture.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

c. Google Bigtable.

- provides storage support: from **throughput-oriented batch-processing jobs** to **latency-sensitive serving** of data.
- design goals: wide applicability, scalability, high performance, and high availability.
- Bigtable organizes the data storage in tables of which the rows are distributed over the distributed file system supporting the middleware, i.e Google File System.
- A table is organized into rows and columns;
- columns can be grouped in column family, which allow for specific optimization for better access control, storage and indexing of data.

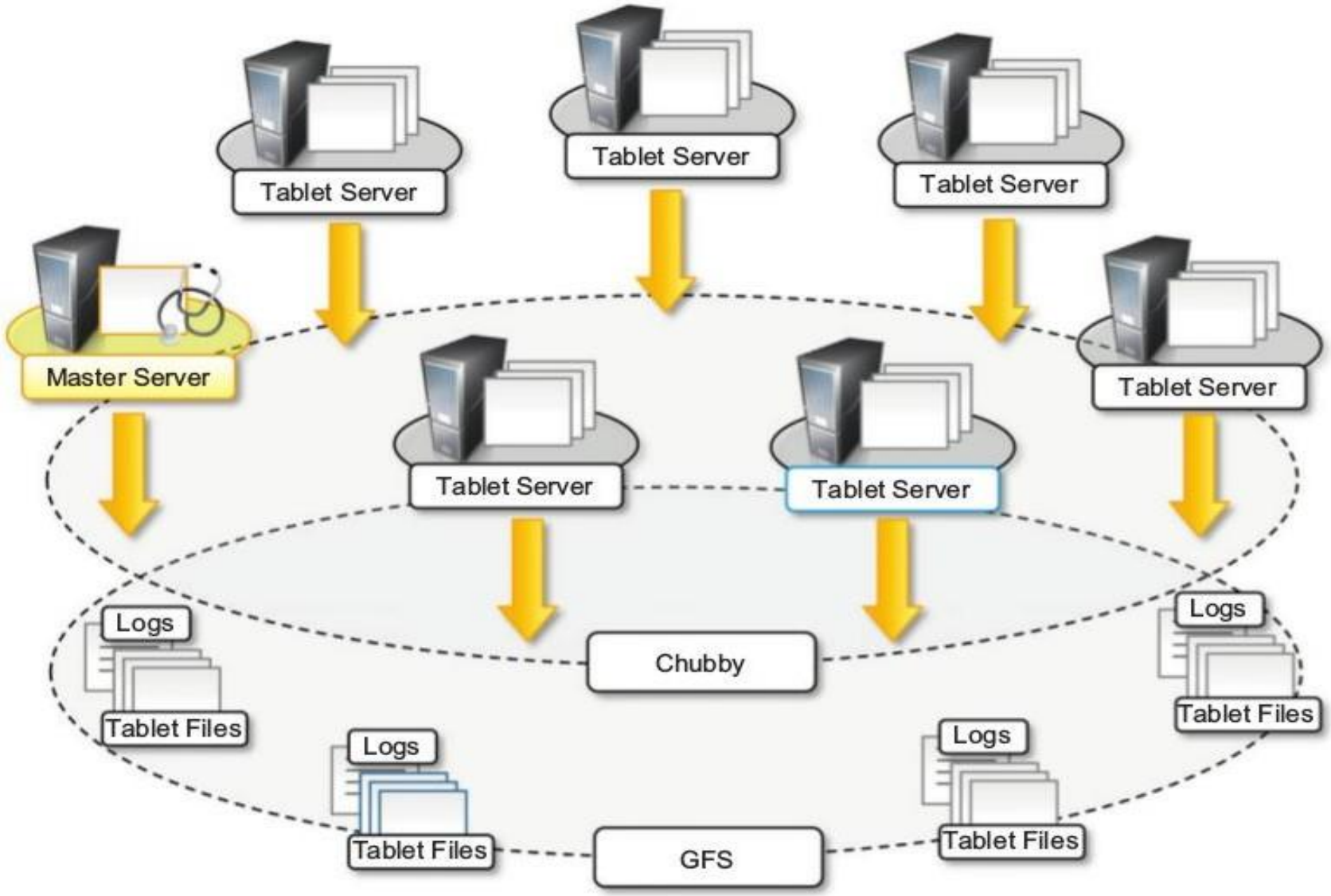


FIGURE 8.4

Bigtable architecture.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

d. Apache Cassandra.

- designed to avoid a single point of failure and offer highly reliable service.
- developed by Facebook; now it is part of Apache incubator initiative.
- provides storage support for Web applications such as Facebook, Digg, Twitter.

- concept: tables implemented as a distributed multidimensional map indexed by a key.
- value corresponding to key is structured object and constitutes the row of a table.
- Cassandra organizes the row of a table into columns, and sets of columns can be grouped into column families.
- APIs provided by the system to access and manipulate data: insertion, retrieval, and deletion.
- The insertion is performed at the row level; retrieval and deletion operate at

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

column level.

8.2 Technologies for data-intensive computing

8.2.1 Storage systems

2. NoSQL systems

e. Hadoop HBase.

- inspiration from Google Bigtable.
- offer real-time read/write operations for tables with billions of rows and millions of columns.
- architecture and logic model of HBase is very similar to Google Bigtable, and the entire system is backed by the Hadoop Distributed File System (HDFS).

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

focus on the processing of data and move into the runtime system the management of transfers, thus making the data always available where needed.

approach followed by the MapReduce programming platform.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

1. The MapReduce programming model.

- the computational logic of an application in two simple functions: map and reduce.
- Data transfer and management are completely handled by the distributed storage infrastructure (i.e., the Google File System)

$$map\ (k1, v1) \rightarrow list(k2, v2)$$
$$reduce(k2, list(v2)) \rightarrow list(v2)$$

- map function reads a key-value pair and produces a list of key-value pairs of different types.
- reduce function reads a pair composed of a key and a list of values and produces a list of values of the same type.
- (k1,v1,k2,kv2) hints how these two functions are connected and are executed.

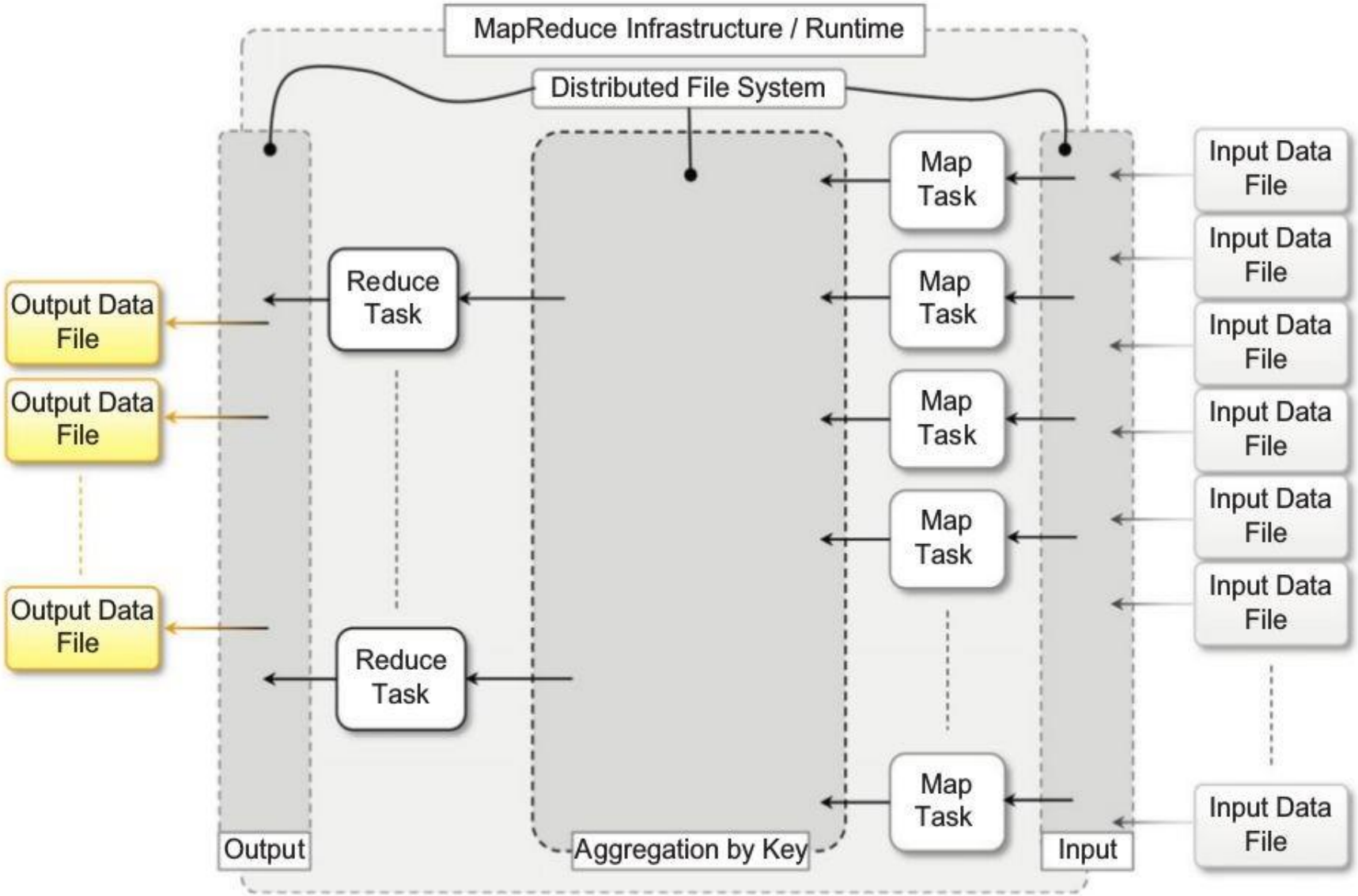


FIGURE 8.5

MapReduce computation workflow.

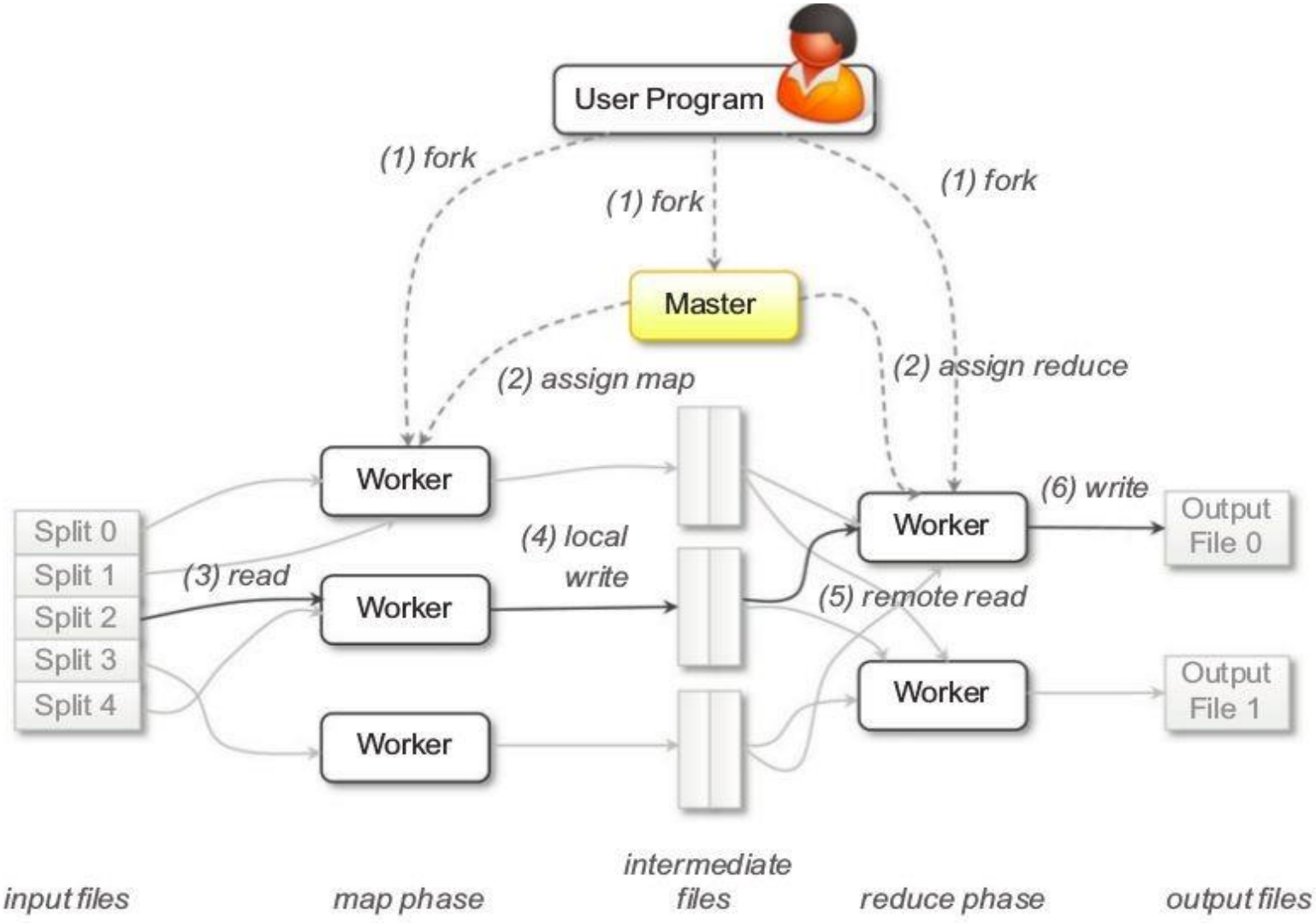


FIGURE 8.6

Google MapReduce infrastructure overview.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

1. The MapReduce programming model.

- **The master process** is in charge of controlling the execution of map and reduce tasks, partitioning, and reorganizing the intermediate output
- **The worker processes** are used to host the execution of map and reduce tasks and provide basic I/O facilities

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

2. Variations and extensions of MapReduce.

A. **Hadoop.**
B. **Pig.**
C. **Hive.**
D. **Map-Reduce-Merge.**
E. **Twister.**

A. Hadoop.

- collection of software projects for reliable and scalable distributed computing.
- Hadoop Distributed File System (HDFS) and Hadoop MapReduce.
- Same as Google File System & Google MapReduce.

B. Pig.

- analysis of large datasets.
- Pig consists of a high-level language for data analysis programs, coupled with infrastructure.
- compiler for high-level language that produces sequence of MapReduce jobs.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

2. Variations and extensions of MapReduce.

C. Hive.

- data warehouse infrastructure on top of Hadoop MapReduce.
- tools for easy data summarization, ad hoc queries, and analysis of large datasets
- major advantages is ability to scale out, since it is based on the Hadoop framework.

D. Map-Reduce-Merge.

- a third phase — Merge phase—that allows efficiently merging data already partitioned and sorted.
- simplifies the management of heterogeneous related datasets.
- provides abstraction to express the common relational algebra operators & several join algorithms.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

2. Variations and extensions of MapReduce.

E. Twister.

- iterative executions of MapReduce jobs.
- Twister proposes the following extensions:
    1. Configure Map
    2. Configure Reduce
    3. While Condition Holds True Do
        a. Run MapReduce
        b. Apply Combine Operation to Result
        c. Update Condition
    4. Close

- additional phase called combine, run at the end of MapReduce job, that aggregates output together.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

3. Alternatives to MapReduce.

a. Sphere.
b. All-Pairs.
c. DryadLINQ.

a. Sphere.

- Sector Distributed File System (SDFS).
- implements the **stream processing model** (**Single Program, Multiple Data**)
- developers to express the computation in terms of **user-defined functions (UDFs)**
- built on top of Sector's API.
- execution of UDFs is achieved through **Sphere Process Engines (SPEs)**,

- Sphere **client** sends a request for processing to the **master** node,
- returns the list of available slaves,
- client will choose the slaves on which to execute Sphere processes.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

3. Alternatives to MapReduce.

b. All-Pairs.

implements All-pairs function

All-pairs(A:set; B:set; F:function) -> M:matrix

The All-pairs function can be easily solved by the following algorithm:

1. For each $i in A
2. For each $j in B
3. Submit job F $i $j

Execution of a distributed application is developed in four stages:

(1) model the system;

(2) distribute the data;

(3) dispatch batch jobs; and

(4) clean up the system.

Ex1: field of biometrics, similarity matrices are composed by comparison of several images.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

3. Alternatives to MapReduce.

Ex2: applications and algorithms in data mining.

8.2 Technologies for data-intensive computing

8.2.2 Programming platforms

3. Alternatives to MapReduce.

c. DryadLINQ.

- Microsoft Research project
- investigates programming models for writing parallel and distributed programs to scale from a small cluster to a large datacenter.
- Developers express distributed applications as set of sequential programs connected by channels.
- Computation is expressed as directed acyclic graph - nodes are sequential programs and vertices are channels connecting programs.
- Superset of the MapReduce model.

8.3 Aneka MapReduce programming

Aneka provides an implementation of MapReduce introduced by **Google** and implemented by **Hadoop**.

8.3.1 Introducing the MapReduce programming model

1 Programming abstractions
2 Runtime support
3 Distributed file system support

8.3.2 Example application

1 Parsing Aneka logs
2 Mapper design and implementation
3 Reducer design and implementation
4 Driver program
5 Running the application

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

defines the abstractions and runtime support for developing MapReduce applications on top of Aneka.

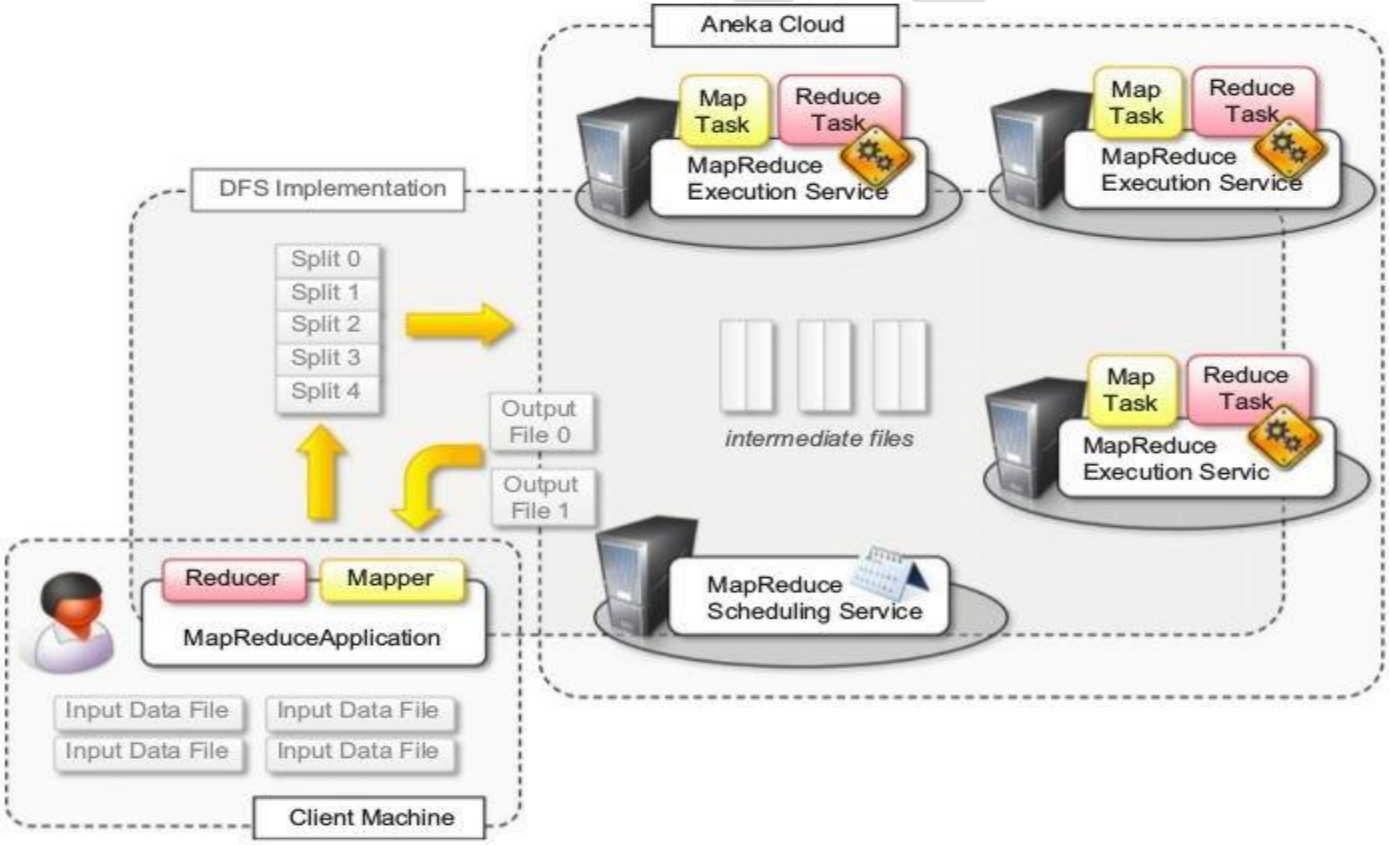


FIGURE 8.7

Aneka MapReduce infrastructure.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

The runtime support is composed of **three** main elements:

1. **MapReduce Scheduling Service**, which plays the role of the master process in the Google and Hadoop implementation.
2. **MapReduce Execution Service**, which plays the role of the worker process in the Google and Hadoop implementation.
3. A specialized **distributed file system** that is used to move data files.

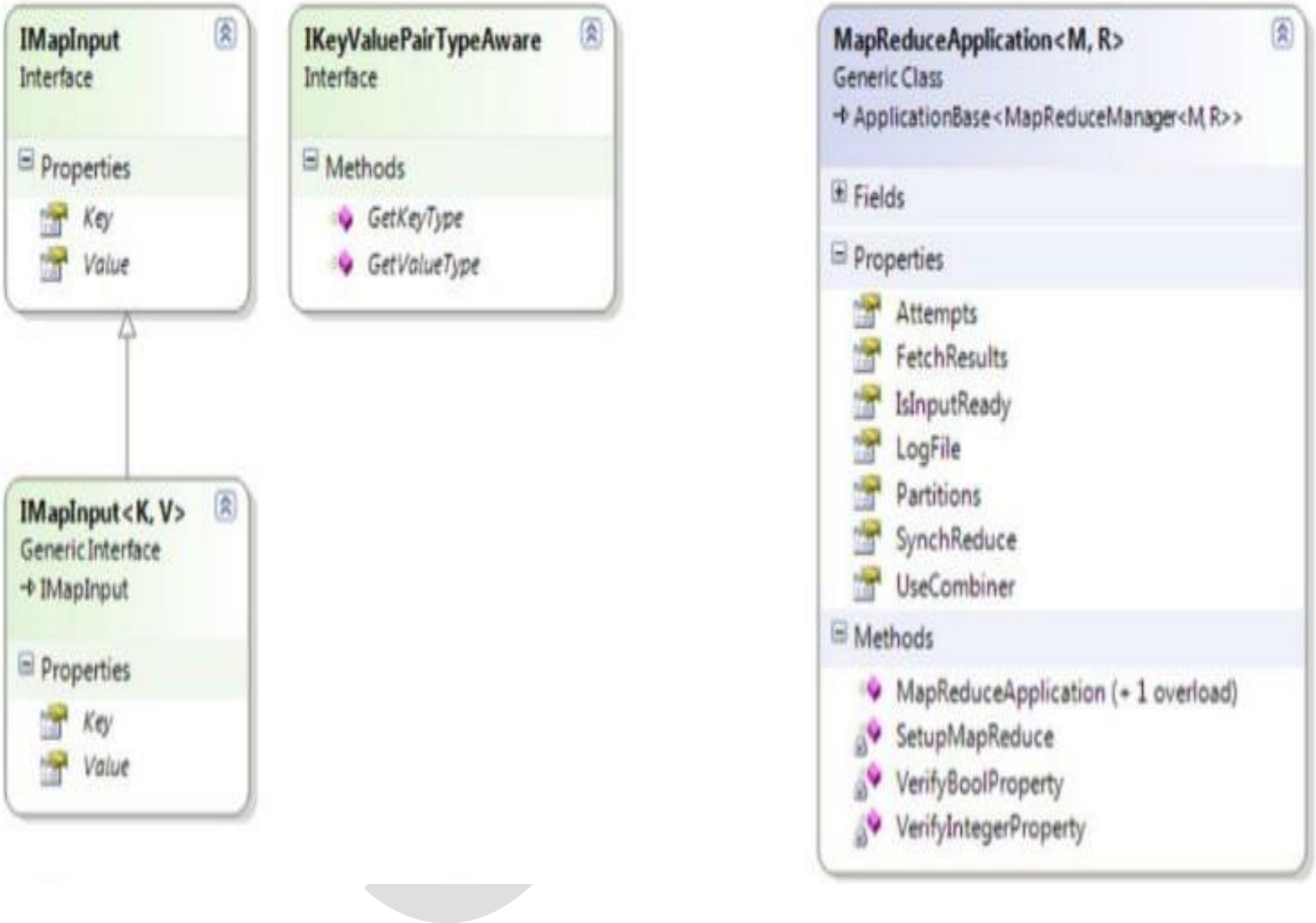
- Client components, MapReduce Application, are used to submit the execution of a MapReduce job, upload data files, and monitor it.
- Local data files are automatically uploaded to Aneka, and output files are automatically downloaded to client machine.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

1 Programming abstractions

- Aneka executes any piece of user code within distributed application.
- task creation is responsibility of the infrastructure once the user has defined the map and reduce functions.


IMapInput
Interface
Properties
Key
Value
IKeyValuePairTypeAware
Interface
Methods
GetKeyType
GetValueType
IMapInput<K, V>
Generic Interface
IMapInput
Properties
Key
Value
MapReduceApplication<M, R>
Generic Class
ApplicationBase<MapReduceManager<M,R>>
Fields
Properties
Attempts
FetchResults
IsInputReady
LogFile
Partitions
SynchReduce
UseCombiner
Methods
MapReduceApplication (+ 1 overload)
SetupMapReduce
VerifyBoolProperty
VerifyIntegerProperty

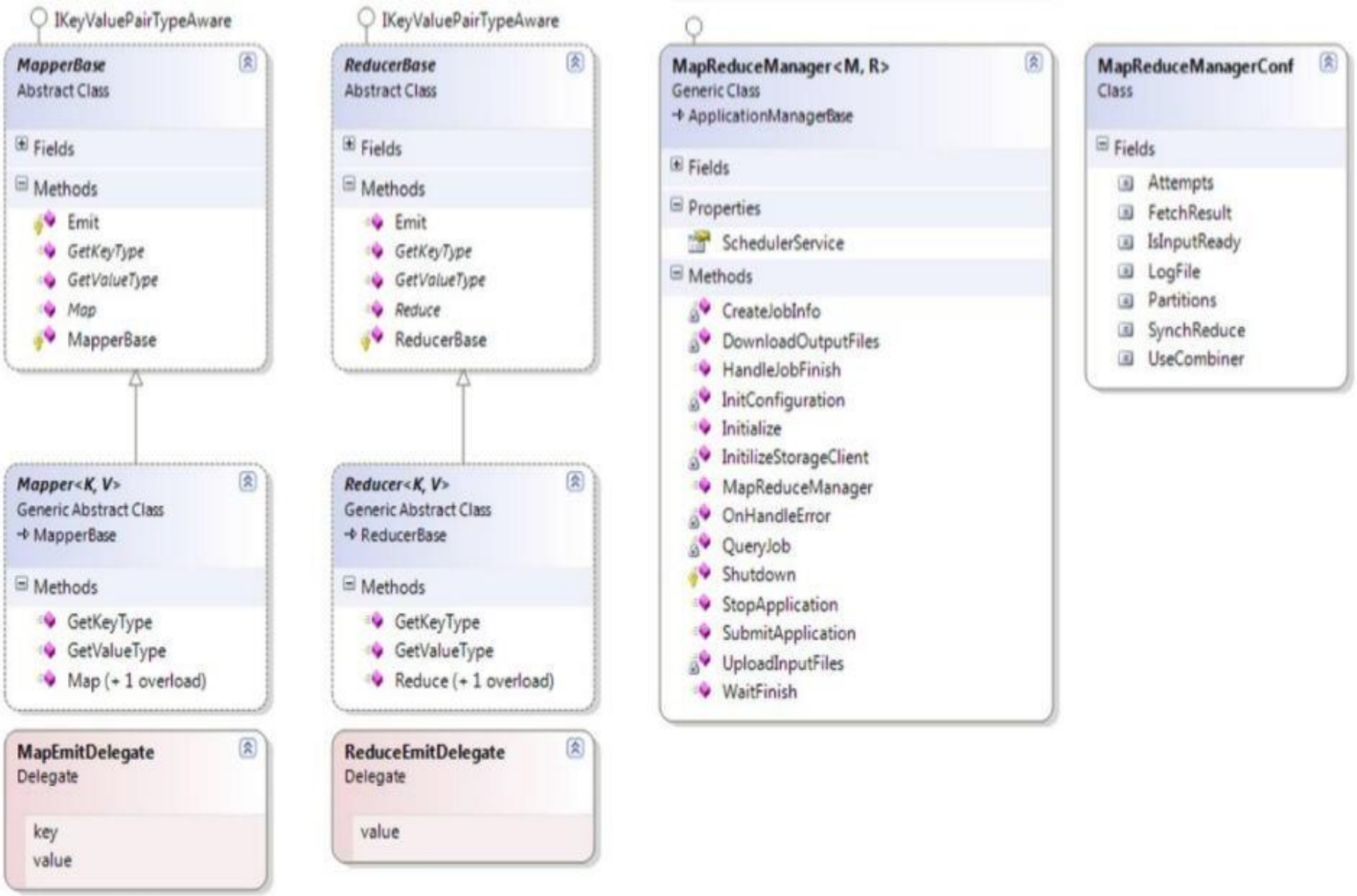


FIGURE 8.8

MapReduce Abstractions Object Model.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

1 Programming abstractions

Listing 8.1 definition of the Mapper<K,V> class and of the related types that developers should be aware of for implementing the map function.
Listing 8.2 implementation of the Mapper<K,V> component for Word Counter sample.
Listing 8.3 definition of Reducer<K,V> class. The implementation of a specific reducer requires specializing the generic class and overriding the abstract method: Reduce(IReduceInputEnumerator<V> input).
Listing 8.4 implement the reducer function for word-counter example.
Listing 8.5 interface of MapReduceApplication<M,R>.
Listing 8.6 displays collection of methods that are of interest in this class for execution of MapReduce jobs.
Listing 8.7 MapReduce application for running the word-counter example defined by the previous WordCounterMapper and WordCounterReducer classes.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

2 Runtime support

comprises the collection of services that deal with scheduling and executing MapReduce tasks.

These are

the MapReduce Scheduling Service and

the MapReduce Execution Service.

Job and Task Scheduling. responsibility of the MapReduce Scheduling Service.

covers the same role as the master process in the Google MapReduce implementation.

The architecture of the Scheduling Service is organized into two major components:

the MapReduceSchedulerService and

the MapReduceScheduler.

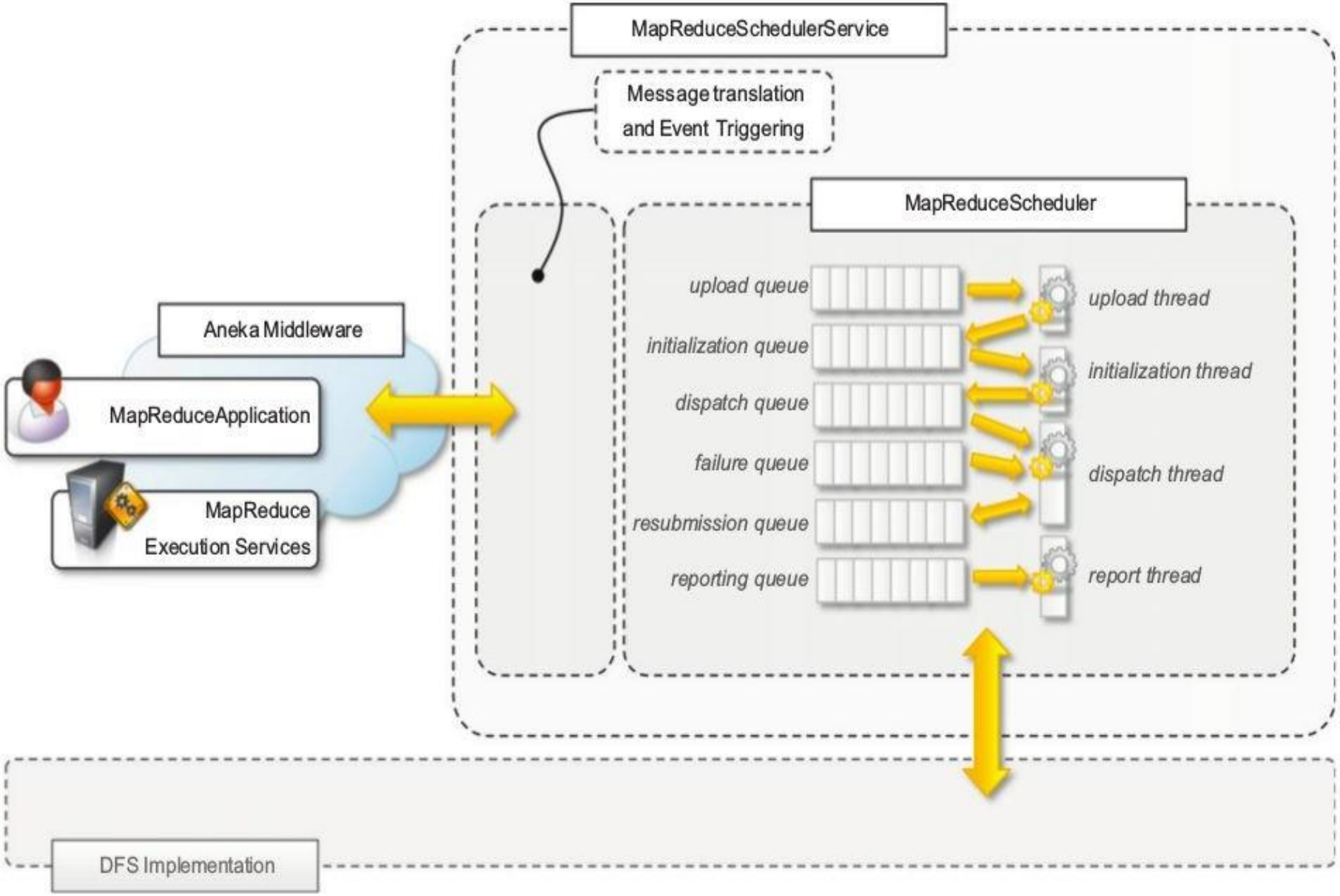


FIGURE 8.9

MapReduce Scheduling Service architecture.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

2 Runtime support

Task Execution.

Controlled by MapReduce Execution Service.

This component plays role of the worker process in the Google MapReduce implementation.

There are three major components that coordinate together for executing tasks:

1. MapReduce- SchedulerService,
2. ExecutorManager, and
3. MapReduceExecutor.

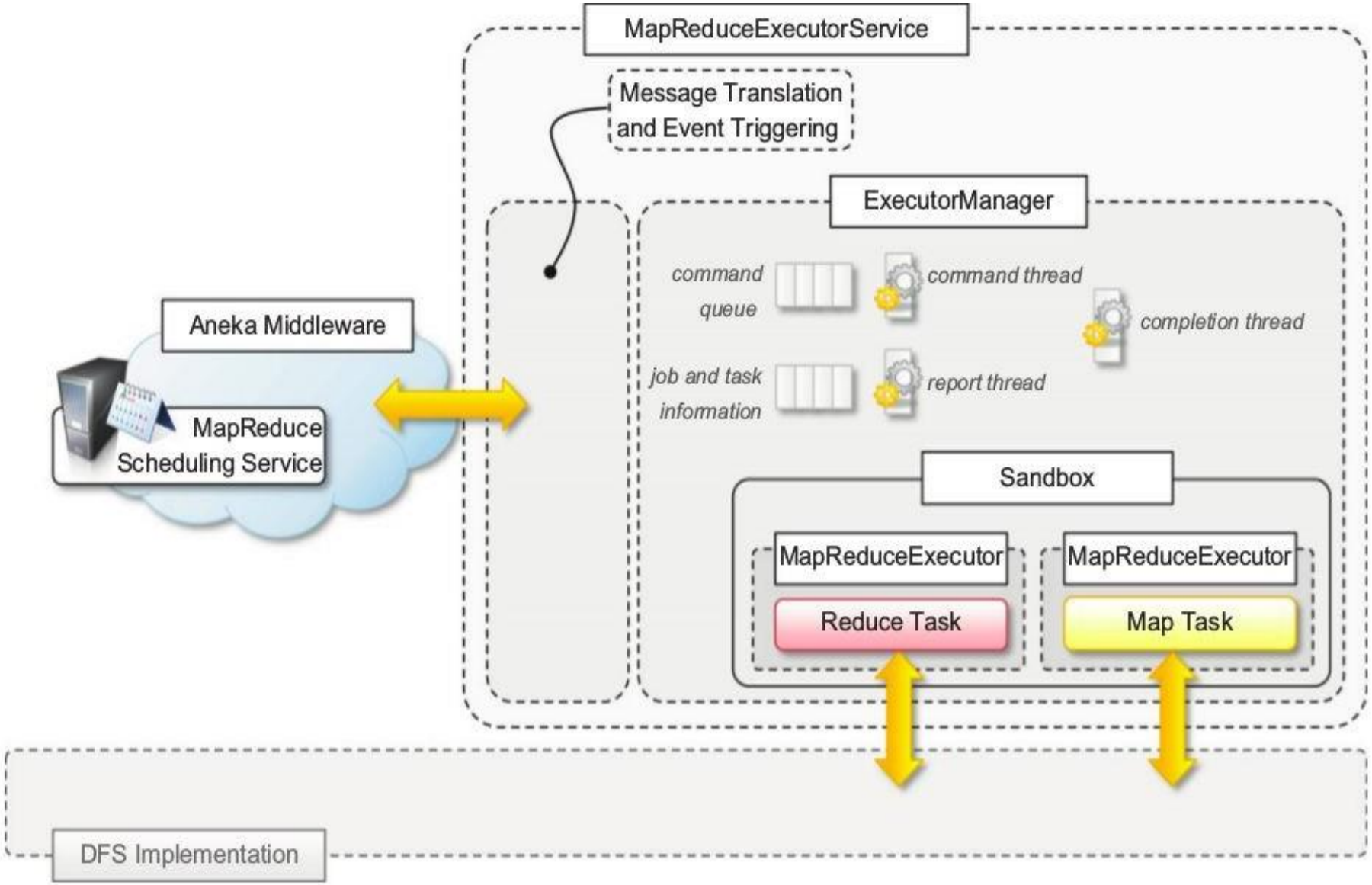


FIGURE 8.10

MapReduce Execution Service architecture.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

3 Distributed file system support

Aneka supports, the MapReduce model.

guarantee high availability and better efficiency by means of replication and distribution.

interfacing with different storage implementations and it maintains the same flexibility for the integration.

The level of integration requires to perform the following tasks:

- Retrieving the location of files and file chunks
- Accessing a file by means of a stream

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

3 Distributed file system support

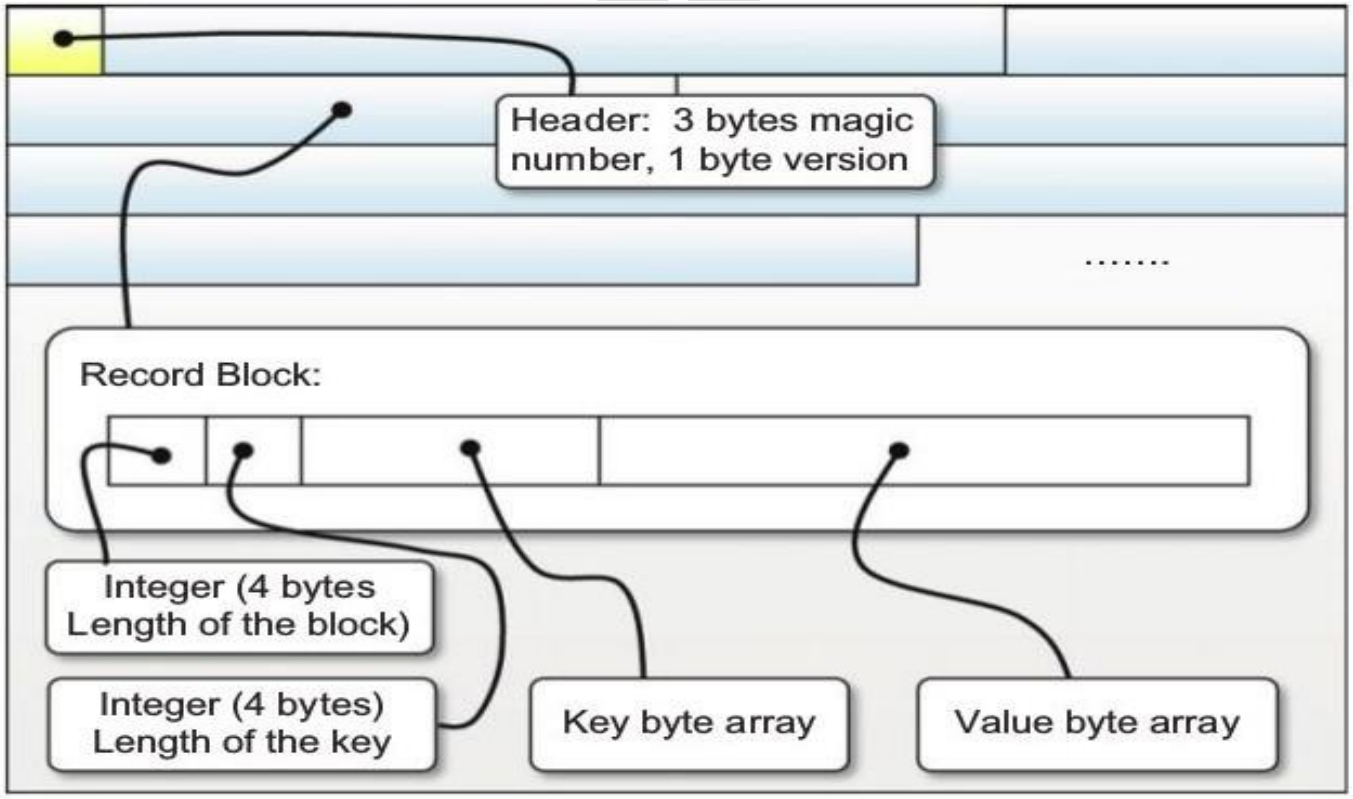


FIGURE 8.11

Aneka MapReduce data file format.

8.3 Aneka MapReduce programming

8.3.1 Introducing the MapReduce programming model

3 Distributed file system support

Listing 8.8 shows the interface of the SeqReader and SeqWriter classes.

Listing 8.9 shows a practical use of the SeqReader class by implementing the callback used in the word-counter example.

8.3 Aneka MapReduce programming

8.3.2 Example application

1 Parsing Aneka logs
2 Mapper design and implementation
3 Reducer design and implementation
4 Driver program
5 Running the application

8.3 Aneka MapReduce programming

1 Parsing Aneka logs

Aneka components (daemons, container instances, and services) produce a lot of information that is stored in the form of log files.

The entire framework leverages the log4net library for collecting and storing the log information.

Some examples of formatted log messages are:

15 Mar 2011 10:30:07 DEBUGSchedulerService: . . .
HandleSubmitApplicationSchedulerService: . . .
15 Mar 2011 10:30:07 INFOSchedulerService: Scanning candidate storage . . .
15 Mar 2011 10:30:10 INFOAdded [WU: 51d55819-b211-490f-b185-8a25734ba705,
4e86fd02. . .
15 Mar 2011 10:30:10 DEBUGStorageService:NotifySchedulerSending
FileTransferMessage. . .
15 Mar 2011 10:30:10 DEBUGIndependentSchedulingService:QueueWorkUnitQueueing. . .
15 Mar 2011 10:30:10 INFOAlgorithmBase::AddTasks[64] Adding 1 Tasks
15 Mar 2011 10:30:10 DEBUGAlgorithmBase:FireProvisionResourcesProvision

8.3 Aneka MapReduce programming

1 Parsing Aneka logs

Possible information that we might want to extract from such logs is the following:

- The distribution of log messages according to the level
- The distribution of log messages according to the components

The structure of the map and reduce functions will be the following:

map: (long; string) => (string; long)
reduce: (long; string) => (string; long)

8.3 Aneka MapReduce programming

2 Mapper design and implementation

The operation performed by the map function is a very simple text extraction that identifies the level of the logging and the name of the component entering the information in the log.
Once this information is extracted, a key-value pair (string, long) is emitted by the function.

Listing 8.10 shows the implementation of the Mapper class for the log parsing task.

8.3 Aneka MapReduce programming

3 Reducer design and implementation

Add all the values that are associated to the same key and emit a key-value pair with the total sum.

Listing 8.11 , the operation to perform is very simple and actually is the same for both of the two different key-value pairs extracted from the log lines.

8.3 Aneka MapReduce programming

4 Driver program

LogParsingMapper and **LogParsingReducer** constitute the core functionality of the MapReduce job, which only requires to be properly configured in the main program in order to process and produce text tiles.

Listing 8.12 shows the implementation of the driver program. With respect to the previous examples, there are three
things to be noted:

- The configuration of the MapReduce job
- The post-processing of the result files
- The management of errors

8.3 Aneka MapReduce programming

5 Running the application

Aneka produces a considerable amount of logging information.

By continuing to run an Aneka Cloud for a few days, it is quite easy to collect enough data to mine for our sample application.

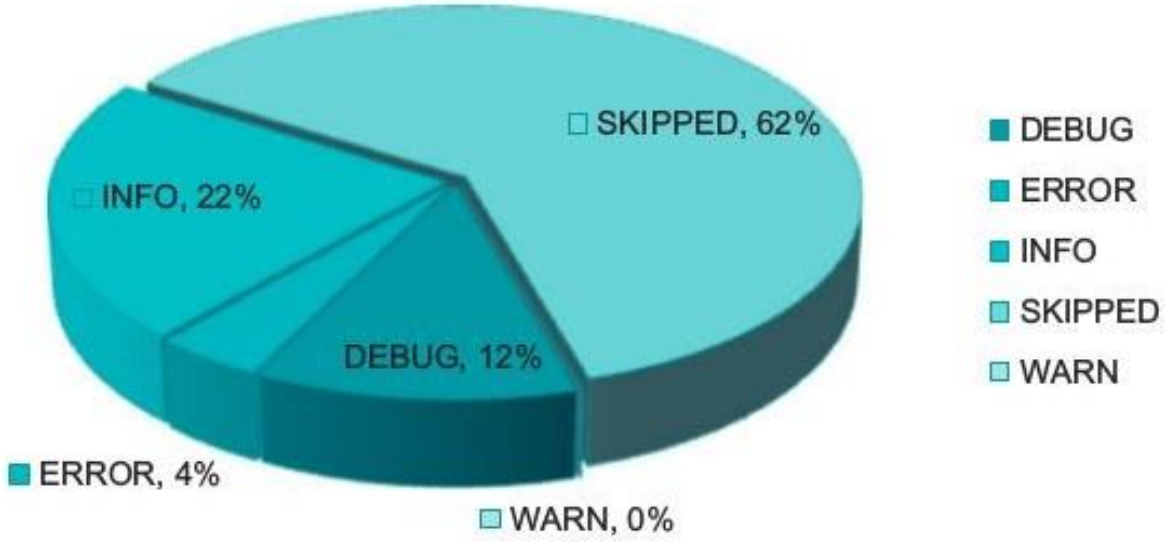


FIGURE 8.12

Log-level entries distribution.

8.

5 R

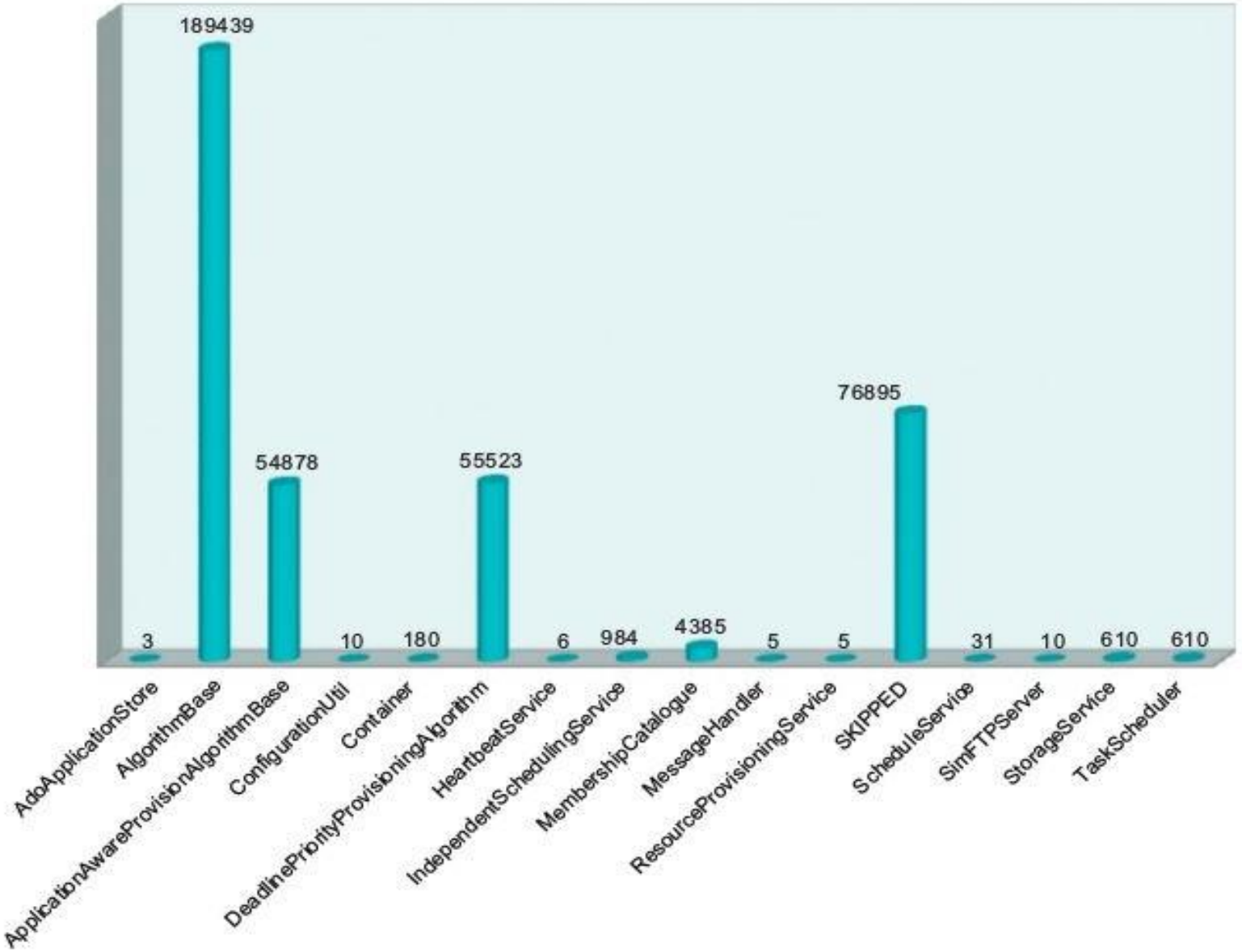


FIGURE 8.13

Component entries distribution.

Cloud Platforms in Industry

Overview

9.1 Amazon web services
- 9.1.1 Compute services
- 9.1.2 Storage services
- 9.1.3 Communication services
- 9.1.4 Additional services

9.2 Google AppEngine
- 9.2.1 Architecture and core concepts
- 9.2.2 Application life cycle
- 9.2.3 Cost model
- 9.2.4 Observations

9.3 Microsoft Azure

9.3.1 Azure core concepts
9.3.2 SQL Azure
9.3.3 Windows Azure platform appliance
9.3.4 Observations

9.1 Amazon web services

- Development of flexible applications by providing solutions for elastic infrastructure scalability, messaging, and data storage.
- Accessible through SOAP (Simple Object Access Protocol) or RESTful (Representational State Transfer) Web service interfaces
- Expenses computed on a pay-as-you-go basis.

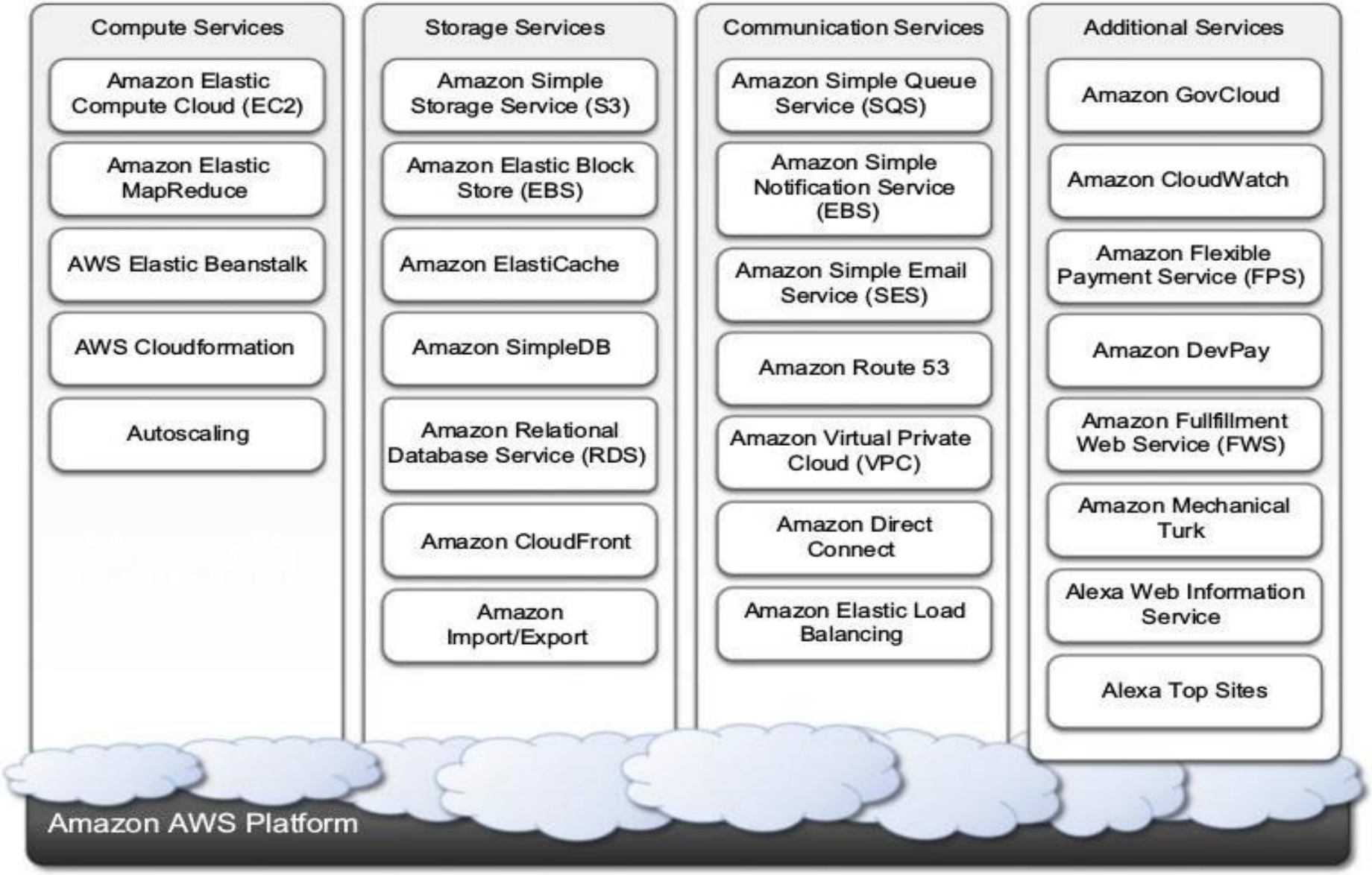


FIGURE 9.1

Amazon Web Services ecosystem.

9.1 Amazon web services

9.1.1 Compute services

- fundamental element of cloud computing systems.
- Amazon EC2, which delivers an IaaS solution.
- Amazon EC2 allows deploying servers in the form of virtual machine images.
- Images come with a preinstalled operating system and a software stack.

1 Amazon machine images

- Amazon Machine Images (AMIs) are templates from which it is possible to create a virtual machine.
- They are stored in Amazon S3.
- An AMI contains a physical file system layout with a predefined operating system installed.
- Predefined operating system installed as
  Amazon Ramdisk Image (ARI, id: ari-yyyyyy) and
  Amazon Kernel Image (AKI, id: aki-zzzzzz)

9.1 Amazon web services

9.1.1 Compute services

2 EC2 instances

EC2 instances represent virtual machines.
The processing power is expressed in terms of virtual cores and EC2 Compute Units (ECUs).

six major categories:

1. Standard instances.
2. Micro instances.
3. High-memory instances.
4. High-CPU instances.
5. Cluster Compute instances.
6. Cluster GPU instances.

9.1 Amazon web services

9.1.1 Compute services

3 EC2 environment

- virtual environment, which provides them with the services they require to host applications.
- In charge of allocating addresses, attaching storage volumes, and configuring security in terms of access control and network connectivity.
- Instances are created with an internal IP address, which makes them capable of communicating within the EC2 network and accessing the Internet as clients.
- Instance owners can partially control where to deploy instances. Instead, they have a finer control over the security of the instances as well as their network accessibility.
- EC2 instances are given domain name in the form ec2-xxx-xxx-xxx.compute-x.amazonaws.com

9.1 Amazon web services

9.1.1 Compute services

4. Advanced compute services

- **AWS CloudFormation -** model that characterizes EC2 instances.
- Concepts of templates, which are JSON formatted text files.
- Templates provide a simple and declarative way to build complex systems.
- **AWS Elastic Beanstalk** constitutes a simple and easy way to package applications and deploy them on the AWS Cloud.
- Service is available for Web applications developed with the Java/Tomcat technology stack.
- **Amazon Elastic MapReduce** provides AWS users with a cloud computing platform for MapReduce applications. It utilizes Hadoop as the MapReduce engine.

9.1 Amazon web services

9.1.2 Storage services

The core service is represented by Amazon Simple Storage Service (S3).
The core components of S3 are two: buckets and objects.

1 S3 key concepts
2 Amazon elastic block store
3 Amazon ElastiCache
4 Structured storage solutions
5 Amazon CloudFront

9.1 Amazon web services

9.1.2 Storage services

1 S3 key concepts

accessible through a Representational State Transfer (REST) interface.

- The storage is organized in a two-level hierarchy.
- Stored objects cannot be manipulated like standard files.
- Content is not immediately available to users.
- Requests will occasionally fail.

HTTP requests (GET, PUT, DELETE, HEAD, and POST).

Resource naming - uniform resource identifiers (URIs)

Amazon offers three different ways of addressing a bucket:

1. Canonical form: http://s3.amazonaws.com/bukect_name/
2. Subdomain form: http://bucketname.s3.amazon.com/
3. Virtual hosting form: http://bucket-name.com/

Buckets - container of objects. virtual drive hosted on the S3 distributed storage

Objects and metadata - object is identified by a name that needs to be unique within the bucket in which the content is stored.

Access control and security - access to buckets and objects by means of Access Control Policies (ACPs). Different permissions can be used.

9.1 Amazon web services

9.1.2 Storage services

2 Amazon elastic block store

Amazon Elastic Block Store (EBS) provide EC2 instances with persistent storage in the form of volumes that can be mounted at instance startup.
EBS volumes normally reside within the same availability zone of the EC2 instances that will use them to maximize the I/O performance.

3 Amazon ElastiCache

implementation of an elastic in-memory cache based on a cluster of EC2 instances.
based on a cluster of EC2 instances running the caching software, which is made available through Web services.
ElastiCache cluster can be dynamically resized according to the demand of the client applications.

9.1 Amazon web services

9.1.2 Storage services

4 Structured storage solutions

Amazon provides applications with structured storage services in three different forms:

- Preconfigured EC2 AMIs,
- Amazon Relational Data Storage (RDS), and
- Amazon SimpleDB.

Preconfigured EC2 AMIs are predefined templates featuring an installation of a given database management system. EC2 instances created from these AMIs can be completed with an EBS volume for storage persistence.

IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Vertica.

RDS is relational database service that relies on the EC2 infrastructure and is managed by Amazon.

Developers do not have to worry about configuring the storage for high availability, designing failover strategies, or keeping the servers up-to-date.

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applications that do not require a fully relational model for their data.

9.1 Amazon web services

9.1.2 Storage services

5 Amazon CloudFront

- Content delivery network.
- Collection of edge servers strategically located around the globe to better serve requests for static and streaming Web content.
- Content that can be delivered through CloudFront is static (HTTP and HTTPS) or streaming (Real Time Messaging Protocol, or RMTP).

9.1 Amazon web services

9.1.3 Communication services

Facilities to structure and facilitate the communication among existing applications and services residing within the AWS infrastructure.
These facilities can be organized into two major categories:
1. Virtual networking and
2. Messaging.

9.1 Amazon web services

9.1.3 Communication services

1. Virtual networking

- Virtual networking comprises a collection of services that allow AWS users to control the connectivity to and between compute and storage services.
- Amazon Virtual Private Cloud (VPC) and Amazon Direct Connect provide connectivity solutions in terms of infrastructure.
- Flexibility in creating virtual private networks within the Amazon infrastructure and beyond.

2. Messaging.

- Amazon Simple Queue Service (SQS),
- Amazon Simple Notification Service (SNS), and
- Amazon Simple Email Service (SES).

9.2 Google AppEngine

- Google AppEngine is a PaaS implementation.
- Distributed and scalable runtime environment that leverages Google's distributed infrastructure to scale out applications.

9.2.1 Architecture and core concepts
9.2.2 Application life cycle
9.2.3 Cost model
9.2.4 Observations

9.2 Google AppEngine

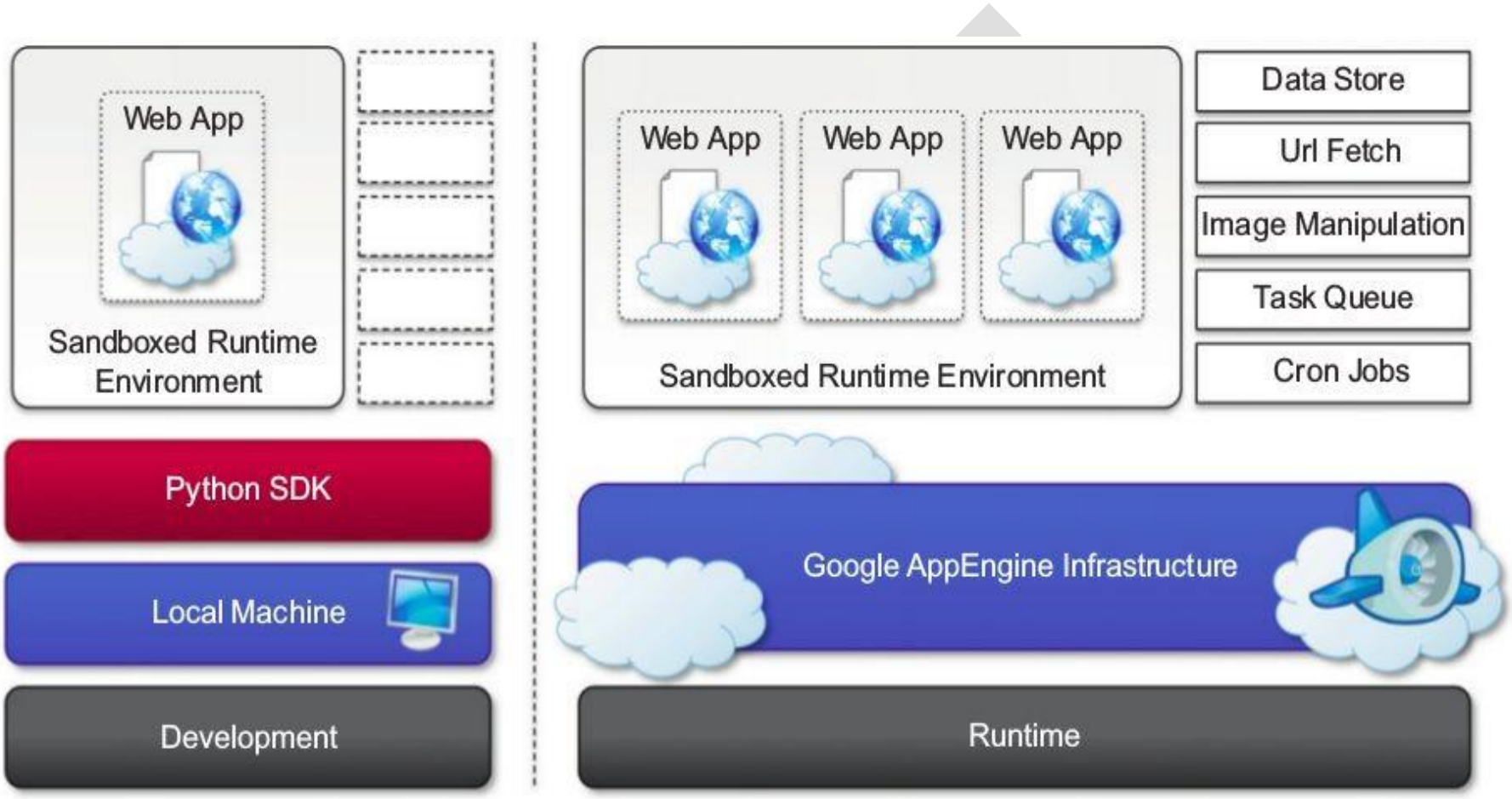


FIGURE 9.2

Google AppEngine platform architecture.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

The platform is logically divided into four major components:

1 Infrastructure
2 Runtime environment
3 Storage
4 Application services
5 Compute services

9.2 Google AppEngine

9.2.1 Architecture and core concepts

1 Infrastructure

- AppEngine’s infrastructure takes advantage of many servers available within Google datacenters. For each HTTP request, AppEngine locates the servers hosting the application that processes the request, evaluates their load, and, if necessary, allocates additional resources (i.e., servers) or redirects the request to an existing server.
-
- Also responsible for monitoring application performance and collecting sta-
- tistics on which the billing is calculated.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

2 Runtime environment

represents the execution context of applications hosted on AppEngine.

Sandboxing

Major responsibilities of the runtime environment is to provide the application environment with an isolated and protected context.

Supported runtimes

- Java, Python, and Go.
- Currently supports Java 6, Java Server Pages (JSP), and applications interact with environment by using the Java Servlet standard.
- Developers can create applications with the AppEngine Java SDK.
- Python is provided by an optimized Python 2.5.2 interpreter.
- Python Web application framework- webapp.
- Go programming language - SDK includes compiler and standard libraries for developing applications in Go and interfacing it AppEngine services.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

3 Storage

operate differently depending on the volatility of the data.

Static file servers

- Dynamic and static data.
- Dynamic data - logic of the application and the interaction with the user.
- Static data constitute of components that define the graphical layout of the application (CSS files, plain HTML files, JavaScript files, images, icons, and sound files) or data files.

DataStore

- DataStore is a service that allows developers to store semistructured data.
- DataStore is based on Bigtable a redundant, distributed, and semistructured
- data store that organizes data in the form of tables.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

4 Application services

Simplify common operations that are performed in Web applications:

- access to data,
- account management,
- integration of external resources,
- messaging and communication,
- image manipulation, and
- asynchronous computation.

UrlFetch

- The sandbox environment does not allow applications to open arbitrary connections.
- HTTP/HTTPS by means of the UrlFetch service.
- Applications can make synchronous and asynchronous Web requests and integrate the resources obtained.
- Ability to set deadlines for requests so that they can be completed (or aborted) within a given time.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

4 Application services

MemCache

- caching services by means of MemCache.
- distributed in-memory cache that is optimized for fast access and provides volatile store for objects that are frequently accessed.
- significantly reduce the access time to data.

Mail and instant messaging

- Email can also be used to trigger activities in Web applications.
- send and receive mails through Mail service - allows sending email on behalf of the application to specific user accounts.
- Extensible Messaging and Presence Protocol (XMPP), Any chat service that supports XMPP, such as Google Talk, can send and receive chat messages to and from the Web application.

Account management

- Web applications can conveniently store profile settings in the form of key-value pairs, attach them to a given Google account, and quickly retrieve them once the user authenticates.

9.2 Google AppEngine

9.2.1 Architecture and core concepts

5 Compute services

- Users navigate the Web pages and get instantaneous feedback.
- Feedback is result of some computation happening on the Web application.
- Immediate feedback and notification once the required operation is completed.
- **Task queues**
- Task Queues allow applications to submit a task for a later execution.
- **Cron jobs**
- Schedule the required operation at the desired time.

9.2 Google AppEngine

9.2.2 Application life cycle

Testing and development, deployment, and monitoring.
Java SDK and Python SDK

1 Application development and testing
2 Application deployment and management

1 Application development and testing

- **Java SDK** - facility for building applications with the Java 5 and Java 6 runtime environments.
- Eclipse development environment by using the Google AppEngine plug-in - install Java SDK, Google Web Toolkit, and Google AppEngine plug-ins into Eclipse.
- **Python SDK** - Python 2.5.
- Standalone tool, called GoogleAppEngineLauncher, for managing Web applications locally and deploying them to AppEngine.
- **webapp** - includes a set of models, components, and tools that simplify development of Web applications.

9.2 Google AppEngine

9.2.2 Application life cycle
2 Application deployment and management

- Deployed on AppEngine with a simple click or command-line tool.
- Also from GoogleAppEngineLauncher and Google AppEngine plug-in.
- http://<application-id>.appspot.com.
- Also map the application with registered DNS domain name.

9.2 Google AppEngine

9.2.3 Cost model

- Set up a billing account and be charged on a pay-per-use basis.
- Developers to identify the appropriate daily budget.
- An application is measured against billable quotas, fixed quotas, and per-minute quotas.
- Once an application reaches the quota, resource is depleted and will not be available to application until the quota is replenished.
- Once a resource is depleted, subsequent requests to that resource will generate an error or an exception.

9.3 Microsoft Azure

Built on top of Microsoft datacenters infrastructure and provides with collection of services.

- Services range from compute, storage, and networking to application connectivity, access control, and business intelligence.

- Figure 9.3 provides an overview of services provided by Azure.

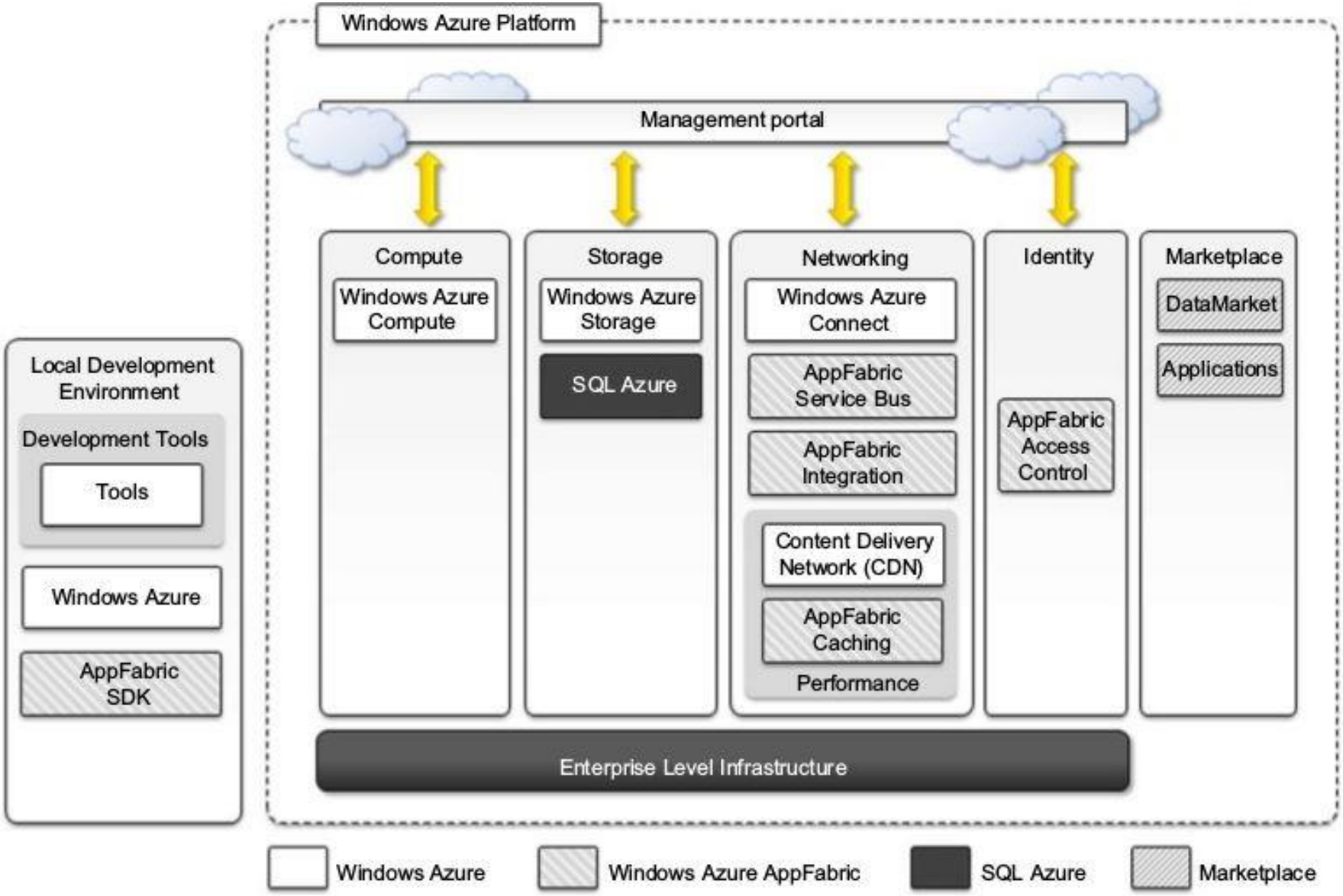


FIGURE 9.3

Microsoft Windows Azure Platform Architecture.

9.3 Microsoft Azure

9.3.1 Azure core concepts

- Made up of **foundation layer** and a set of developer services that can be used to build scalable applications.
- Services cover **compute, storage, networking**, and **identity management**, which are tied together by middleware called **AppFabric**.

1 Compute services
- Web role
- Worker role
- Virtual machine role

2 Storage services
- Blobs
- Azure drive
- Tables
- Queues

9.3 Microsoft Azure

9.3.1 Azure core concepts

3 Core infrastructure: AppFabric
- Access control
- Service bus
- Azure cache

4 Other services
- Windows Azure virtual network
- Windows Azure content delivery network

9.3 Microsoft Azure

9.3.1 Azure core concepts

1 Compute services

- Compute services are core components, they are delivered by means of abstraction of roles.
- A role is a runtime environment that is customized for specific compute task.
- Roles are managed by Azure OS and instantiated on demand.

Web role

- The Web role is designed to implement scalable Web applications.
- hosted on the IIS 7 Web Server, which is a component of the infrastructure that supports Azure.
- .NET technology supports Web roles; developers can directly develop their applications in Visual Studio, test them locally, and upload to Azure.
- Used to run and scale PHP Web applications on Azure.

Worker role

- Worker roles are designed to host general compute services on Azure.
- Quickly provide compute power or to host services.
- Runs continuously from the creation of its instance.
- Azure SDK provides APIs and libraries that allow connecting the role with the service provided by runtime and easily controlling its startup.

9.3 Microsoft Azure

9.3.1 Azure core concepts

1 Compute services

Virtual machine role

- allows to fully control the computing stack of compute service by defining a custom image of Windows Server 2008 R2 OS & their applications.
- Hyper-V virtualization technology.
- Developers can image Windows server installation into a **Virtual Hard Disk (VHD).**

Table 9.7 Windows Azure Compute Instances Characteristics, 2011–2012

Compute Instance Type	CPU	Memory	Instance Storage	I/O Performance	Hourly Cost (USD)
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.04
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 × 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 × 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 × 1.6 GHz	14 GB	2,040 GB	High	$0.96

9.3 Microsoft Azure

9.3.1 Azure core concepts

2 Storage services

- Windows Azure provides different types of storage solutions that complement compute services with a more durable and redundant option compared to local storage.

Blobs

- Store large amount of data in the form of binary large objects (BLOBs).
- Two types of blobs are available:
  - **Block blobs**. composed of blocks optimized for sequential access;
  - Blocks are of 4 MB, and a single block blob can reach 200 GB.
  - **Page blobs**. pages that are identified by an offset from the beginning of blob.
  - Split into multiple pages or constituted of single page.
  - Optimized for random access.
  - Maximum dimension of a page blob can be 1 TB.

Azure drive

- Entire file system in the form of single Virtual Hard Drive (VHD) file**.**
- NTFS file system, providing persistent and durable storage.

Tables

- semistructured storage solution. Tables are more similar to spreadsheets.
- Handle large amounts of data and queries returning huge result sets.
- Currently, table can contain up to 100 TB of data, and rows can have up to 255 properties, with a maximum of 1 MB for each row.

9.3 Microsoft Azure

9.3.1 Azure core concepts

2 Storage services

Queues

- Queue storage allows applications to communicate by exchanging messages through durable queues, thus avoiding lost or unprocessed messages.
- Applications enter messages into a queue, and other applications can read them in a first-in, first-out (FIFO) style.
- When an application reads a message it is marked as invisible; hence it will not be available to other clients.
- Once Application has completed processing message, it explicitly deletes message from queue.
- Alternative to reading a message is **peeking**, which allows retrieving message but letting it stay visible in the queue.

- All the services described are **geo-replicated three times** to ensure their availability in case of major disasters.
- **Geo-replication** involves the copying of data into a different datacenter that is **hundreds or thousands of miles away** from the original datacenter.

9.3 Microsoft Azure

9.3.1 Azure core concepts

3 Core infrastructure: AppFabric

- Middleware for developing, deploying, and managing applications on cloud or for integrating existing applications with cloud services.
- Scaling out and high availability; sandboxing and multitenancy; state management; and dynamic address resolution and routing.
- Simplify many common tasks in distributed application, such as communication, authentication and authorization, and data access.

Access control

- AppFabric provides capability of encoding access control to resources in Web applications.
- Services into set of rules that are expressed outside application code base.
- Applications can leverage Active Directory, Windows Live, Google, Facebook, and other services to authenticate users.

Service bus

- Messaging and connectivity infrastructure.
- Designed to allow transparent network traversal and to simplify development of loosely coupled applications letting developers focus on logic of interaction.
- Applications need to be connected to bus, which provides these services.

9.3 Microsoft Azure

9.3.1 Azure core concepts

3 Core infrastructure: AppFabric

Azure cache

- provides a set of durable storage solutions that allow applications to persist their data.
- Azure Cache is a service that allows developers to quickly access data persisted on Windows Azure storage or in SQL Azure.
- implements a distributed in-memory cache of which, size can be dynamically adjusted by applications.

9.3 Microsoft Azure

9.3.1 Azure core concepts

4 Other services

- Simplify the development and integration of applications.

Windows Azure virtual network

- includes **Windows Azure Connect** and **Windows Azure Traffic Manager**.
- **Windows Azure Connect** allows easy setup of IP-based network connectivity among machines hosted on private premises and roles deployed on Azure Cloud.
- **Windows Azure Traffic Manager** provides load-balancing features for services listening to HTTP or HTTPS ports and hosted on multiple roles.

Windows Azure content delivery network

- Content delivery network solution that improves content delivery capabilities of Windows Azure Storage and Microsoft Windows Update, Bing maps.

9.3 Microsoft Azure

9.3.2 SQL Azure

- SQL Azure is a relational database service hosted on Windows Azure and built on the SQL Server technologies.
- via SQL Server - provides developers with scalable, highly available, and fault-tolerant relational database.
- Service is manageable using REST APIs, allowing developers to control databases.

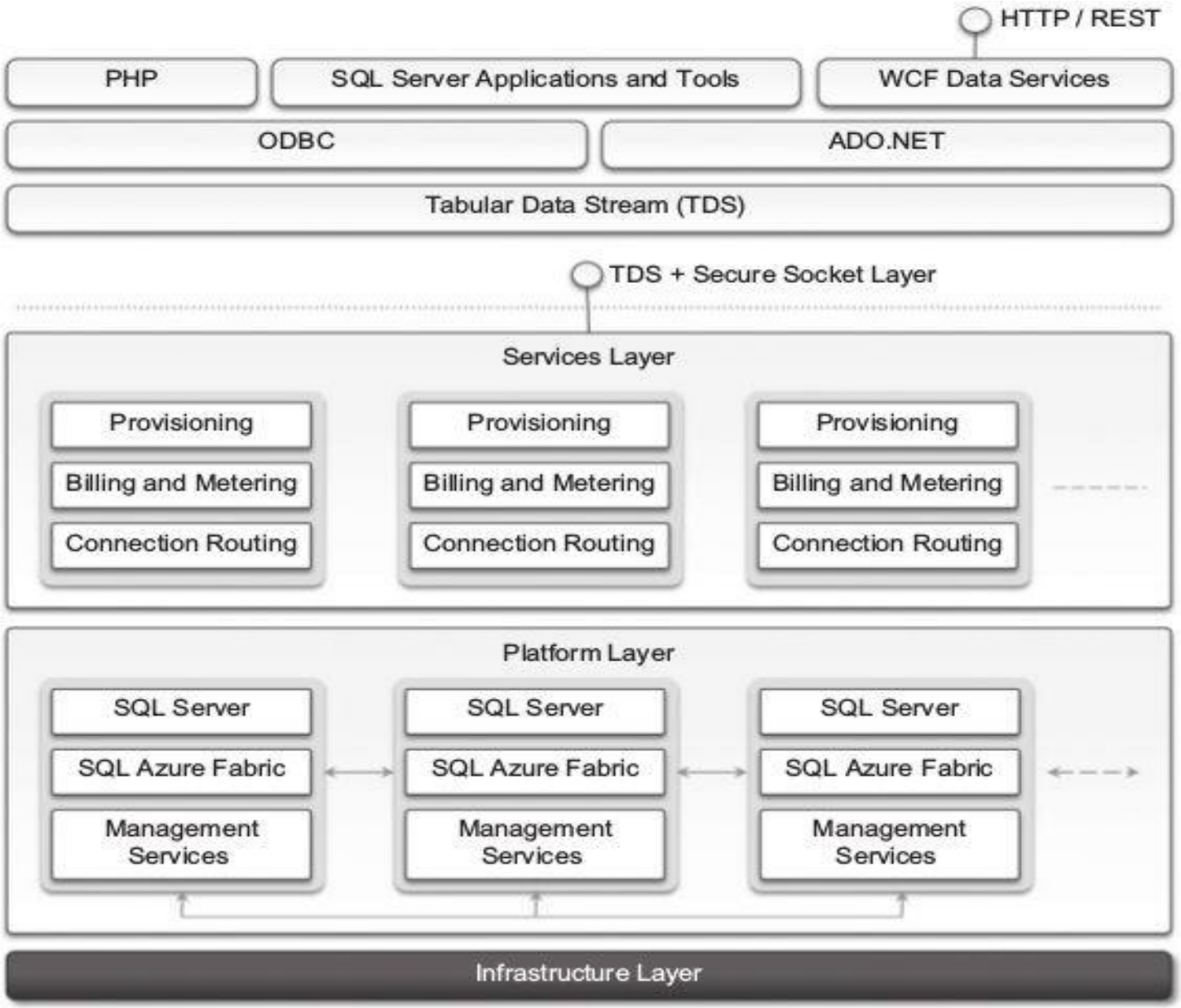


FIGURE 9.4

SQL Azure architecture.

9.3 Microsoft Azure

9.3.2 SQL Azure

- Developers have to sign up for Windows Azure account in order to use SQL Azure.
- Closely resemble physical SQL Servers: They have a fully qualified domain name under the database.windows.net
  (i.e., **server-name.database.windows.net**) domain name.
- This simplifies management tasks and the interaction with SQL Azure from client applications.

9.3 Microsoft Azure

9.3.3 Windows Azure platform appliance

- Can also be deployed as appliance on third-party data centers and constitutes the cloud infrastructure governing the physical servers of datacenter.
- Includes Windows Azure, SQL Azure, and Microsoft- specified configuration of network, storage, and server hardware.
- The appliance targets governments and service providers who want to have their own cloud computing infrastructure.
- Full-featured implementation of Windows Azure.
- Goal is to replicate Azure on a third-party infrastructure and make available its services beyond boundaries of Microsoft Cloud.

The appliance addresses two major scenarios:

- institutions that have very large computing needs (such as government agencies) and
- institutions that cannot afford to transfer their data outside their premises.

9.3 Microsoft Azure

9.3.4 Observations

- Windows Azure is Microsoft's solution for developing cloud computing applications.
- PaaS layer.
- Provides developer with collection of services and scal- able middleware hosted on Microsoft datacenters that address compute, storage, networking, and identity management needs of applications.
- Individually or all together for building both applications.
- Compute services are based on the abstraction of roles.
- AppFabric, constitutes distributed and scalable middleware of Azure.
- SQL Azure is another important element of Windows Azure and provides support for relational data in the cloud.
- Platform is based on .NET technology and Windows systems.

Chapter 10 – Cloud Applications

- Huge popularity in industry due to its ability to host applications for which the services can be delivered to consumers rapidly at minimal cost.
- We discuss Application case studies, detailing their architecture and how they leveraged various cloud technologies.
- Range of domains, from scientific to engineering, gaming, and social networking.

Overview

10.1 Scientific applications

- 10.1.1 Healthcare: ECG analysis in the cloud
- 10.1.2 Biology: protein structure prediction
- 10.1.3 Biology: gene expression data analysis for cancer diagnosis
- 10.1.4 Geoscience: satellite image processing

10.2 Business and consumer applications

- 10.2.1 CRM and ERP
  - 1 Salesforce.com
  - 2 Microsoft dynamics CRM
  - 3 NetSuite
- 10.2.2 Productivity
  - 1 Dropbox and iCloud
  - 2 Google docs
  - 3 Cloud desktops: EyeOS and XIOS/3
- 10.2.3 Social networking
  - 1 Facebook
- 10.2.4 Media applications
  - 1 Animoto
  - 2 Maya rendering with Aneka
  - 3 Video encoding on the cloud: Encoding.com
- 10.2.5 Multiplayer online gaming

10.1 Scientific applications

- increasingly using cloud computing systems and technologies.
- high-performance computing (HPC) applications, high-throughput computing (HTC) applications, and data-intensive applications.
- minimal changes need to be made to existing applications.
- IaaS solutions, offer optimal environment for running bag-of-tasks applications and workflows.
- Scientists explore new programming models for tackling computationally challenging problems.
- Applications - redesigned and implemented on top of cloud programming application models and platforms to leverage their unique capabilities.

10.1 Scientific applications

10.1.1 Healthcare: ECG analysis in the cloud

- cloud technologies to support doctors in providing more effective diagnostic processes.
- Internet connectivity and its accessibility from any device at any time has made cloud technologies an attractive option for developing health-monitoring systems.

10.1 Scientific applications

10.1.1 Healthcare: ECG analysis in the cloud

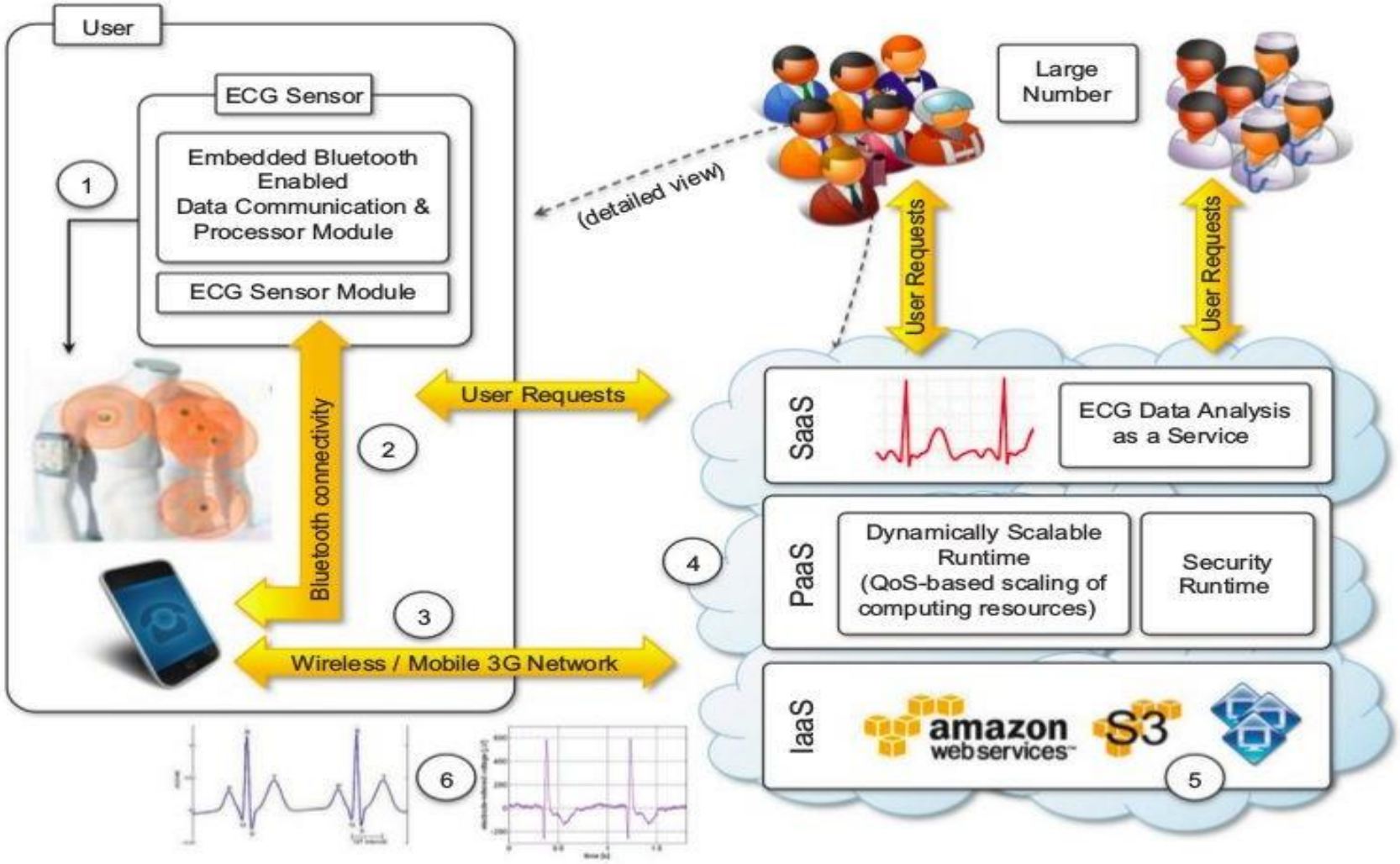


FIGURE 10.1

An online health monitoring system hosted in the cloud.

10.1 Scientific applications

10.1.2 Biology: protein structure prediction

- Require high computing capabilities and often operate on large data-sets that cause extensive I/O operations.
- extensive use of supercomputing and cluster computing infrastructures.
- Protein structure prediction is a computationally intensive task.
- The geometric structure of a protein cannot be directly inferred from the sequence of genes that compose its structure, but it is the result of complex computations aimed at identifying the structure.
- This task requires investigation of space with massive number of states, creating large number of computations for each of these states.

- One project that investigates the use of cloud technologies for protein structure prediction is Jeeva — an integrated Web portal that enables scientists to offload the prediction task to a computing cloud based on Aneka (**Figure 10.2**).
- Uses machine learning tecniques.
- E,H,C classification.

10.1 Scientific applications

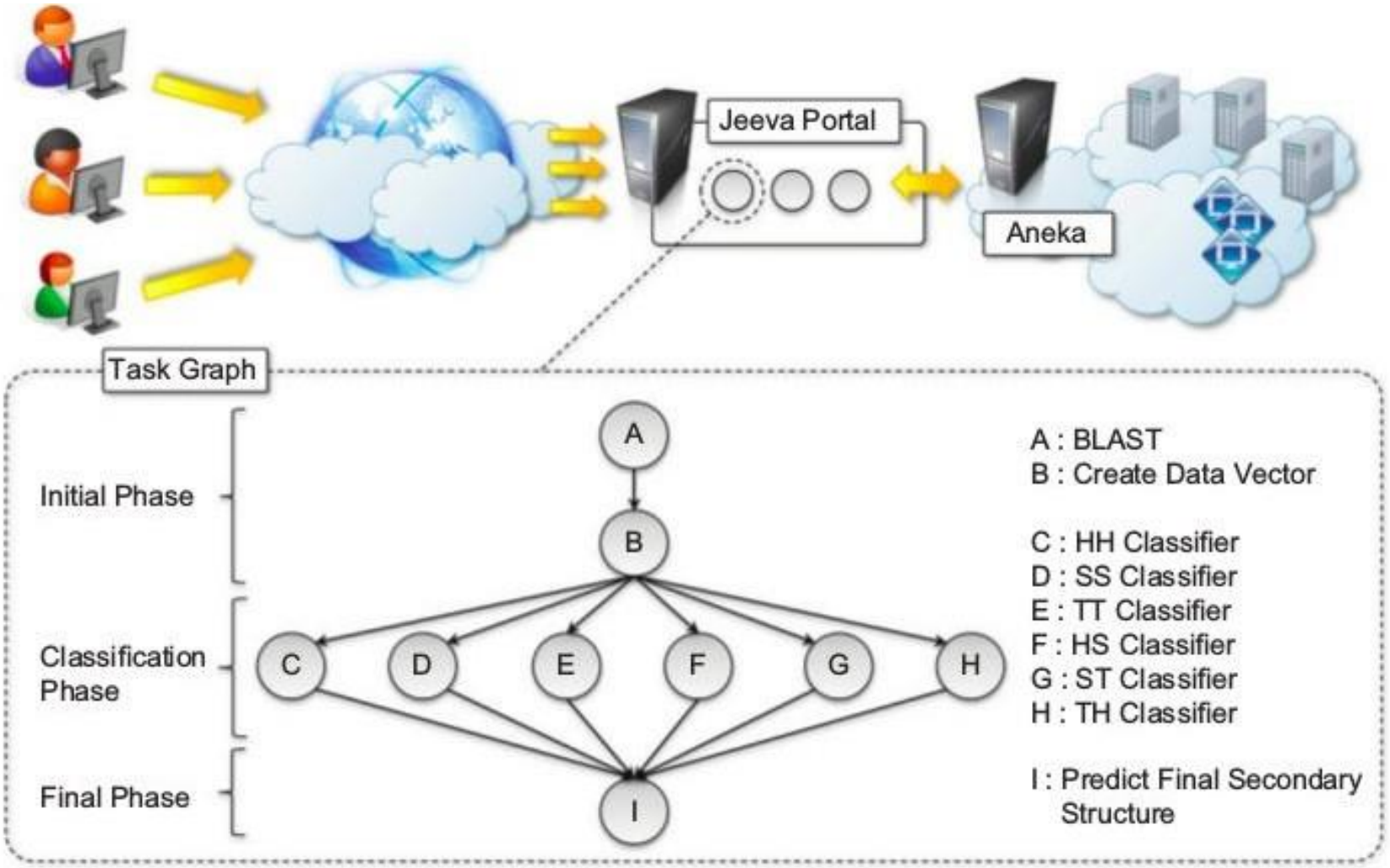


FIGURE 10.2

Architecture and overview of the Jeeva Portal.

10.1 Scientific applications

10.1.3 Biology: gene expression data analysis for cancer diagnosis

- measurement of the expression levels of thousands of genes at once.
- used to understand the biological processes that are triggered by medical treatment at a cellular level.
- Gene expression profiling is utilized to provide a more accurate classification of tumors.

- Learning classifiers, which generate a population of condition-action rules that guide the classification process.
- **eXtended Classifier System (XCS)** has been utilized for classifying large datasets in bioinformatics and computer science domains.
- **CoXCS** divides the entire search space into subdomains and employs standard XCS algorithm in each of these subdomains.

10.1 Scientific applications

10.1.3 Biology: gene expression data analysis for cancer diagnosis

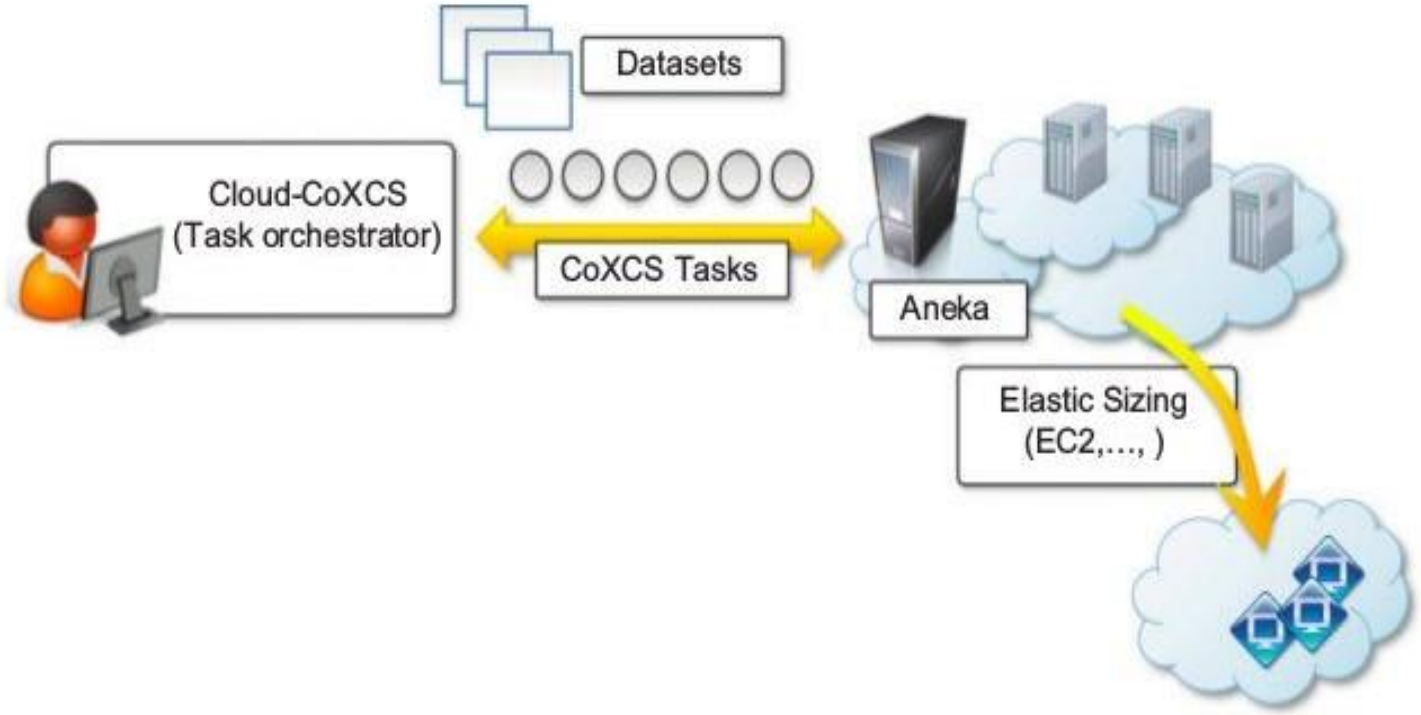


FIGURE 10.3

Cloud-CoXCS: An environment for microarray data processing on the cloud.

10.1 Scientific applications

10.1.4 Geoscience: satellite image processing

- collect, produce, and analyze massive amounts of geospatial and nonspatial data.
- the **geographic information system (GIS)** is a major element of geoscience applications. GIS applications capture, store, manipulate, analyze, manage, and present all types of geographically referenced data.

10.1 Scientific applications

10.1.4 Geoscience: satellite image processing

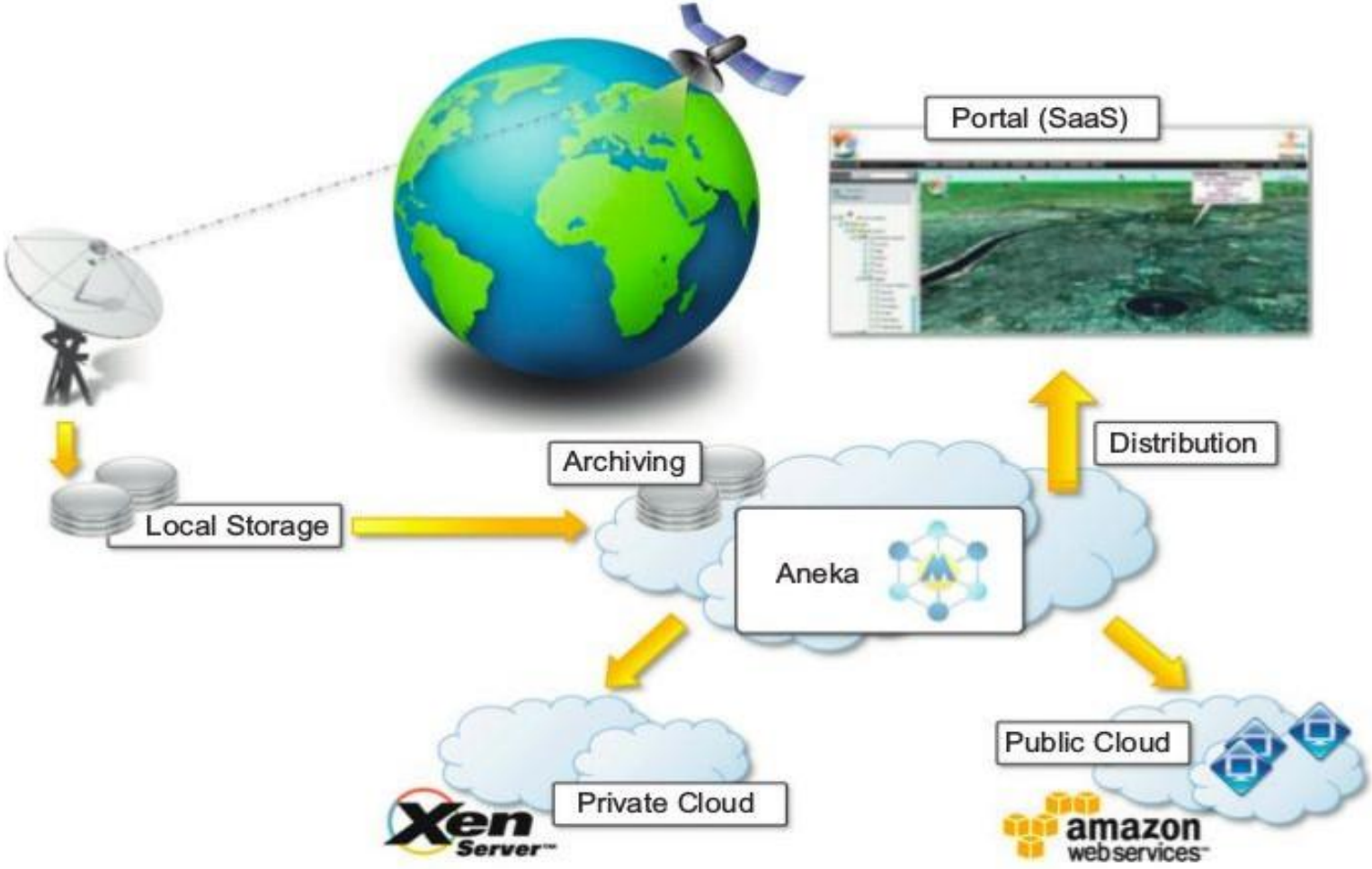


FIGURE 10.4

A cloud environment for satellite data processing.

10.2 Business & consumer applications

- 10.2.1 CRM and ERP
  - 1 Salesforce.com
  - 2 Microsoft dynamics CRM
  - 3 NetSuite
- 10.2.2 Productivity
  - 1 Dropbox and iCloud
  - 2 Google docs
  - 3 Cloud desktops: EyeOS and XIOS/3
- 10.2.3 Social networking
  - 1 Facebook
- 10.2.4 Media applications
  - 1 Animoto
  - 2 Maya rendering with Aneka
  - 3 Video encoding on the cloud: Encoding.com
- 10.2.5 Multiplayer online gaming

10.2 Business & consumer applications

10.2.1 CRM and ERP

- **Customer relationship management (CRM)** and **enterprise resource planning (ERP)** applications are market segments that are flourishing in the cloud.
- access to business and customer data from everywhere and from any device.
- ERP systems integrate several aspects of enterprise: finance and accounting, human resources, manufacturing, supply chain management, project management, and CRM.
- ERP solutions are less popular than CRM solutions at this time.

1 Salesforce.com

- most popular and developed CRM solution available today.
- **Salesforce.com** is based on **Force.com** cloud development platform.

10.2 Business & consumer applications

10.2.1 CRM and ERP

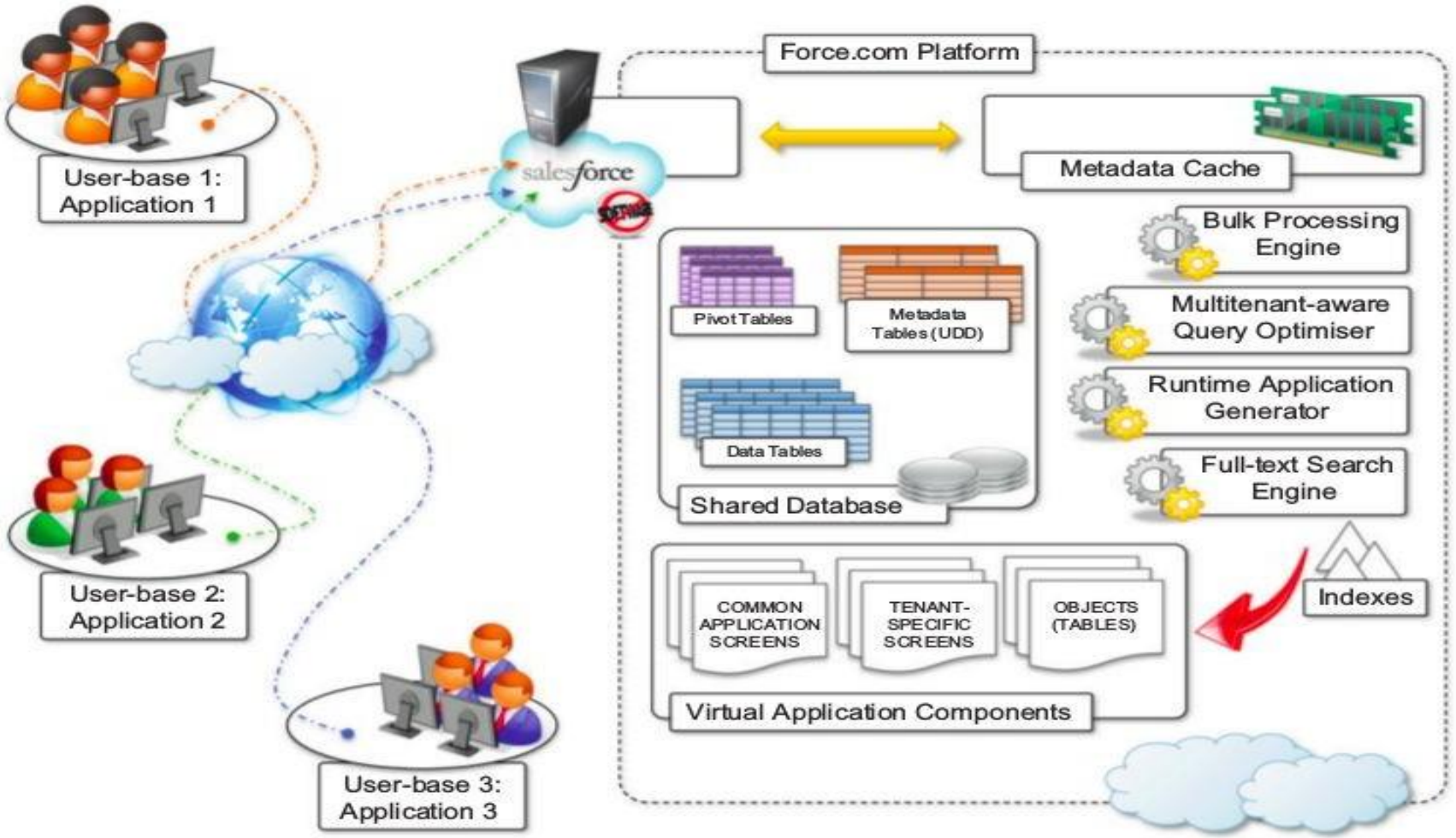


FIGURE 10.5

Salesforce.com and Force.com architecture.

10.2 Business & consumer applications

10.2.1 CRM and ERP

2 Microsoft dynamics CRM

- completely hosted in Microsoft's datacenters across the world and offers to customers a 99.9% SLA.
- Each CRM
- instance is deployed on a separate database, and the application provides users with facilities for marketing, sales, and advanced customer relationship management.
- SOAP and RESTful Web services.
- integrate with other Microsoft products and line-of-business applications.

3 NetSuite

- Three major products: **NetSuite Global ERP, NetSuite Global CRM1 , and NetSuite Global Ecommerce**.
- All-in-one solution: **NetSuite One World**, integrates all three products together.
- Two large datacenters on the East and West coasts of the United States, connected by redundant links.
- The **NetSuite Business Operating System (NS-BOS)** is complete stack of technologies for building SaaS.

10.2 Business & consumer applications

10.2.2 Productivity

1 Dropbox and iCloud

Also Windows Live, Amazon Cloud Drive, and CloudMe.

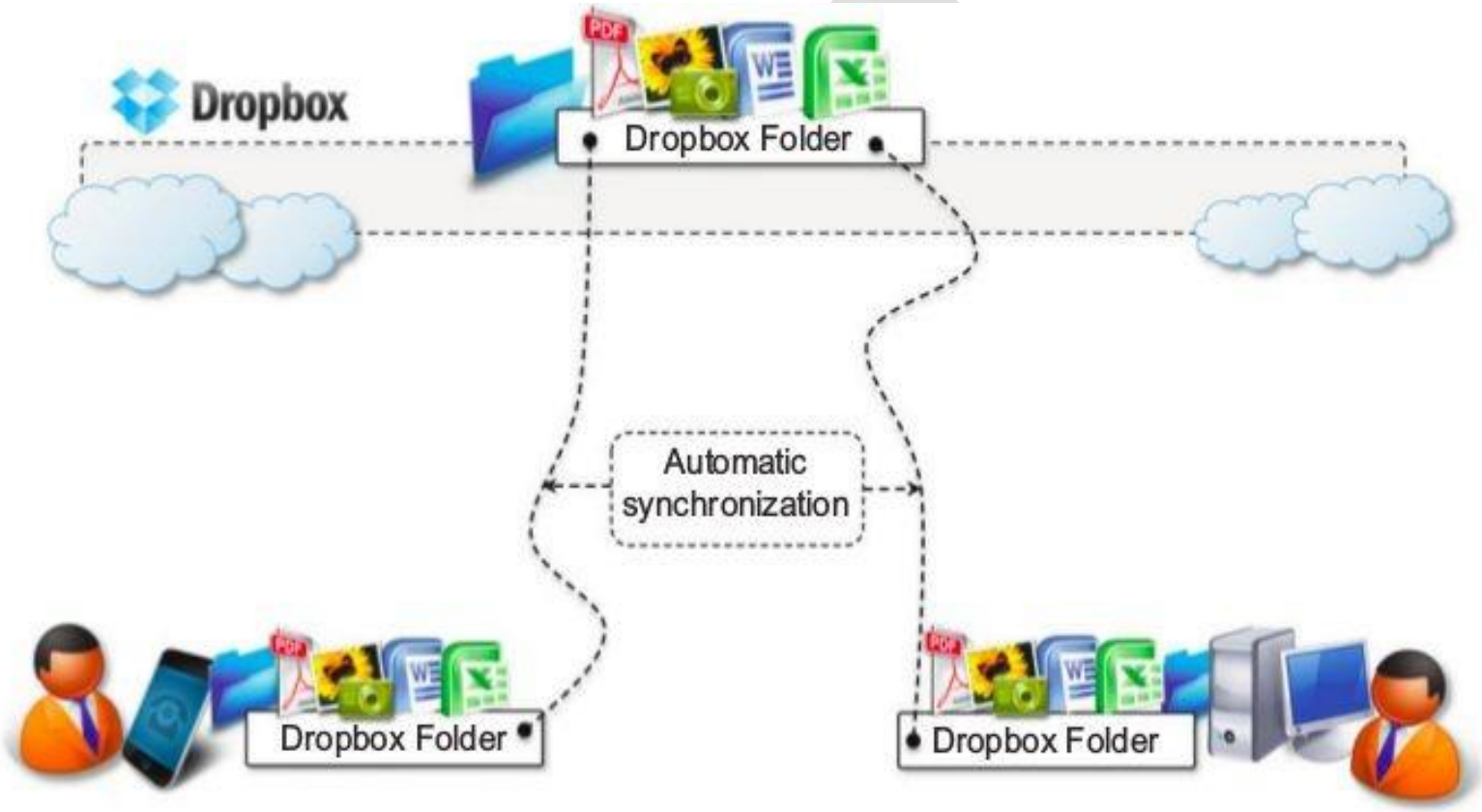


FIGURE 10.6

Dropbox usage scenario.

10.2 Business & consumer applications

10.2.2 Productivity

2 Google docs

- Google Docs allows users to create and edit text documents, spreadsheets, presentations, forms, and drawings.

- It aims to replace desktop products such as Microsoft Office and OpenOffice and provide similar interface and functionality as a cloud service.

10.2 Business & consumer applications

10.2.2 Productivity

3 Cloud desktops: EyeOS and XIOS/3

- EyeOS 1 is one of the most popular Web desktop solutions.
- Replicates the functionalities of a classic desktop environment and comes with pre-installed applications for the most common file and document management tasks.

Xcerion XML Internet OS/3 (XIOS/3)

- Another example of a Web desktop environment.
- Strong leverage of XML, used to implement many of the tasks of the OS: rendering user interfaces, defining application business logics, structuring file system organization, and even application development.

10.2 Business & consumer applications

10.2.2 Productivity

3 Cloud desktops: EyeOS

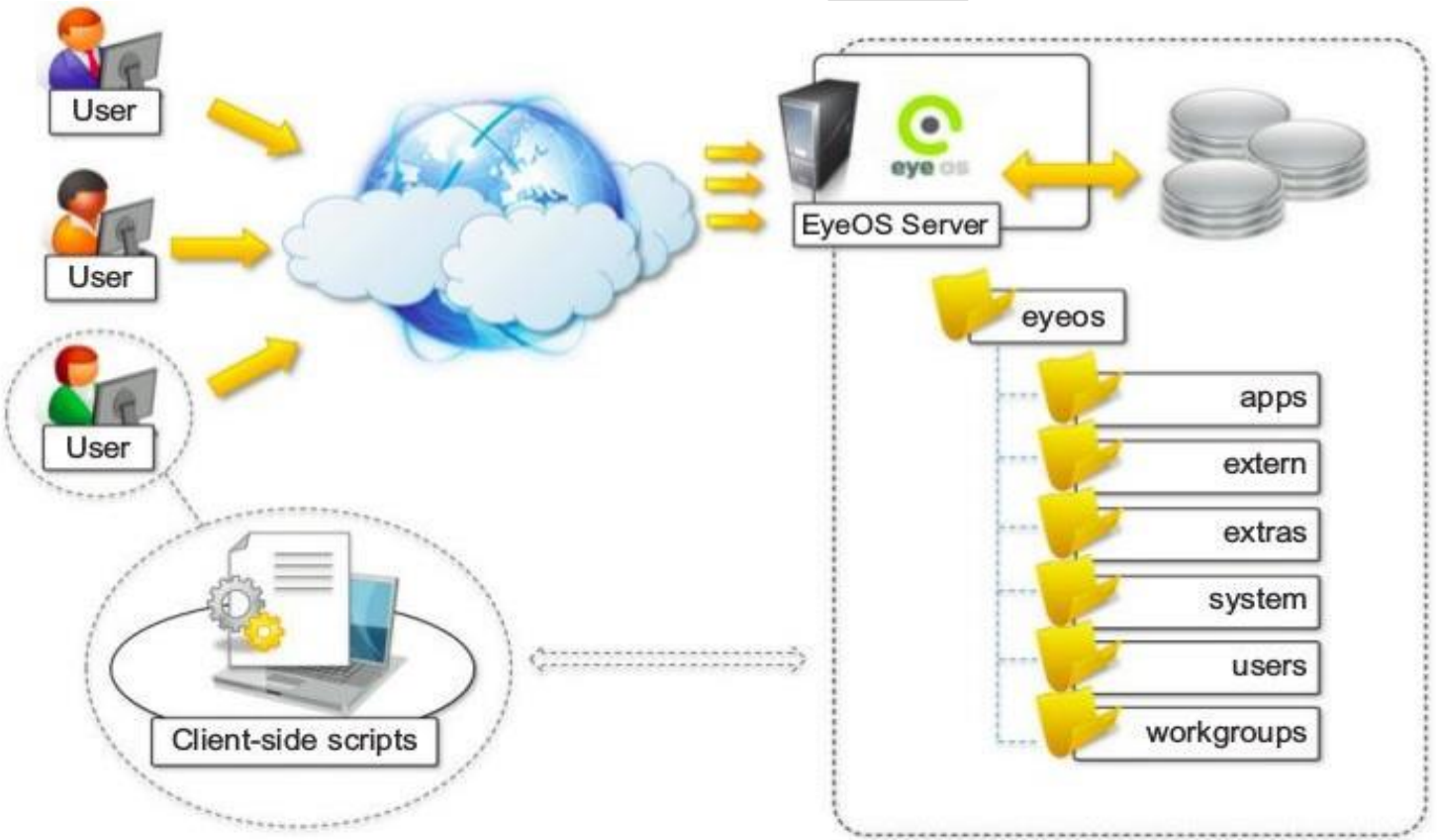


FIGURE 10.7

EyeOS architecture.

10.2 Business & consumer applications

10.2.3 Social networking

1 Facebook

- 800 million users
- Two data centers built and optimized to reduce costs and impact on environment.
- Technologies constitute a powerful platform for developing cloud applications.
- The reference stack serving Facebook is based on LAMP (Linux, Apache, MySQL, and PHP).
- The social graph identifies collection of interlinked information that is of relevance for a given user.
- Most of the user data are served by querying a distributed cluster of MySQL.
- These data are then cached for faster retrieval.
- Thrift - collection of abstractions that allow cross-language development.

10.2 Business & consumer applications

10.2.4 Media applications

- Video-processing operations, such as encoding, transcoding, composition, and rendering, are good candidates for a cloud-based environment.
- Computationally intensive tasks offloaded to cloud computing infrastructures.

1 Animoto

- interface for quickly creating videos out of images, music, and video fragments submitted by users.
- process is executed in the background and the user is notified via email once the video is rendered.
- ability to quickly create videos with stunning effects without user intervention.
-
- Artificial intelligence (AI) engine, which selects the animation and transition effects according to pictures and music, drives the rendering operation.

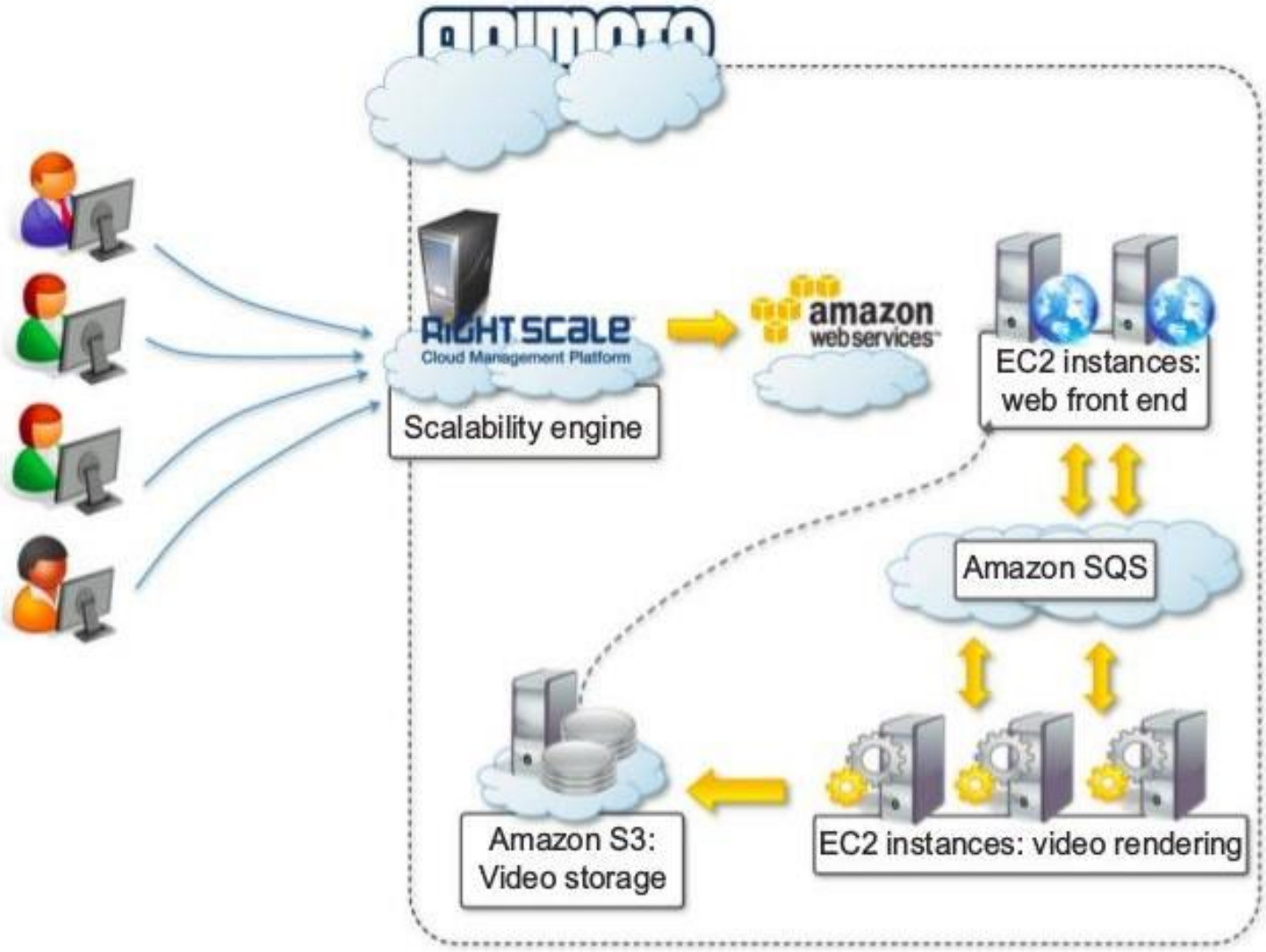


FIGURE 10.8

Animoto reference architecture.

10.2 Business & consumer applications

10.2.4 Media applications

2 Maya rendering with Aneka

- A private cloud solution for rendering train designs by engineering department of GoFront group, a division of China Southern Railway.

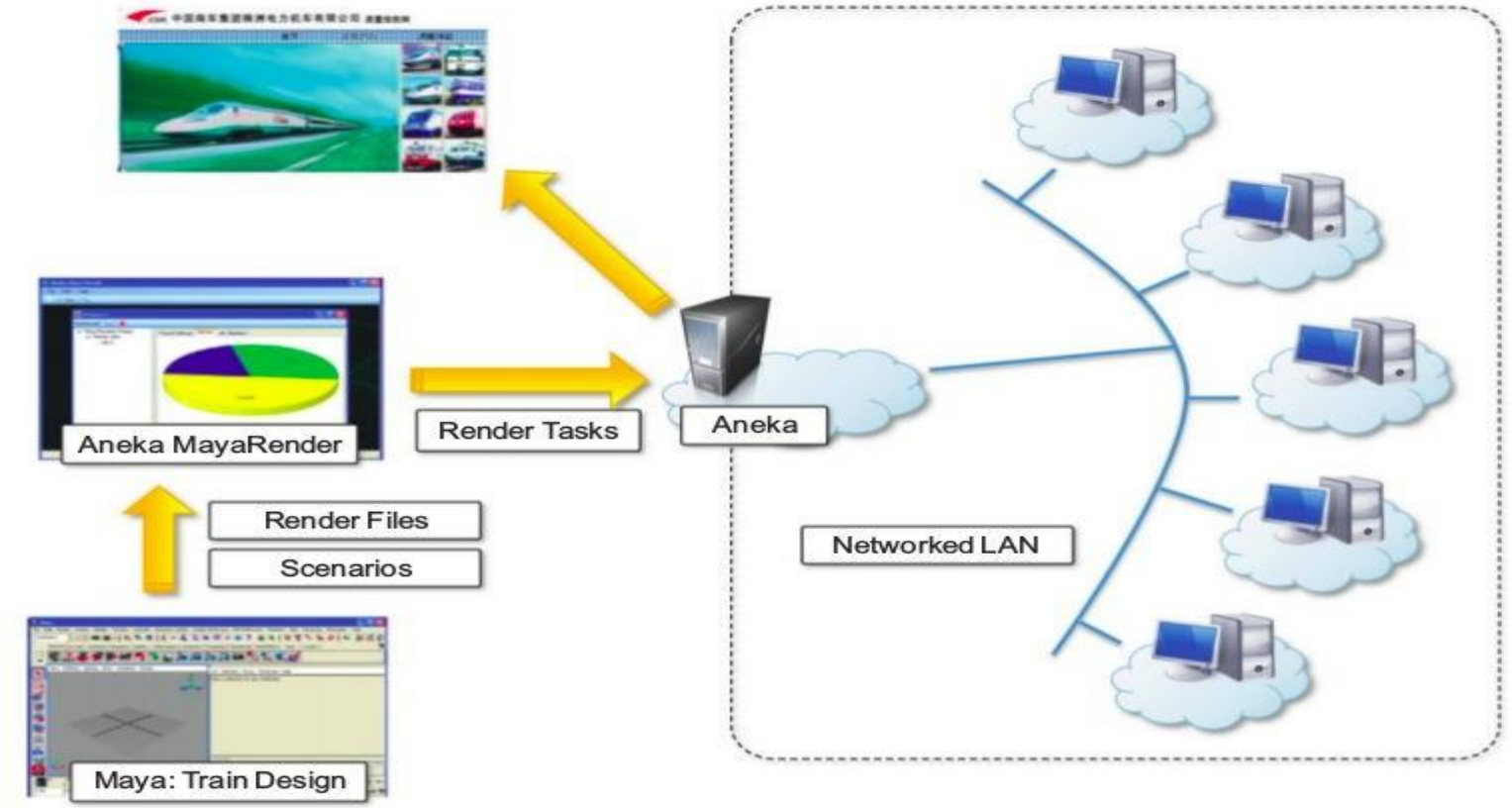


FIGURE 10.9

3D rendering on private clouds.

10.2 Business & consumer applications

10.2.4 Media applications

3 Video encoding on the cloud: Encoding.com

- Video-transcoding services on demand.
- Leverages cloud technology to provide both horsepower required for video conversion and storage for staging videos.
- Service integrates with both Amazon Web Services technologies (EC2, S3, and CloudFront) and Rackspace (Cloud Servers, Cloud Files, and Limelight CDN access).
- Also offers other video-editing operations such as the insertion of thumbnails, watermarks, or logos.

10.2 Business & consumer applications

10.2.5 Multiplayer online gaming

- based on game log processing.
- Players update game server hosting game session, and server integrates all updates into a log that is made available to all players through a TCP port.
- Client software used for game connects to the log port and, by reading log, updates local user interface with actions of other players.

10.2 Business & consumer applications

10.2.5 Multiplayer online gaming

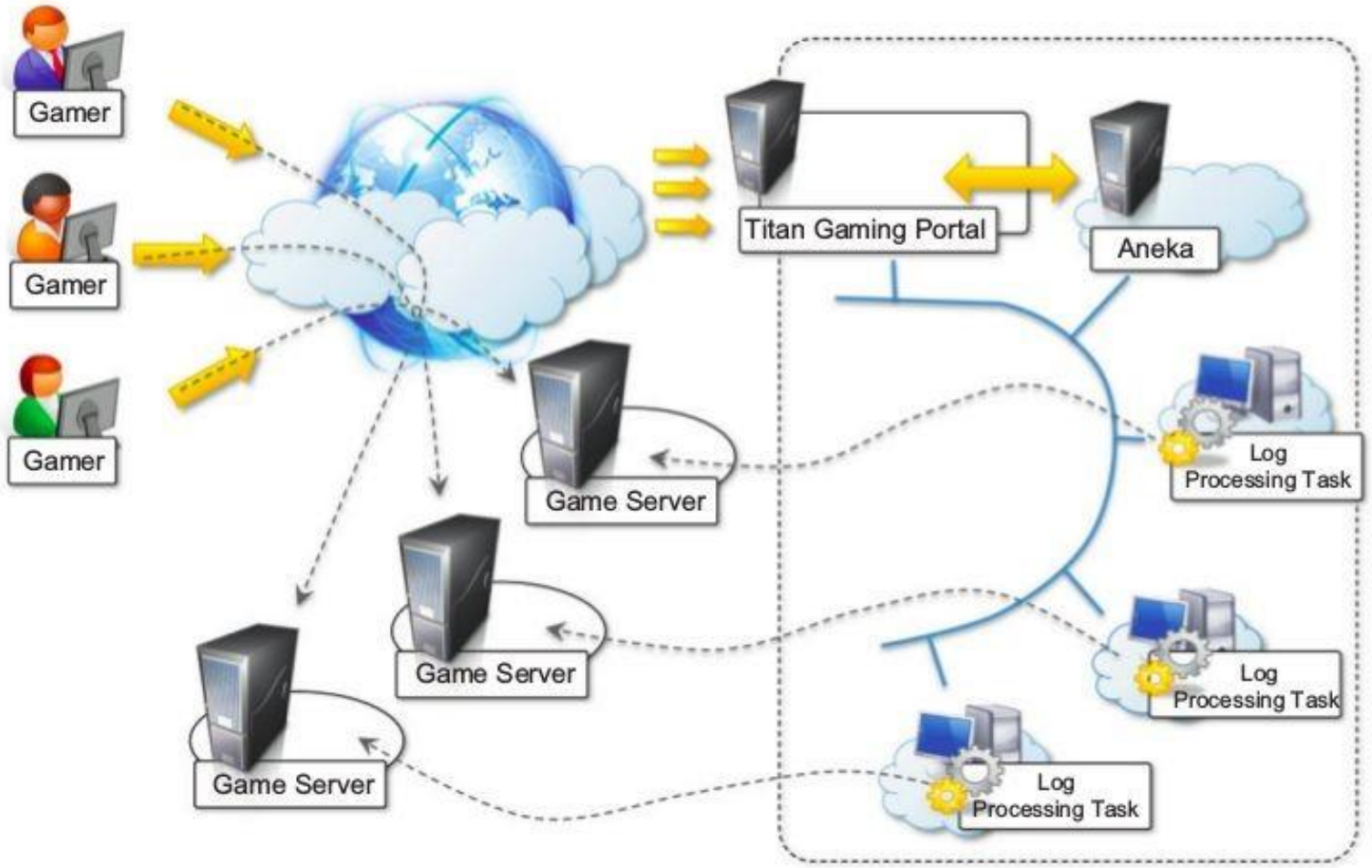


FIGURE 10.10

Scalable processing of logs for network games.