MODULE – III

High-Throughput Computing – Task Programming (Chapter 7)

Task computing is a wide area of distributed system programming encompassing several different models of architecting distributed applications,

A task represents a program, which require input files and produce output files as a result of its execution.

Applications are then constituted of a collection of tasks. These are submitted for execution and their output data are collected at the end of their execution.

This chapter characterizes the abstraction of a task and provides a brief overview of the distrib- uted application models that are based on the task abstraction.

The Aneka Task Programming Model is taken as a reference implementation to illustrate the execution of bag-of-tasks (BoT) applications on a distributed infrastructure.

7.1 Task computing

A task identifies one or more operations that produce a distinct output and that can be isolated as a single logical unit.

In practice, a task is represented as a distinct unit of code, or a program, that can be separated and executed in a remote run time environment.

Multithreaded programming is mainly concerned with providing a support for parallelism within a single machine. Task computing provides distribution by harnessing the compute power of several computing nodes. Hence, the presence of a distributed infrastructure is explicit in this model.

Now clouds have emerged as an attractive solution to obtain a huge computing power on demand for the execution of distributed applications. To achieve it, suitable middleware is needed. A reference scenario for task computing is depicted in **Figure 7.1**.

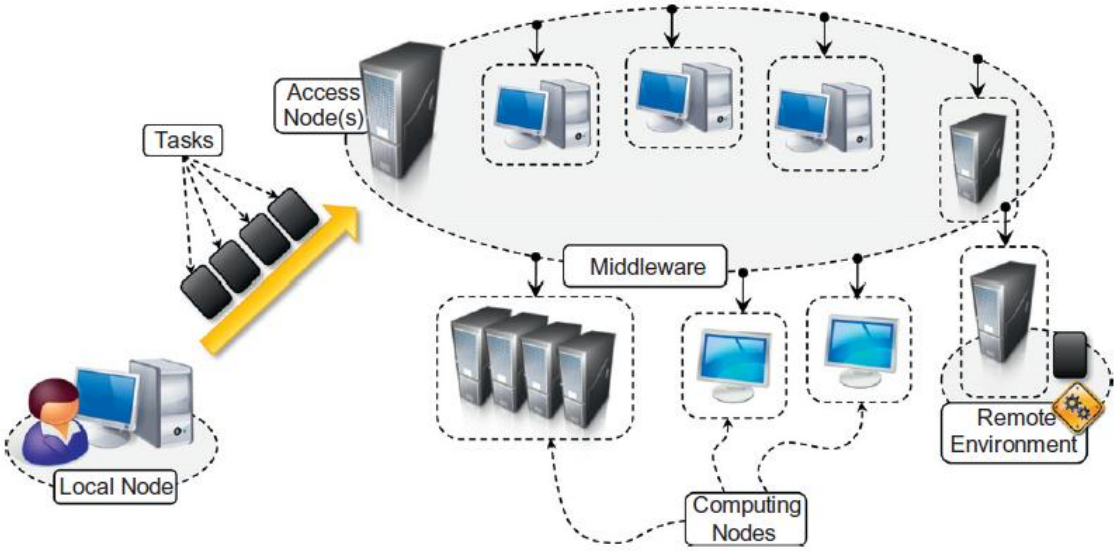


FIGURE 7.1

Task computing scenario.

The middleware is a software layer that enables the coordinated use of multiple resources, which are drawn from a datacentre or geographically distributed networked computers.

A user submits the collection of tasks to the access point(s) of the middleware, which will take care of scheduling and monitoring the execution of tasks.

Each computing resource provides an appropriate runtime environment.

Task submission is done using the APIs provided by the middleware, whether a Web or programming language interface.

Appropriate APIs are also provided to monitor task status and collect their results upon completion.

It is possible to identify a set of common operations that the middleware needs to support the creation and execution of task-based applications. These operations are:

- Coordinating and scheduling tasks for execution on a set of remote nodes
- Moving programs to remote nodes and managing their dependencies
- Creating an environment for execution of tasks on the remote nodes
- Monitoring each task's execution and informing the user about its status
- Access to the output produced by the task.

7.1.1 Characterizing a task

A task represents a component of an application that can be logically isolated and executed separately.

A task can be represented by different elements:

- A shell script composing together the execution of several applications
- A single program
- A unit of code (a Java/C11/.NET class) that executes within the context of a specific runtime environment.

A task is characterized by input files, executable code (programs, shell scripts, etc.), and output files.

The runtime environment in which tasks execute is the operating system or an equivalent sandboxed environment.

A task may also need specific software appliances on the remote execution nodes.

7.1.2 Computing categories

These categories provide an overall view of the characteristics of the problems. They implicitly impose requirements on the infrastructure and the middleware.

Applications falling into this category are:

1 High-performance computing
2 High-throughput computing
3 Many-task computing

1 High-performance computing

High-performance computing (HPC) is the use of distributed computing facilities for solving problems that need large computing power.

The general profile of HPC applications is constituted by a large collection of compute-intensive tasks that need to be processed in a short period of time.
The metrics to evaluate HPC systems are floating-point operations per second (FLOPS), now tera-FLOPS or even peta-FLOPS, which identify the number of floating- point operations per second.
Ex: supercomputers and clusters are specifically designed to support HPC applications that are developed to solve "Grand Challenge" problems in science and engineering.

2 High-throughput computing

High-throughput computing (HTC) is the use of distributed computing facilities for applications requiring large computing power over a long period of time.
HTC systems need to be robust and to reliably operate over a long time scale.
The general profile of HTC applications is that they are made up of a large number of tasks of which the execution can last for a considerable amount of time.
Ex: scientific simulations or statistical analyses.
It is quite common to have independent tasks that can be scheduled in distributed resources because they do not need to communicate.
HTC systems measure their performance in terms of jobs completed per month.

3 Many-task computing

MTC denotes high-performance computations comprising multiple distinct activities coupled via file system operations.
MTC is the heterogeneity of tasks that might be of different nature: Tasks may be small or large, single-processor or multiprocessor, compute-intensive or data-intensive, static or dynamic, homogeneous or heterogeneous.
MTC applications includes loosely coupled applications that are communication-intensive but not naturally expressed using the message-passing interface.
It aims to bridge the gap between HPC and HTC. MTC is similar to HTC, but it concentrates on the use of many computing resources over a short period of time to accomplish many computational tasks.

7.1.3 Frameworks for task computing

Some popular software systems that support the task-computing framework are:

1. **Condor**
2. **Globus Toolkit**
3. **Sun Grid Engine (SGE)**
4. **BOINC**
5. **Nimrod/G**

Architecture of all these systems is similar to the general reference architecture depicted in **Figure 7.1**.
They consist of two main components: a scheduling node (one or more) and worker nodes. The organization of the system components may vary.

1. Condor

Condor is the most widely used and long-lived middleware for managing clusters, idle workstations, and a collection of clusters.
Condor supports features of batch-queuing systems along with the capability to checkpoint jobs and manage overload nodes.
It provides a powerful job resource-matching mechanism, which schedules jobs only on resources that have the appropriate runtime environment.
Condor can handle both serial and parallel jobs on a wide variety of resources.
It is used by hundreds of organizations in industry, government, and academia to manage infrastructures.
Condor-G is a version of Condor that supports integration with grid computing resources, such as those managed by Globus.

2. Globus Toolkit

The Globus Toolkit is a collection of technologies that enable grid computing.
It provides a comprehensive set of tools for sharing computing power, databases, and other services across corporate, institutional, and geographic boundaries.
The toolkit features software services, libraries, and tools for resource monitoring, discovery, and management as well as security and file management.
The toolkit defines a collection of interfaces and protocol for interoperation that enable different systems to

integrate with each other and expose resources outside their boundaries.

3. Sun Grid Engine (SGE)

Sun Grid Engine (SGE), now Oracle Grid Engine, is middleware for workload and distributed resource management.

Initially developed to support the execution of jobs on clusters, SGE integrated additional capabilities and now is able to manage heterogeneous resources and constitutes middleware for grid computing.

It supports the execution of parallel, serial, interactive, and parametric jobs and features advanced scheduling capabilities such as budget-based and group- based scheduling, scheduling applications that have deadlines, custom policies, and advance reservation.

4. BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) is framework for volunteer and grid computing. It allows us to turn desktop machines into volunteer computing nodes that are leveraged to run jobs when such machines become inactive.

BOINC supports job check pointing and duplication.

BOINC is composed of two main components: the BOINC server and the BOINC client.

The BOINC server is the central node that keeps track of all the available resources and scheduling jobs.

The BOINC client is the software component that is deployed on desktop machines and that creates the BOINC execution environment for job submission.

BOINC systems can be easily set up to provide more stable support for job execution by creating computing grids with dedicated machines.

When installing BOINC clients, users can decide the application project to which they want to donate the CPU cycles of their computer.

Currently several projects, ranging from medicine to astronomy and cryptography, are running on the BOINC infrastructure.

5. Nimrod/G

Tool for automated modeling and execution of parameter sweep applications over global computational grids.

It provides a simple declarative parametric modeling language for expressing parametric experiments.

It uses novel resource management and scheduling algorithms based on economic principles.

It supports deadline- and budget-constrained scheduling of applications on distributed grid resources to minimize the execution cost and at the same deliver results in a timely manner.

It has been used for a very wide range of applications over the years, ranging from quantum chemistry to policy and environmental impact.

7.2 Task-based application models

There are several models based on the concept of the task as the fundamental unit for composing distributed applications.

What makes these models different from one another is the way in which tasks are generated, the relationships they have with each other, and the presence of dependencies or other conditions.

In this section, we quickly review the most common and popular models based on the concept of the task.

7.2.1 Embarrassingly parallel applications

Embarrassingly parallel applications constitute the most simple and intuitive category of distributed applications.

The tasks might be of the same type or of different types, and they do not need to communicate among themselves.

This category of applications is supported by the majority of the frameworks for distributed computing. Since tasks do not need to communicate, there is a lot of freedom regarding the way they are scheduled.

Tasks can be executed in any order, and there is no specific requirement for tasks to be executed at the same time.

Scheduling these applications is simplified and concerned with the optimal mapping of tasks to available resources. Frameworks and tools supporting embarrassingly parallel applications are the Globus Toolkit, BOINC, and Aneka.

There are several problems: image and video rendering, evolutionary optimization, and model forecasting.

In image and video rendering the task is represented by the rendering of a pixel or a frame, respectively.

For evolutionary optimization meta heuristics, a task is identified by a single run of the algorithm with a given parameter set.
The same applies to model forecasting applications.
In general, scientific applications constitute a considerable source of embarrassingly parallel applications.

7.2.2 Parameter sweep applications

Parameter sweep applications are a specific class of embarrassingly parallel applications for which the tasks are identical in their nature and differ only by the specific parameters used to execute.
Parameter sweep applications are identified by a template task and a set of parameters. The template task defines the operations that will be performed on the remote node for the execution of tasks.
The parameter set identifies the combination of variables whose assignments specialize the template task into a specific instance.
Any distributed computing framework that provides support for embarrassingly parallel applications can also support the execution of parameter sweep applications.
The only difference is that the tasks that will be executed are generated by iterating over all the possible and admissible combinations of parameters.
Nimrod/G is natively designed to support the execution of parameter sweep applications, and Aneka provides client-based tools for visually composing a template task, defining parameters, and iterating over all the possible combinations.
A plethora of applications fall into this category. Scientific computing domain: evolutionary optimization algorithms, weather-forecasting models, computational fluid dynamics applications, Monte Carlo methods.
For example, in the case of evolutionary algorithms it is possible to identify the domain of the applications as a combination of the relevant parameters.
For genetic algorithms these might be the number of individuals of the population used by the optimizer and the number of generations for which to run the optimizer.
The following example in pseudo-code demonstrates how to use parameter sweeping for the execution of a generic evolutionary algorithm.

```
individuals 5 {100, 200,300,500,1000}
generations 5 {50, 100,200,400}
foreach indiv in individuals do
foreach generation in generations do
    task = generate_task(indiv, generation)
    submit_task(task)
```

In this case 20 tasks are generated. The function generate_task is specific to the application and creates the task instance by substituting the values of indiv and generation to the corresponding variables in the template definition. The function submit_task is specific to the middleware used and performs the actual task submission.

A template task is in general a composition of operations template task is in general a composition of operations concerning the execution of legacy applications with the appropriate parameters and set of file system operations. Frameworks that natively support the execution of parameter sweep applications provide a set of useful commands for manipulating or operating on files.
The commonly available commands are:
• Execute. Executes a program on the remote node.
• Copy. Copies a file to/from the remote node.
• Substitute. Substitutes the parameter values with their placeholders inside a file.
• Delete. Deletes a file.

Figures 7.2 and 7.3 provide examples of two possible task templates, the former as defined according to the notation used by Nimrod/G, and the latter as required by Aneka.

```
parameter x float range from 1 to 10 step 1;
parameter y float range from -4 to 5 step 1;

task main
  node:execute /bin/echo X:${x} Y:${y} > output
  copy node:output output.`expr ${y}\*10+${x}`
endtask
```

FIGURE 7.2

Nimrod/G task template definition.

The template file has two sections: a header for the definition of the parameters, and a task definition section that includes shell commands mixed with Nimrod/G commands.
The prefix node: identifies the remote location where the task is executed. Parameters are identified with the ${. . .} notation.

```
<psm>
  <name>Aneka Blast</name>
  <description>BLAST simulation</description>
  <workspace>C:\Projects\Explorer\blast</workspace>
  <parameters>
    <single name="p" type="String" comment="The name of the program" value="blastn"/>
    <single name="d" type="String" comment="The database file" value="ecoli.nt"/>
    <range name="s" type="String" comment="The sequence file" from="0" to="2" interval="1"/>
  </parameters>
  <sharedFiles>
    <file path="blastall.exe" vpath="blastall.exe"/>
    <file path="ecoli.nt.nhr" vpath="ecoli.nt.nhr"/>
    <file path="ecoli.nt.nin" vpath="ecoli.nt.nin"/>
    <file path="ecoli.nt.nnd" vpath="ecoli.nt.nnd"/>
    <file path="ecoli.nt.nni" vpath="ecoli.nt.nni"/>
    <file path="ecoli.nt.nsd" vpath="ecoli.nt.nsd"/>
    <file path="ecoli.nt.nsi" vpath="ecoli.nt.nsi"/>
    <file path="ecoli.nt.nsq" vpath="ecoli.nt.nsq"/>
  </sharedFiles>
  <task>
    <inputs>
      <file path="seq($s).txt" vpath="seq($s).txt"/>
    </inputs>
    <outputs>
      <file path="output($s).txt" vpath="output($s).txt"/>
    </outputs>
    <commands>
      <execute cmd="blastall.exe" args="-p ($p) -d ($d) -i seq($s).txt -o output($s).txt"/>
    </commands>
  </task>
</psm>
```

FIGURE 7.3

Aneka parameter sweep file.

The file is an XML document containing several sections, the most important of which are shared Files, parameters, and task. Parameters contains the definition of the parameters that will customize the template task.
Two different types of parameters are defined: a single value and a range parameter. The shared Files section contains the files that are required to execute the task;
The task has a collection of input and output files for which local and remote paths are defined, as well as a collection of commands.

7.2.3 MPI applications

Message Passing Interface (MPI) is a specification for developing parallel programs that communicate by exchanging messages.

MPI has originated as an attempt to create common ground from the several distributed shared memory and message-passing infrastructures available for distributed computing. Now a days, MPI has become a de facto standard for developing portable and efficient message-passing HPC applications.

MPI provides developers with a set of routines that:

- Manage the distributed environment where MPI programs are executed
- Provide facilities for point-to-point communication
- Provide facilities for group communication
- Provide support for data structure definition and memory allocation
- Provide basic support for synchronization with blocking calls

The general reference architecture is depicted in Figure 7.4. A distributed application in MPI is composed of a collection of MPI processes that are executed in parallel in a distributed infrastructure that supports MPI.

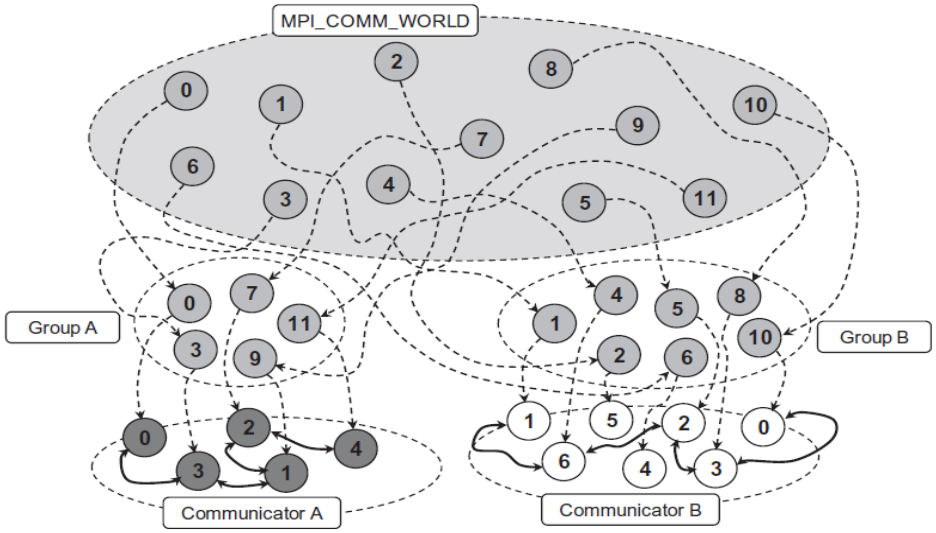


FIGURE 7.4

MPI reference scenario.

MPI applications that share the same MPI runtime are by default as part of a global group called MPI_COMM_WORLD. Within this group, all the distributed processes have a unique identifier that allows the MPI runtime to localize and address them.

Each MPI process is assigned a rank within the group.

The rank is a unique identifier that allows processes to communicate with each other within a group.

To create an MPI application it is necessary to define the code for the MPI process that will be executed in parallel. This program has, in general, the structure described in **Figure 7.5**.

The section of code that is executed in parallel is clearly identified by two operations that set up the MPI environment and shut it down, respectively.

In the code section, it is possible to use all the MPI functions to send or receive messages in either asynchronous or synchronous mode.

The diagram in **Figure 7.5** might suggest that the MPI might allow the definition of completely symmetrical applications, since the portion of code executed in each node is the same.

A common model used in MPI is the master-worker model, where by one MPI process coordinates the execution of others that perform the same task.

Once the program has been defined in one of the available MPI implementations, it is compiled with a modified version of the compiler for the language.

The output of the compilation process can be run as a distributed application by using a specific tool provided with the MPI implementation.

One of the most popular MPI software environments is developed by the Argonne National Laboratory in the United States.

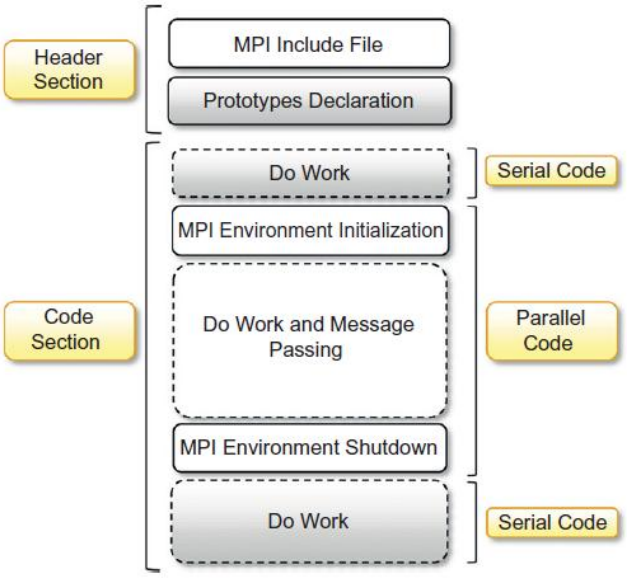


FIGURE 7.5
MPI program structure.

7.2.4 Workflow applications with task dependencies

Workflow applications are characterized by a collection of tasks that exhibit dependencies among them.
Such dependencies, which are mostly data dependencies determine the way in which the applications are scheduled as well as where they are scheduled.

1 What is a workflow?

A workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant (a resource; human or machine) to another for action, according to a set of procedural rules.
The concept of workflow as a structured execution of tasks that have dependencies on each other has demonstrated itself to be useful for expressing many scientific experiments and gave birth to the idea of scientific workflow.
In the case of scientific workflows, the process is identified by an application to run, the elements that are passed among participants are mostly tasks and data, and the participants are mostly computing or storage nodes. The set of procedural rules is defined by a workflow definition scheme that guides the scheduling of the application.
A scientific workflow generally involves data management, analysis, simulation, and middleware supporting the execution of the workflow.
A scientific workflow is generally expressed by a directed acyclic graph (DAG), which defines the dependencies among tasks or operations.
The nodes on the DAG represent the tasks to be executed in a workflow application; the arcs connecting the nodes identify the dependencies among tasks and the data paths that connect the tasks.
The most common dependency that is realized through a DAG is data dependency, which means that the output files of a task constitute the input files of another task.

The DAG in **Figure 7.6** describes a sample Montage workflow. Montage is a toolkit for assembling images into mosaics; it has been specially designed to support astronomers in composing the images taken from different telescopes or points of view into a coherent image.
The workflow depicted in **Figure 7.6** describes the general process for composing a mosaic; the labels on the right describe the different tasks that have to be performed to compose a mosaic.
In the case presented in the diagram, a mosaic is composed of seven images.
For each of the image files, the following process has to be performed: image file transfer, reprojection, calculation of the difference, and common plane placement.
Therefore, each of the images can be processed in parallel for these tasks.

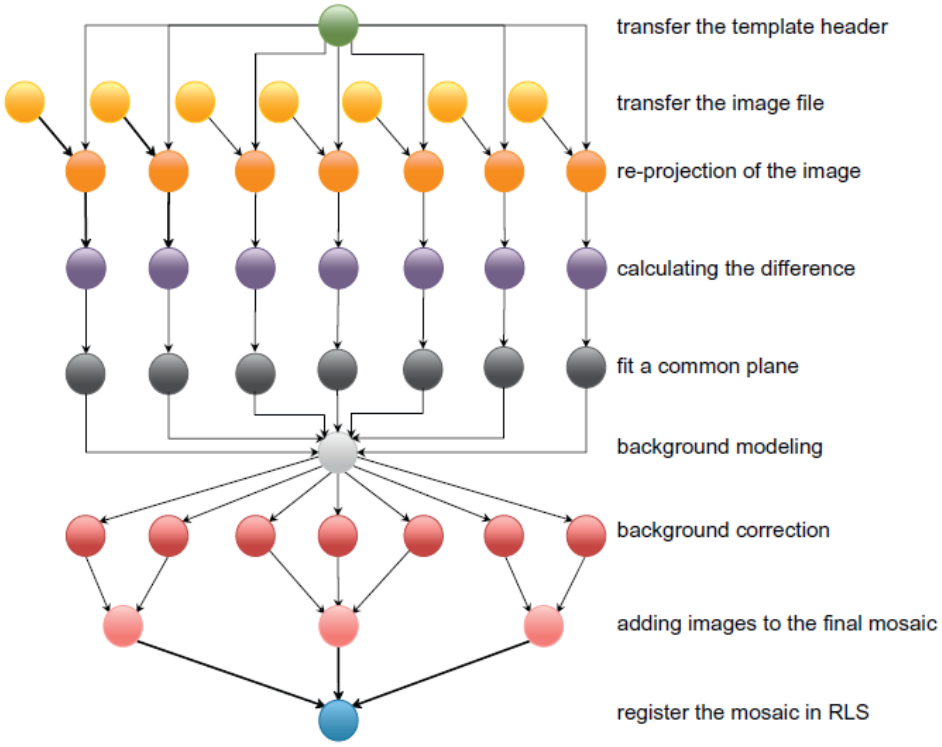


FIGURE 7.6
Sample Montage workflow.

2 Workflow technologies

Business-oriented computing workflows are defined as compositions of services.

There are specific languages and standards for the definition of workflows, such as Business Process Execution Language (BPEL).

An abstract reference model for a workflow management system, as depicted in **Figure 7.7**.

Design tools allow users to visually compose a workflow application.

This specification is stored in the form of an XML document based on a specific workflow language and constitutes the input of the workflow engine, which controls the execution of the workflow by leveraging a distributed infrastructure.

The workflow engine is a client- side component that might interact directly with resources or with one or several middleware components for executing the workflow.

Some of the most relevant technologies for designing and executing workflow-based applications are:

1. **Kepler,**
2. **DAGMan,**
3. **Cloudbus Workflow Management System,** and
4. **Offspring.**

1. Kepler

Kepler is an open-source scientific workflow engine.

The system is based on the Ptolemy II system, which provides a solid platform for developing dataflow-oriented workflows.

Kepler provides a design environment based on the concept of actors, which are reusable and independent blocks of computation such as Web services, data- base calls.

The connection between actors is made with ports.

An actor consumes data from the input ports and writes data/results to the output ports.

Kepler supports different models, such as synchronous and asynchronous models.

The workflow specification is expressed using a proprietary XML language.

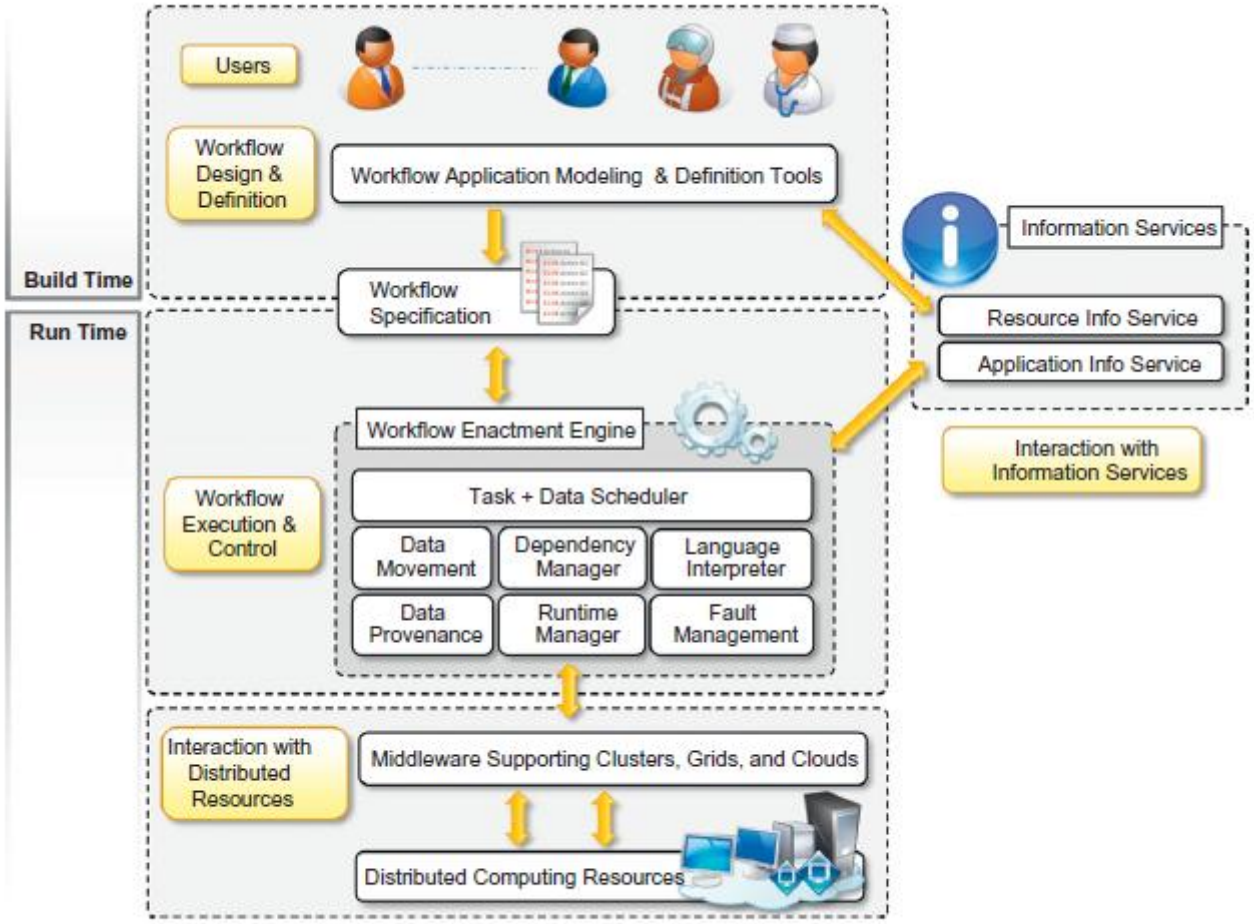


FIGURE 7.7

Abstract model of a workflow system.

2. DAGMan

DAGMan (Directed Acyclic Graph Manager) constitutes an extension to the Condor scheduler to handle job interdependencies.

DAGMan acts as a metascheduler for Condor by submitting the jobs to the scheduler in the appropriate order.

The input of DAGMan is a simple text file that contains the information about the jobs, pointers to their job submission files, and the dependencies among jobs.

3. Cloudbus Workflow Management System

Cloudbus Workflow Management System (WfMS) is a middleware platform built for managing large application workflows on distributed computing platforms such as grids and clouds.

It comprises software tools that help end users compose, schedule, execute, and monitor workflow applications through a Web-based portal.

The portal provides the capability of uploading workflows or defining new ones with a graphical editor.

To execute workflows, WfMS relies on the Gridbus Broker, a grid/cloud resource broker that supports the execution of applications with quality-of- service (QoS) attributes.

4. Offspring

It offers a programming-based approach to developing workflows.

Users can develop strategies and plug them into the environment, which will execute them by leveraging a specific distribution engine.

The advantage provided by Offspring is the ability to define dynamic workflows.

This strategy represents a semi structured workflow that can change its behaviour at runtime according to the execution of specific tasks.

This allows developers to dynamically control the dependencies of tasks at runtime.

Offspring supports integration with any distributed computing middleware that can manage a simple bag-of-tasks application.

Offspring allows the definition of workflows in the form of plug-ins.

7.3 Aneka task-based programming

Aneka provides support for all the flavors of task-based programming by means of the Task Programming Model, which constitutes the basic support given by the framework for supporting the execution of bag-of-tasks applications.

Task programming is realized through the abstraction of the Aneka.Tasks.ITask. By using this abstraction as a basis support for execution of legacy applications, parameter sweep applications and workflows have been integrated into the framework.

7.3.1 Task programming model

The Task Programming Model provides a very intuitive abstraction for quickly developing distributed applications on top of Aneka.

It provides a minimum set of APIs that are mostly centered on the Aneka.Tasks.ITask interface.

Figure 7.8 provides an overall view of the components of the Task Programming Model and their roles during application execution.

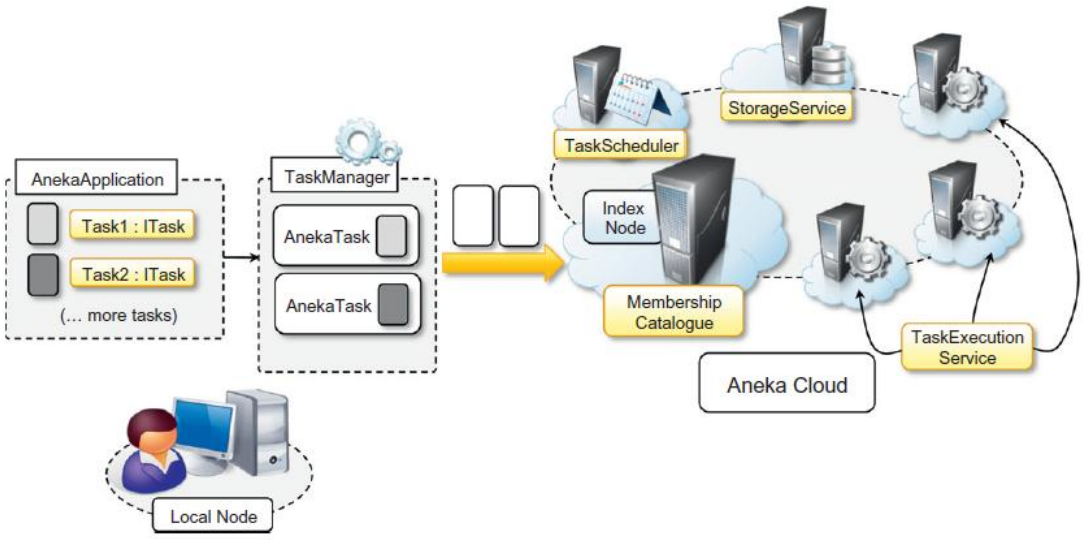


FIGURE 7.8

Task programming model scenario.

Developers create distributed applications in terms of ITask instances, the collective execution of which describes a running application.

These tasks, together with all the required dependencies (data files and libraries), are grouped and managed through the Aneka Application class, which is specialized to support the execution of tasks.

Two other components, AnekaTask and TaskManager, constitute the client-side view of a task-based application. The former constitutes the runtime wrapper Aneka uses to represent a task within the middleware; the latter is the underlying component that interacts with Aneka, submits the tasks, monitors their execution, and collects the results.

In the middleware, four services coordinate their activities inorder to execute task-based applications. These are MembershipCatalogue, TaskScheduler, ExecutionService, and StorageService.

MembershipCatalogue constitutes the main access point of the cloud and acts as a service directory to locate the TaskScheduler service that is incharge of managing the execution of task-based applications.

Its main responsibility is to allocate task instances to resources featuring the Execution Service for task execution and for monitoring task state.

7.3.2 Developing applications with the task model

Execution of task-based applications involves several components.

The development of such applications is limited to the following operations:

- Defining classes implementing the ITask interface
- Creating a properly configured AnekaApplication instance
- Creating ITask instances and wrapping them into AnekaTask instances
- Executing the application and waiting for its completion.

1. ITask and AnekaTask
2. Controlling task execution
3. File management
4. Task libraries
5. Web services integration

1. ITask and AnekaTask

All the client-side features for developing task-based applications with Aneka are contained in the Aneka.Tasks namespace (Aneka.Tasks.dll).

The most important component for designing tasks is the ITask interface, which is defined in **Listing7.1**.

This interface exposes only one method: Execute. The method is invoked in order to execute the task on the remote node.

```
namespace Aneka.Tasks
{
    ///<summary>
    ///Interface ITask. Defines the interface for implementing a task.
    ///</summary>
    public interface ITask
    {
        ///<summary>
        ///Executes the sine function.
        ///</summary>
        public void Execute();
    }
}
```

LISTING 7.1

ITask interface.

The ITask interface provides a programming approach for developing native tasks, which means tasks implemented in any of the supported programming languages of the .NET framework.

The restrictions on implementing task classes are minimal; they need to be serializable, since task instances are created and moved over the network.

ITask provides minimum restrictions on how to implement a task class and decouples the specific operation of the task from the runtime wrapper classes.

It is required for managing tasks within Aneka. This role is performed by the AnekaTask class that represents the task instance in accordance with the Aneka application model APIs. This class extends the Aneka.Entity.WorkUnit class and provides the feature for embedding ITask instances.

AnekaTask is mostly used internally, and for end users it provides facilities for specifying input and output files for the task.

Listing 7.2 describes a simple implementation of a task class that computes the Gaussian distribution for a given point x.

```
// File: GaussTask.cs
using System;
using Aneka.Tasks;

namespace GaussSample
{
    /// <summary>
    /// Class GaussTask. Implements the ITask interface for computing the Gauss function.
    /// </summary>
    [Serializable]
    public class GaussTask : ITask
    {
        /// <summary>
        /// Input value.
        /// </summary>
        private double x;
        /// <summary>
        /// Gets the input value of the Gauss function.
        /// </summary>
        public double X { get { return this.x; } set { this.x = value; } }
        /// <summary>
        /// Result value.
        /// </summary>
        private double y;
        /// <summary>
        /// Gets the result value of the Gauss function.
        /// </summary>
        public double Y { get { return this.y; } set { this.y = value; } }

        /// <summary>
        /// Executes the Gauss function.
        /// </summary>
        public void Execute()
        {
            this.y = Math.Exp(-this.x*this.x);
        }
    }
}
```

LISTING 7.2

ITask interface implementation.

Listing 7.3 describes how to wrap an ITask instance into an AnekaTask. It also shows how to add input and output files specific to a given task. The Task Programming Model leverages the basic capabilities for file management that belong to the WorkUnit class, from which the AnekaTask class inherits.
WorkUnit has two collections of files, Input Files and Output Files; developers can add files to these collections and the runtime environment will automatically move these files where it is necessary.
Input files will be staged into the Aneka Cloud and moved to the remote node where the task is executed.
Output files will be collected from the execution node and moved to the local machine or a remote FTP server.

```
// create a Gauss task and wraps it into an AnekaTaskinstance
GaussTaskgauss = new GaussTask();
AnekaTask task = newAnekaTask(gauss);
// add one input and one output files
task.AddFile("input.txt", FileDataType.Input, FileAttributes.Local);
task.AddFile("result.txt", FileDataType.Output, FileAttributes.Local);
```

LISTING 7.3

Wrapping an *ITask* into an *AnekaTask* Instance.

2. Controlling task execution

Task classes and AnekaTask define the computation logic of a task-based application.
Aneka Application class provides the basic feature for implementing the coordination logic of the

application.
In task programming, it assumes the form of Aneka Application<AnekaTask, TaskManager>.
The operations provided for the task model are:
• Static and dynamic task submission
• Application state and task state monitoring
• Event-based notification of task completion or failure.

Static submission is a very common pattern in the case of task-based applications, and it involves the creation of all the tasks that need to be executed in one loop and their submission as a single bag.
Dynamic submission of tasks is a more efficient technique and involves the submission of tasks as a result of the event-based notification mechanism implemented in the Aneka Application class.

Listing 7.4 shows how to create and submit 400 Gauss tasks as a bag by using the static submission approach.
Each task can be referenced using its unique identifier (Work Unit.Id) by the indexer operator [] applied to the application class.
In the case of static submission, the tasks are added to the application, and the method SubmitExecution() is called.

```
// get an instance of the Configuration class from file
Configuration conf = Configuration.GetConfiguration("conf.xml");
// specify that the submission of task is static (all at once)
conf.SingleSubmission = true;
AnekaApplication<AnekaTask, TaskManager> app =
new AnekaApplication<Task,TaskManager>(conf);
for(int i=0; i<400; i++)
{
   GaussTaskgauss = new GaussTask();
   gauss.X = i;
   AnekaTask task = new AnekaTask(gauss);
   // add the task to the bag of work units to submit
   app.AddWorkunit(task);
}
// submit the entire bag
app.SubmitExecution();
```

LISTING 7.4
Static task submission.

A different scenario is constituted by dynamic submission, where tasks are submitted as a result of other events that occur during the execution—for example, the completion or the failure of previously submitted tasks or other conditions that are not related to the interaction of Aneka.
Listing 7.5 extends the previous example and implements a dynamic task submission strategy for refining the computation of Gaussian distribution.

To capture the failure and the completion of tasks, it is necessary to listen to the events WorkUnitFailed and WorkUnitFinished.
This class exposes a WorkUnit property that, if not null, gives access to the task instance. The event handler for the task failure simply dumps the information that the task is failed to the console with, if possible, additional information about the error that occurred.
The event handler for task completion checks whether the task completed was submitted within the original bag, and in this case submits another task by using the ExecuteWorkUnit(AnekaTasktask) method.
To discriminate tasks submitted within the initial bag and other tasks, the value of GaussTask.X is used.
If X contains a value with no fractional digits, it is an initial task; otherwise, it is not.

In designing the coordination logic of the application, it is important to note that the task sub- mission identifies an asynchronous execution pattern, which means that the method SubmitExecution, as well as the method ExecuteWorkUnit, returns when the submission of tasks is completed, but not the actual completion of tasks.
This requires the developer to put in place the proper synchronization logic to let the main thread of the application wait until all the tasks are terminated and the application is completed.

This behaviour can be implemented using the synchronization APIs provided by the System.Threading namespace: System.Threading.AutoResetEvent or System.Threading.ManualResetEvent. These two APIs, together with a minimal logic, count all the tasks to be collected and signal the main thread once all tasks are terminated.

```
///<summary>
///Main method for submitting tasks.
///</summary>
public void SubmitApplication()
{
   // get an instance of the Configuration class from file
   Configuration conf = Configuration.GetConfiguration("conf.xml");
   // specify that the submission of task is dynamic
   conf.SingleSubmission = false;
   AnekaApplication<AnekaTask, TaskManager> app =
                  new AnekaApplication<Task,TaskManager>(conf);
   // attach methods to the event handler that notify the client code
   // when tasks are completed or failed
   app.WorkUnitFailed +=
                  new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFailed);
   app.WorkUnitFinished +=
                  new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFinished);
   for(int i=0; i<400; i++)
   {
      GaussTask gauss = new GaussTask();
      gauss.X = i;
      AnekaTask task = newAnekaTask(gauss);
      // add the task to the bag of work units to submit
      app.AddWorkunit(task);
   }
   // submit the entire bag
   app.SubmitExecution();
}
/// <summary>
/// Event handler for task failure.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFailed(object sender, WorkUnitEventArgs<AnekaTask>args)
{
   // do nothing, we are not interested in task failure at the moment
   // just dump to console the failure.
   if (args.WorkUnit != null)
   {
      Exception error = args.WorkUnit.Exception;
      Console.WriteLine("Task {0} failed - Exception: {1}",
      args.WorkUnit.Name, (error == null ? "[Not given]" : error.Message);
   }
}
```

LISTING 7.5

Dynamic task submission.

Listing 7.6 provides a complete implementation of the task submission program, implementing dynamic submission and the appropriate synchronization logic.

The GaussApp application keeps track of the number of currently running tasks by using the taskCount field. When this value reaches zero, there are no more tasks to wait for and the application is stopped by calling StopExecution. This method fires the ApplicationFinished event whose event handler unblocks them a in thread by signalling the semaphore.

A final aspect that can be considered for controlling the execution of the task application is the resubmission strategy that is used. By default the configuration of the application sets the resubmis- sion strategy as manual.

In automatic resubmission, Aneka will keep resubmitting the task until a maximum number of attempts is reached. If the task keeps failing, the task failure event will eventually be fired.

```
// File: GaussApp.cs
using System;
using System.Threading;

using Aneka.Entity;
using Aneka.Tasks;

namespace GaussSample
{
   /// <summary>
   /// Class GaussApp. Defines the coordination logic of the
   /// distributed application for computing the gaussian distribution.
   /// </summary>
   public class GaussApp
   {
      /// <summary>
      /// Semaphore used to make the main thread wait while
      /// all the tasks are terminated.
      /// </summary>
      private ManualResetEvent semaphore;
      /// <summary>
      /// Counter of the running tasks.
      /// </summary>
      private int taskCount = 0;
      /// <summary>
      /// Aneka application instance.
      /// </summary>
      private AnekaApplication<AnekaTask, TaskManager> app;

      /// <summary>
      /// Main entry point for the application.
      /// </summary>
      /// <param name="args">An array of strings containing the command line.</param>
      public static void Main(string[] args)
      {
         try
         {
            // initialize the logging system
            Logger.Start();

            string confFile = "conf.xml";
            if (args.Length > 0)
            {
               confFile = args[0];
            }
            // get an instance of the Configuration class from file
            Configuration conf = Configuration.GetConfiguration(confFile);
            // create an instance of the GaussApp and starts its execution
            // with the given configuration instance
            GaussApp application = new GaussApp();
            application.SubmitApplication(conf);
         }

         catch(Exception ex)
         {
            IOUtil.DumpErrorReport(ex, "Fatal error while executing application.");
         }
         finally
         {
            // terminate the logging thread
            Logger.Stop();

            }
         }
```

```
/// <summary>
/// Application submission method.
/// </summary>
/// <param name="conf">Application configuration.</param>
public void SubmitApplication(Configuration conf)
{
   // initialize the semaphore and the number of
   // task initially submitted
   this.semaphore = new ManualResetEvent(false);
   this.taskCount = 400;

   // specify that the submission of task is dynamic
   conf.SingleSubmission = false;
   this.app = new AnekaApplication<Task,TaskManager>(conf);
   // attach methods to the event handler that notify the client code
   // when tasks are completed or failed
   this.app.WorkUnitFailed +=
        new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFailed);
   this.app.WorkUnitFinished +=
        new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFinished);

   // attach the method OnAppFinished to the Finished event so we can capture
   // the application termination condition, this event will be fired in case of
   // both static application submission or dynamic application submission
   app.Finished += new EventHandler<ApplicationEventArgs>(this.OnAppFinished);

   for(int i=0; i<400; i++)
   {
      GaussTask gauss = new GaussTask();
      gauss.X = i;
      AnekaTask task = newAnekaTask(gauss);
      // add the task to the bag of work units to submit
      app.AddWorkunit(task);
   }
   // submit the entire bag
   app.SubmitExecution();

   // wait until signaled, once the thread is signaled the application is completed
   this.semaphore.Wait();
}

/// <summary>
/// Event handler for task failure.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFailed(object sender, WorkUnitEventArgs<AnekaTask> args)
{
   // do nothing, we are not interested in task failure at the moment
   // just dump to console the failure.
   if (args.WorkUnit != null)
   {
      Exception error = args.WorkUnit.Exception;
      Console.WriteLine("Task {0} failed - Exception: {1}",
      args.WorkUnit.Name,
      (error == null ? "[Not given]" : error.Message);
   }
   // we do not have to synchronize this operation because
   // events handlers are run all in the same thread, and there
   // will not be other threads updating this variable

   this.taskCount--;
   if (this.taskCount == 0)
   {
      this.app.StopExecution();
   }
}
```

```
    /// <summary>
    /// Event handler for task completion.
    /// </summary>
    /// <param name="sender">Event source: the application instance.</param>
    /// <param name="args">Event arguments.</param>
    private void OnWorkUnitFinished(object sender, WorkUnitEventArgs<AnekaTask> args)
    {
       // we do not have to synchronize this operation because
       // events handlers are run all in the same thread, and there
       // will not be other threads updating this variable
       this.taskCount--;

       // if the task is completed for sure we have a WorkUnit instance
       // and we do not need to check as we did before.
       GaussTask gauss = (GaussTask) args.WorkUnit.Task;
       // we check whether it is an initially submitted task or a task
       // that we submitted as a reaction to the completion of another task
       if (task.X - Math.Abs(task.X) == 0)
       {
          // ok it was an original task, then we increment of 0.5 the
          // value of X and submit another task
          GaussTask fraq = GaussTask();
          fraq.X = gauss.X + 0.5;
          AnekaTask task =new AnekaTask(fraq);

                this.taskCount++;
                // we call the ExecuteWorkUnit method that is used
                // for dynamic submission
                app.ExecuteWorkUnit(task);
             }
             Console.WriteLine("Task {0} completed - [X:{1},Y:{2}]",
                               args.WorkUnit.Name, gauss.X, gauss.Y);
             if (this.taskCount == 0)
             {
                this.app.StopExecution();
             }
          }
          /// <summary>
          /// Event handler for the application termination.
          /// </summary>
          /// <param name="sender">Event source: the application instance.</param>
          /// <param name="args">Event arguments.</param>
          private void OnWorkUnitFinished(object sender, ApplicationEventArgs args)
          {
              // unblock the main thread, because we have identified the termination
              // of the application
              this.semaphore.Set();
          }
       }
    }
```

LISTING 7.6

GaussApplication.

3. File management

Task-based applications normally deal with files to perform their operations.

Files may constitute input data for tasks, may contain the result of a computation, or may represent executable code or library dependencies.

Any model based on the WorkUnit and ApplicationBase classes has built-in support for file management.

A fundamental component for the management of files is the FileData class, which constitutes the logic representation of physical files, as defined in the Aneka.Data.Entity namespace (Aneka. Data.dll).

A File Data instance provides information about a file:

• Its nature: whether it is a shared file, an input file, or an outputfile

• Its path both in the local and in the remote file system, including a different name

• A collection of attributes that provides other information.

Listing7.7 demonstrates how to add file dependencies to the application and to tasks. It is possible to add both FileData instances, thus having more control of the information attached to the file.

```
// get an instance of the Configuration class from file
Configuration conf = Configuration.GetConfiguration("conf.xml");
AnekaApplication<Task,TaskManager>app =new AnekaApplication<Task,TaskManager>(conf);

// attach shared files with different methods by using the FileData class and directly
// using the API provided by the AnekaApplication class

// create a local shared file whose local and remote name is "pi.tab"
FileDatapiTab = newFileData("pi.tab",FileDataType.Shared);
app.AddSharedFile(piTab);
// once the file is added to the collection of shared files, its OwnerId property
// references app.Id

// create a remote shared file by specifying the attributes whose name is "pi.dat"
FileDatapiDat = newFileData("pi.dat",FileDataType.Shared, FileAttributes.None);
// the StorageBucketId property points a specific configuration section that is
// used to store the information for retrieving the file from the remote server
piDat.StorageBucketId = "FTPStore";
app.AddSharedFile(piDat);
// once the file is added to the collection of shared files, its OwnerId property
// references app.Id

// adds a local shared file
app.AddSharedFile("pi.xml");

for(int i=0; i<400; i++)
{
   GaussTask gauss = new GaussTask();
   gauss.X = i;
   AnekaTask task = newAnekaTask(gauss);

   // adds a local input file for the current task whose name is "<i>.txt"
   // where <i> is the value of the loop variable
   FileData input = new FileData(string.Format("{0}.txt", i);
   FileDataType.Input, FileAttributes.Local);
   // once transferred to the remote node, the file will have the name
   // "input.txt". Since tasks are executed in separate directories there
   // will be no name clashing
   input.VirtualPath ="input.txt";
   task.AddFile(input);
   // once the file is added to the task, it will be stored in the InputFiles
   // collection and its OwnerId property will referenced task.Id

   // adds anoutput file for the current task whose name is "out.txt" that will
   // be stored on S3
```

```
    FileData output =new FileData("out.txt", FileDataType.Input, FileAttributes.None);
    // once transferred to the remote server, the file will have the name
    // "<i>.out"where <i> is the value of the loop variable. In this way we
    // easily avoid name clashing while storing output files into a single
    // directory
    output.VirtualPath =string.Format("{0}.out", i);
    output.StorageBucketId = "S3Store";
    task.AddFile(output);
    // once the file is added to the task, it will be stored in the InputFiles
    // collection and its OwnerId property will referenced task.Id

    // adds a localoutput file for the current task whose name is "trace.log".
    // The file is optional, this means that if after the execution of the task the file

    // is not present no exception or task failure will be risen.
    FileData trace =new FileData("trace.log", FileDataType.Input,
    FileAttributes.Local | FileAttributes.Optional);
    // once transferred to the local machine, the file will have the name
    // "<i>.log"where <i> is the value of the loop variable. In this way we
    // easily avoid name clashing while storing output files into a single
    // directory
    trace.VirtualPath =string.Format("{0}.log", i);
    task.AddFile(trace);
    // once the file is added to the task, it will be stored in the InputFiles
    // collection and its OwnerId property will referenced task.Id

    // add the task to the bag of work units to submit
    app.AddWorkunit(task);
}

// submit the entire bag, files will be moved automatically by the Aneka APIs
app.SubmitExecution();
```

LISTING 7.7

File dependencies management.

The general interaction flow for file management is as follows:

• Once the application is submitted, the shared files are staged into the Aneka Cloud.

• If the file is local it will be searched into the directory location identified by the property Configuration. Workspace; if the file is remote, the specific configuration settings mapped by the FileData. Storage Bucket Id property will be used to access the remote server and stage in the file.

• If there is any failure in staging input files, the application will be terminated with an error.

• For each of the tasks belonging to the application, the corresponding input files are staged into the Aneka Cloud, as is done for shared files.

• Once the task is dispatched for execution to a remote node, the runtime will transfer all the shared files of the application and the input files of the task into the working directory of the task and eventually get renamed if the FileData.VirtualPath property is not null.

• After the execution of the task, the runtime will look for all the files that have been added into the WorkUnit.OutputFiles collection. If not null, the value of the FileData.VirtualPath property will be used to locate the files; otherwise, the FileData.FileName property will be the reference. All the files that do not contain the File Attributes. Optional attribute need to be present; otherwise, the execution of the task is classified as a failure.

• Despite the successful execution or the failure of a task, the runtime tries to collect and move to their respective destinations all the files that are found. Files that contain the File Attributes. Local attribute are moved to the local machine from where the application is saved and stored in the directory location identified by the property Configuration.Workspace. Files that have a Storage BucketId property set will be staged out to the corresponding remote server.

Listing 7.8 shows a sample configuration file containing the settings required to access the remote files

through the FTP and S3 protocols. Within the <Groups> tag, there is a specific group named StorageBuckets; this group maintains the configuration settings for each storage bucket that needs to be used in for file transfer.

```
<?xmlversion="1.0"encoding="utf-8"?>
<Aneka>
  <UseFileTransfer value="true" />
  <Workspace value="." />
  <SingleSubmission value="false" />
  <ResubmitMode value="Manual" />
  <PollingTime value="1000" />
  <LogMessages value="true" />
  <SchedulerUri value="tcp://localhost:9090/Aneka"/>
  <UserCredential type="Aneka.Security.UserCredentials" assembly="Aneka.dll">
    <UserCredentials username="Administrator" password=""/>
  </UserCredential>
  <Groups>
    <Group name="StorageBuckets">
      <Groups>
        <Group name="FTPStore">
          <Property name="Scheme" value="ftp"/>
          <Property name="Host" value="www.remoteftp.org"/>
          <Property name="Port" value="21"/>
          <Property name="Username" value="anonymous"/>
          <Property name="Password" value="nil"/>
        </Group>
        <Group name="S3Store">
          <Propertyname="Scheme" value="S3"/>
          <Propertyname="Host"value="www.remoteftp.org"/>
          <Propertyname="Port"value="21"/>
          <Propertyname="Username"value="anonymous"/>
          <Propertyname="Password"value="nil"/>
        </Group>
      </Groups>
    </Group>
  </Groups>
</Aneka>
```

LISTING 7.8

Aneka application configuration file.

4 Task libraries

Aneka provides a set of ready-to-use tasks for performing the most basic operations for remote file management.

These tasks are part of the Aneka.Tasks.BaseTasks namespace, which is part of the Aneka.Tasks.dll library.

The following operations are implemented:

• File copy. The Local Copy Task performs the copy of a file on the remote node; it takes a file as input and produces a copy of it under a different name or path.

• Legacy application execution. The Execute Task allows executing external and legacy applications by using the System.Diagnostics.Process class. It requires the location of the executable file to run, and it is also possible to specify command-line parameters.

• Substitute operation. The Substitute Task performs a search-and-replace operation within a given file by saving the resulting file under a different name.

• File deletion. The Delete Task deletes a file that is accessible through the file system on the remote node.

• Timed delay. The Wait Task introduces a timed delay. This task can be used in several scenarios; it can be used for profiling or for simulation of the execution.

• Task composition. The Composite Task implements the composite pattern and allows expressing a task as a composition of multiple tasks that are executed in sequence.

5 Web services integration

The task submission Web service is an additional component that can be deployed in any ASP.NET Web

server and that exposes a simple interface for job submission, which is compliant with the Aneka Application Model.
The task Web service provides an interface that is more compliant with the traditional way fostered by grid computing.
The reference scenario for Web-based submission is depicted in **Figure 7.9.**
Users create a distributed application instance on the cloud, they can submit jobs querying the status of the application or a single job.
It is up to the users to then terminate the application when all the jobs are completed or abort it if there is no need to complete job execution.

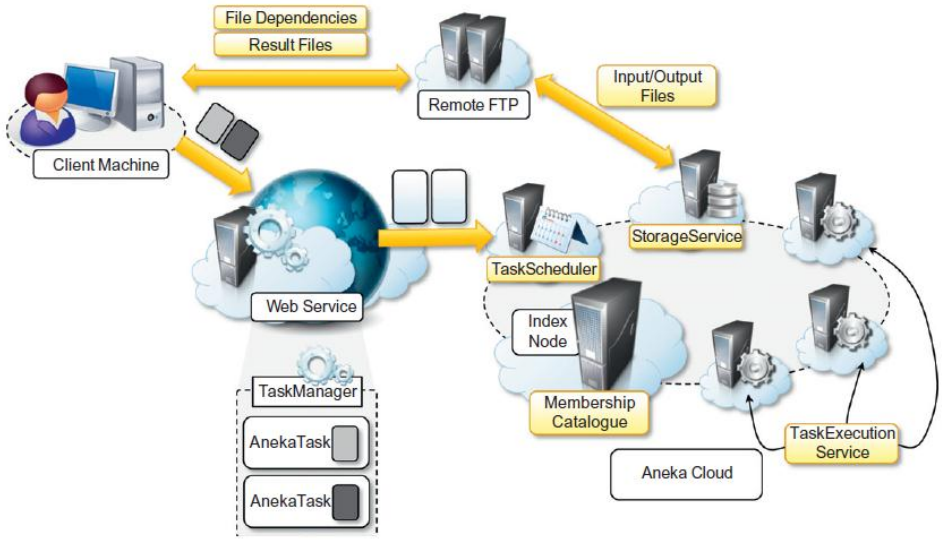


FIGURE 7.9
Web service submission scenario.

Operations supported through the Web service interface are the following:
• Local file copy on the remote node
• File deletion
• Legacy application execution through the common shell services
• Parameter substitution.

7.3.3 Developing a parameter sweep application

Aneka integrates support for parameter-sweeping applications on top of the task model by means of a collection of client components that allow developers to quickly prototype applications through either programming APIs or graphical user interfaces (GUIs).
The PSM is organized into several name spaces under the common root Aneka.PSM.
More precisely:
• Aneka.PSM.Core (Aneka.PSM.Core.dll) contains the base classes for defining a template task and the client components managing the generation of tasks, given the set of parameters.
• Aneka.PSM.Workbench (Aneka.PSM.Workbench.exe) and Aneka.PSM.Wizard (Aneka.PSM. Wizard.dll) contain the user interface support for designing and monitoring parameter sweep applications. Mostly they contain the classes and components required by the Design Explorer, which is the main GUI for developing parameter sweep applications.
• Aneka.PSM.Console (Aneka.PSM.Console.exe) contains the components and classes supporting the execution of parameter sweep applications in console mode.

1 Object model
2 Development and monitoring tools

1 Object model

The fundamental elements of the Parameter Sweep Model are defined in the Aneka.PSM.Core namespace.
This model introduces the concept of job (Aneka.PSM.Core.PSM JobInfo).
Figure 7.11 shows the most relevant components of the object model.

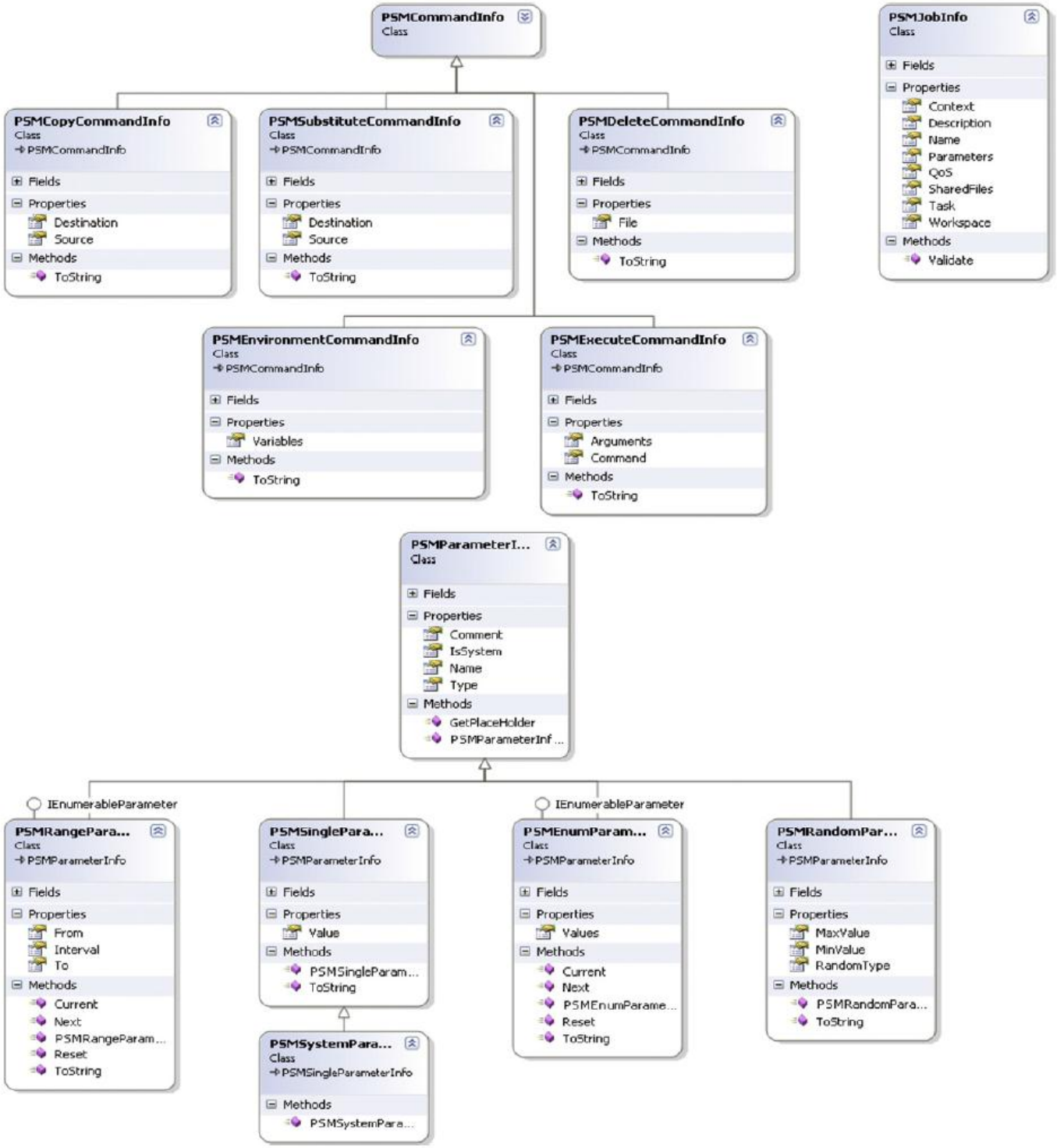


FIGURE 7.11

PSM object model (relevant classes).

Currently, it is possible to specify five different types of parameters:

• Constant parameter (PSM Single Parameter Info). This parameter identifies a specific value that is set at design time and will not change during the execution of the application.

• Range parameter (PSM Range Parameter Info). This parameter allows defining a range of allowed values, which might be integer or real. The parameter identifies a domain composed of discrete values and requires the specification of a lower bound, an upper bound, and a step for the generation of all the admissible values.

• Random parameter (PSM Random Parameter Info). This parameter allows the generation of a random value in between a given range defined by a lower and an upper bound.

• Enumeration parameter (PSM Enum Parameter Info). This parameter allows for specifying a discrete set of values of any type. It is useful to specify discrete sets that are not based on numeric values.

• System parameter (PSM System Parameter Info). This parameter allows for mapping specific value that

will be substituted at runtime while the task instance is executed on the remote node.

The available commands for composing the task template perform the following operations:
• Local file copy on the remote node (PSM Copy Command Info)
• Remote file deletion (PSM Delete Command Info)
• Execution of programs through the shell (PSM Execute Command Info)
• Environment variable setting on the remote node (PSM Environment Command Info)
• String pattern replacement within files (PSM Substitute Command Info)

A parameter sweep application is executed by means of a job manager (IJob Manager), which interfaces the developer with the underlying APIs of the task model.
Figure 7.12 shows the relationships among the PSM APIs, with a specific reference to the job manager, and the task model APIs.
The implementation of IJob Manager will then create a corresponding Aneka application instance and leverage the task model API to submit all the task instances generated from the template task. The interface also exposes facilities for controlling and monitoring the execution of the parameter sweep application as well as support for registering the statistics about the application.

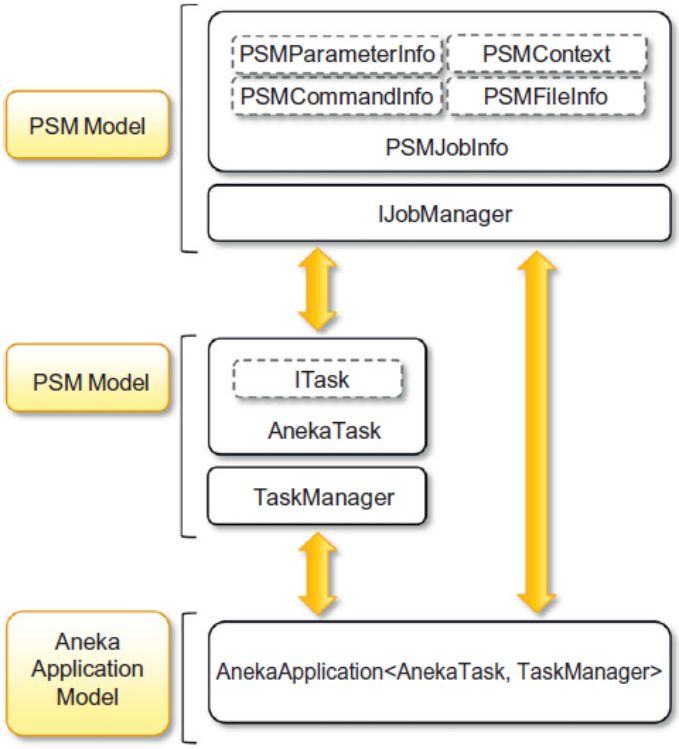


FIGURE 7.12
Parameter sweep model APIs.

2 Development and monitoring tools
The core libraries allow developers to directly program parameter sweep applications and embed them into other applications.
Additional tools simplify design and development of parameter sweep applications.
These tools are the:
Aneka Design Explorer and
The **Aneka PSM Console**.
Aneka Design Explorer
The Aneka Design Explorer is an integrated visual environment for quickly prototyping parameter sweep applications, executing them, and monitoring their status.
It provides a simple wizard that helps the user visually define any aspect of parameter sweep applications, such as file dependencies and result files, parameters, and template tasks.
The environment also provides a collection of components that help users monitor application execution, aggregate statistics about application execution, gain detailed task transition monitoring, and gain extensive

access to application logs.

The Aneka PSM Console

The Aneka PSM Console is a command-line utility designed to run parameter sweep applications in non-interactive mode.

The console offers a simplified interface for running applications with essential features for monitoring their execution.

7.3.4 Managing workflows

Two different workflow managers can leverage Aneka for task execution:

1. **The Workflow Engine** and
2. **Offspring**.

1. The Workflow Engine

The former leverages the task submission Web service exposed by Aneka; the latter directly interacts with the Aneka programming APIs.

The Workflow Engine plug-in for Aneka which allows client applications developed with any technology and language to leverage Aneka for task execution.

2. Offspring

Figure 7.13 describes the Offspring architecture. The system is composed of two types of components: **plug-ins** and a **distribution engine**.

Plug-ins are used to enrich the environment of features; the distribution engine represents access to the distributed computing infrastructure leveraged for task execution.

Auto Plugin provides facilities for the definition of workflows in terms of strategies.

A strategy generates the tasks that are submitted for execution and defines the logic, in terms of sequencing, coordination, and dependencies, used to submit the task through the engine.

The Strategy Controller, decouples the strategies from the distribution engine.

The connection with Aneka is realized through the Aneka Engine, which implements the operations of IDistribution Engine for the Aneka middleware and relies on the services exposed by the task model programming APIs.

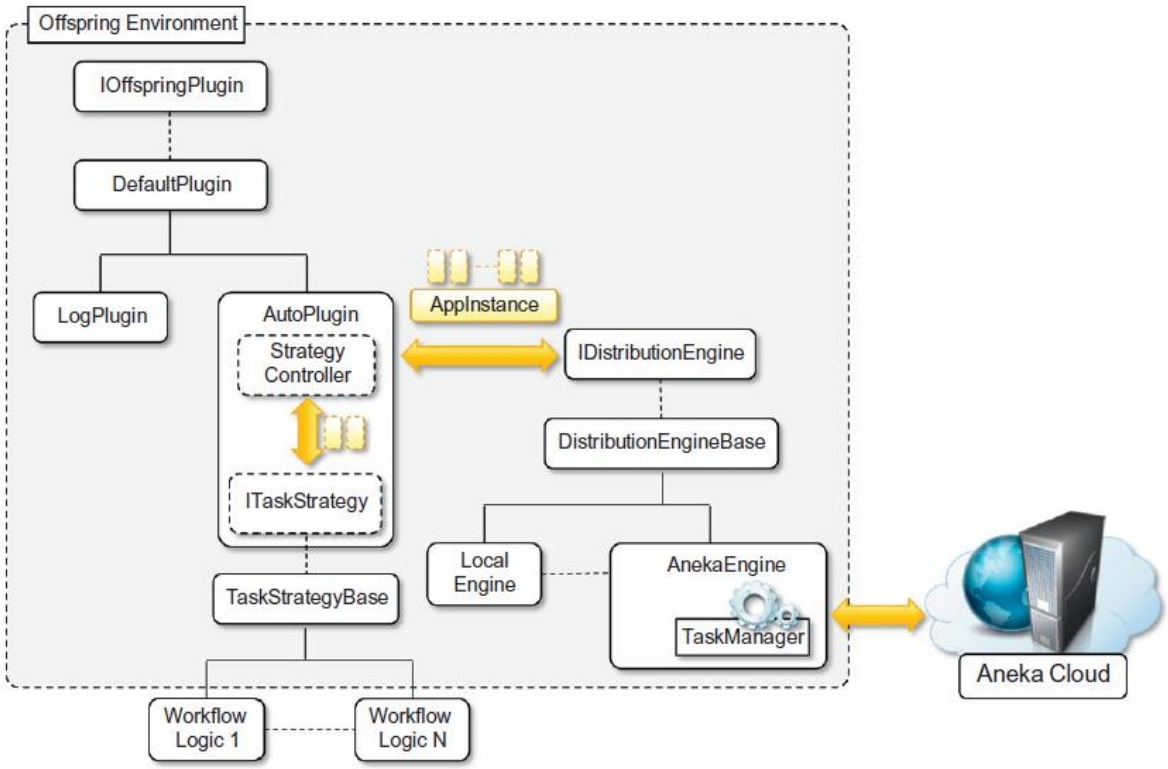


FIGURE 7.13

Offspring architecture.

Two different types of tasks can be defined: **native tasks** and **legacy tasks**.

Native tasks are completely implemented in managed code.

Legacy tasks manage file dependencies and wrap all the data necessary for the execution of legacy programs

on a remote node.
Figure7.14 describes the interactions among these components. Two main execution threads control the execution of a strategy.
A control thread manages the execution of the strategy, where as a monitoring thread collects the feedback from the distribution engine and allows for the dynamic reaction of the strategy to the execution of previously submitted tasks.
The execution of a strategy is composed of three macro steps: setup, execution, and finalization. The first step involves the setup of the strategy and the application mapping it. Correspondingly, the finalization step is in charge of releasing all the internal resources allocated by the strategy and shutting down the application.
The core of the workflow execution resides in the execution step, which is broken down into a set of iterations. During each of the iterations a collection of tasks is submitted; these tasks do not have dependencies from each other and can be executed in parallel. As soon as a task completes or fails, the strategy is queried to see whether a new set of tasks needs to be executed.
At the end of each iteration, the controller checks to see whether the strategy has completed the execution, and in this case, the finalization step is performed.

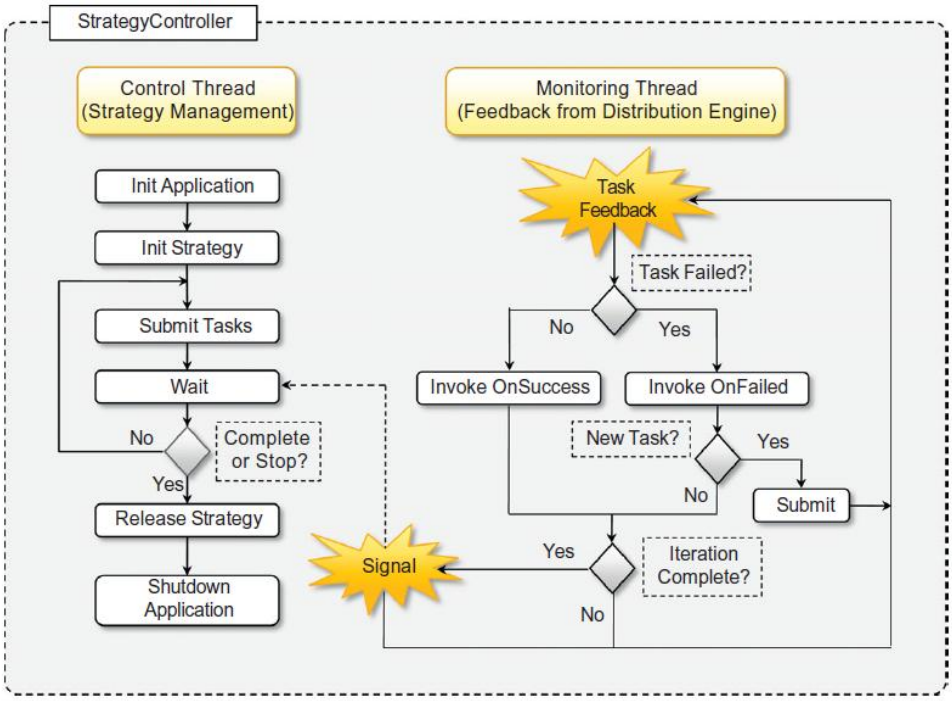


FIGURE 7.14
Workflow coordination.

Module - 4

Data-Intensive Computing: MapReduce Programming (Chapter - 8)

Data-intensive computing focuses on aa class of applications that deal with a large amount of data. Several application fields, ranging from computational 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 by providing more scalable and efficient storage architectures and a better performance in terms of data computation and processing.

This chapter characterizes the nature of data-intensive computing and presents an overview of the challenges introduced by production of large volumes of data and how they are handled by storage systems and computing models. It describes MapReduce, which is a popular programming model for creating data-intensive applications and their deployment on clouds.

8.1 What is data-intensive computing?

Data-intensive computing is concerned with **production, manipulation, and analysis** of **large-scale data** in the range of hundreds of **megabytes (MB) to petabytes (PB)** and **beyond**.

Dataset is commonly used to identify a collection of information elements that is relevant to one or more applications. Datasets are often maintained in repositories, which are infrastructures supporting the storage, retrieval, and indexing of large amounts of information.

To facilitate classification and search, relevant bits of information, called **metadata**, are attached to datasets.

Data-intensive computations occur in many application domains.

Computational science is one of the most popular ones. People conducting scientific simulations and experiments are often keen to produce, analyze, and process huge volumes of data. Hundreds of gigabytes of data are produced every second by telescopes mapping the sky; the collection of images of the sky easily reaches the scale of petabytes over a year.

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.

8.1.1 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

8.1.1 Characterizing data-intensive computations

Data-intensive applications dealS with huge volumes of data, also exhibit compute-intensive properties.

Figure 8.1 identifies the domain of data-intensive computing in the two upper quadrants of the graph.

Data-intensive applications handle datasets on the scale of multiple terabytes and petabytes.

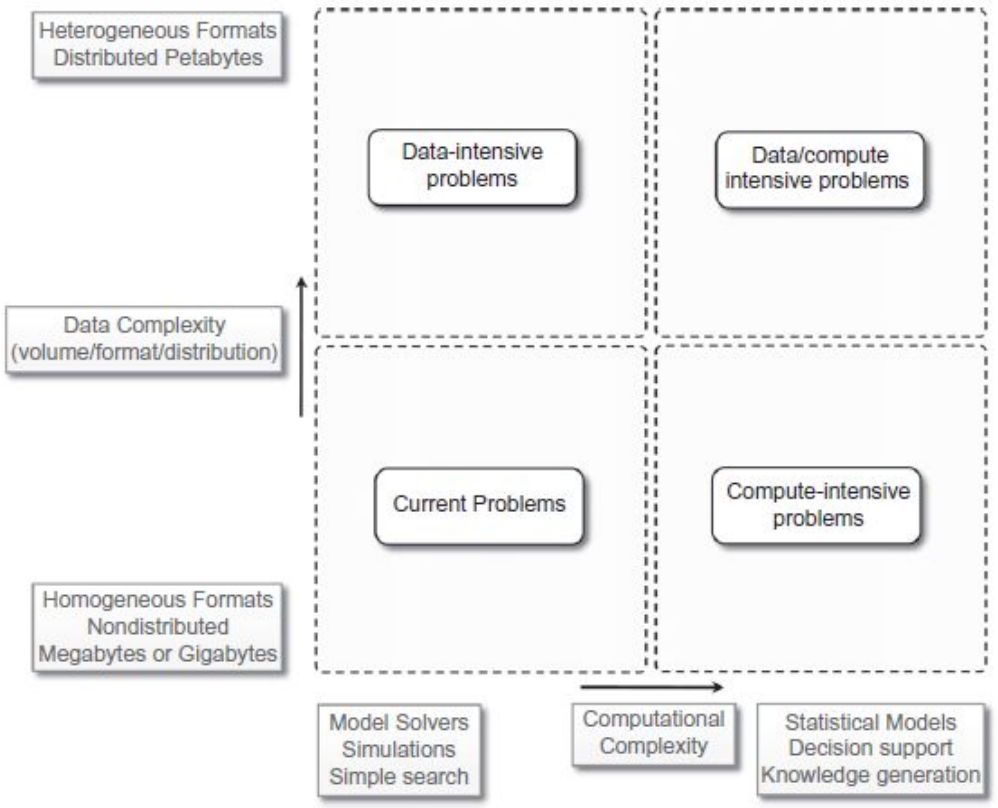


FIGURE 8.1

Data-intensive research issues.

8.1.2 Challenges ahead

The huge amount of data produced, analyzed, or stored imposes requirements on the supporting infrastructures and middleware that are hardly found in the traditional solutions.

Moving terabytes of data becomes an obstacle for high-performing computations.

Data partitioning, **content replication** and **scalable algorithms** help in improving the performance.

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

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** aimed at providing a better support for accessing in-memory multiterabyte 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** that are able to move the computation to where the data are located.
7. **Interconnection architectures** that provide better support for filtering multi gigabyte datastreams coming from high-speed networks and scientific instruments.
8. **Software integration** techniques that facilitate the combination of software modules running on different platforms to quickly form analytical pipelines.

8.1.3 Historical perspective

Data-intensive computing involves the production, management, and analysis of large volumes of data.

Support for data-intensive computations is provided by harnessing storage, networking, technologies, algorithms, and infrastructure software all together.

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 led the way.

Two years later, the potential of using high-speed wide area networks for enabling high-speed, TCP/IP-based distributed applications was demonstrated at **Supercomputing 1991 (SC91).**

Kaiser project, leveraged the **Wide Area Large Data Object (WALDO)** system, used to provide following capabilities:

1. automatic generation of metadata;
2. automatic cataloguing of data and metadata processing data in real time;
3. facilitation of cooperative research by providing local and remote users access to data; and
4. mechanisms to incorporate data into databases and other documents.

The **Distributed Parallel Storage System (DPSS)** was developed, later used to support TerraVision, a terrain visualization application that lets users explore and navigate a tridimensional real landscape.

Clipper project, the goal of designing and implementing a collection of independent, architecturally consistent service components to support data-intensive computing. The challenges addressed by Clipper project include management of computing resources, generation or consumption of high-rate and high-volume data flows, human interaction management, and aggregation of resources.

2 Data grids

Huge computational power and storage facilities could be obtained by harnessing heterogeneous resources across different administrative domains.

Data grids emerge as infrastructures that support data-intensive computing.

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.

Data grids offer two main functionalities:

- high-performance and reliable file transfer for moving large amounts of data, and
- scalable replica discovery and management mechanisms.

Data grids mostly provide storage and dataset management facilities as support for scientific experiments that produce huge volumes of data.

Datasets are replicated by infrastructure to provide better availability.

Data grids have their own characteristics and introduce new challenges:

1. **Massive datasets**. The size of datasets can easily be on the scale of gigabytes, terabytes, and beyond. It is therefore necessary to minimize latencies during bulk transfers, replicate content with appropriate strategies, and manage storage resources.
2. **Shared data collections.** Resource sharing includes distributed collections of data. For example, repositories can be used to both store and read data.
3. **Unified namespace.** Data grids impose a unified logical namespace where to locate data collections and resources. Every data element has a single logical name, which is eventually mapped to different physical filenames for the purpose of replication and accessibility.
4. **Access restrictions.** Even though one of the purposes of data grids is to facilitate sharing of results and data for experiments, some users might want to ensure confidentiality for their data and restrict access to them to their collaborators. Authentication and authorization in data grids involve both coarse-grained and fine-grained access control over shared data collections.

As a result, several scientific research fields, including high-energy physics, biology, and astronomy, leverage data grids.

3 Data clouds and "Big Data"

Together with the diffusion of cloud computing technologies that support data-intensive computations, the term **Big Data** has become popular. **Big Data** characterizes the nature of data-intensive computations today and currently identifies datasets that grow so large that they become complex to work with using on-hand database management tools.

In general, 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. Therefore, Big Data sizes are a constantly moving target, currently ranging from a few dozen tera-bytes to many petabytes of data in a single dataset.

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.

A data cloud is a combination of these components.

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

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

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

Ex 4: Greenplum uses a shared-nothing massively parallel processing (MPP) architecture based on commodity hardware.

4 Databases and data-intensive computing

Distributed databases are a collection of data stored at different sites of a computer network. Each site might expose a degree of autonomy, providing services for the execution of local applications, but also participating in the execution of a global application.

A distributed database can be created by splitting and scattering the data of an existing database over different sites or by federating together multiple existing databases. 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

Data-intensive computing concerns the development of applications that are mainly focused on processing large quantities of data.

Therefore, storage systems and programming models constitute a natural classification of the technologies supporting data-intensive computing.

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.

8.2.1 Storage systems

Traditionally, database management systems constituted the de facto storage.

Due to the explosion of unstructured data in the form of blogs, Web pages, software logs, and sensor readings, the relational model in its original formulation does not seem to be the preferred solution for supporting data analytics on a large scale.

Some factors contributing to change in database are:

A. **Growing of popularity of Big Data.** The management of large quantities of data is no longer a rare case but instead has become common in several fields: scientific computing, enterprise applications, media entertainment, natural language processing, and social network analysis.
B. **Growing importance of data analytics in the business chain.** The management of data is no longer considered a cost but a key element of business profit. This situation arises in popular social networks such as Facebook, which concentrate their focus on the management of user profiles, interests, and connections among people.
C. **Presence of data in several forms, not only structured.** As previously mentioned, what constitutes relevant information today exhibits a heterogeneous nature and appears in several forms and formats.
D. **New approaches and technologies for computing.** Cloud computing promises access to a massive amount of computing capacity on demand. This allows engineers to design software systems that incrementally scale to arbitrary degrees of parallelism.

1. High-performance distributed file systems and storage clouds

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

a. Lustre. The Lustre file system is a massively parallel distributed file system that covers the needs of a small workgroup of clusters to a large-scale computing cluster. The file system is used by several of the Top 500 supercomputing systems.

Lustre is 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 a metadata server that contains the metadata about the file system and a collection of object storage servers that are in charge of providing storage.

b. IBM General Parallel File System (GPFS). GPFS is the high-performance distributed file system developed by IBM that provides support for the RS/6000 supercomputer and Linux computing clusters. GPFS is a multiplatform distributed file system built over several years of academic research and provides advanced recovery mechanisms. GPFS is built on the concept of shared disks, in which a collection of disks is attached to the file system nodes by means of some 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.

c. Google File System (GFS). GFS is the storage infrastructure that supports the execution of distributed applications in Google's computing cloud.

GFS is designed with the following assumptions:

1. The system is built on top of commodity hardware that often fails.
2. The system stores a modest number of large files; multi-GB files are common and should be treated efficiently, and small files must be supported, but there is no need to optimize for that.
3. The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
4. The workloads also have many large, sequential writes that append data to files.
5. High-sustained bandwidth is more important than low latency.

The architecture of the file system is organized into a single master, which contains the metadata of the entire file system, and a collection of chunk servers, which provide storage space. From a logical point of view the system is composed of a collection of software daemons, which implement either the master server or the chunk server.

d. Sector. Sector is the storage cloud that supports the execution of data-intensive applications defined according to the Sphere framework. It is a user space file system that can be deployed on commodity hardware across a wide-area network. The system's architecture is composed of four nodes: a security server, one or more master nodes, slave nodes, and client machines. The security server maintains all the information about access control policies for user and files, whereas master servers coordinate and serve the I/O requests of clients, which ultimately interact with slave nodes to access files. The protocol used to exchange data with slave nodes is UDT, which is a lightweight connection-oriented protocol.

e. Amazon Simple Storage Service (S3). Amazon S3 is the online storage service provided by Amazon.

The system offers a flat storage space organized into buckets, which are attached to an Amazon Web Services (AWS) account. Each bucket can store multiple objects, each identified by a unique key. Objects are identified by unique URLs and exposed through HTTP, thus allowing very simple get-put semantics.

2. NoSQL systems

The term **Not Only SQL (NoSQL)** was coined in 1998 to identify a 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 most of the simplest and more common database tasks by using text files as information stores.

Two main factors have determined the growth of the NoSQL:

1. simple data models are enough to represent the information used by applications, and
2. the quantity of information contained in unstructured formats has grown.

Let us now examine some prominent implementations that support data-intensive applications.

a. Apache CouchDB and MongoDB.

Apache CouchDB and MongoDB are two examples of document stores. 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.

The databases expose a RESTful interface and represent data in JSON format. Both allow querying and indexing data by using the MapReduce programming model, expose JavaScript as a base language for data querying and manipulation rather than SQL, and support large files as documents.

b. Amazon Dynamo.

The main goal of Dynamo is to provide an incrementally scalable and highly available storage system. This goal helps in achieving reliability at a massive scale, where thousands of servers and network components build an infrastructure serving 10 million requests per day. Dynamo provides a simplified interface based on get/put semantics, where objects are stored and retrieved with a unique identifier (key).

The architecture of the Dynamo system, shown in **Figure 8.3**, is composed of a collection of storage peers organized in a ring that shares the key space for a given application. The key space is partitioned among the storage peers, and the keys are replicated across the ring, avoiding adjacent peers. Each peer is configured with access to a local storage facility where original objects and replicas are stored.

Each node provides facilities for distributing the updates among the rings and to detect failures and unreachable nodes.

c. Google Bigtable.

Bigtable provides storage support for several Google applications that expose different types of workload: from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users.

Bigtable's key design goals are wide applicability, scalability, high performance, and high availability. To achieve these goals, Bigtable organizes the data storage in tables of which the rows are distributed over the distributed file system supporting the middleware, which is the Google File System.

From a logical point of view, a table is a multidimensional sorted map indexed by a key that is represented by a string of arbitrary length. A table is organized into rows and columns; columns can be grouped in column family, which allow for specific optimization for better access control, the storage and the indexing of data.
Bigtable APIs also allow more complex operations such as single row transactions and advanced data manipulation.

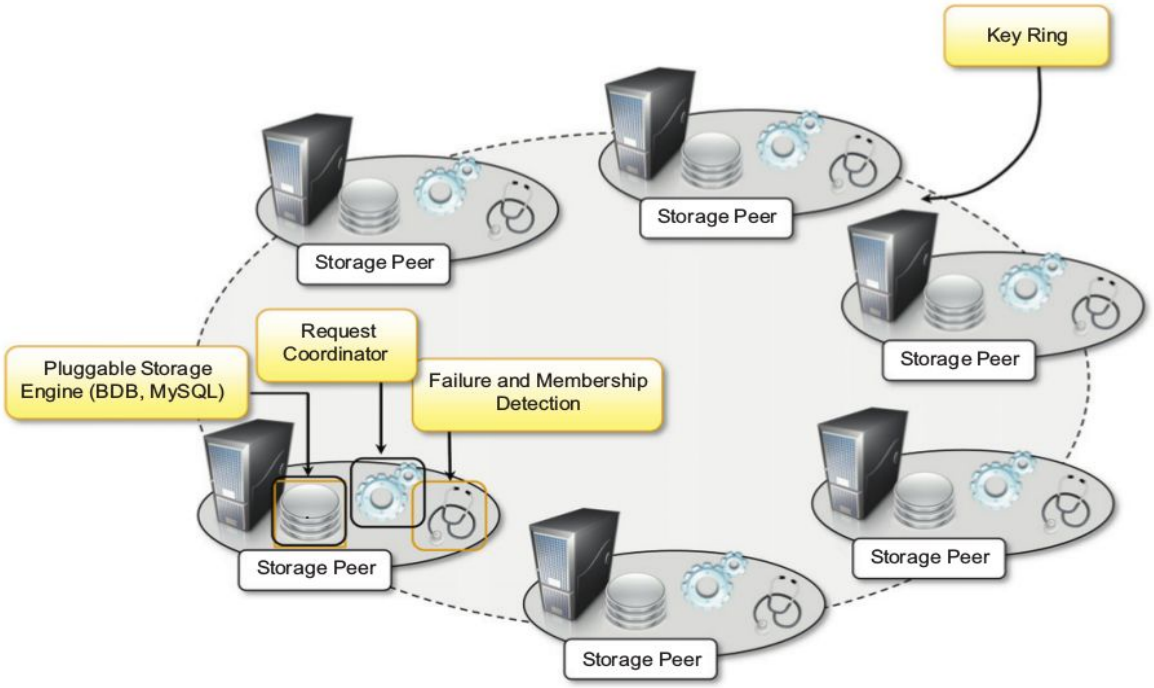


FIGURE 8.3
Amazon Dynamo architecture.

Figure 8.4 gives an overview of the infrastructure that enables Bigtable.
The service is the result of a collection of processes that coexist with other processes in a cluster-based environment. Bigtable identifies two kinds of processes: master processes and tablet server processes. A tablet server is responsible for serving the requests for a given tablet that is a contiguous partition of rows of a table. Each server can manage multiple tablets (commonly from 10 to 1,000). The master server is responsible for keeping track of the status of the tablet servers and of the allocation of tablets to tablet servers. The server constantly monitors the tablet servers to check whether they are alive, and in case they are not reachable, the allocated tablets are reassigned and eventually partitioned to other servers.

d. Apache Cassandra.

The system is designed to avoid a single point of failure and offer a highly reliable service. Cassandra was initially developed by Facebook; now it is part of the Apache incubator initiative. Currently, it provides storage support for several very large Web applications such as Facebook itself, Digg, and Twitter.
The data model exposed by Cassandra is based on the concept of a table that is implemented as a distributed multidimensional map indexed by a key. The value corresponding to a key is a highly 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. The APIs provided by the system to access and manipulate the data are very simple: insertion, retrieval, and deletion. The insertion is performed at the row level; retrieval and deletion can operate at the column level.

e. Hadoop HBase.

HBase is designed by taking inspiration from Google Bigtable; its main goal is to offer real-time read/write operations for tables with billions of rows and millions of columns by leveraging clusters of commodity hardware. The internal 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).

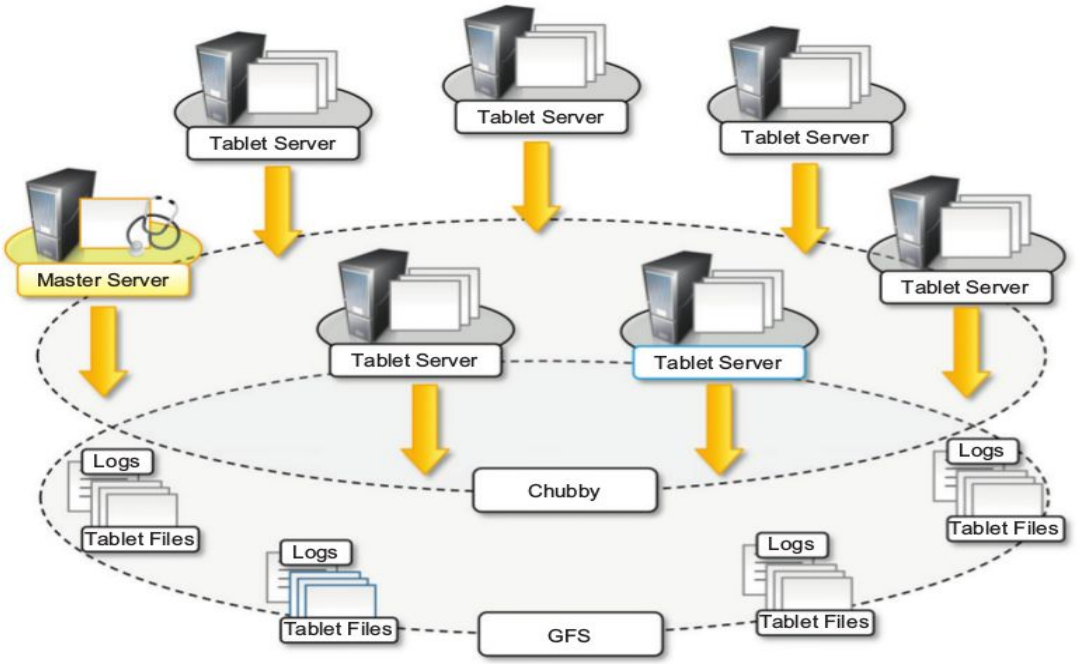


FIGURE 8.4
Bigtable architecture.

8.2.2 Programming platforms

Programming platforms for data-intensive computing provide higher-level abstractions, which focus on the processing of data and move into the runtime system the management of transfers, thus making the data always available where needed.

This is the approach followed by the MapReduce programming platform, which expresses the computation in the form of two simple functions—map and reduce—and hides the complexities of managing large and numerous data files into the distributed file system supporting the platform. In this section, we discuss the characteristics of MapReduce and present some variations of it.

1. The MapReduce programming model.

MapReduce expresses 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), which is in charge of providing access to data, replicating files, and eventually moving them where needed.

the MapReduce model is expressed in the form of the two functions, which are defined as follows:

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

The map function reads a key-value pair and produces a list of key-value pairs of different types. The reduce function reads a pair composed of a key and a list of values and produces a list of values of the same type. The types (k1,v1,k2,kv2) used in the expression of the two functions provide hints as to how these two functions are connected and are executed to carry out the computation of a MapReduce job: The output of map tasks is aggregated together by grouping the values according to their corresponding keys and constitutes the input of reduce tasks that, for each of the keys found, reduces the list of attached values to a single value. Therefore, the input of a MapReduce computation is expressed as a collection of key-value pairs < k1,v1 >, and the final output is represented by a list of values: list(v2).

Figure 8.5 depicts a reference workflow characterizing MapReduce computations. As shown, the user submits a collection of files that are expressed in the form of a list of < k1,v1 > pairs and specifies the map and reduce functions. These files are entered into the distributed file system that supports MapReduce and, if necessary, partitioned in order to be the input of map tasks. Map tasks generate intermediate files that store collections of

< k2, list(v2) > pairs, and these files are saved into the distributed file system. These files constitute the input of reduce tasks, which finally produce output files in the form of list(v2).

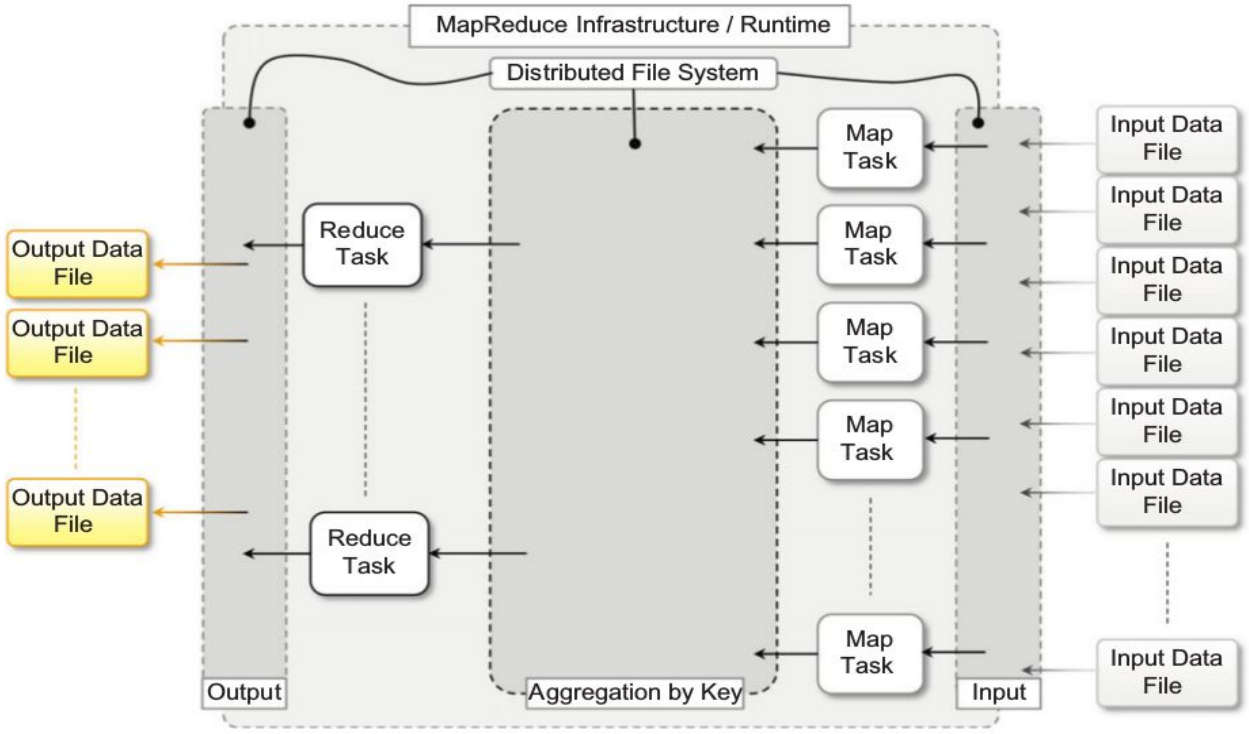


FIGURE 8.5

MapReduce computation workflow.

The computation model expressed by MapReduce is very straightforward and allows greater productivity for people who have to code the algorithms for processing huge quantities of data.

In general, any computation that can be expressed in the form of two major stages can be represented in terms of MapReduce computation.

These stages are:

1. Analysis. This phase operates directly on the data input file and corresponds to the operation performed by the map task. Moreover, the computation at this stage is expected to be embarrassingly parallel, since map tasks are executed without any sequencing or ordering.

2. Aggregation. This phase operates on the intermediate results and is characterized by operations that are aimed at aggregating, summing, and/or elaborating the data obtained at the previous stage to present the data in their final form. This is the task performed by the reduce function.

Figure 8.6 gives a more complete overview of a MapReduce infrastructure, according to the implementation proposed by Google.

As depicted, the user submits the execution of MapReduce jobs by using the client libraries that are in charge of submitting the input data files, registering the map and reduce functions, and returning control to the user once the job is completed. A generic distributed infrastructure (i.e., a cluster) equipped with job-scheduling capabilities and distributed storage can be used to run MapReduce applications.

Two different kinds of processes are run on the distributed infrastructure:

a master process and

a worker process.

The master process is in charge of controlling the execution of map and reduce tasks, partitioning, and reorganizing the intermediate output produced by the map task in order to feed the reduce tasks.

The master process generates the map tasks and assigns input splits to each of them by balancing the load.

The worker processes are used to host the execution of map and reduce tasks and provide basic I/O facilities that are used to interface the map and reduce tasks with input and output files.

Worker processes have input and output buffers that are used to optimize the performance of map and reduce tasks. In particular, output buffers for map tasks are periodically dumped to disk to create intermediate files. Intermediate files are partitioned using a user-defined function to evenly split the output of map tasks.

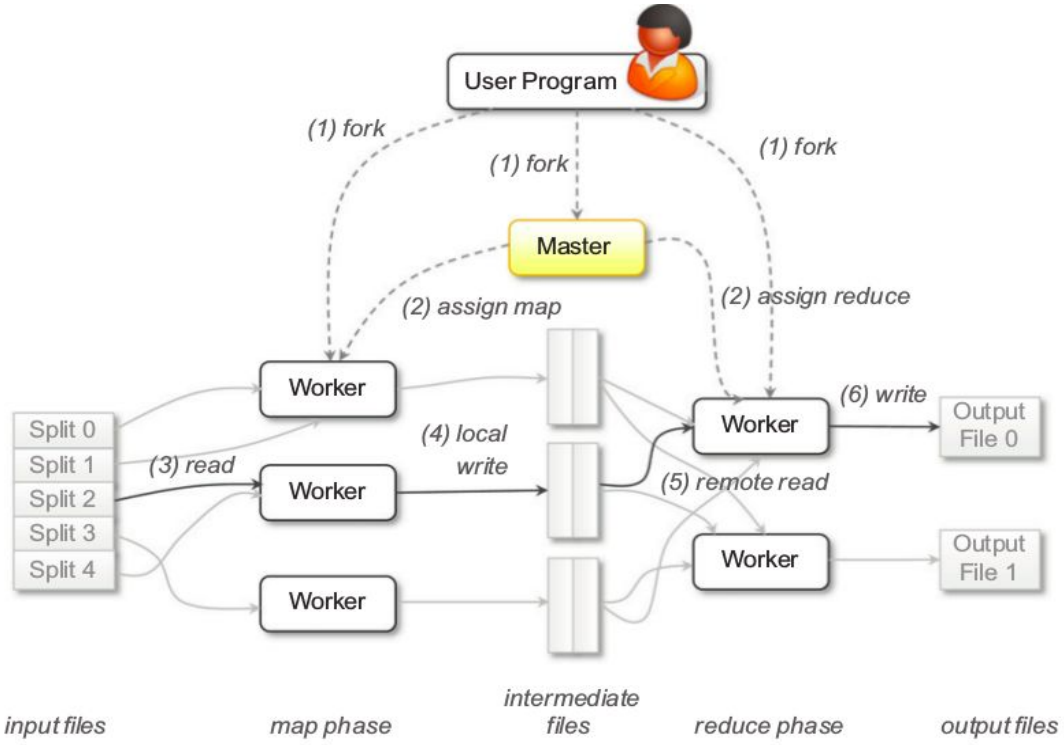


FIGURE 8.6

Google MapReduce infrastructure overview.

2. Variations and extensions of MapReduce.

MapReduce constitutes a simplified model for processing large quantities of data and imposes constraints on the way distributed algorithms should be organized to run over a MapReduce infrastructure.

Therefore, a series of extensions to and variations of the original MapReduce model have been proposed. They aim at extending the MapReduce application space and providing developers with an easier interface for designing distributed algorithms.

We briefly present a collection of MapReduce-like frameworks and discuss how they differ from the original MapReduce model.

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

A. Hadoop.

Apache Hadoop is a collection of software projects for reliable and scalable distributed computing. The initiative consists of mostly two projects: Hadoop Distributed File System (HDFS) and Hadoop MapReduce. The former is an implementation of the Google File System; the latter provides the same features and abstractions as Google MapReduce.

B. Pig.

Pig is a platform that allows the analysis of large datasets. Developed as an Apache project, Pig consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The Pig infrastructure's layer consists of a compiler for a high-level language that produces a sequence of MapReduce jobs that can be run on top of distributed infrastructures.

C. Hive.

Hive is another Apache initiative that provides a data warehouse infrastructure on top of Hadoop MapReduce. It provides tools for easy data summarization, ad hoc queries, and analysis of large datasets stored in Hadoop MapReduce files.

Hive's major advantages reside in the ability to scale out, since it is based on the Hadoop framework, and in the ability to provide a data warehouse infrastructure in environments where there is already a Hadoop system running.

D. Map-Reduce-Merge.

Map-Reduce-Merge is an extension of the MapReduce model, introducing a third phase to the standard MapReduce pipeline—the Merge phase—that allows efficiently merging data already partitioned and sorted (or hashed) by map and reduce modules. The Map-Reduce-Merge framework simplifies the management of heterogeneous related datasets and provides an abstraction able to express the common relational algebra operators as well as several join algorithms.

E. Twister.

Twister is an extension of the MapReduce model that allows the creation of iterative executions of MapReduce jobs. With respect to the normal MapReduce pipeline, the model proposed by 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

Twister provides additional features such as the ability for map and reduce tasks to refer to static and in-memory data; the introduction of an additional phase called combine, run at the end of the MapReduce job, that aggregates the output together.

3. Alternatives to MapReduce.

a. Sphere.

b. All-Pairs.

c. DryadLINQ.

a. Sphere.

Sphere is the distributed processing engine that leverages the **Sector Distributed File System (SDFS)**.

Sphere implements the **stream processing model** (**Single Program, Multiple Data**) and allows developers to express the computation in terms of **user-defined functions (UDFs)**, which are run against the distributed infrastructure.

Sphere is built on top of Sector's API for data access.

UDFs are expressed in terms of programs that read and write streams. A stream is a data structure that provides access to a collection of data segments mapping one or more files in the SDFS.

The execution of UDFs is achieved through **Sphere Process Engines (SPEs)**, which are assigned with a given stream segment.

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

b. All-Pairs.

It provides a simple abstraction—in terms of the All-pairs function—that is common in many scientific computing domains:

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

Ex 1: field of biometrics, where similarity matrices are composed as a result of the comparison of several images that contain subject pictures.

Ex 2: applications and algorithms in data mining.

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

The execution of a distributed application is controlled by the engine and develops in four stages:

(1) model the system;

(2) distribute the data;

(3) dispatch batch jobs; and

(4) clean up the system.

c. DryadLINQ.

Dryad is a Microsoft Research project that investigates programming models for writing parallel and distributed programs to scale from a small cluster to a large datacenter.

In Dryad, developers can express distributed applications as a set of sequential programs that are connected by means of channels.

Dryad computation expressed in terms of a directed acyclic graph in which nodes are the sequential programs and vertices represent the channels connecting such programs.

Dryad is considered a superset of the MapReduce model, its application model allows expressing graphs representing MapReduce computation.

8.3 Aneka MapReduce programming

Aneka provides an implementation of the MapReduce abstractions 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

8.3.1 Introducing the MapReduce programming model

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

Figure 8.7 provides an overview of the infrastructure supporting MapReduce in Aneka.

The application instance is specialized, with components that identify the map and reduce functions to use.

These functions are expressed in terms of **Mapper and Reducer classes** that are extended from the Aneka MapReduce APIs.

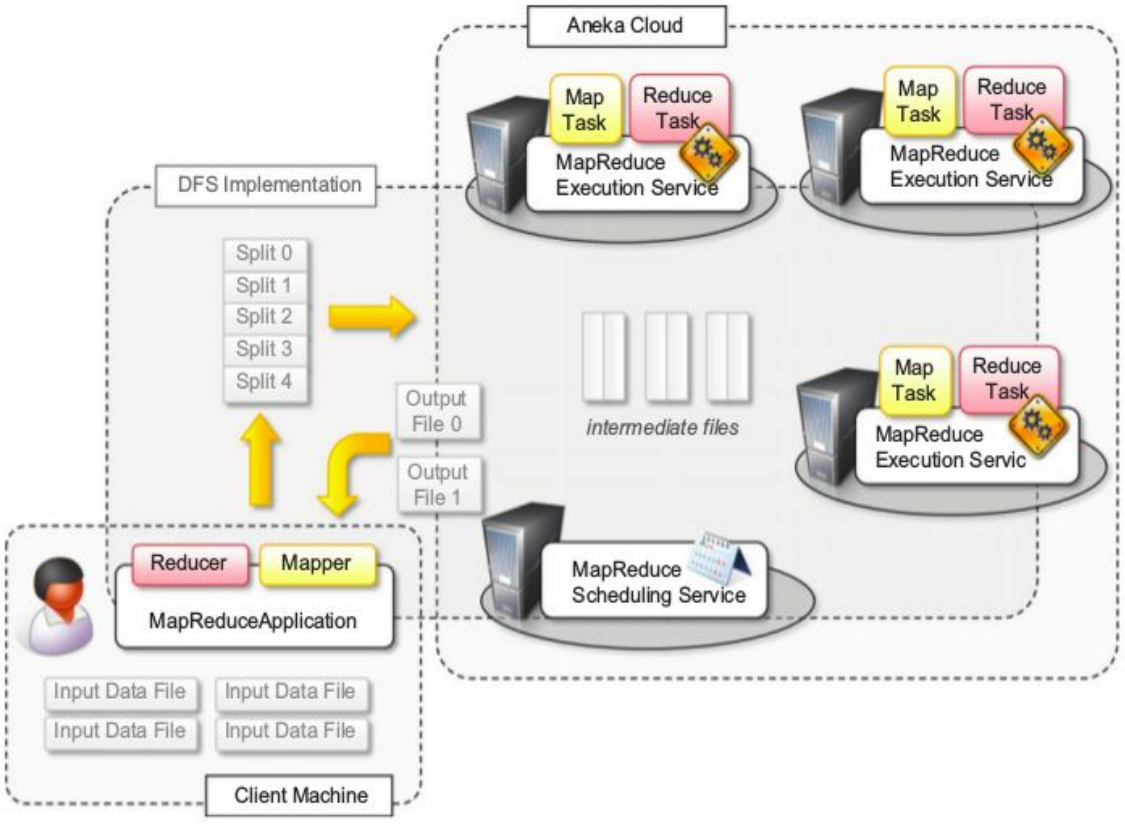


FIGURE 8.7

Aneka MapReduce infrastructure.

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, namely the MapReduce Application, are used to submit the execution of a MapReduce job, upload data files, and monitor it.

The management of data files is transparent: local data files are automatically uploaded to Aneka, and output files are automatically downloaded to the client machine if requested.

In the following sections, we introduce these major components and describe how they collaborate to execute MapReduce jobs.

1 Programming abstractions

Aneka executes any piece of user code within distributed application.

The task creation is responsibility of the infrastructure once the user has defined the map and reduce functions.

Therefore, the Aneka MapReduce APIs provide developers with base classes for developing Mapper and Reducer types and use a specialized type of application class—Map Reduce Application — that supports needs of this programming model.

Figure 8.8 provides an overview of the client components defining the MapReduce programming model. Three classes are of interest for application development: Mapper<K,V>, Reducer<K,V>, and MapReduceApplication<M,R>. The other classes are internally used to implement all the functionalities

required by the model and expose simple interfaces that require minimum amounts of coding for implementing the map and reduce functions and controlling the job submission. Mapper<K,V> and Reducer<K,V> constitute the starting point of the application design and implementation.

The submission and execution of MapReduce job is performed through class MapReduceApplication<M,R>, which provides the interface to Aneka Cloud to support MapReduce programming model. This class exposes two generic types: M and R. These two placeholders identify the specific types of Mapper<K,V> and Reducer<K,V> that will be used by the application.

Listing 8.1 shows in detail the 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 shows the implementation of the Mapper<K,V> component for Word Counter sample. This sample counts frequency of words in a set of large text files. The text files are divided into lines, each of which will become the value component of a key-value pair, whereas the key will be represented by the offset in the file where the line begins.

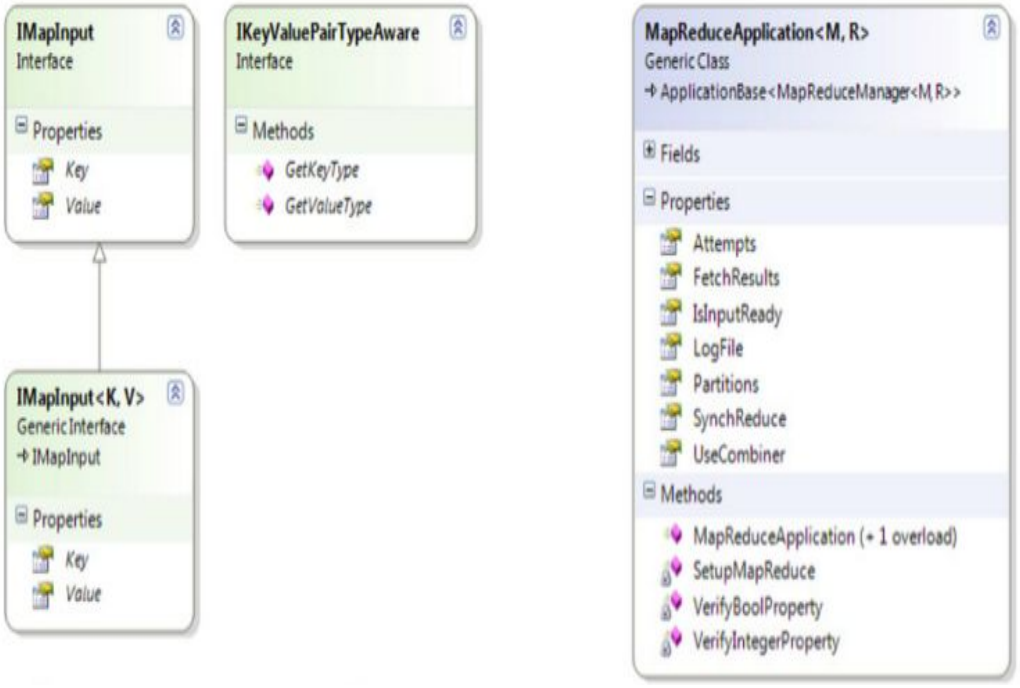
Listing 8.3 shows the definition of Reducer<K,V> class. The implementation of a specific reducer requires specializing the generic class and overriding the abstract method: Reduce(IReduceInputEnumerator<V> input).

Listing 8.4 shows how to implement the reducer function for word-counter example.

Listing 8.5 shows the 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 shows how to create a MapReduce application for running the word-counter example defined by the previous WordCounterMapper and WordCounterReducer classes.



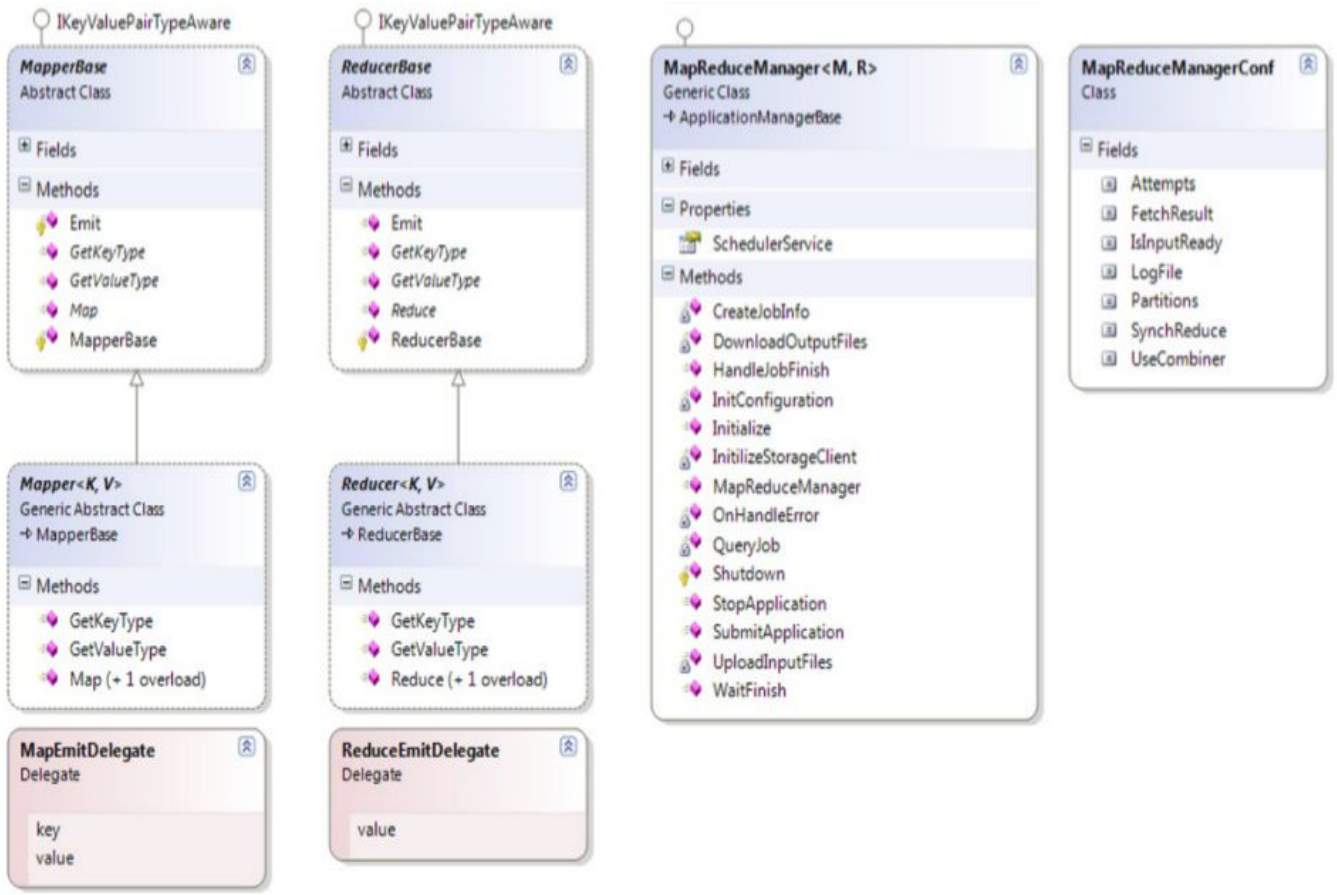


FIGURE 8.8

MapReduce Abstractions Object Model.

```
using Aneka.MapReduce.Internal;
namespace Aneka.MapReduce
{
        /// Interface IMapInput<K,V>. Extends IMapInput and provides strongly-typed version of extended
        /// interface.
        public interface IMapInput<K,V>: IMapInput
        {
                /// Property <i>Key</i> returns the key of key/value pair.
                K Key { get; }
                /// Property <i>Value</i> returns the value of key/value pair.
                V Value { get; }
        }
        /// Delegate MapEmitDelegate. Defines signature of method that is used to doEmit intermediate results
        /// generated by mapper.
        public delegate void MapEmitDelegate(object key, object value);
        /// Class Mapper. Extends MapperBase and provides a reference implementation that can be further
        /// extended in order to define the specific mapper for a given application.
        public abstract class Mapper<K,V> : MapperBase
        {
                /// Emits the intermediate result source by using doEmit.
                /// output stream the information about the output of the Map operation.</para
                public void Map(IMapInput input, MapEmitDelegate emit) { ... }
```

```
            /// Gets the type of the <i>key</i> component of a <i>key-value</i> pair.
            /// <returns>A Type instance containing the metadata
            public override Type GetKeyType(){ return typeof(K); }
            /// Gets the type of the <i>value</i> component of a <i>key-value</i> pair.
            /// <returns>A Type instance containing the metadata
            public overrideType GetValueType(){ return typeof(V); }
            #region Template Methods
            /// Function Map is overrided by users to define a map function.
            protected abstract void Map(IMapInput<K, V> input);
            #endregion
        }
}
```

LISTING 8.1 Map Function APIs.

```
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.WordCounter
{
        /// Class WordCounterMapper. Extends Mapper<K,V> and provides an
        /// implementation of the map function for the Word Counter sample.
        public class WordCounterMapper: Mapper<long,string>
        {
                /// Reads the source and splits into words. For each of the words found
                /// emits the word as a key with a vaue of 1.
                protected override void Map(IMapInput<long,string> input)
                {
                        // we don't care about the key, because we are only interested on
                        // counting the word of each line.
                        string value = input.Value;
                        string[] words = value.Split(" \t\n\r\f\"\'|!-=()[]<>:{}.#".ToCharArray(),
                        StringSplitOptions.RemoveEmptyEntries);
                        // we emit each word without checking for repetitions. The word becomes
                        // the key and the value is set to 1, the reduce operation will take care
                        // of merging occurrences of the same word and summing them.
                        foreach(string word in words)
                        {
                                this.Emit(word, 1);
                        }
                }
        }
}
```

LISTING 8.2 Simple Mapper <K,V> Implementation.

```
using Aneka.MapReduce.Internal;
namespace Aneka.MapReduce
{
        /// Delegate ReduceEmitDelegate. Defines the signature of a method
```

```
        public delegate void ReduceEmitDelegate(object value);
        /// Class <i>Reducer</i>. Extends the ReducerBase class
        public abstract class Reducer<K,V> : ReducerBase
        {
                /// Performs the <i>reduce</i> phase of the <i>map-reduce</i> model.
                public void Reduce(IReduceInputEnumerator input, ReduceEmitDelegate emit) { ... }
                /// Gets the type of the <i>key</i> component of a <i>key-value</i> pair.
                public override Type GetKeyType(){return typeof(K);}
                /// Gets the type of the <i>value</i> component of a <i>key-value</i> pair.
                public override Type GetValueType(){return typeof(V);}
                #region Template Methods
                /// Recuces the collection of values that are exposed by
                /// <paramref name="source"/> into a single value.
                protected abstract void Reduce(IReduceInputEnumerator<V> input);
                #endregion
        }
}
```

LISTING 8.3 Reduce Function APIs.

```
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.WordCounter
{
        /// Class <b><i>WordCounterReducer</i></b>. Reducer implementation for the Word
        /// Counter application.
        public class WordCounterReducer: Reducer<string,int>
        {
                /// Iterates all over the values of the enumerator and sums up
                /// all the values before emitting the sum to the output file.
                protected override void Reduce(IReduceInputEnumerator<int>input)
                {
                        int sum = 0;
                        while(input.MoveNext())
                        {
                                int value = input.Current;
                                sum += value;
                        }
                        this.Emit(sum);
                }
        }
}
```

LISTING 8.4 Simple Reducer <K,V> Implementation.

```
using Aneka.MapReduce.Internal;
namespace Aneka.MapReduce
{
        /// Class <b><i>MapReduceApplication</i></b>. Defines a distributed application
        /// based on the MapReduce Model. It extends the ApplicationBase<M> and specializes
        /// it with the MapReduceManager<M,R> application manager.
        public class MapReduceApplication<M, R> : ApplicationBase<MapReduceManager<M, R>>
        where M: MapReduce.Internal.MapperBase
```

```
        where R: MapReduce.Internal.ReducerBase
        {
                /// Default value for the Attempts property.
                public const intDefaultRetry = 3;
                /// Default value for the Partitions property.
                public const intDefaultPartitions = 10;
                /// Default value for the LogFile property.
                public const stringDefaultLogFile = "mapreduce.log";
                /// List containing the result files identifiers.
                private List<string>resultFiles = new List<string>();
                /// Property group containing the settings for the MapReduce application.
                private PropertyGroupmapReduceSetup;
                /// Gets, sets an integer representing the number of partions for the key space.
                public int Partitions { get { ... } set { ... } }
                /// Gets, sets an boolean value indicating in whether to combine the result
                /// after the map phase in order to decrease the number of reducers used in the
                /// reduce phase.
                public bool UseCombiner { get { ... } set { ... } }
                /// Gets, sets an boolean indicating whether to synchronize the reduce phase.
                public bool SynchReduce { get { ... } set { ... } }
                /// Gets or sets a boolean indicating whether the source files required by the
                /// required by the application is already uploaded in the storage or not.
                public bool IsInputReady { get { ... } set { ... } }
                /// Gets, sets the number of attempts that to run failed tasks.
                public int Attempts { get { ... } set { ... } }
                /// Gets or sets a string value containing the path for the log file.
                public string LogFile { get { ... } set { ... } }
                /// Gets or sets a boolean indicating whether application should download the
                /// result files on the local client machine at the end of the execution or not.
                public bool FetchResults { get { ... } set { ... } }
                /// Creates a MapReduceApplication<M,R> instance and configures it with
                /// the given configuration.
                public MapReduceApplication(Configurationconfiguration) :
                base("MapReduceApplication", configuration){ ... }
                /// Creates MapReduceApplication<M,R> instance and configures it with
                /// the given configuration.
                public MapReduceApplication(string displayName, Configuration configuration) : base(displayName,
                configuration) { ... }
                // here follows the private implementation…
        }
}
```

LISTING 8.5 MapReduceApplication<M,R>.

```
namespace Aneka.Entity
{
        /// Class <b><i>ApplicationBase<M></i></b>. Defines the base class for the
        /// application instances for all the programming model supported by Aneka.
        public class ApplicationBase<M> where M : IApplicationManager, new()
        {
                /// Gets the application unique identifier attached to this instance.
                public string Id { get { ... } }
                /// Gets the unique home directory for the AnekaApplication<W,M>.
```

```
                public string Home { get { ... } }
                /// Gets the current state of the application.
                public ApplicationState State{get{ ... }}
                /// Gets a boolean value indicating whether the application is terminated.
                public bool Finished { get { ... } }
                /// Gets the underlying IApplicationManager that ismanaging the execution of the
                /// application instanceon the client side.
                public M ApplicationManager { get { ... } }
                /// Gets, sets the application display name.
                public string DisplayName { get { ... } set { ... } }
                /// Occurs when the application instance terminates its execution.
                public event EventHandler<ApplicationEventArgs> ApplicationFinished;
                /// Creates an application instance with the given settings and sets the
                /// application display name to null.
                public ApplicationBase(Configuration configuration): this(null, configuration){ ... }
                /// Creates an application instance withthe given settings and display name.
                public ApplicationBase(string displayName, Configuration configuration){ ... }
                /// Starts the execution of the application instanceon Aneka.
                public void SubmitExecution() { ... }
                /// Stops the execution of the entire application instance.
                public void StopExecution() { ... }
                /// Invoke the application and wait until the application finishes.
                public void InvokeAndWait() { this.InvokeAndWait(null); }
                /// Invoke the application and wait until the application finishes, then invokes
                /// the given callback.
                public void InvokeAndWait(EventHandler<ApplicationEventArgs> handler) { ... }
                /// Adds a shared file to the application.
                public virtual void AddSharedFile(string file) { ... }
                /// Adds a shared file to the application.
                public virtual void AddSharedFile(FileData fileData) { ... }
                /// Removes a file from the list of the shared files of the application.
                public virtual void RemoveSharedFile(string filePath) { ... }
                // here come the private implementation.
        }
}
```

LISTING 8.6 ApplicationBase<M>

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.WordCounter
{
        /// Class <b><i>Program<M></i></b>. Application driver for the Word Counter sample.
        public class Program
        {
                /// Reference to the configuration object.
                private static Configuration configuration = null;
                /// Location of the configuration file.
                private static string confPath = "conf.xml";
                /// Processes arguments given to application & read runs application or shows help.
                private static void Main(string[] args)
                {
```

```
                try
                {
                        Logger.Start();
                        // get the configuration
                        configuration = Configuration.GetConfiguration(confPath);
                        // configure MapReduceApplication
                        MapReduceApplication<WordCountMapper, WordCountReducer> application =
                        new MapReduceApplication<WordCountMapper, WordCountReducer>
                ("WordCounter",
                        configuration);
                        // invoke and wait for result
                        application.InvokeAndWait(new
                        EventHandler<ApplicationEventArgs>(OnDone));
                }
                catch(Exception ex)
                {
                        Usage();
                        IOUtil.DumpErrorReport(ex, "Aneka WordCounter Demo - Error Log");
                }
                finally
                {
                        Logger.Stop();
                }
        }
        /// Hooks the ApplicationFinished events and Process the results if the application has been successful.
        private static void OnDone(object sender, ApplicationEventArgs e) { ... }
        /// Displays a simple informative message explaining the usage of the application.
        private static void Usage() { ... }
    }
}
```

LISTING 8.7 WordCounter Job.

2 Runtime support

The runtime support for the execution of MapReduce jobs 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. The scheduling of jobs and tasks is the responsibility of the MapReduce Scheduling Service, which 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.

Main role of the service wrapper is to translate messages coming from the Aneka runtime or the client applications into calls or events directed to the scheduler component, and vice versa. The relationship of the two components is depicted in **Figure 8.9**.

The core functionalities for job and task scheduling are implemented in the MapReduceScheduler class. The scheduler manages multiple queues for several operations, such as uploading input files into the distributed file system; initializing jobs before scheduling; scheduling map and reduce tasks; keeping track of unreachable nodes; resubmitting failed tasks; and reporting execution statistics.

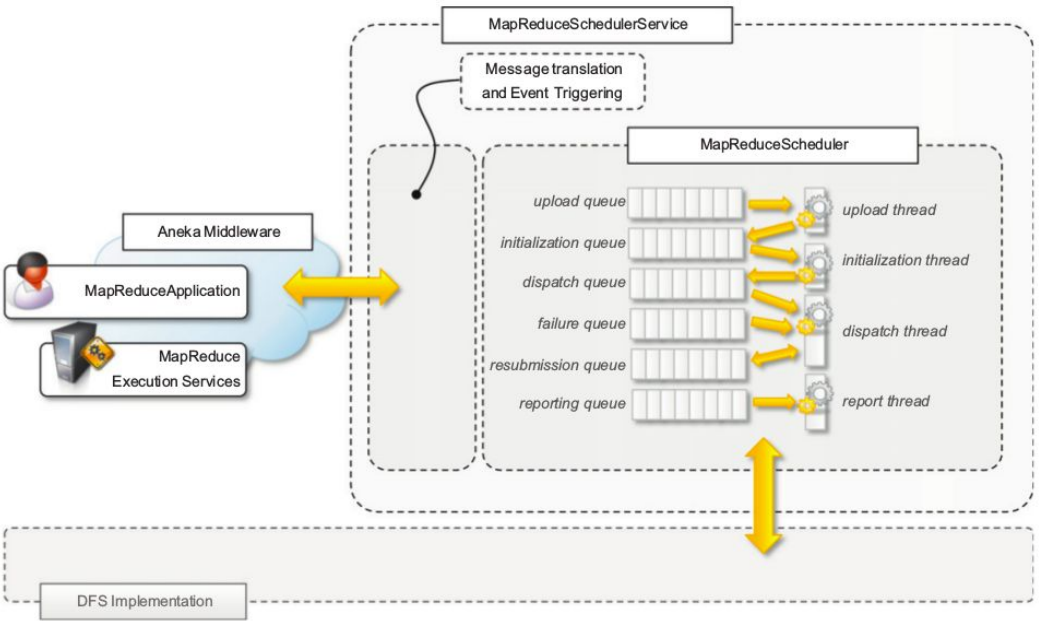


FIGURE 8.9

MapReduce Scheduling Service architecture.

Task Execution. The execution of tasks is controlled by the MapReduce Execution Service. This component plays the role of the worker process in the Google MapReduce implementation. The service manages the execution of map and reduce tasks and performs other operations, such as sorting and merging intermediate files. The service is internally organized, as described in **Figure 8.10**.

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

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

The MapReduceSchedulerService interfaces the ExecutorManager with the Aneka middleware; the ExecutorManager is in charge of keeping track of the tasks being executed by demanding the specific execution of a task to the MapReduceExecutor and of sending the statistics about the execution back to the Scheduler Service.

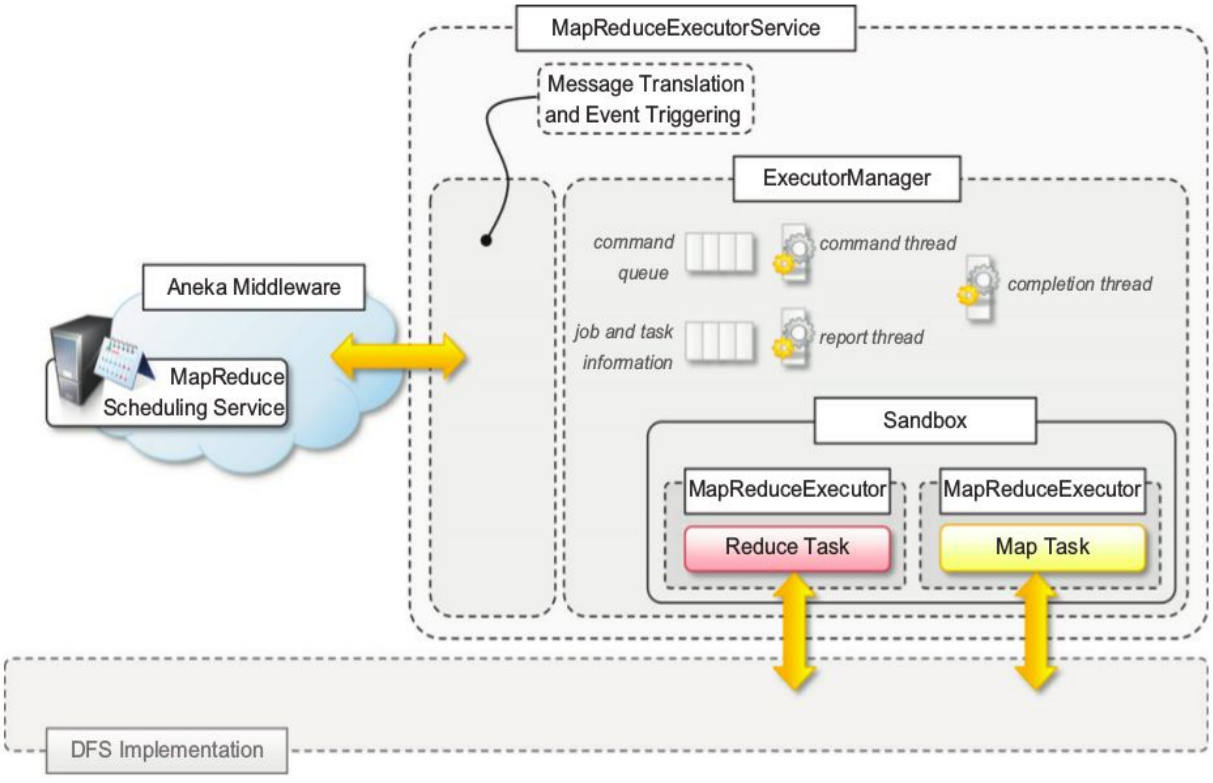


FIGURE 8.10

MapReduce Execution Service architecture.

3 Distributed file system support

Aneka supports, the MapReduce model that uses a distributed file system implementation.

Distributed file system implementations guarantee high availability and better efficiency by means of replication and distribution.

the original MapReduce implementation assumes the existence of a distributed and reliable storage; hence, the use of a distributed file system for implementing the storage layer is natural.

Aneka provides the capability of interfacing with different storage implementations and it maintains the same flexibility for the integration of a distributed file system.

The level of integration required by MapReduce requires the ability to perform the following tasks:

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

The first operation is useful to the scheduler for optimizing the scheduling of map and reduce tasks according to the location of data; the second operation is required for the usual I/O operations to and from data files.

On top of these low-level interfaces, MapReduce programming model offers classes to read from and write to files in a sequential manner. These are classes **SeqReader** and **SeqWriter**. They provide sequential access for reading and writing key-value pairs, and they expect specific file format, which is described in **Figure 8.11**.

An Aneka MapReduce file is composed of a header, used to identify the file, and a sequence of record blocks, each storing a key-value pair. The header is composed of 4 bytes: the first 3 bytes represent the character sequence SEQ and the fourth byte identifies the version of the file. The record block is composed as follows: the first 8 bytes are used to store two integers representing the length of the rest of the block and the length of the key section, which is immediately following. The remaining part of the block stores the data of the value component of the pair. The SeqReader and SeqWriter classes are designed to read and write files in this format by transparently handling the file format information and translating key and value instances to and from their binary representation.

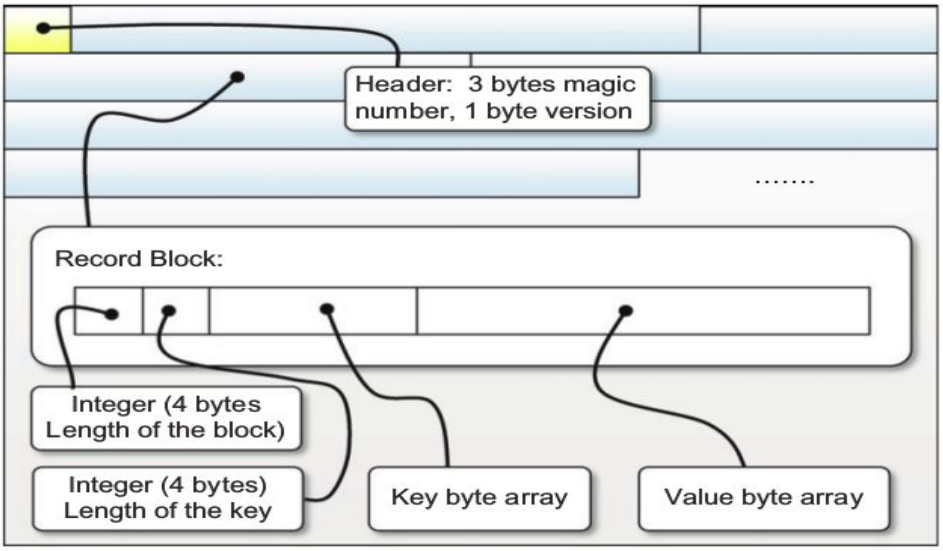


FIGURE 8.11

Aneka MapReduce data file format.

Listing 8.8 shows the interface of the SeqReader and SeqWriter classes. The SeqReader class provides an enumerator-based approach through which it is possible to access the key and the value sequentially by calling the NextKey() and the NextValue() methods, respectively. It is also possible to access the raw byte data of keys and values by using the NextRawKey() and NextRawValue(). HasNext() returns a Boolean, indicating whether there are more pairs to read or not.

```
namespace Aneka.MapReduce.DiskIO
{
        /// Class <b><i>SeqReader</i></b>. This class implements a file reader for the sequence
        /// file, which isa standard file split used by MapReduce.NET to store a partition of a fixed size of a data file.
        public class SeqReader
```

```
{
        /// Creates a SeqReader instance and attaches it to the given file.
        public SeqReader(string file) : this(file, null, null) { ... }
        /// Creates a SeqReader instance, attaches it to the given file, and sets the
        /// internal buffer size to bufferSize.
        public SeqReader(string file, int bufferSize) : this(file,null,null,bufferSize) { ... }
        /// Creates a SeqReader instance, attaches it to the given file, and provides
        /// metadata information about the content of the file in the form of keyType andcvalueType.
        public SeqReader(string file, Type keyType, Type valueType) : this(file, keyType, valueType,
        SequenceFile.DefaultBufferSize) { ... }
        /// Creates a SeqReader instance, attaches it to the given file, and provides
        /// metadata information about the content of the file in the form of keyType and valueType.
        public SeqReader(string file, Type keyType, Type valueType, int bufferSize){ ... }
        /// Sets the metadata information about the keys and the values contained in the data file.
        public void SetType(Type keyType, Type valueType) { ... }
        /// Checks whether there is another record in data file and moves current file pointer to its beginning.
        public bool HaxNext() { ... }
        /// Gets the object instance corresponding to the next key in the data file. in the data file.
        public object NextKey() { ... }
        /// Gets the object instance corresponding to the next value in the data file. in the data file.
        public object NextValue() { ... }
        /// Gets the raw bytes that contain the value of the serializedinstance of the current key.
        public BufferInMemory NextRawKey() { ... }
        /// Gets the raw bytes that contain the value of the serialized instance of the current value.
        public BufferInMemory NextRawValue() { ... }
        /// Gets the position of the file pointer as an offset from its beginning.
        public long CurrentPosition() { ... }
        /// Gets the size of the file attached to this instance of SeqReader.
        public long StreamLength() { ... }
        /// Moves file pointer to position. If value of position is 0 or -ve, returns current position of file pointer.
        public long Seek(long position) { ... }
        /// Closes the SeqReader instanceand releases all resources that have been allocated to read fromthe file.
        public void Close() { ... }
        // private implementation follows
}
/// Class SeqWriter. This class implements a file writer for the sequence file, which is a standard file split used by
///MapReduce.NET to store a partition of a fixed size of a data file. This classprovides an interface to add a
///sequence of key-value pair incrementally.
public class SeqWriter
{
        /// Creates a SeqWriter instance for writing to file. This constructor initializes
        /// the instance with the default value for the internal buffers.
        public SeqWriter(string file) : this(file, SequenceFile.DefaultBufferSize){ ... }
        /// Creates a SeqWriter instance, attachesit to the given file, and sets the
        /// internal buffer size to bufferSize.
        public SeqWriter(string file, int bufferSize) { ... }
        /// Appends a key-value pair to the data file split.
        public void Append(object key, object value) { ... }
        /// Appends a key-value pair to the data file split.
        public void AppendRaw(byte[] key, byte[] value) { ... }
        /// Appends a key-value pair to the data file split.
        public void AppendRaw(byte[] key, int keyPos, int keyLen,
```

```
                byte[] value, int valuePos, int valueLen) { ... }
                /// Gets the length of the internal buffer or 0 if no buffer has been allocated.
                public longLength() { ... }
                /// Gets the length of data file split on disk so far.
                public long FileLength() { ... }
                /// Closes SeqReader instance and releases all the resources that have been allocated to write to the file.
                public void Close() { ... }
                // private implementation follows
        }
}
```

LISTING 8.8 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. To visualize the results of the application, we use the SeqReader class to read the content of the output files and dump it into a proper textual form that can be visualized with any text editor, such as the Notepad application.

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.WordCounter
{
        /// Class Program. Application driver for the Word Counter sample.
        public class Program
        {
                /// Reference to the configuration object.
                private static Configuration configuration = null;
                /// Location of the configuration file.
                private static string confPath = "conf.xml";
                /// Processes the arguments given to the application and according
                /// to the parameters read runs the application or shows the help.
                private static void Main(string[] args)
                {
                try
                {
                        Logger.Start();
                        // get the configuration
                        Program.configuration = Configuration.GetConfiguration(confPath);
                        // configure MapReduceApplication
                        MapReduceApplication<WordCountMapper, WordCountReducer> application =
                        new MapReduceApplication<WordCountMapper, WordCountReducer>("WordCounter",
                        configuration);
                        // invoke and wait for result
                        application.InvokeAndWait(newEventHandler<ApplicationEventArgs>(OnDone));
                        // alternatively we can use the following call
                }
                catch(Exception ex)
                {
                        Program.Usage();
                        IOUtil.DumpErrorReport(ex, "Aneka WordCounter Demo - Error Log");
                }
                finally
                {
```

```
                Logger.Stop();
            }
        }
        /// Hooks the ApplicationFinished events and process the results
        /// if the application has been successful.
        private static void OnDone(object sender, ApplicationEventArgs e)
        {
            if (e.Exception != null)
            {
                IOUtil.DumpErrorReport(e.Exception, "Aneka WordCounter Demo - Error");
            }
            else
            {
                string outputDir = Path.Combine(configuration.Workspace, "output");
                try
                {
                    FileStream resultFile = new FileStream("WordResult.txt",FileMode.Create,
                    FileAccess.Write);
                    Stream WritertextWriter = new StreamWriter(resultFile);
                    DirectoryInfo sources = new DirectoryInfo(outputDir);
                    FileInfo[] results = sources.GetFiles();
                    foreach(FileInfo result in results)
                    {
                        SeqReader seqReader = newSeqReader(result.FullName);
                        seqReader.SetType(typeof(string), typeof(int));
                        while(seqReader.HaxNext() == true)
                        {
                            object key = seqReader.NextKey();
                            object value = seqReader.NextValue();
                            textWriter.WriteLine("{0}\t{1}", key, value);
                        }
                        seqReader.Close();
                    }
                    textWriter.Close();
                    resultFile.Close();
                    // clear the output directory
                    sources.Delete(true);
                    Program.StartNotePad("WordResult.txt");
                }
                catch(Exception ex)
                {
                    IOUtil.DumpErrorReport(e.Exception, "Aneka WordCounter Demo - Error");
                }
            }
            }
            /// Starts the notepad process and displays the given file.
            private static voidStartNotepad(string file) { ... }
            /// Displays a simple informative message explaining the usage of the application.
            private static void Usage() { ... }
        }
}
```

LISTING 8.9 WordCounter Job.

8.3.2 Example application

To demonstrate how to program real applications with Aneka MapReduce, we consider a very common task: log parsing. We design a MapReduce application that processes the logs produced by the Aneka container in order to extract some summary information about the behavior of the Cloud.

1 Parsing Aneka logs

Aneka components produce a lot of information that is stored in the form of log files.
In this example, we parse these logs to extract useful information about the execution of applications and the usage of services in the Cloud.
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

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

This information can be easily extracted and composed into a single view by creating Mapper tasks that count the occurrences of log levels and component names and emit one simple key-value pair in the form (level-name, 1) or (component-name, 1) for each of the occurrences. The Reducer task will simply sum up all the key-value pairs that have the same key. For both problems, the structure of the map and reduce functions will be the following:

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

The Mapper class will then receive a key-value pair containing the position of the line inside the file as a key and the log message as the value component. It will produce a key-value pair containing a string representing the name of the log level or the component name and 1 as value. The Reducer class will sum up all the key-value pairs that have the same name.

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. The Map method simply locates the position of the log-level label into the line, extracts it, and emits a corresponding key-value pair (label, 1).

```
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.LogParsing
{
        /// Class LogParsingMapper. Extends Mapper<K,V> and provides an
        /// implementation of the map function for parsing the Aneka container log files .
        /// This mapper emits a ke y-value (log-level, 1) and potentially another key -value
        /// (_aneka -component-name,1) if it is able to extract such information from the
        /// input.
```

```
public class LogParsingMapper: Mapper<long,string>
{
    /// Reads the input and extracts the information about the log level and if
    /// found the name of the aneka component that entered the log line .
    protected override void Map(IMapInput<long,string>input)
    {
        // we don't care about the key, because we are only interested on
        // counting the word of each line.
        string value = input.Value;
        long quantity = 1;
        // first we extract the log level name information. Since the date is reported
        // in the standard format DD MMM YYYY mm:hh:ss it is possible to skip the first
        // 20 characters (plus one space) and then extract the next following characters
        // until the next position of the space character.
        int start = 21;
        int stop = value.IndexOf( ' ', start);
        string key = value.Substring(start, stop – start);
        this.Emit(key, quantity);
        //now we are looking for the Aneka component name that entered the log line
        //if this is inside the log line it is just right after the log level preceeded
        //by the character sequence <space><dash><space> and terminated by the <c olon> character.
        start = stop + 3; // we skip the <space><dash><space> sequence.
        stop = value.IndexOf( ':', start);
        key = value.Substring(start, stop – start);
        // we now check whether the key contains any space, if not then it is the name
        // of an Aneka component and the line does not need to be skipped.
        if (key.IndexOf(' ') == -1)
        {
            this.Emit("_" + key, quantity);
        }
    }
}
}
```

LISTING 8.10 Log-Parsing Mapper Implementation.

3 Reducer design and implementation

The implementation of the reduce function is even more straightforward; the only operation that needs to be performed is to 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.

```
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.LogParsing
{
    /// Class <b><i>LogParsingReducer</i></b>. Extends Reducer<K,V> and provides an
    /// implementation of the redcuce function for parsing the Aneka container log files .
    /// The Reduce method iterates all ov er values of the enumerator and sums the values
    /// before emitting the sum to the output file.
    public class LogParsingReducer : Reducer<string,long>
    {
        /// Iterates all over the values of the enum erator and sums up
        /// all the values before emitting the sum to the output file.
```

```
        protected override void Reduce(IReduceInputEnumerator<long>input)
        {
            long sum = 0;
            while(input.MoveNext())
            {
                long value = input.Current;
                sum += value;
            }
            this.Emit(sum);
        }
    }
}
```

LISTING 8.11 Aneka Log-Parsing Reducer Implementation.

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.

Another task that is performed in the driver application is the separation of these two statistics into two different files for further analysis.

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

The configuration of the job is performed in the Initialize method. This method reads the configuration file from the local file system and ensures that the input and output formats of files are set to text.

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;
namespace Aneka.MapReduce.Examples.LogParsing
{
    /// Class Program. Application driver. This class sets up the MapReduce
    /// job and configures it with the <i>LogParsingMapper</i> and <i>LogParsingReducer</i>
    /// classes. It also configures the MapReduce runtime in order sets the appropriate
    /// format for input and output files.
    public class Program
    {
    /// Reference to the configuration object.
    private static Configuration configuration = null;
    /// Location of the configuration file.
    private static string confPath = "conf.xml";
    /// Processes the arguments given to the application and according
    /// to the parameters read runs the application or shows the help.
    private static void Main(string[] args)
    {
        try
        {
            Logger.Start();
            // get the configuration
            Program.configuration = Program.Initialize(confPath);
            // configure MapReduceApplication
```

```
                MapReduceApplication<LogParsingMapper, LogParsingReducer> application =
                new MapReduceApplication<LogParsingMapper, LogParsingReducer>("LogParsing",
                configuration);
                // invoke and wait for result
                application.InvokeAndWait(newEventHandler<ApplicationEventArgs>(OnDone));
                // alternatively we can use the following call
                // application.InvokeAndWait();
        }
        catch(Exception ex)
        {
                Program.ReportError(ex);
        }
        finally
        {
                Logger.Stop();
        }
        Console.Readline();
}
/// Initializes the configuration and ensures that the appropriate input
/// and output formats . are set
private static Configuration Initialize(stringconfigFile)
{
        Configuration conf = Configuration.GetConfiguration(confPath);
        // we ensure that the input and the output formats are simple text files.
        PropertyGroup mapReduce = conf["MapReduce"];
        if (mapReduce == null)
        {
                mapReduce = newPropertyGroup("MapReduce");
                conf.Add("MapReduce") = mapReduce;
        }
        // handling input properties
        PropertyGroup group = mapReduce.GetGroup("Input");
        if (group == null)
        {
                group = newPropertyGroup("Input");
                mapReduce.Add(group);
        }
        string val = group["Format"];
        if (string.IsNullOrEmpty(val) == true)
        {
                group.Add("Format","text");
        }
        val = group["Filter"];
        if (string.IsNullOrEmpty(val) == true)
        {
                group.Add("Filter","*.log");
        }
        // handling output properties
        group = mapReduce.GetGroup("Output");
        if (group == null)
        {
                group = newPropertyGroup("Output");
```

```
                mapReduce.Add(group);
        }
        val = group["Format"];
        if (string.IsNullOrEmpty(val) == true)
        {
                group.Add("Format","text");
        }
        return conf;
}
/// Hooks the ApplicationFinished events and process the results
/// if the application has been successful.
private static void OnDone(object sender, ApplicationEventArgs e)
{
        if (e.Exception != null)
        {
                Program.ReportError(ex);
        }
else
{
        Console.Write("Aneka Log Parsing–Job Terminated: SUCCESS");
        FileStream logLevelStats = null;
        FileStream componentStats = null;
        string workspace = Program.configuration.Workspace;
        string outputDir = Path.Combine(workspace, "output");
        DirectoryInfo sources = new DirectoryInfo(outputDir);
        FileInfo[] results = sources.GetFiles();
        try
        {
                logLevelStats = new FileStream(Path.Combine(workspace,"loglevels.txt"),
                FileMode.Create,FileAccess.Write));
                componentStats = new FileStream(Path.Combine(workspace,"components.txt"),
                FileMode.Create,FileAccess.Write));
                using(StreamWriter logWriter = new StreamWriter(logLevelStats))
                {
                        using(StreamWritercompWriter = newStreamWriter(componentStats))
                        {
                                foreach(FileInfo result in results)
                                {
                                        using(StreamReader reader =new StreamReader(result.OpenRead()))
                                        {
                                                while(reader.EndOfStream == false)
                                                {
                                                        string line = reader.ReadLine();
                                                        if (line != null)
                                                        {
                                                                if (line.StartsWith("_ ") == true)
                                                                {
                                                                    compWriter.WriteLine(line.Substring(1));
                                                                }
                                                                else
                                                                {
                                                                    logWriter.WriteLine(line);
```

```
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                            // clear the output directory
                                            sources.Delete(true);
                                            Console.WriteLine("Statistics saved to:[loglevels.txt, components.txt]");
                                            Environment.ExitCode = 0;
                                    }
                                    catch(Exception ex)
                                    {
                                            Program.ReportError(ex);
                                    }
                                    Console.WriteLine("<Press Return>");
                            }
}
/// Displays a simple informative message explaining the usage of the application.
private static void Usage()
{
        Console.WriteLine("Aneka Log Parsing – Usage Log.Parsing.Demo.Console.exe"+ " [conf.xml]");
}
/// Dumps the error to the console, sets the exit code of the application to -1 and saves the error dump into a file.
private static void ReportError(Exception ex)
{
        IOUtil.DumpErrorReport(Console.Out, ex,"Aneka Log Parsing–Job Terminated: "+ "ERROR");
        IOUtil.DumpErrorReport(ex, "Aneka Log Parsing–Job Terminated: ERROR");
        Program.Usage();j
        Environment.ExitCode = -1;
}
}
}
```

LISTING 8.12 Driver Program Implementation.

Instead of a programming approach for the initialization of the configuration, it is also possible to embed these settings into the standard Aneka configuration file, as demonstrated in **Listing 8.13.**

```
<?xml version="1.0" encoding="utf -8" ?>
<Aneka>
        <UseFileTransfervalue="false" />
        <Workspacevalue="Workspace" />
        <SingleSubmission value="AUTO" />
        <PollingTimevalue="1000"/>
        <LogMessagesvalue="false" />
        <SchedulerUrivalue="tcp://localhost:9090/Aneka" />
        <UserCredential type="Aneka.Security.UserCredentials" assembly="Aneka.dll">
                <UserCredentials username="Administrator" password=""/>
        </UserCredentials>
        <Groups>
                <Group name="MapReduce">
```

```
                    <Groups>
                        <Group name="Input">
                        <Property name="Format" value="text" />
                        <Property name="Filter" value="*.log" />
                    </Group>
                    <Group name="Output">
                    <Property name="Format" value="text" />
                </Group>
        </Groups>
        <Property name="LogFile" value="Execution.log"/>
        <Property name="FetchResult" value="true" />
        <Property name="UseCombiner" value="true" />
        <Property name="SynchReduce" value="false" />
        <Property name="Partitions" value="1" />
        <Property name="Attempts" value="3" />
        </Group>
        </Groups>
</Aneka>
```

LISTING 8.13 Driver Program Configuration File (conf.xml)

5 Running the application

Aneka produces a considerable amount of logging information. The default configuration of the logging infrastructure creates a new log file for each activation of the Container process or as soon as the dimension of the log file goes beyond 10 MB. Therefore, by simply continuing to run an Aneka Cloud for a few days, it is quite easy to collect enough data to mine for our sample application.

In the execution of the test, we used a distributed infrastructure consisting of seven worker nodes and one master node interconnected through a LAN. We processed 18 log files of several sizes for a total aggregate size of 122 MB. The execution of the MapReduce job over the collected data produced the results that are stored in the loglevels.txt and components.txt files and represented graphically in **Figures 8.12 and 8.13**, respectively.

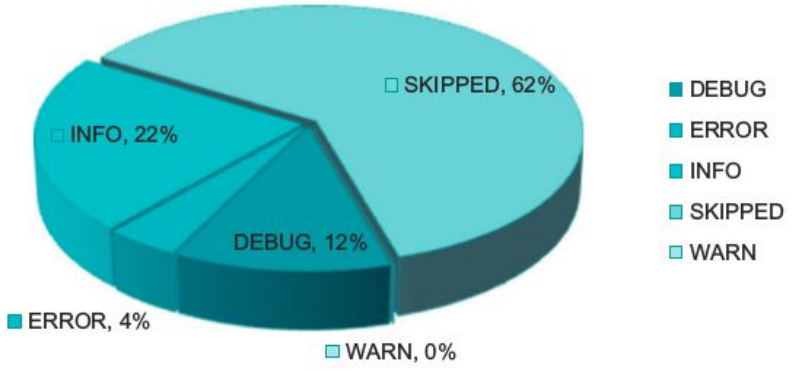


FIGURE 8.12

Log-level entries distribution.

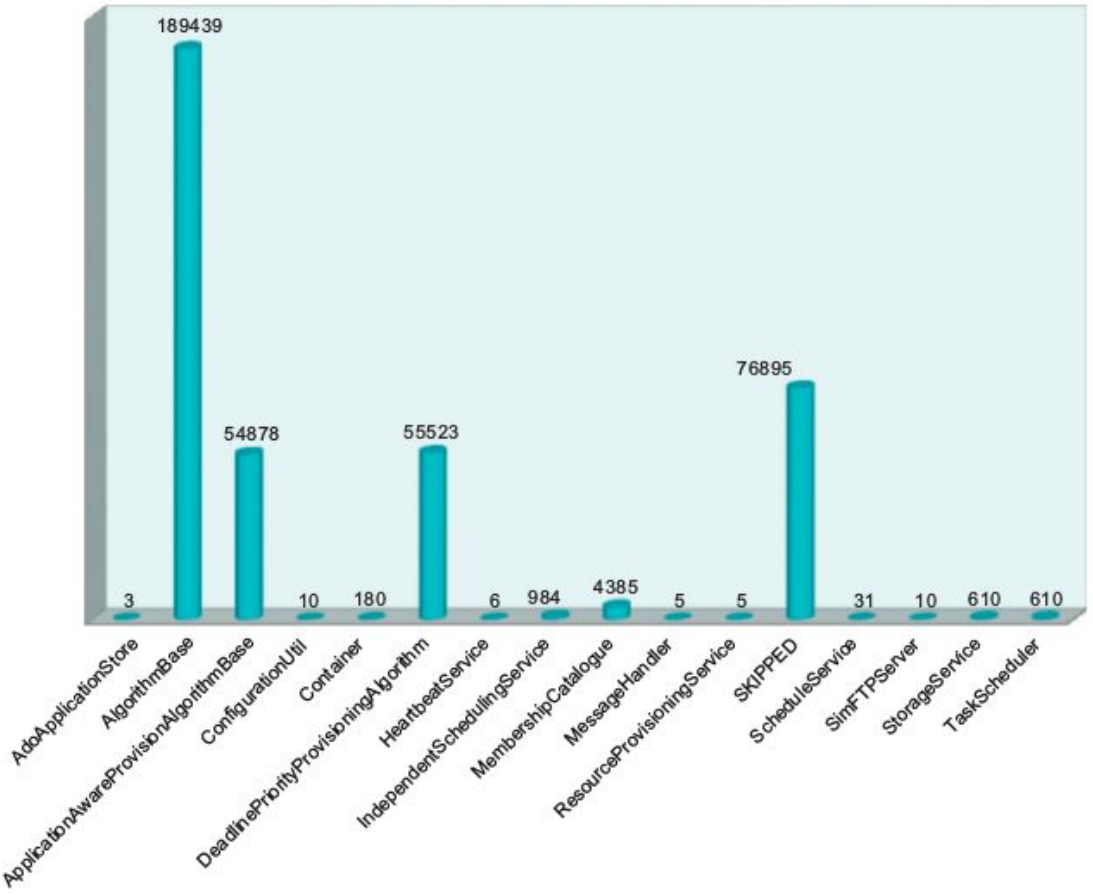


FIGURE 8.13

Component entries distribution.

The two graphs show that there is a considerable amount of unstructured information in the log files produced by the Container processes. In particular, about 60% of the log content is skipped during the classification. This content is more likely due to the result of stack trace dumps into the log file, which produces—as a result of ERROR and WARN entries—a sequence of lines that are not recognized. Figure 8.13 shows the distribution among the components that use the logging APIs. This distribution is computed over the data recognized as a valid log entry, and the graph shows that just about 20% of these entries have not been recognized by the parser implemented in the map function. We can then infer that the meaningful information extracted from the log analysis constitutes about 32% (80% of 40% of the total lines parsed) of the entire log data.

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.

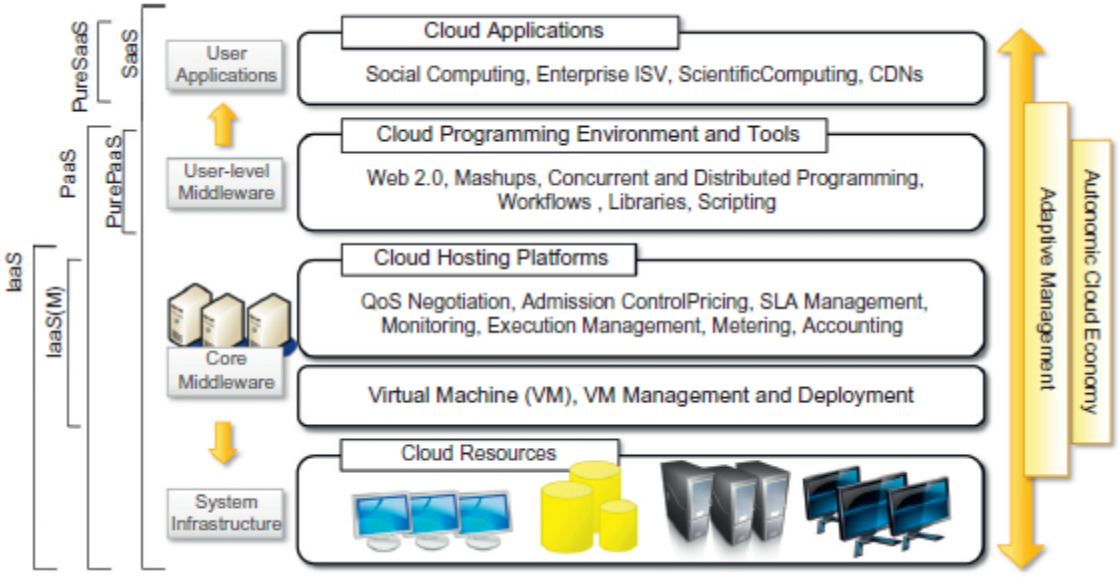


Figure 3.1 The cloud computing architecture.

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.

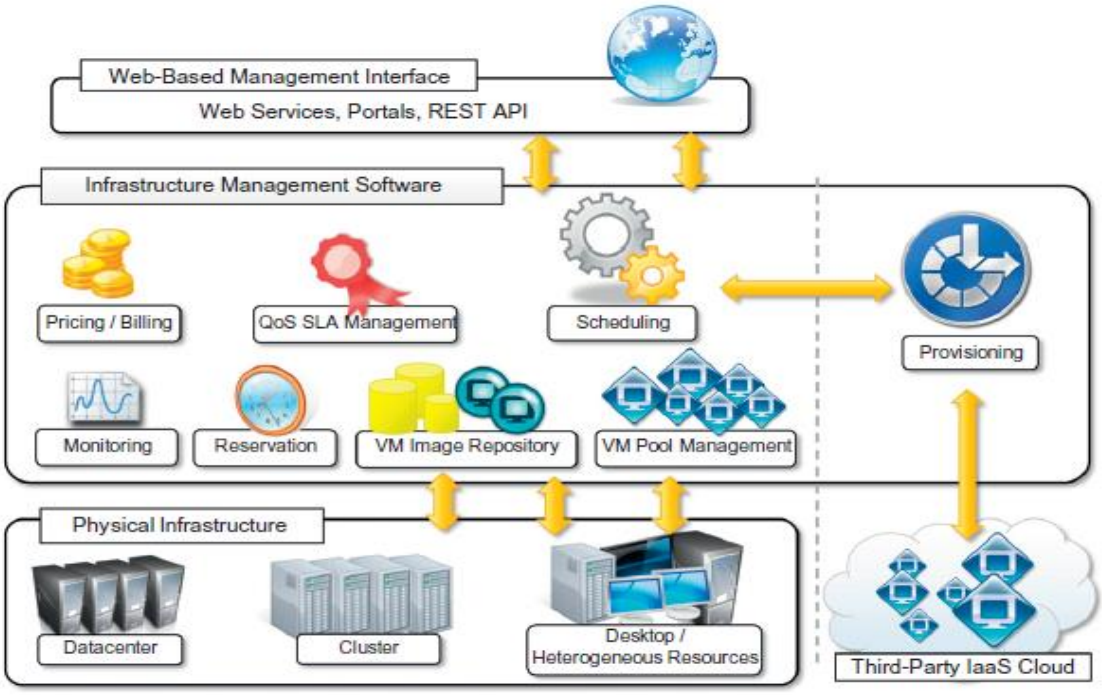


Figure 3.2 IaaS Reference implementation

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.

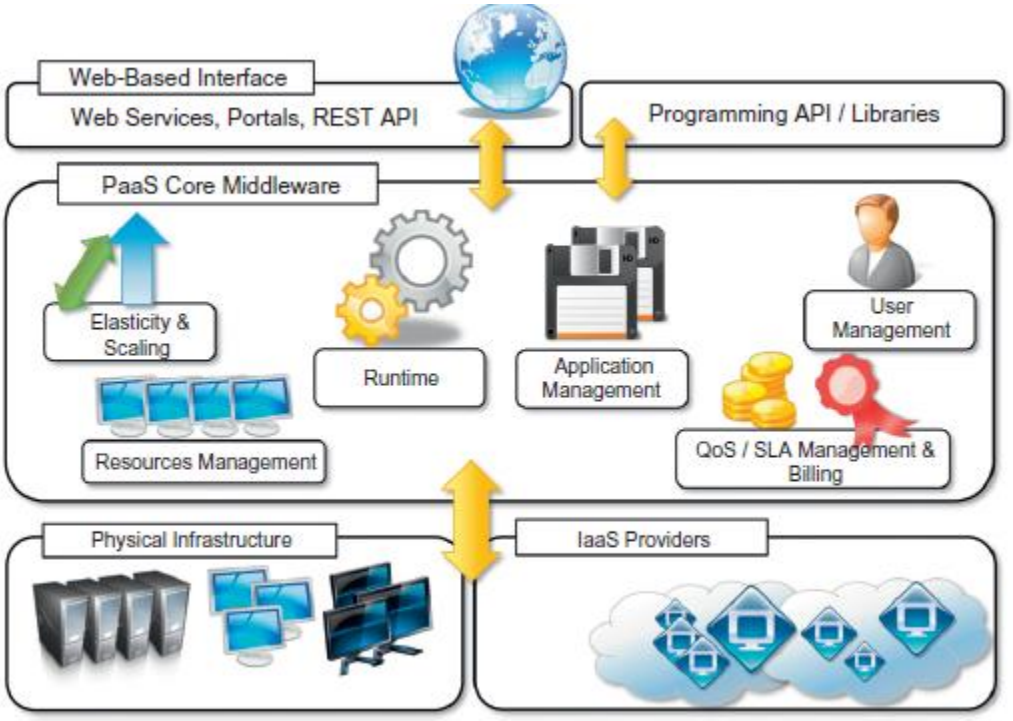


Figure 3.3 The Platform-as-a-Service reference model

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.

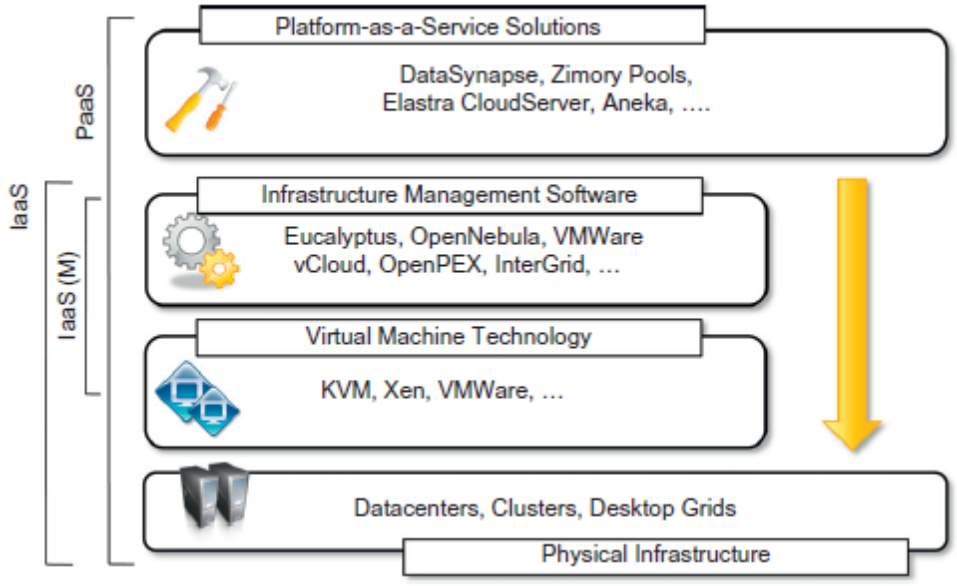


Figure 3.4 Private cloud software and hardware stack

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.

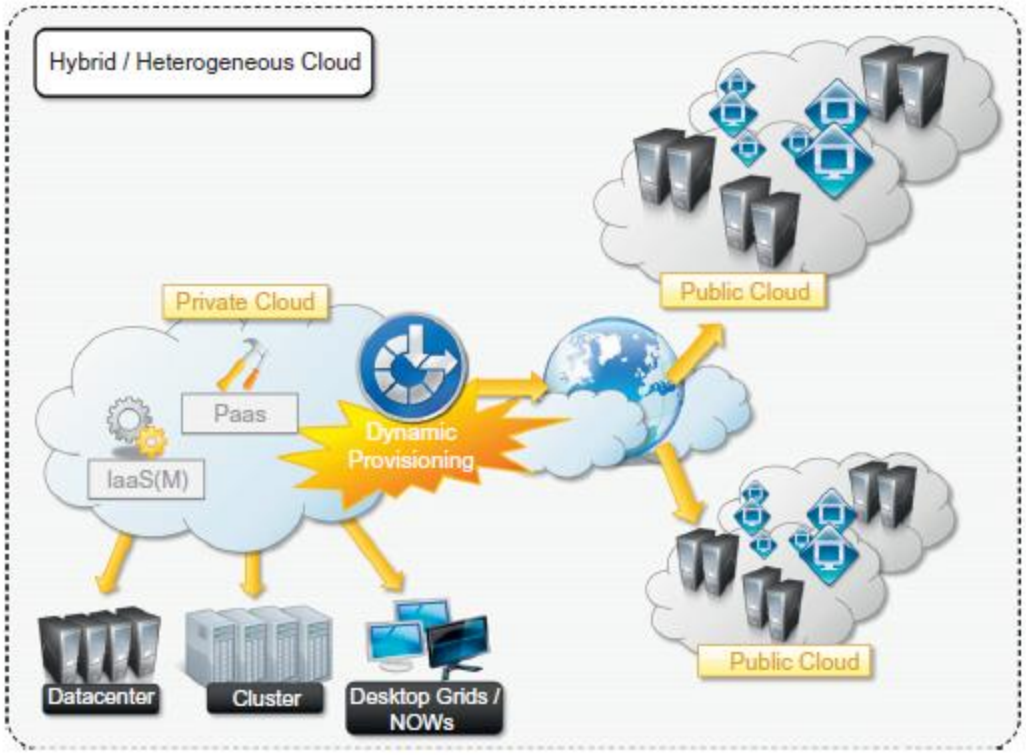


Figure 3.5 Hybrid/heterogeneous cloud overview.

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.

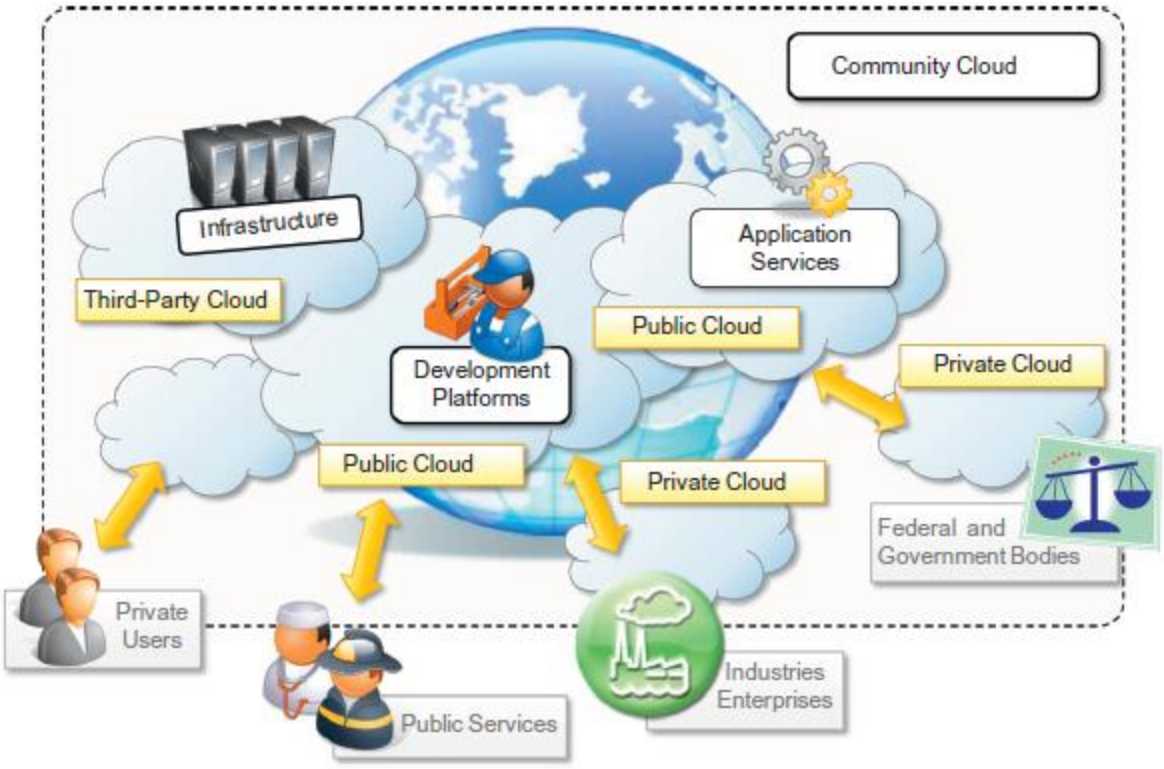


Figure 3.5 A Community Cloud

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

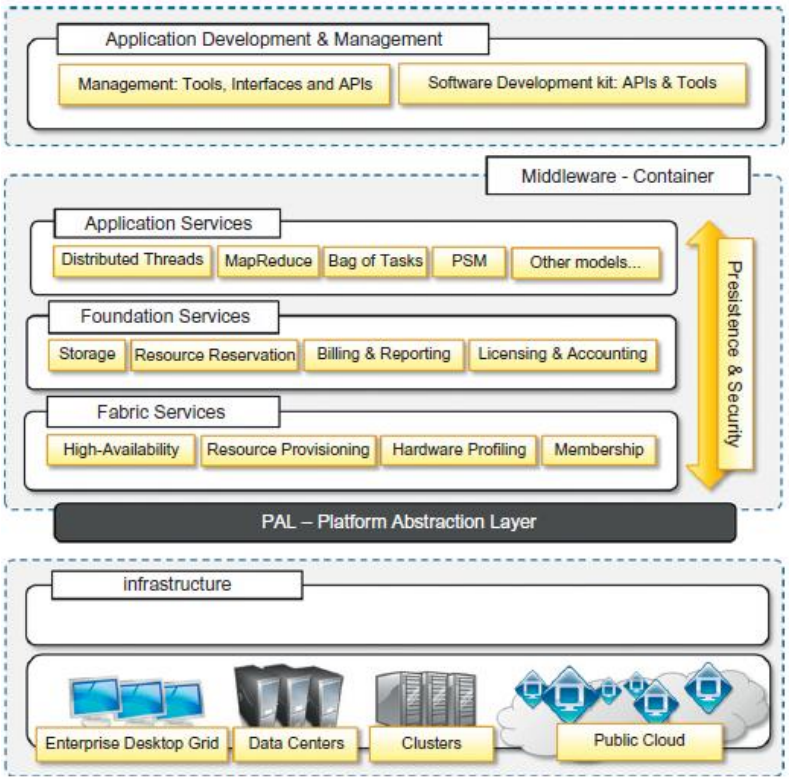
Module-2 (Section-2)

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.

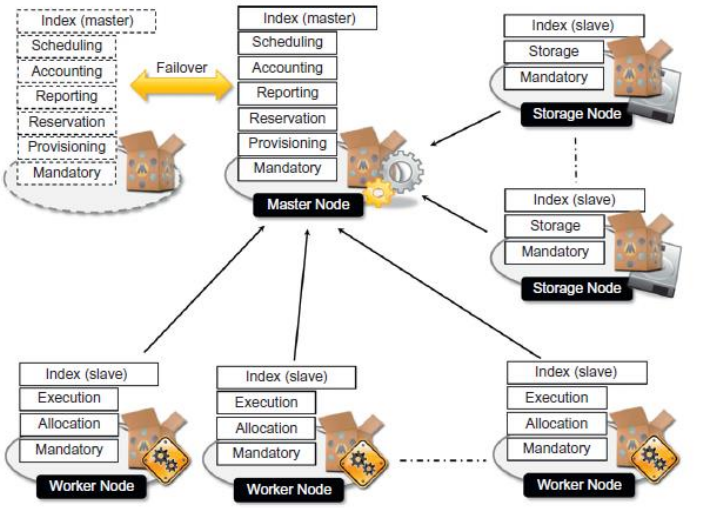


FIGURE 5.4
Logical organization of an Aneka cloud.

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.

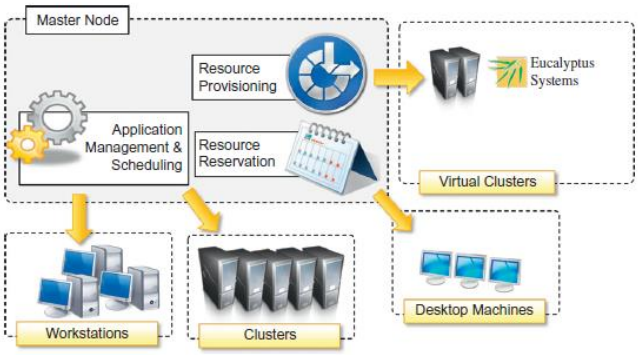


FIGURE 5.5
Private cloud deployment.

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.

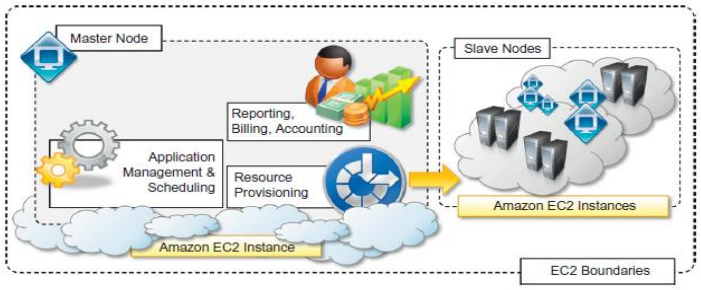


FIGURE 5.6
Public Aneka cloud deployment.

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

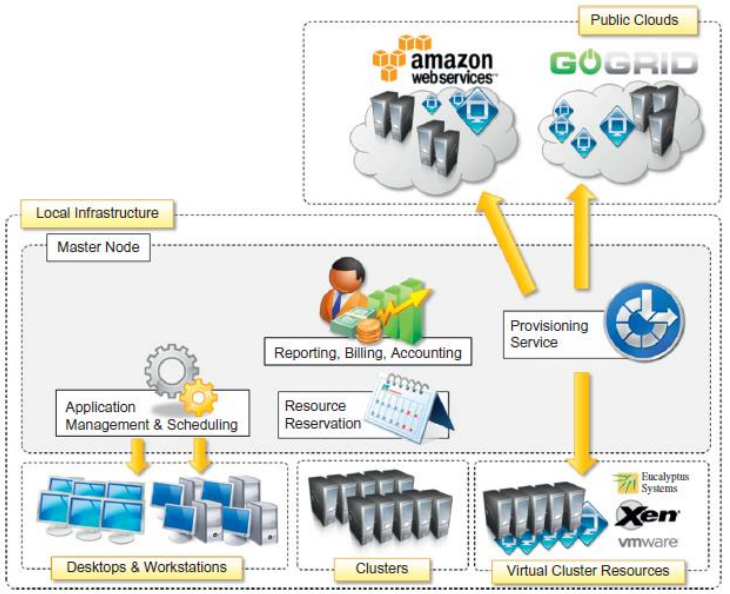


FIGURE 5.7
Hybrid cloud deployment.

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.

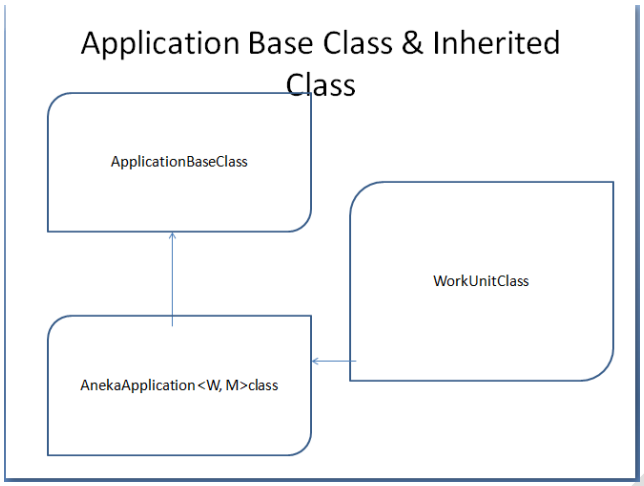
The SDK provides support for both programming models and services by

- The Application Model
- The Service Model.

Application Model

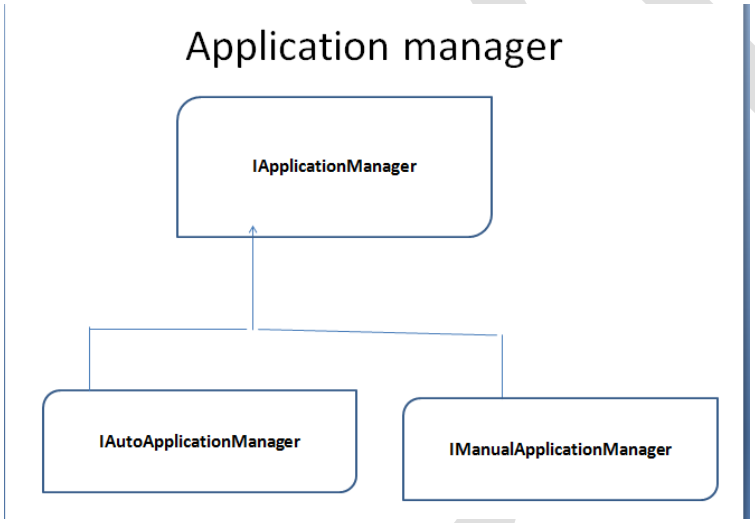
- The Application Model covers the development of applications and new programming models
- It Consists of Application Class & Application Manager
- Application Class – Provide user/developer view about distributed applications of the Aneka cloud
- Application Manager – Are Internal components that control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

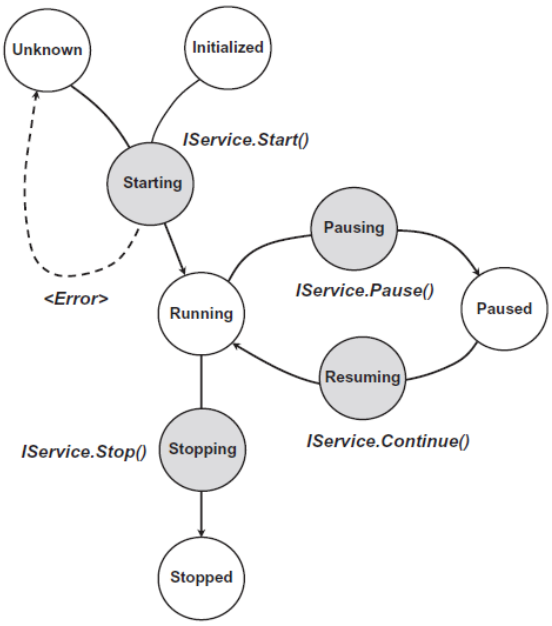


FIGURE 5.9
Service life cycle.

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 - 5

Cloud Platforms in Industry. (Chapter - 9)

9. Cloud Platforms in Industry

An overview of a few prominent cloud computing platforms and a brief description of the types of service they offer.

A cloud computing system can be developed using either a single technology and vendor or a combination of them.

This chapter presents some of representative cloud computing solutions offered as Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) services in the market.

9.1 Amazon web services

9.1 Amazon web services

Amazon Web Services (AWS) is a platform that allows the development of flexible applications by providing solutions for elastic infrastructure scalability, messaging, and data storage.

The platform is accessible through SOAP or RESTful Web service interfaces and provides a Web-based console where users can handle administration and monitoring of the resources required, as well as their expenses computed on a pay-as-you-go basis.

Figure 9.1 shows all the services available in the AWS ecosystem. At the base of the solution stack are services that provide raw compute and raw storage: Amazon Elastic Compute (EC2) and Amazon Simple Storage Service (S3).

FIGURE 9.1

Amazon Web Services ecosystem.

9.1.1 Compute services

The fundamental service in this space is Amazon EC2, which delivers an IaaS solution that has served as a

reference model for several offerings from other vendors in the same market segment.
Amazon EC2 allows deploying servers in the form of virtual machines created as instances of a specific image. Images come with a preinstalled operating system and a software stack, and instances can be configured for memory, number of processors, and storage.
Users are provided with credentials to remotely access the instance and further configure or install software if needed.
1. Amazon machine images
2. EC2 instances
3. EC2 environment
4. Advanced compute services

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 and identified by a unique identifier in the form of ami-xxxxxx and a manifest XML file.
AMI contains physical file system layout with a predefined operating system installed as Amazon Ramdisk Image (ARI, id: ari-yyyyyy) and the Amazon Kernel Image (AKI, id: aki-zzzzzz)
A common practice is to prepare new AMIs to create an instance from a preexisting AMI, log into it once it is booted and running, and install all the software needed. Using the tools provided by Amazon, we can convert the instance into a new image. Once an AMI is created, it is stored in an S3 bucket and the user can decide whether to make it available to other users or keep it for personal use.

2. EC2 instances
EC2 instances represent virtual machines. They are created using AMI as templates, which are specialized by selecting the number of cores, their computing power, and the installed memory. The processing power is expressed in terms of virtual cores and EC2 Compute Units (ECUs).
the use of ECUs helps give users a consistent view of the performance offered by EC2 instances.
One ECU is defined as giving the same performance as a 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor.
We can identify six major configurations for EC2 instances.

- **Standard instances.** offers set of configurations that are suitable for most applications. EC2 provides three different categories of increasing computing power, storage, and memory.
- **Micro instances.** suitable for those applications that consume limited amount of computing power and memory and need bursts in CPU cycles to process surges in workload.
- **High-memory instances.** targets applications that need to process huge workloads and require large amounts of memory. Three-tier Web applications characterized by high traffic are the target profile.
- **High-CPU instances.** targets compute-intensive applications.
- **Cluster Compute instances.** used to provide virtual cluster services. Instances in this category are characterized by high CPU compute power and large memory and an extremely high I/O and network performance, which makes it suitable for HPC applications.
- **Cluster GPU instances.** provides instances featuring graphic processing units (GPUs) and high compute power, large memory, and extremely high I/O and network performance. This class is particularly suited for cluster applications that perform heavy graphic computations.

3. EC2 environment
EC2 instances are executed within a virtual environment. The EC2 environment is in charge of allocating addresses, attaching storage volumes, and configuring security in terms of access control and network connectivity.
It is possible to associate an Elastic IP to each instance. Elastic IPs allow instances running in EC2 to act as servers reachable from the Internet.
EC2 instances are given domain name in the form ec2-xxx-xxx-xxx.compute-x.amazonaws.com,
where xxx-xxx-xxx normally represents the four parts of the external IP address separated by a dash, and compute-x gives information about the availability zone where instances are deployed.
Currently, there are five availability zones: two in the United States (Virginia and Northern California), one in Europe (Ireland), and two in Asia Pacific (Singapore and Tokyo).

4. Advanced compute services
AWS CloudFormation constitutes an extension of the simple deployment model that characterizes EC2 instances. CloudFormation introduces the concepts of templates, which are JSON formatted text files that

describe the resources needed to run an application or a service in EC2 together.
Templates provide a simple and declarative way to build complex systems and integrate EC2 instances with other AWS services such as S3, SimpleDB, SQS, SNS, Route 53, Elastic Beanstalk, and others.
AWS Elastic Beanstalk constitutes a simple and easy way to package applications and deploy them on the AWS Cloud. This service simplifies the process of provisioning instances and deploying application code and provides appropriate access to them.
Currently, this service is available for Web applications developed with the Java/Tomcat technology stack.
Beanstalk simplifies tedious tasks without removing the user's capability of accessing—and taking over control of—the underlying EC2 instances.
Amazon Elastic MapReduce provides AWS users with a cloud computing platform for MapReduce applications. It utilizes Hadoop as the MapReduce engine, deployed on a virtual infrastructure composed of EC2 instances, and uses Amazon S3 for storage needs.
Elastic MapReduce introduces elasticity and allows users to dynamically size the Hadoop cluster according to their needs, as well as select the appropriate configuration of EC2 instances to compose the cluster.

9.1.2 Storage services

The core service is represented by Amazon Simple Storage Service (S3). This is a distributed object store that allows users to store information in different formats. The core components of S3 are two: buckets and objects. Buckets represent virtual containers in which to store objects; objects represent the content that is actually stored. Objects can also be enriched with metadata that can be used to tag the stored content with additional information.
1 S3 key concepts
2 Amazon elastic block store
3 Amazon ElastiCache
4 Structured storage solutions
5 Amazon CloudFront

1 S3 key concepts
S3 has been designed to provide a simple storage service that's 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.

Access to S3 is provided with RESTful Web services. These express all the operations that can be performed on the storage in the form of HTTP requests (GET, PUT, DELETE, HEAD, and POST).
Resource naming
Buckets, objects, and attached metadata are made accessible through a REST interface. Therefore, they are represented by uniform resource identifiers (URIs) under the s3.amazonaws.com domain.
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
A bucket is a container of objects. It can be thought of as a virtual drive hosted on the S3 distributed storage, which provides users with a flat store to which they can add objects. Buckets are top-level elements of the S3 storage architecture and do not support nesting. That is, it is not possible to create "subbuckets" or other kinds of physical divisions.
Objects and metadata
Objects constitute the content elements stored in S3. Users either store files or push to the S3 text stream representing the object's content. An object is identified by a name that needs to be unique within the bucket in which the content is stored. The name cannot be longer than 1,024 bytes when encoded in UTF-8, and it allows almost any character. Buckets do not support nesting.
Access control and security

Amazon S3 allows controlling the access to buckets and objects by means of Access Control Policies (ACPs). An ACP is a set of grant permissions that are attached to a resource expressed by means of an XML configuration file.

A policy allows defining up to 100 access rules, each of them granting one of the available permissions to a grantee.

Currently, five different permissions can be used:

A. READ allows the grantee to retrieve an object and its metadata and to list the content of a bucket as well as getting its metadata.
B. WRITE allows the grantee to add an object to a bucket as well as modify and remove it.
C. READ_ACP allows the grantee to read the ACP of a resource.
D. WRITE_ACP allows the grantee to modify the ACP of a resource.
E. FULL_CONTROL grants all of the preceding permissions.

2 Amazon elastic block store

The Amazon Elastic Block Store (EBS) allows AWS users to provide EC2 instances with persistent storage in the form of volumes that can be mounted at instance startup. They accommodate up to 1 TB of space and are accessed through a block device interface, thus allowing users to format them according to the needs of the instance they are connected to.

EBS volumes normally reside within the same availability zone of the EC2 instances that will use them to maximize the I/O performance. It is also possible to connect volumes located in different availability zones. Once mounted as volumes, their content is lazily loaded in the background and according to the request made by the operating system. This reduces the number of I/O requests that go to the network.

3 Amazon ElastiCache

ElastiCache is an implementation of an elastic in-memory cache based on a cluster of EC2 instances.

It provides fast data access through a Memcached-compatible protocol so that applications can transparently migrate to ElastiCache.

ElastiCache is based on a cluster of EC2 instances running the caching software, which is made available through Web services.

An ElastiCache cluster can be dynamically resized according to the demand of the client applications.

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. Available AMIs include installations of 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 with patches. Moreover, the service provides users with automatic backups, snapshots, point-in-time recoveries, and facilities for implementing replications.

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. SimpleDB provides support for semistructured data, the model for which is based on the concept of domains, items, and attributes.

SimpleDB uses domains as top-level elements to organize a data store. These domains are roughly comparable to tables in the relational model. Unlike tables, they allow items not to have all the same column structure; each item is therefore represented as a collection of attributes expressed in the form of a key-value pair.

5 Amazon CloudFront

CloudFront is an implementation of a content delivery network on top of the Amazon distributed storage infrastructure. It leverages a collection of edge servers strategically located around the globe to better serve requests for static and streaming Web content so that the transfer time is reduced.

AWS provides users with simple Web service APIs to manage CloudFront. To make available content through CloudFront, it is necessary to create a distribution. This identifies an origin server, which contains the original version of the content being distributed, and it is referenced by a DNS domain under the Cloudfront.net domain name.

The content that can be delivered through CloudFront is static (HTTP and HTTPS) or streaming (Real Time Messaging Protocol, or RMTP).

9.1.3 Communication services

Amazon provides 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.

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.
Amazon VPC provides flexibility in creating virtual private networks within the Amazon infrastructure and beyond. The service providers prepare templates for network service for advanced configurations. Templates include public subnets, isolated networks, private networks accessing Internet through network address translation (NAT), and hybrid networks including AWS resources and private resources.
Amazon Direct Connect allows AWS users to create dedicated networks between the user private network and Amazon Direct Connect locations, called ports. This connection can be further partitioned in multiple logical connections and give access to the public resources hosted on the Amazon infrastructure. The advantage is the consistent performance of the connection between the users premises and the Direct Connect locations.
Amazon Route 53 implements dynamic DNS services that allow AWS resources to be reached through domain names different from the amazon.com domain. By leveraging the large and globally distributed network of Amazon DNS servers.

2. Messaging.

The three different types of messaging services offered are

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

Amazon SQS constitutes disconnected model for exchanging messages between applications by means of message queues, hosted within the AWS infrastructure. Using the AWS console or directly the underlying Web service AWS, users can create an unlimited number of message queues and configure them to control their access. Applications can send messages to any queue they have access to. These messages are securely and redundantly stored within the AWS infrastructure for a limited period of time, and they can be accessed by other (authorized) applications.
Amazon SNS provides a publish-subscribe method for connecting heterogeneous applications. Amazon SNS allows applications to be notified when new content of interest is available. This feature is accessible through a Web service whereby AWS users can create a topic, which other applications can subscribe.
Amazon SES provides AWS users with a scalable email service that leverages the AWS infrastructure. Once users are signed up for the service, they have to provide an email that SES will use to send emails on their behalf. To activate the service, SES will send an email to verify the given address and provide the users with the necessary information for the activation.

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

AppEngine is a platform for developing scalable applications accessible through the Web. **Figure 9.2.**
The platform is logically divided into four major components: infrastructure, the runtime environment, the underlying storage, and the set of scalable services.

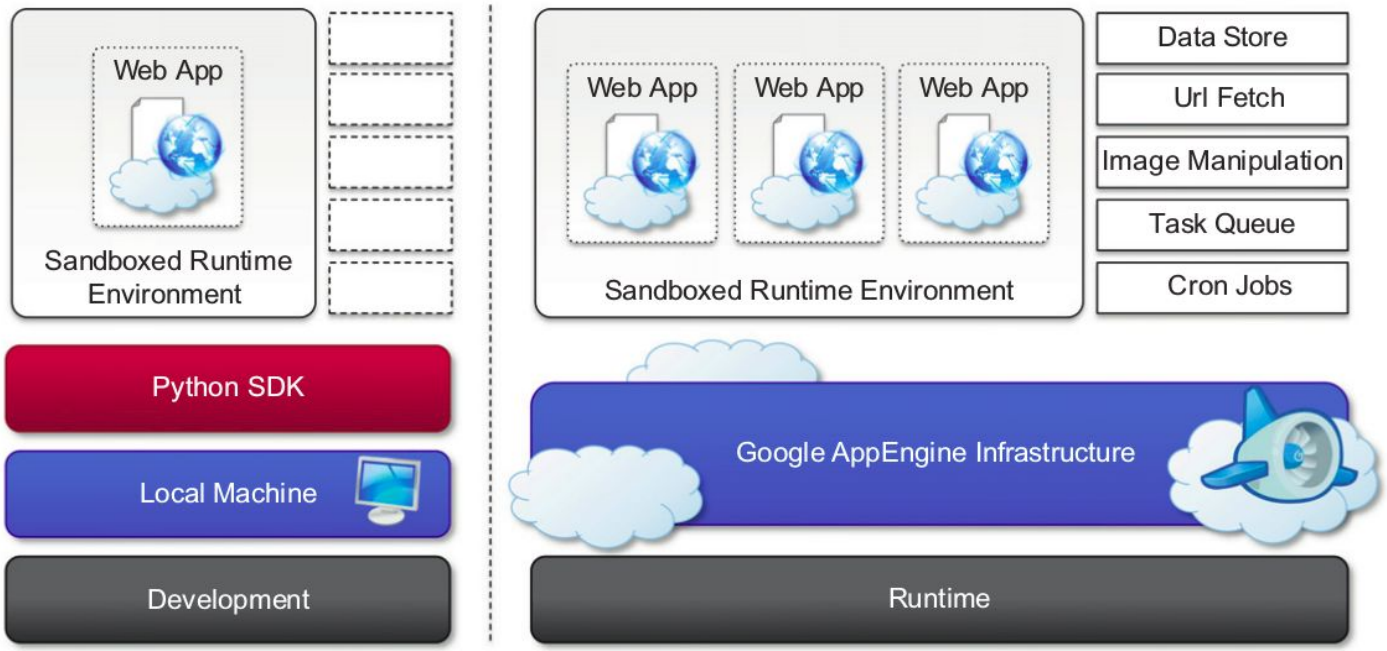


FIGURE 9.2

Google AppEngine platform architecture.

1 Infrastructure

AppEngine hosts Web applications, and its primary function is to serve users requests efficiently.

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 or redirects the request to an existing server.

The infrastructure is also responsible for monitoring application performance and collecting statistics on which the billing is calculated.

2 Runtime environment

The runtime environment represents the execution context of applications hosted on AppEngine.

Sandboxing- One of the major responsibilities of the runtime environment is to provide the application environment with an isolated and protected context in which it can execute without causing a threat to the server and without being influenced by other applications. In other words, it provides applications with a sandbox.

If an application tries to perform any operation that is considered potentially harmful, an exception is thrown and the execution is interrupted.

Supported runtimes- Currently, it is possible to develop AppEngine applications using three different languages and related technologies: Java, Python, and Go.

AppEngine currently supports Java 6, and developers can use the common tools for Web application development in Java, such as the Java Server Pages (JSP), and the applications interact with the environment by using the Java Servlet standard.

Support for Python is provided by an optimized Python 2.5.2 interpreter. As with Java, the runtime environment supports the Python standard library.

Developers can use a specific Python Web application framework, called webapp, simplifying the development of Web applications.

The Go runtime environment allows applications developed with the Go programming language to be hosted and executed in AppEngine. Currently the release of Go that is supported by AppEngine is r58.1. The SDK includes the compiler and the standard libraries for developing applications in Go and interfacing it with AppEngine services.

3 Storage

AppEngine provides various types of storage, which operate differently depending on the volatility of the data.

Static file servers- Web applications are composed of dynamic and static data. Dynamic data are a result of the logic of the application and the interaction with the user. Static data often are mostly constituted of the components that define the graphical layout of the application or data files.

DataStore- DataStore is a service that allows developers to store semistructured data. The service is designed

to scale and optimized to quickly access data. DataStore can be considered as a large object database in which to store objects that can be retrieved by a specified key.
DataStore imposes less constraint on the regularity of the data but, at the same time, does not implement some of the features of the relational model.
The underlying infrastructure of DataStore is based on Bigtable, a redundant, distributed, and semistructured data store that organizes data in the form of tables.
DataStore provides high-level abstractions that simplify interaction with Bigtable. Developers define their data in terms of entity and properties, and these are persisted and maintained by the service into tables in Bigtable.
DataStore also provides facilities for creating indexes on data and to update data within the context of a transaction. Indexes are used to support and speed up queries. A query can return zero or more objects of the same kind or simply the corresponding keys.

4 Application services

Applications hosted on AppEngine take the most from the services made available through the runtime environment. These services simplify most of the common operations that are performed in Web applications

UrlFetch - The sandbox environment does not allow applications to open arbitrary connections through sockets, but it does provide developers with the capability of retrieving a remote resource through HTTP/HTTPS by means of the UrlFetch service. Applications can make synchronous and asynchronous Web requests and integrate the resources obtained in this way into the normal request- handling cycle of the application.
UrlFetch is not only used to integrate meshes into a Web page but also to leverage remote Web services in accordance with the SOA reference model for distributed applications.

MemCache- This is a distributed in-memory cache that is optimized for fast access and provides developers with a volatile store for the objects that are frequently accessed. The caching algorithm implemented by MemCache will automatically remove the objects that are rarely accessed. The use of MemCache can significantly reduce the access time to data; developers can structure their applications so that each object is first looked up into MemCache and if there is a miss, it will be retrieved from DataStore and put into the cache for future lookups.

Mail and instant messaging- AppEngine provides developers with the ability to send and receive mails through Mail. The service allows sending email on behalf of the application to specific user accounts. It is also possible to include several types of attachments and to target multiple recipients.
AppEngine provides also another way to communicate with the external world: the 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, which is identified by its own address.

Account management- AppEngine simplifies account management by allowing developers to leverage Google account management by means of Google Accounts.
Using Google Accounts, 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.

5 Compute services

AppEngine offers additional services such as Task Queues and Cron Jobs that simplify the execution of computations.

Task queues- A task is defined by a Web request to a given URL, and the queue invokes the request handler by passing the payload as part of the Web request to the handler. It is the responsibility of the request handler to perform the "task execution," which is seen from the queue as a simple Web request.

Cron jobs- the required operation needs to be performed at a specific time of the day, which does not coincide with the time of the Web request. In this case, it is possible to schedule the required operation at the desired time by using the Cron Jobs service.

9.2.2 Application life cycle

AppEngine provides support for all the phases characterizing the life cycle of an application: testing and development, deployment, and monitoring.

1 Application development and testing

Developers can start building their Web applications on a local development server.
The development server simulates the AppEngine runtime environment by providing a mock implementation of DataStore, MemCache, UrlFetch, and the other services leveraged by Web applications.

AppEngine builds indexes for each of the queries performed by a given application in order to speed up access to the relevant data. This capability is enabled by a priori knowledge about all the possible queries made by the application; such knowledge is made available to AppEngine by the developer while uploading the application.

Java SDK- The Java SDK provides developers with the facility for building applications with the Java 5 and Java 6 runtime environments. Using the Eclipse software installer, it is possible to download and install Java SDK, Google Web Toolkit, and Google AppEngine plug-ins into Eclipse. These three components allow developers to program powerful and rich Java applications for AppEngine.

Python SDK- The Python SDK allows developing Web applications for AppEngine with Python 2.5. It provides a standalone tool, called GoogleAppEngineLauncher, for managing Web applications locally and deploying them to AppEngine.

The Python implementation of the SDK also comes with an integrated Web application framework called webapp that includes a set of models, components, and tools that simplify the development of Web applications and enforce a set of coherent practices.

The webapp framework has been reimplemented and made available in the Python SDK so that it can be used with AppEngine.

2 Application deployment and management

Once application has been developed and tested, it can be deployed on AppEngine with simple click or command-line tool. Before performing such task, it is necessary to create application identifier, which will be used to locate application from Web browser by typing the address http://<application-id>.appspot.com.

An application identifier is mandatory because it allows unique identification of the application while it's interacting with AppEngine. Developers use an app identifier to upload and update applications.

Once an application identifier has been created, it is possible to deploy an application on AppEngine. This task can be done using either respective development environment (GoogleAppEngineLauncher and Google AppEngine plug-in) or the command-line tools.

9.2.3 Cost model

Once application has been tested and tuned for AppEngine, it is possible to set up a billing account and obtain more allowance and be charged on a pay-per-use basis. This allows developers to identify appropriate daily budget that they want to allocate for given application.

An application is measured against **billable quotas, fixed quotas, and per-minute quotas.**

Google AppEngine uses these quotas to ensure that users do not spend more than the allocated budget and that applications run without being influenced by each other from a performance point of view.

Billable quotas identify the daily quotas that are set by application administrator and are defined by daily budget allocated for application.

Free quotas are part of the billable quota and identify the portion of the quota for which users are not charged.

Fixed quotas are internal quotas set by AppEngine that identify infrastructure boundaries and define operations that application can carry out on infrastructure.

9.2.4 Observations

AppEngine, a framework for developing scalable Web applications, leverages Google's infrastructure.

The core components of service are scalable and sandboxed runtime environment for executing applications and a collection of services that implement most of the common features.

One of the characteristic elements of AppEngine is use of simple interfaces that allow applications to perform specific operations that are optimized and designed to scale.

Building on top of these blocks, developers can build applications and let AppEngine scale them out when needed.

9.3 Microsoft Azure

Microsoft Windows Azure is a cloud operating system built on top of Microsoft datacenters' infrastructure and provides developers with a collection of services for building applications with cloud technology.

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. These services can be managed and controlled through the Windows Azure Management Portal, which acts as an administrative console for all the services

offered by the Azure platform.

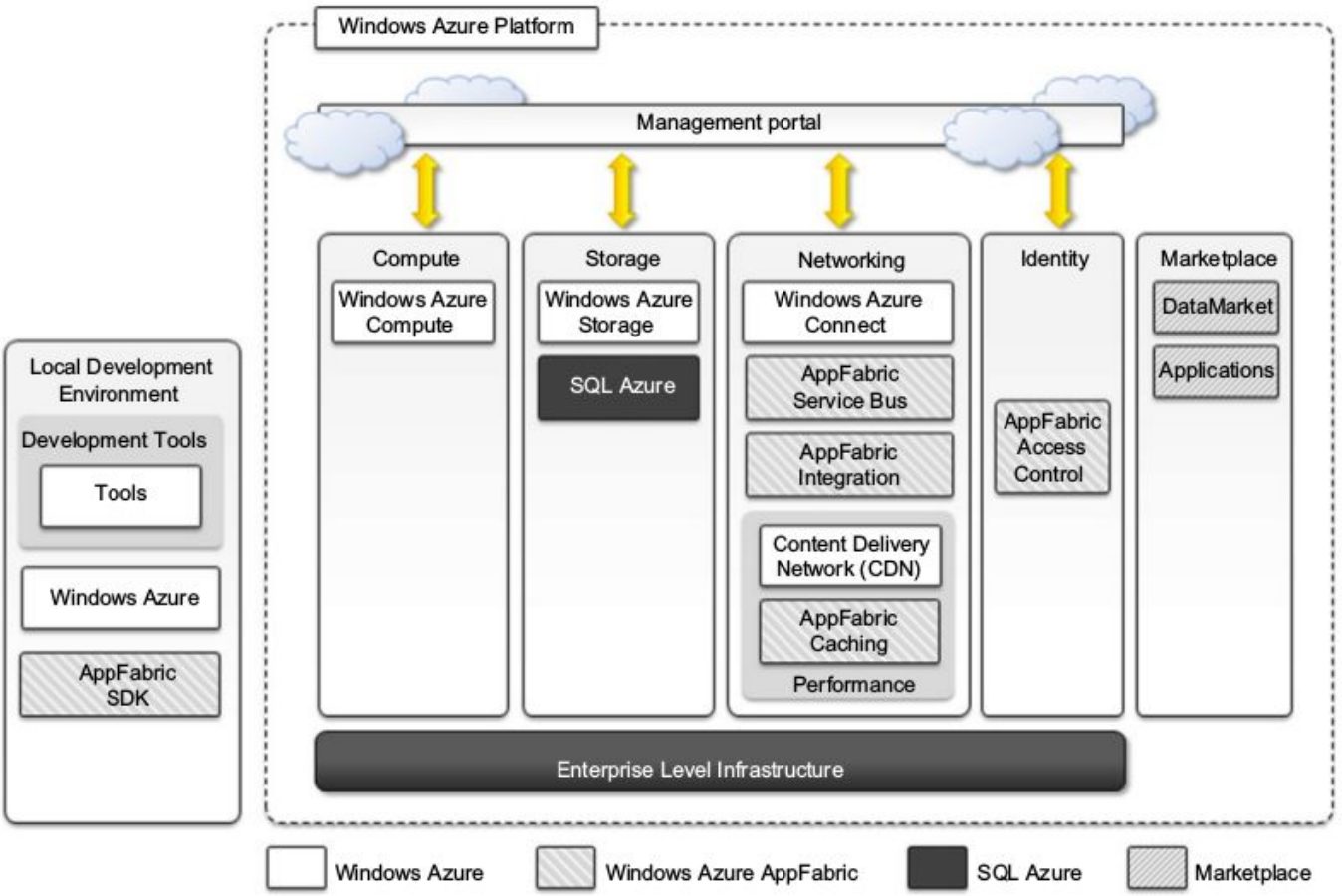


FIGURE 9.3

Microsoft Windows Azure Platform Architecture.

9.3.1 Azure core concepts

The Windows Azure platform is made up of a foundation layer and a set of developer services that can be used to build scalable applications. These services cover compute, storage, networking, and identity management, which are tied together by middleware called AppFabric.

1 Compute services

Compute services are the core components of Microsoft Windows Azure, and they are delivered by means of the abstraction of roles.

Currently, there are three different roles: Web role, Worker role, and Virtual Machine (VM) role.

Web role- The Web role is designed to implement scalable Web applications. Web roles represent the units of deployment of Web applications within the Azure infrastructure. They are hosted on the IIS 7 Web Server.

Since version 3.5, the .NET technology natively supports Web roles; developers can directly develop their applications in Visual Studio, test them locally, and upload to Azure.

Web roles can be used to run and scale PHP Web applications on Azure (CGI Web Role).

Worker role - Worker roles are designed to host general compute services on Azure. They can be used to quickly provide compute power or to host services that do not communicate with the external world through HTTP. A common practice for Worker roles is to use them to provide background processing for Web applications developed with Web roles.

A Worker role runs continuously from the creation of its instance until it is shut down. The Azure SDK provides developers with convenient APIs and libraries that allow connecting the role with the service provided by the runtime and easily controlling its startup as well as being notified of changes in the hosting environment.

Virtual machine role- The Virtual Machine role allows developers to control computing stack of their compute service by defining a custom image of the Windows Server 2008 R2 operating system and all the service stack required by their applications. The Virtual Machine role is based on the Windows Hyper-V virtualization technology.

Developers can image a Windows server installation complete with all the required applications and

components, save it into a Virtual Hard Disk (VHD).

2 Storage services

Compute resources are equipped with local storage in the form of a directory on the local file system.

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

Blobs

Azure allows storing large amount of data in the form of binary large objects (BLOBs) by means of the blobs service.

Block blobs- Block blobs are composed of blocks and are optimized for sequential access; therefore they are appropriate for media streaming. Currently, blocks are of 4 MB, and a single block blob can reach 200 GB in dimension.

Page blobs- Page blobs are made of pages that are identified by offset from beginning of blob. A page blob can be split into multiple pages or constituted of single page. This type of blob is optimized for random access and can be used to host data different from streaming. Maximum dimension of page blob can be 1TB.

Azure drive

Page blobs can be used to store an entire file system in the form of a single Virtual Hard Drive (VHD) file. This can then be mounted as a part of the NTFS file system by Azure compute resources, thus providing persistent and durable storage.

Tables

Tables constitute a semistructured storage solution, allowing users to store information in the form of entities with a collection of properties. Entities are stored as rows in the table and are identified by a key, which also constitutes the unique index built for the table. Users can insert, update, delete, and select a subset of the rows stored in the table.

Currently, a 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. The maximum dimension of a row key and partition keys is 1 KB.

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.

3 Core infrastructure: AppFabric

AppFabric is a comprehensive middleware for developing, deploying, and managing applications on the cloud or for integrating existing applications with cloud services.

AppFabric implements an optimized infrastructure supporting scaling out and high availability; sandboxing and multitenancy; state management; and dynamic address resolution and routing.

Access control- AppFabric provides the capability of encoding access control to resources in Web applications and services into a set of rules that are expressed outside the application code base. These rules give a great degree of flexibility in terms of the ability to secure components of the application and define access control policies for users and groups.

Service bus - Service Bus constitutes the messaging and connectivity infrastructure provided with AppFabric for building distributed and disconnected applications The service is designed to allow transparent network traversal and to simplify the development of loosely coupled applications, without renouncing security and reliability and letting developers focus on the logic of the interaction rather than the details of its implementation. Service Bus allows services to be available by simple URLs, which are untied from their deployment location.

Azure cache- Windows Azure 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. The service implements a distributed in-memory cache of which the size can be dynamically adjusted by applications according to their needs.

4 Other services

Windows Azure virtual network- Networking services for applications are offered under the name Windows Azure Virtual Network, which includes Windows Azure Connect and Windows Azure Traffic Manager. Windows Azure Connect allows easy setup of IP-based network connectivity among machines hosted on the private premises and the roles deployed on the Azure Cloud. This service is particularly useful in the case of

VM roles.

Windows Azure content delivery network- Windows Azure Content Delivery Network (CDN) is the content delivery network solution that improves the content delivery capabilities of Windows Azure Storage and several other Microsoft services, such as Microsoft Windows Update and Bing maps.

9.3.2 SQL Azure

SQL Azure is a relational database service hosted on Windows Azure and built on the SQL Server technologies. The service extends the capabilities of SQL Server to the cloud and provides developers with a scalable, highly available, and fault-tolerant relational database. SQL Azure is accessible from either the Windows Azure Cloud or any other location that has access to the Azure Cloud. It is fully compatible with the interface exposed by SQL Server, so applications built for SQL Server can transparently migrate to SQL Azure. **Figure 9.4** shows the architecture of SQL Azure. Access to SQL Azure is based on the Tabular Data Stream (TDS) protocol, which is the communication protocol underlying all the different interfaces used by applications to connect to a SQL Server-based installation such as ODBC and ADO.NET.

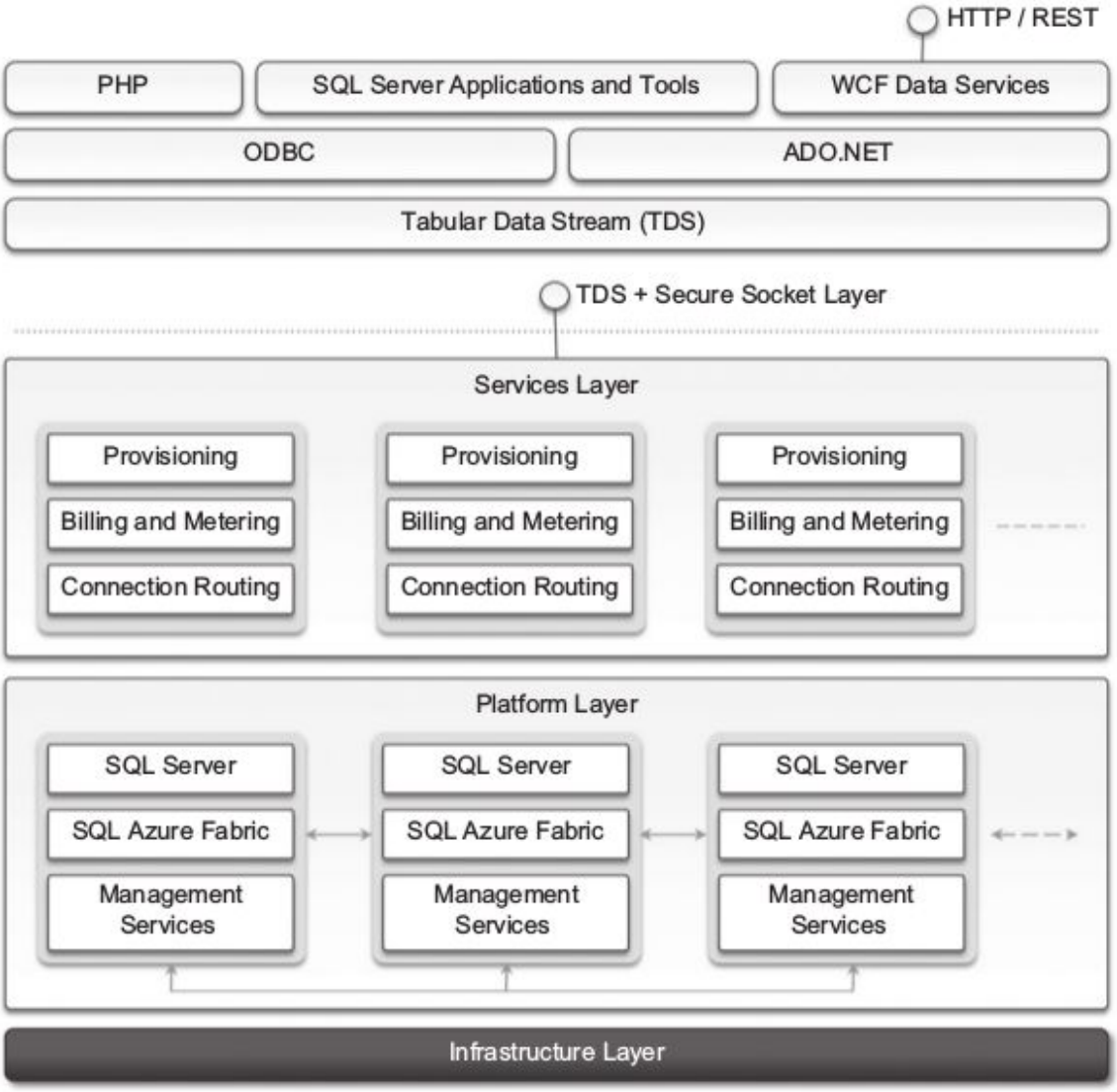


FIGURE 9.4

SQL Azure architecture.

Developers have to sign up for a Windows Azure account in order to use SQL Azure. Once the account is activated, they can either use the Windows Azure Management Portal or the REST APIs to create servers and logins and to configure access to servers.

SQL Azure servers are abstractions that 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 the management tasks and the interaction with SQL Azure from client applications.

Currently, two different editions are available: Web Edition and Business Edition. The former is suited for small Web applications and supports databases with a maximum size of 1 GB or 5 GB. The latter is suited for

independent software vendors, line-of-business applications, and enterprise applications and supports databases with a maximum size from 10 GB to 50 GB, in increments of 10 GB.

9.3.3 Windows Azure platform appliance

The Windows Azure platform can also be deployed as an appliance on third-party data centers and constitutes the cloud infrastructure governing the physical servers of the datacenter. The Windows Azure Platform Appliance includes Windows Azure, SQL Azure, and Microsoft- specified configuration of network, storage, and server hardware. The appliance is a solution that targets governments and service providers who want to have their own cloud computing infrastructure.

9.3.4 Observations

Windows Azure is Microsoft's solution for developing cloud computing applications. Azure is an implementation of the PaaS layer and provides the developer with a collection of services and scalable middleware hosted on Microsoft datacenters that address compute, storage, networking, and identity management needs of applications.

The core components of the platform are composed of compute services, storage services, and middleware. Compute services are based on the abstraction of roles, which identify a sandboxed environment where developers can build their distributed and scalable components.

SQL Azure is another important element of Windows Azure and provides support for relational data in the cloud. SQL Azure is an extension of the capabilities of SQL Server adapted for the cloud environment and designed for dynamic scaling.

Module - 5

Cloud Applications. (Chapter - 10)

Cloud computing has gained huge popularity in industry due to its ability to host applications for which the services can be delivered to consumers rapidly at minimal cost.

This chapter discusses some application case studies, detailing their architecture and how they leveraged various cloud technologies.

Applications from a range of domains, from scientific to engineering, gaming, and social networking, are considered.

10.1 Scientific Applications

Scientific applications are a sector that is increasingly using cloud computing systems and technologies.

Cloud computing systems meet the needs of different types of applications in the scientific domain: high-performance computing (HPC) applications, high-throughput computing (HTC) applications, and data-intensive applications.

The opportunity to use cloud resources is even more appealing because minimal changes need to be made to existing applications in order to leverage cloud resources.

10.1.1 Healthcare: ECG analysis in the cloud

Healthcare is a domain in which computer technology has found several and diverse applications: from supporting the business functions to assisting scientists in developing solutions to cure diseases.

An illustration of the infrastructure and model for supporting remote ECG monitoring is shown in **Figure 10.1**.

Wearable computing devices equipped with ECG sensors constantly monitor the patient's heartbeat. Such information is transmitted to the patient's mobile device, which will eventually forward it to the cloud-hosted Web service for analysis.

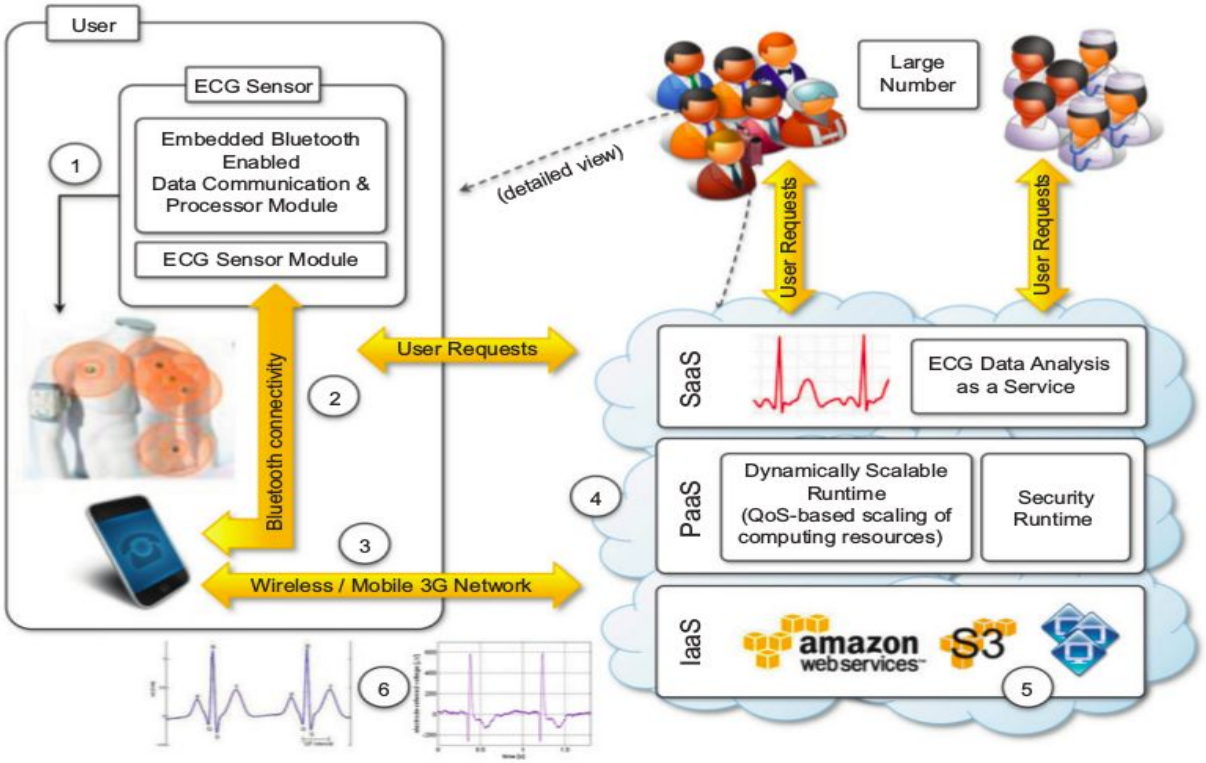


FIGURE 10.1

An online health monitoring system hosted in the cloud.

The Web service forms the front-end of a platform that is hosted in cloud and leverages three layers of cloud computing stack: SaaS, PaaS, and IaaS.

The Web service constitute SaaS application that will store ECG data in the Amazon S3 service and issue a processing request to the scalable cloud platform.

The runtime platform is composed of a dynamically sizable number of instances running the workflow engine and Aneka.

The number of workflow engine instances is controlled according to the number of requests in the queue of each instance, while Aneka controls the number of EC2 instances used to execute the single tasks defined by the workflow engine for a single ECG processing job.

Advantages

1. The elasticity of cloud infrastructure that can grow and shrink according to the requests served. As a result, doctors and hospitals do not have to invest in large computing infrastructures designed after capacity planning, thus making more effective use of budgets.

2. Ubiquity. Cloud computing technologies are easily accessible and promise to deliver systems with minimum or no downtime. Computing systems hosted in cloud are accessible from any Internet device through simple interfaces (such as SOAP and REST-based Web services). This makes systems easily integrated with other systems maintained on hospital's premises.

3. Cost savings. Cloud services are priced on a pay-per-use basis and with volume prices for large numbers of service requests.

10.1.2 Biology: protein structure prediction

Applications in biology require high computing capabilities and operate on large data-sets that cause extensive I/O operations.

Therefore biology applications have made use of supercomputing and cluster computing infrastructures.

Similar capabilities can be leveraged using cloud computing technologies in a more dynamic fashion, thus opening new opportunities for bioinformatics applications.

Protein structure prediction is a computationally intensive task that is fundamental to different types of research in the life sciences.

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 that minimizes the required energy.

This task requires the investigation of a space with a massive number of states, consequently creating a 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**).

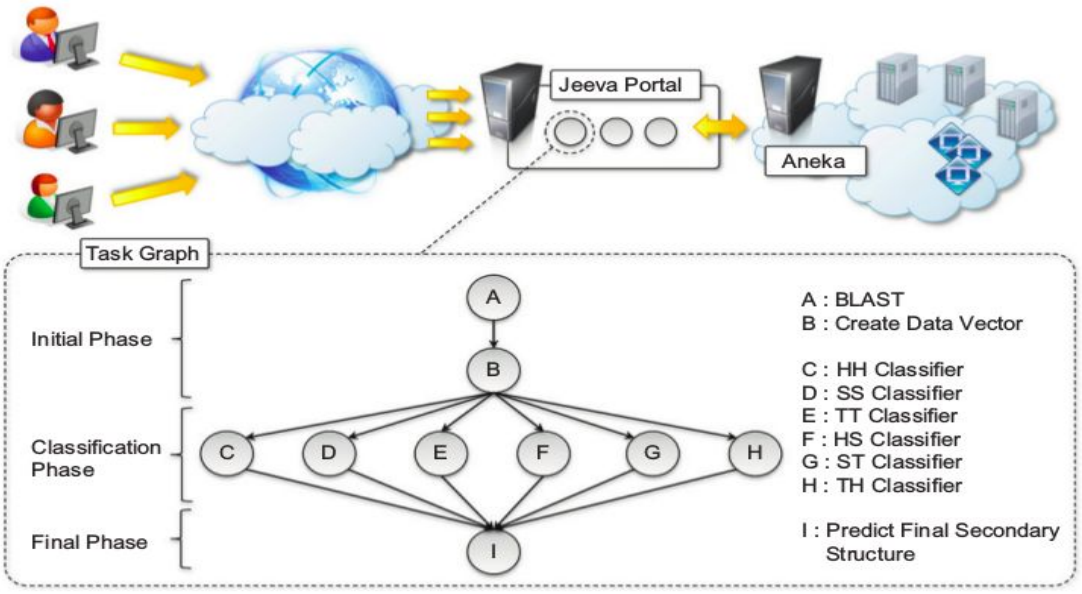


FIGURE 10.2

Architecture and overview of the Jeeva Portal.

The prediction task uses machine learning techniques (support vector machines) for determining the secondary structure of proteins.

These techniques translate problem into one of pattern recognition, where a sequence has to be classified into one of three possible classes (E, H, and C).

A popular implementation based on support vector machines divides the pattern recognition problem into three phases: initialization, classification, and a final phase.

These three phases have to be executed in sequence, we can perform parallel execution in the classification phase, where multiple classifiers are executed concurrently.

This reduces computational time of the prediction.

The prediction algorithm is then translated into a task graph that is submitted to Aneka.

Once the task is completed, the middleware makes the results available for visualization through the portal.

The advantage of using cloud technologies is the capability to leverage a scalable computing infrastructure that can be grown and shrunk on demand.

10.1.3 Biology: gene expression data analysis for cancer diagnosis

Gene expression profiling is the measurement of the expression levels of thousands of genes at once. It is used to understand the biological processes that are triggered by medical treatment at a cellular level.

important application of gene expression profiling is cancer diagnosis and treatment.

Cancer is a disease characterized by uncontrolled cell growth and proliferation. This behavior occurs because genes regulating the cell growth mutate.

Gene expression profiling is utilized to provide a more accurate classification of tumors.

The dimensionality of typical gene expression datasets ranges from several thousands to over tens of thousands of genes.

This problem is approached with learning classifiers, which generate a population of condition-action rules that guide the classification process. The eXtended Classifier System (XCS) has been utilized for classifying large datasets in bioinformatics and computer science domains.

A variation of this algorithm, CoXCS [162], has proven to be effective in these conditions. CoXCS divides the entire search space into subdomains and employs the standard XCS algorithm in each of these subdomains. Such a process is computationally intensive but can be easily parallelized because the classifications problems on the subdomains can be solved concurrently.

Cloud-CoXCS (**Figure 10.3**) is a cloud-based implementation of CoXCS that leverages Aneka to solve the classification problems in parallel and compose their outcomes. The algorithm is controlled by strategies, which define the way the outcomes are composed together and whether the process needs to be iterated.

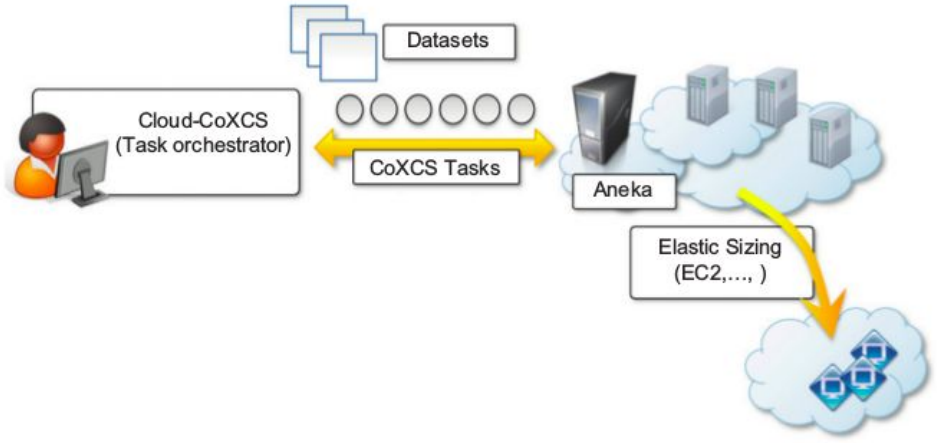


FIGURE 10.3

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

10.1.4 Geoscience: satellite image processing

Geoscience applications collect, produce, and analyze massive amounts of geospatial and nonspatial data. As the technology progresses and our planet becomes more instrumented, volume of data that needs to be processed increases significantly.

Geographic information system (GIS) applications capture, store, manipulate, analyze, manage, and present all types of geographically referenced data.

Cloud computing is an attractive option for executing these demanding tasks and extracting meaningful information to support decision makers.

Satellite remote sensing generates hundreds of gigabytes of raw images that need to be further processed to become the basis of several different GIS products. This process requires both I/O and compute-intensive tasks. Large images need to be moved from a ground station's local storage to compute facilities, where several transformations and corrections are applied.

The system shown in **Figure 10.4** integrates several technologies across the entire computing stack.

A SaaS application provides a collection of services for such tasks as geocode generation and data visualization.

At the PaaS level, Aneka controls the importing of data into the virtualized infrastructure and the execution of image-processing tasks that produce the desired outcome from raw satellite images.

The platform leverages a Xen private cloud and the Aneka technology to dynamically provision the required resources.

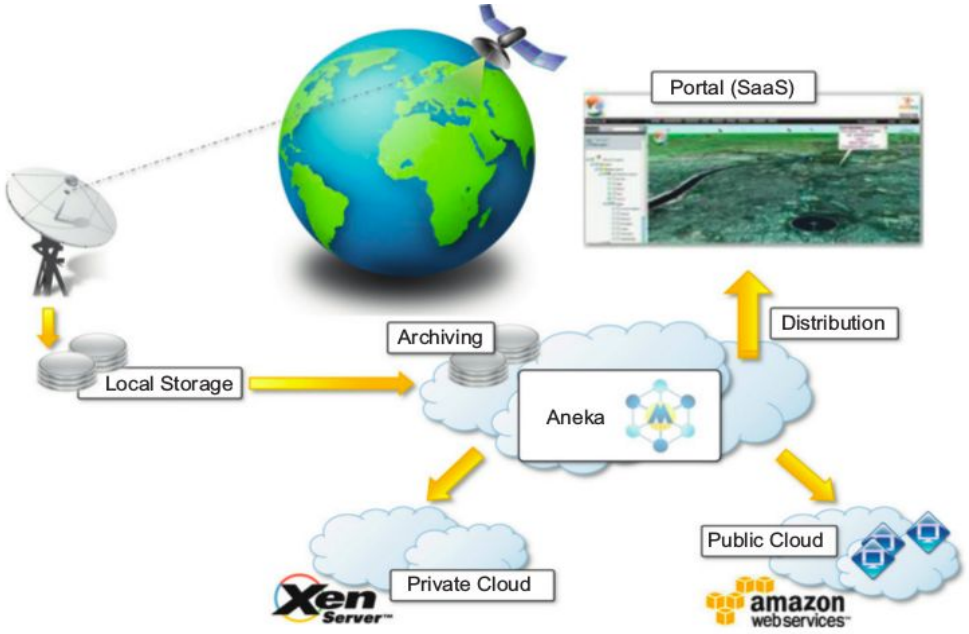


FIGURE 10.4
A cloud environment for satellite data processing.

10.2 Business and consumer applications

The business and consumer sector is the one that benefits the most from cloud computing technologies.
On one hand, the opportunity to transform capital costs into operational costs makes clouds an attractive option for all enterprises that are IT-centric.
On the other hand, the sense of ubiquity that the cloud offers for accessing data and services makes it interesting for end users.
The combination of all these elements has made cloud computing the preferred technology for a wide range of 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
Cloud CRM applications constitute a great opportunity for small enterprises and start-ups to have fully functional CRM software without large up-front costs and by paying subscriptions.
Your business and customer data from everywhere and from any device, has fostered the spread of cloud CRM applications.
ERP solutions on the cloud are less mature and have to compete with well-established in-house solutions. ERP systems integrate several aspects of an enterprise: finance and accounting, human resources, manufacturing, supply chain management, project management, and CRM.

1 Salesforce.com

Salesforce.com is most popular and developed CRM solution available today.
As of today more than 100,000 customers have chosen Safesforce.com to implement their CRM solutions.
The application provides customizable CRM solutions that can be integrated with additional features developed by third parties.
Salesforce.com is based on the Force.com cloud development platform.
This represents scalable and high-performance middleware executing all operations of all Salesforce.com applications.
The architecture of the Force.com platform is shown in Figure 10.5.

At the core of the platform resides its metadata architecture, which provides the system with flexibility and scalability.

Application core logic and business rules are saved as metadata into the Force.com store.

Both application structure and application data are stored in the store. A runtime engine executes application logic by retrieving its metadata and then performing the operations on the data.

A full-text search engine supports the runtime engine. This allows application users to have an effective user experience The search engine maintains its indexing data in a separate store.

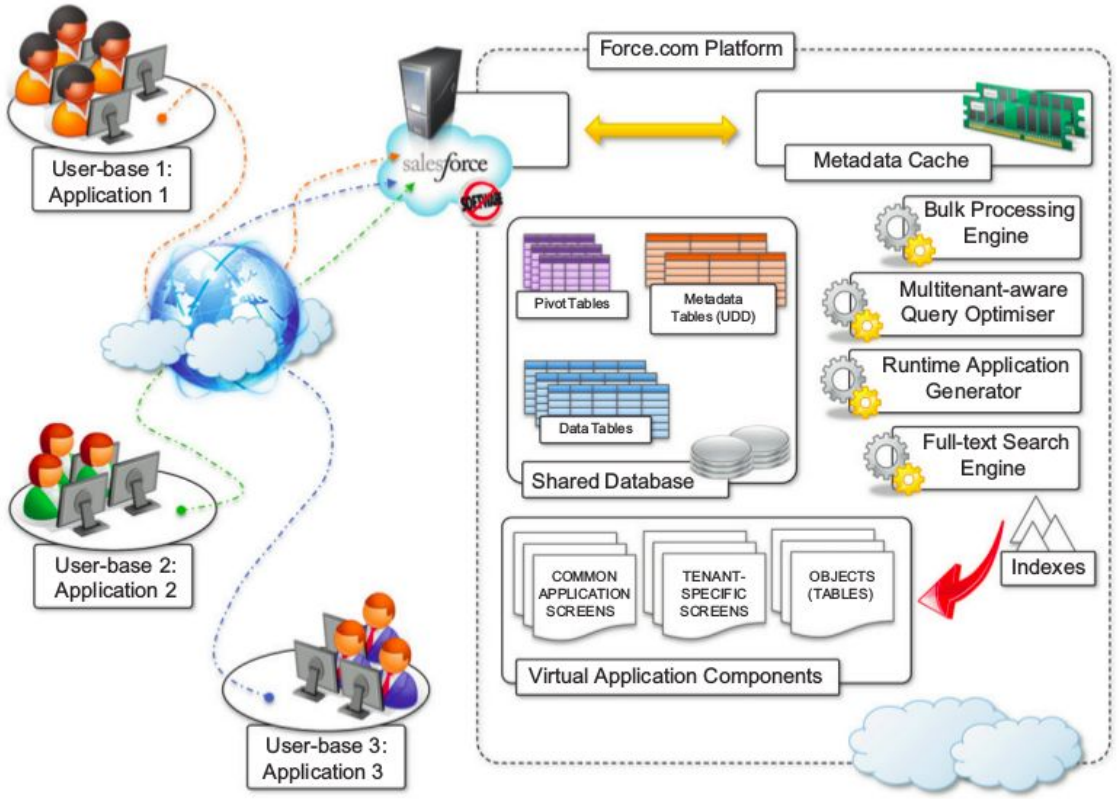


FIGURE 10.5

Salesforce.com and Force.com architecture.

2 Microsoft dynamics CRM

The system is 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 application provides users with facilities for marketing, sales, and advanced customer relationship management.

Dynamics CRM Online features can be accessed either through a Web browser interface or by means of SOAP and RESTful Web services.

This allows Dynamics CRM to be easily integrated with both other Microsoft products and line-of-business applications.

Dynamics CRM can be extended by developing plug-ins that allow implementing specific behaviors triggered on the occurrence of given events.

3 NetSuite

NetSuite provides a collection of applications that help customers manage every aspect of the business enterprise.

Its offering is divided into three major products: NetSuite Global ERP, NetSuite Global CRM1 , and NetSuite Global Ecommerce.

Moreover, an all-in-one solution: NetSuite One World, integrates all three products together.

The services NetSuite delivers are powered by two large datacenters on the East and West coasts of the United States, connected by redundant links.
This allows NetSuite to guarantee 99.5% uptime to its customers.
The NetSuite Business Operating System (NS-BOS) is a complete stack of technologies for building SaaS business applications that leverage the capabilities of NetSuite products.
On top of the SaaS infrastructure, the NetSuite Business Suite components offer accounting, ERP, CRM, and ecommerce capabilities.

10.2.2 Productivity

Productivity applications replicate in cloud. The most common tasks that we are used to performing on our desktop: from document storage to office automation and complete desktop environments hosted in the cloud.

1 Dropbox and iCloud

Online storage solutions have turned into SaaS applications and become more usable as well as more advanced and accessible.
The most popular solution for online document storage is **Dropbox**, that allows users to synchronize any file across any platform and any device in a seamless manner (**Figure 10.6**). Dropbox provides users with a free storage that is accessible through the abstraction of a folder. Users can either access their Dropbox folder through a browser or by downloading and installing a Dropbox client, which provides access to the online storage by means of a special folder. All the modifications into this folder are silently synched so that changes are notified to all the local instances of the Dropbox folder across all the devices.
Another interesting application in this area is **iCloud**, a cloud-based document-sharing application provided by Apple to synchronize iOS-based devices in a completely transparent manner.
Documents, photos, and videos are automatically synched as changes are made, without any explicit operation. This allows the system to efficiently automate common operations without any human intervention.
This capability is limited to iOS devices, and currently there are no plans to provide iCloud with a Web-based interface that would make user content accessible from even unsupported platforms.

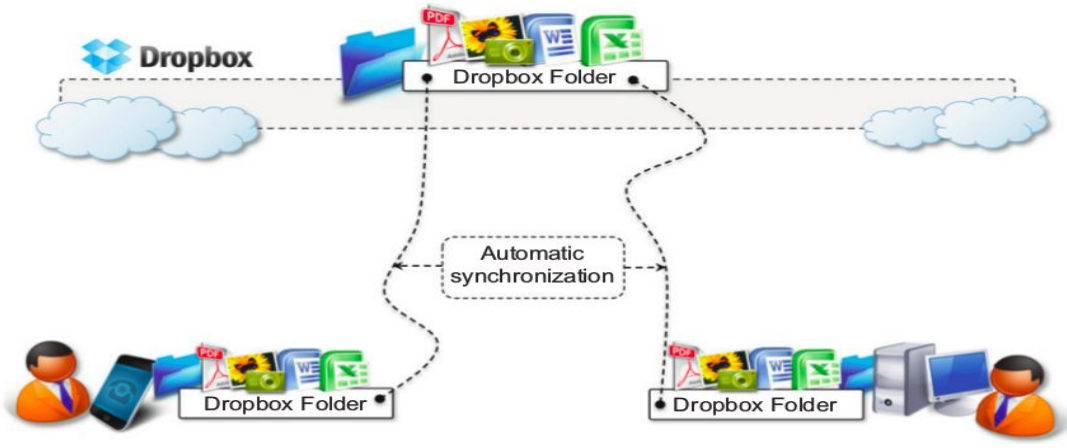


FIGURE 10.6
Dropbox usage scenario.

2 Google docs

Google Docs is a SaaS application that delivers the basic office automation capabilities with support for collaborative editing over the Web.
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.
By being stored in the Google infrastructure, these documents are always available from anywhere and from any device that is connected to the Internet.

Google Docs is a good example of what cloud computing can deliver to end users: ubiquitous access to resources, elasticity, absence of installation and maintenance costs, and delivery of core functionalities as a service.

3 Cloud desktops: EyeOS and XIOS/3

EyeOS is one of the most popular Web desktop solutions based on cloud technologies. It replicates the functionalities of a classic desktop environment and comes with pre-installed applications for the most common file and document management tasks (**Figure 10.7**). Single users can access the EyeOS desktop environment from anywhere and through any Internet-connected device, whereas organizations can create a private EyeOS Cloud on their premises to virtualize the desktop environment of their employees and centralize their management.

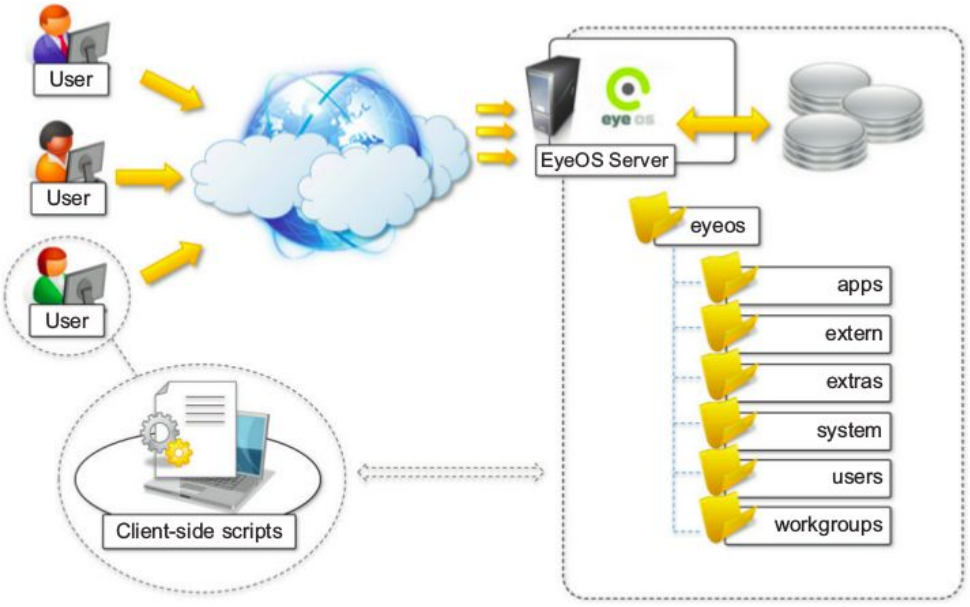


FIGURE 10.7
EyeOS architecture.

The EyeOS architecture is quite simple: On the server side, the EyeOS application maintains the information about user profiles and their data, and the client side constitutes the access point for users and administrators to interact with the system. EyeOS stores the data about users and applications on the server file system. Once the user has logged in by providing credentials, the desktop environment is rendered in the client's browser by downloading all the JavaScript libraries required to build the user interface and implement the core functionalities of EyeOS.

EyeOS also provides APIs for developing new applications and integrating new capabilities into the system. EyeOS applications are server-side components that are defined by at least two files (stored in the eyeos/apps/appname directory): appname.php and appname.js. The first file defines and implements all the operations that the application exposes; the JavaScript file contains the code that needs to be loaded in the browser in order to provide user interaction with the application.

Xcerion XML Internet OS/3 (XIOS/3) is another example of a Web desktop environment. The service is delivered as part of the CloudMe application, which is a solution for cloud document storage. The key differentiator of XIOS/3 is its 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.

XIOS/3 is released as open-source software and implements a marketplace where third parties can easily deploy applications that can be installed on top of the virtual desktop environment. It is possible to develop any type of

application and feed it with data accessible through XML Web services: developers have to define the user interface, bind UI components to service calls and operations, and provide the logic on how to process the data.

10.2.3 Social networking

Social networking applications have grown considerably in the last few years to become the most active sites on the Web.

To sustain their traffic and serve millions of users seamlessly, services such as Twitter and Facebook have leveraged cloud computing technologies.

1 Facebook

Facebook is probably the most evident and interesting environment in social networking.

With more than 800 million users, it has become one of the largest Websites in the world.

To sustain this incredible growth, it has been fundamental that Facebook be capable of continuously adding capacity and developing new scalable technologies and software systems while maintaining high performance to ensure a smooth user experience.

Currently, the social network is backed by two data centers that have been built and optimized to reduce costs and impact on the environment.

On top of this highly efficient infrastructure, built and designed out of inexpensive hardware, a completely customized stack of opportunely modified and refined open-source technologies constitutes the back-end of the largest social network.

The reference stack serving Facebook is based on LAMP (Linux, Apache, MySQL, and PHP). This collection of technologies is accompanied by a collection of other services developed in-house.

These services are developed in a variety of languages and implement specific functionalities such as search, news feeds, notifications, and others.

While serving page requests, the social graph of the user is composed.

The social graph identifies a 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 instances, which mostly contain key-value pairs.

10.2.4 Media applications

Media applications has taken a considerable advantage from leveraging cloud computing technologies.

The computationally intensive tasks can be easily offloaded to cloud computing infrastructures.

1 Animoto

Animoto is the most popular example of media applications on the cloud. The Website provides users with a very straightforward interface for quickly creating videos out of images, music, and video fragments submitted by users. Users select a specific theme for a video, upload the photos and videos and order them in the sequence they want to appear, select the song for the music, and render the video. The process is executed in the background and the user is notified via email once the video is rendered.

A proprietary artificial intelligence (AI) engine, which selects the animation and transition effects according to pictures and music, drives the rendering operation. Users only have to define the storyboard by organizing pictures and videos into the desired sequence.

The infrastructure of Animoto is complex and is composed of different systems that all need to scale (**Figure 10.8**). The core function is implemented on top of the Amazon Web Services infrastructure. It uses Amazon EC2 for the Web front-end and worker nodes; Amazon S3 for the storage of pictures, music, and videos; and Amazon SQS for connecting all the components.

The system's auto-scaling capabilities are managed by Rightscale, which monitors the load and controls the creation of new worker instances.

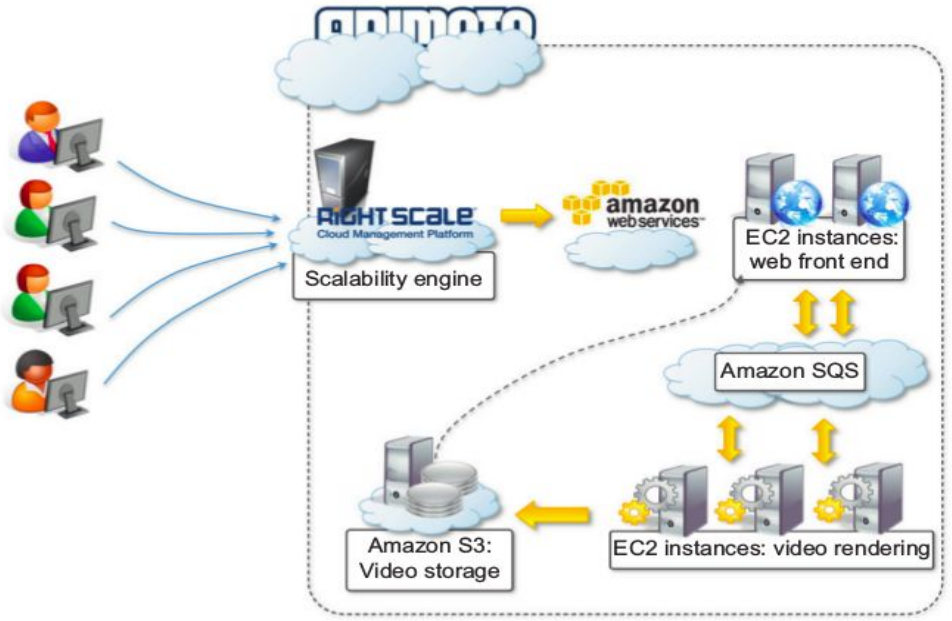


FIGURE 10.8
Animoto reference architecture.

2 Maya rendering with Aneka

A private cloud solution for rendering train designs has been implemented by the engineering department of GoFront group, a division of China Southern Railway (**Figure 10.9**). The department is responsible for designing models of high-speed electric locomotives, metro cars, urban transportation vehicles, and motor trains. The design process for prototypes requires high-quality, three-dimensional (3D) images. The analysis of these images can help engineers identify problems and correct their design.

Three-dimensional rendering tasks take considerable amounts of time, especially in the case of huge numbers of frames, but it is critical for the department to reduce the time spent in these iterations. This goal has been achieved by leveraging cloud computing technologies, which turned the network of desktops in the department into a desktop cloud managed by Aneka.

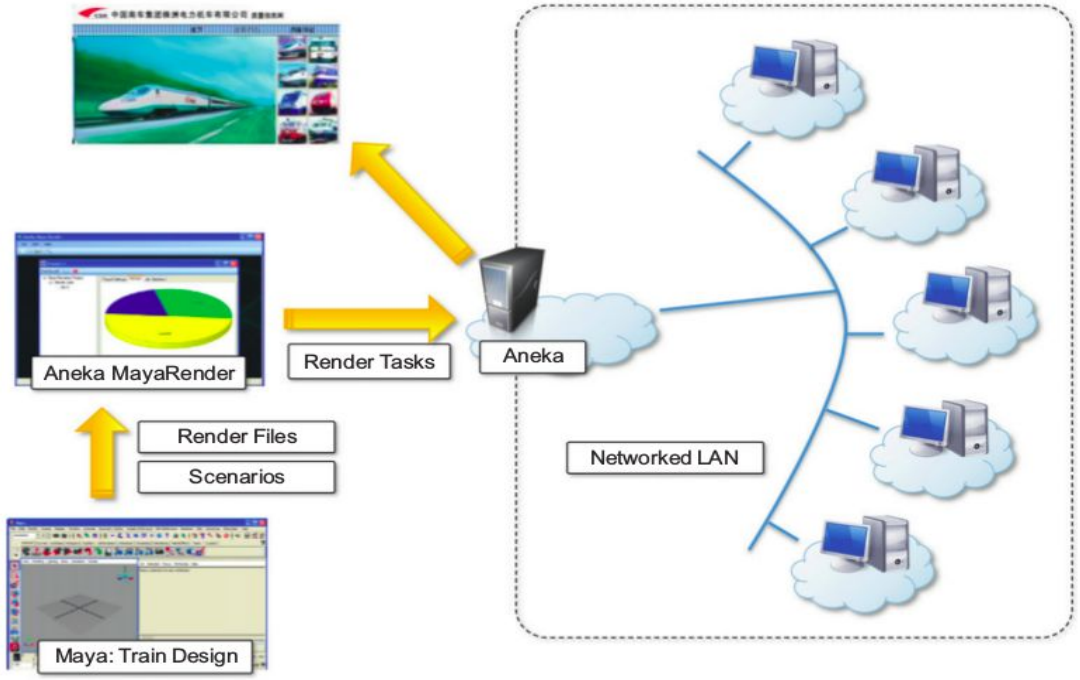


FIGURE 10.9
3D rendering on private clouds.

3 Video encoding on the cloud: Encoding.com

Video encoding and transcoding are operations that can greatly benefit from using cloud technologies: They are computationally intensive and potentially require considerable amounts of storage.

Encoding.com is a software solution that offers video-transcoding services on demand and leverages cloud technology to provide both the horsepower required for video conversion and the storage for staging videos. The service integrates with both Amazon Web Services technologies (EC2, S3, and CloudFront) and Rackspace (Cloud Servers, Cloud Files, and Limelight CDN access).

To use the service, users have to specify the location of the video to transcode, the destination format, and the target location of the video. Encoding.com also offers other video-editing operations such as the insertion of thumbnails, watermarks, or logos. Moreover, it extends its capabilities to audio and image conversion.

10.2.5 Multiplayer online gaming

Online multiplayer gaming attracts millions of gamers around the world who share a common experience by playing together in a virtual environment that extends beyond the boundaries of a normal LAN. Online games support hundreds of players in the same session, made possible by the specific architecture used to forward interactions, which is based on game log processing.

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

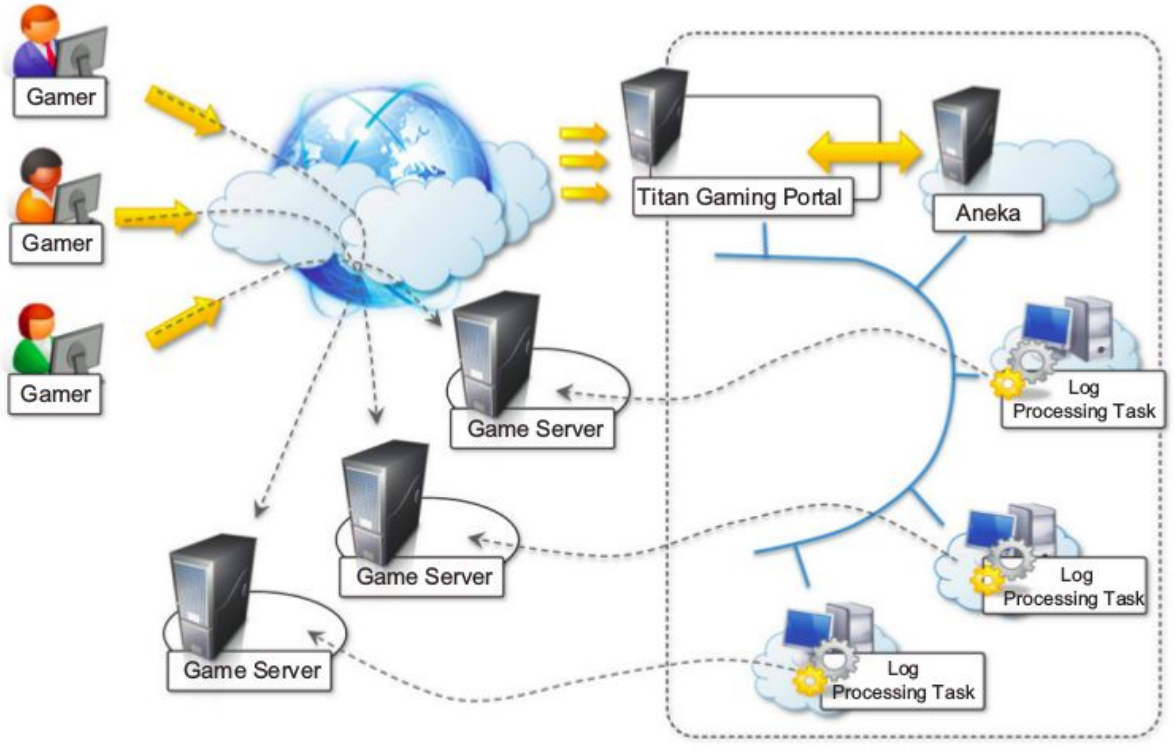


FIGURE 10.10

Scalable processing of logs for network games.

Game log processing is also utilized to build statistics on players and rank them. These features constitute the additional value of online gaming portals that attract more and more gamers. The processing of game logs is a potentially compute-intensive operation that strongly depends on the number of players online and the number of games monitored.

The use of cloud computing technologies can provide the required elasticity for seamlessly processing these workloads and scale as required when the number of users increases. A prototype implementation of cloud-based game log processing has been implemented by Titan Inc. (**Figure 10.10**)

MODULE – I
INTRODUCTION, VIRTUALIZATION
Chapters – 1 & 3.

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 a technological advancement that focuses on the way we design computing systems, develop applications, and leverage existing services for building software. 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.

1.1 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. This principle finds several practical applications and then gives different images of cloud computing to different people. Chief information and technology officers of large enterprises see opportunities for scaling their infrastructure on demand and sizing it according to their business needs. End users leveraging cloud computing services can access their documents and data anytime,anywhere, and from any device connected to the Internet. Many other points of view exist.

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."

1.1.1 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.
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. Different stake-holders leverage clouds for a variety of services. The need for ubiquitous storage and compute power on demand is the most common reason to consider cloud computing. A scalable runtime for applications is an attractive option for application and system developers that do not have infrastructure or cannot afford any further expansion of existing infrastructure.

This approach provides opportunities for optimizing datacenter facilities and fully utilizing their capabilities to serve multiple users. This consolidation model will reduce the waste of energy and carbon emissions, thus contributing to a greener IT on one end and increasing revenue on the other end.

FIGURE 1.1 : Cloud computing vision.

1.1.2 Defining a cloud

The term cloud has historically been used in the telecommunications industry as an abstraction of the network in system diagrams. It then became the symbol of the most popular computer network, the Internet. This meaning also applies to cloud computing, which refers to an Internet-centric way of computing. 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 et al.:
"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."

The 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.

1.1.3 A closer look

Cloud computing is helping enterprises, governments, public and private institutions, and research organizations shape more effective and demand-driven computing systems. Access to, as well as integration of, cloud computing resources and systems is now as easy as performing a credit card transaction over the Internet. Practical examples of such systems exist across all market segments:

- Large enterprises can offload some of their activities to cloud-based systems.
- Small enterprises and start-ups can afford to translate their ideas into business results more quickly, without excessive up-front costs.
- System developers can concentrate on the business logic rather than dealing with the complexity of infrastructure management and scalability.
- End users can have their documents accessible from everywhere and any device.

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.

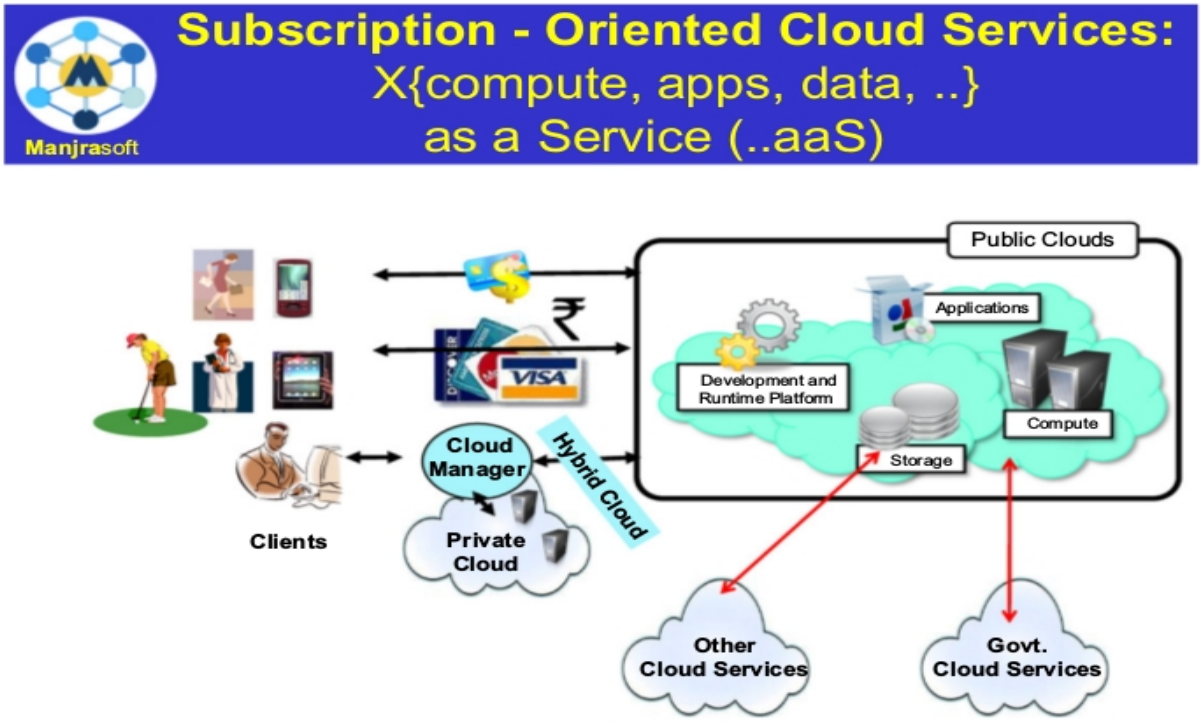


FIGURE 1.3

A bird's-eye view of cloud computing.

1.1.4 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. 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, which provides an organic view of cloud computing.
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.
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.
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. 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 spe- cific software stack. IaaS services are therefore used to develop scalable Websites or for back- ground processing.

1.1.5 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

The most evident benefit from the use of cloud computing systems and technologies is the increased economical return due to the reduced maintenance costs and operational costs related to IT software and infrastructure.
This is mainly because IT assets, namely software and infrastructure, are turned into utility costs, which are paid for as long as they are used, not paid for up front.
IT infrastructure and software generated capital costs, since they were paid up front so that business start-ups could afford a computing infrastructure, enabling the business activities of the organization. The revenue of the business is then utilized to compensate over time for these costs.
End users can benefit from cloud computing by having their data and the capability of operating on it always available, from anywhere, at any time, and through multiple devices. Information and services stored in the cloud are exposed to users by Web-based interfaces that make them accessible from portable devices as well as desktops at home.

1.1.6 Challenges ahead

New, interesting problems and challenges are regularly being posed to the cloud community, including IT practitioners, managers, governments, and regulators. 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.

1.2 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.

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.

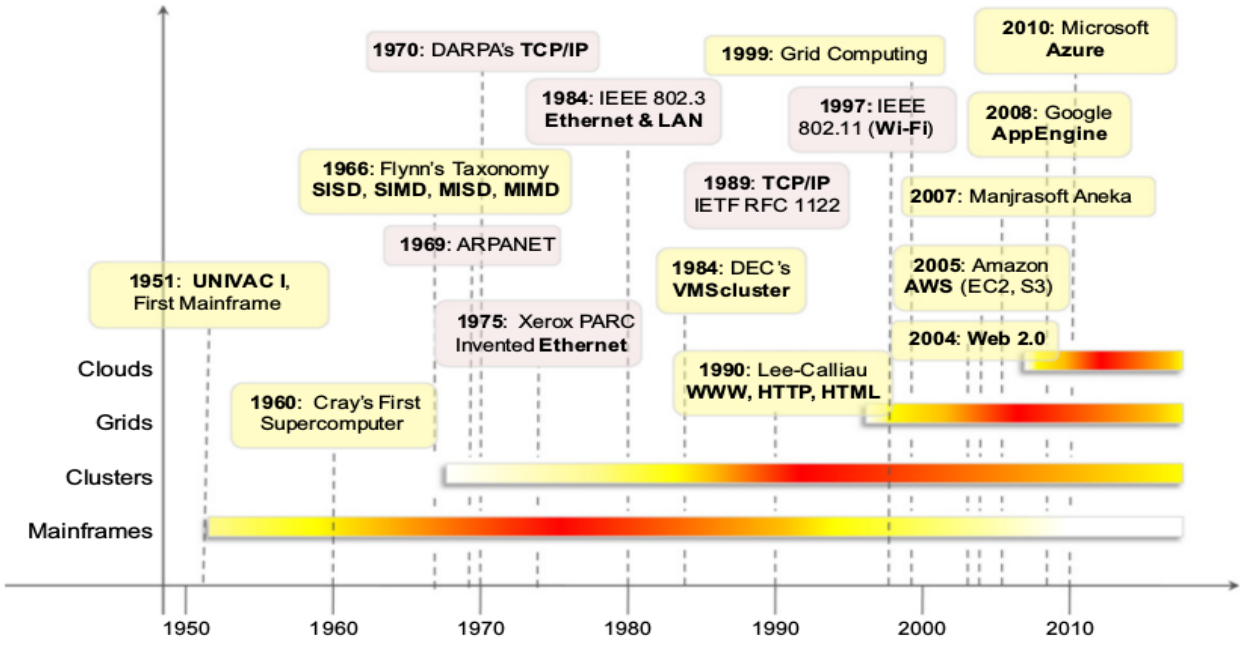


FIGURE 1.6

The evolution of distributed computing technologies, 1950s–2010s.

1.2.1 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.:

"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.

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. Starting in the 1980s.

Cluster technology contributed considerably to the evolution of tools and frameworks for distributed computing, including Condor, Parallel Virtual Machine (PVM), and Message Passing Interface (MPI).

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, hugeBstorage facilities, and a variety of services.

A computing grid was a dynamic aggregation of heterogeneous computing nodes, and its scale was nationwide or even worldwide. 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.

1.2.2 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. Virtualization has been around for more than 40 years, but its application has always been limited by technologies that did not allow an efficient use of virtualization solutions.

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.

Virtualization technologies are also used to replicate runtime environments for programs. Applications in the case of process virtual machines (which include the foundation of technologies such as Java or .NET), instead of being executed by the operating system, are run by a specific program called a virtual machine. This technique allows isolating the execution of applications and providing a finer control on the resource they access.

1.2.3 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. These technologies allow us to build applications leveraging the contribution of users, who now become providers of content.

Web 2.0 applications are extremely dynamic: they improve continuously, and new updates and features are integrated at a constant rate by following the usage trend of the community. There is no need to deploy new software releases on the installed base at the client side.

Web 2.0 applications aim to leverage the "long tail" of Internet users by making themselves available to everyone in terms of either media accessibility or affordability.

Examples of Web 2.0 applications are Google Documents, Google Maps, Flickr, Facebook, Twitter, YouTube, de.li.cious, Blogger, and Wikipedia. In particular, social networking Websites take the biggest advantage of Web 2.0. The level of interaction in Websites such as Facebook or Flickr would not have been possible without the support of AJAX, Really Simple Syndication (RSS), and other tools that make the user experience incredibly interactive.

This idea of the Web as a transport that enables and enhances interaction was introduced in 1999 by Darcy DiNucci 5 and started to become fully realized in 2004. Today it is a mature platform for supporting the needs of cloud computing, which strongly leverages Web 2.0. Applications and frameworks for delivering rich Internet applications (RIAs) are fundamental for making cloud services accessible to the wider public.

1.2.4 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.

A service is supposed to be loosely coupled, reusable, programming language independent, and location transparent. Loose coupling allows services to serve different scenarios more easily and makes them reusable. Independence from a specific platform increases services accessibility. Thus, a wider range of clients, which can look up services in global registries and consume them in a location-transparent manner, can be served.

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.
- 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.

1.2.5 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.
The American scientist John McCarthy, who, in a speech for the Massachusetts Institute of Technology (MIT) centennial in 1961, observed:
"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility, just as the telephone system is a public utility . . . The computer utility could become the basis of a new and important industry."

The first traces of this service-provisioning model can be found in the mainframe era. IBM and other mainframe providers offered mainframe power to organizations such as banks and government agencies throughout their datacenters.
From an application and system development perspective, service-oriented computing and service-oriented architectures (SOAs) introduced the idea of leveraging external services for performing a specific task within a software system.

1.3 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.

1.3.1 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. Their performance is mostly influenced by the workload generated by varying user demands. With the diffusion of Web 2.0 technologies, the Web has become a platform for developing rich and complex applications, including enterprise applications that now leverage the Internet as the preferred channel for service delivery and user interaction.
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.
Cloud computing provides a solution for on-demand and dynamic scaling across the entire stack of computing.
This is achieved by
(a) providing methods for renting compute power, storage, and networking;
(b) offering runtime environments designed for scalability and dynamic sizing; and
(c) providing application services that mimic the behavior of desktop applications but that are completely hosted and managed on the provider side.

1.3.2 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.
Distributed computing is a foundational model for cloud computing because cloud systems are distributed systems. Besides administrative tasks mostly connected to the accessibility of resources in the cloud, the extreme dynamism of cloud systems—where new nodes and services are provisioned on demand—constitutes the major challenge for engineers and developers.

Web 2.0 technologies constitute the interface through which cloud computing services are delivered, managed, and provisioned.
Cloud computing is often summarized with the acronym XaaS—Everything-as-a-Service—that clearly underlines the central role of service orientation.
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.

1.3.3 Computing platforms and technologies

Development of a cloud computing application happens by leveraging platforms and frameworks that provide different types of services, from the bare-metal infrastructure to customizable applications serving specific purposes.

1 Amazon web services (AWS)
2 Google AppEngine
3 Microsoft Azure
4 Hadoop
5 Force.com and Salesforce.com
6 Manjrasoft Aneka

1 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. EC2 also provides the capability to save a specific running instance as an image, thus allowing users to create their own templates for deploying systems. These templates are stored into S3 that delivers persistent storage on demand. S3 is organized into buckets; these are containers of objects that are stored in binary form and can be enriched with attributes. Users can store objects of any size, from simple files to entire disk images, and have them accessible from everywhere.

2 Google AppEngine

Google AppEngine is a scalable runtime environment mostly devoted to executing Web applications. These take advantage of the large computing infrastructure of Google to dynamically scale as the demand varies over time. 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). Once development is complete, developers can easily migrate their application to AppEngine, set quotas to contain the costs generated, and make the application available to the world. The languages currently supported are Python, Java, and Go.

3 Microsoft Azure

Microsoft Azure is a cloud operating system and a platform for developing applications in the cloud. Applications in Azure are organized around the concept of roles, which identify a distribution unit for applications and embody the application’s logic. Currently, there are three types ofrole: 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.

4 Hadoop

Apache Hadoop is an open-source framework that is suited for processing large data sets on commodity hardware. Hadoop is an implementation of MapReduce, an application programming model developed by Google, which provides two fundamental operations for data processing: map and reduce. The former transforms and synthesizes the input data provided by the user; the latter aggragates the output obtained by the map operations. Hadoop provides the runtime environment, and developers need only provide the input data and specify the map and reduce functions that need to be executed.

5 Force.com and Salesforce.com

Force.com is a cloud computing platform for developing social enterprise applications. The platform is the basis for SalesForce.com, a Software-as-a-Service solution for customer relationship management. 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 platform provides complete support for developing applications, from the design of the data layout to the definition of business rules and workflows and the definition of the user interface.

6 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).

Developers can choose different abstractions to design their application: tasks, distributed threads, and map-reduce. These applications are then executed on the distributed service-oriented runtime environment, which can dynamically integrate additional resource on demand.

CHAPTER - 3

3. Virtualization

Virtualization technology is one of the fundamental components of cloud computing, especially in regard to infrastructure-based services. Virtualization allows the creation of a secure, customizable, and isolated execution environment for running applications, even if they are untrusted, without affecting other users' applications. The basis of this technology is the ability of a computer program—or a combination of software and hardware—to emulate an executing environment separate from the one that hosts such programs.

3.1 Introduction

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.

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

- Increased performance and computing capacity.

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. Using these resources for other purposes after hours could improve the efficiency of the IT infrastructure.
- Lack of space.
  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
- Greening initiatives.
  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.
  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.

3.2 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 3.1).

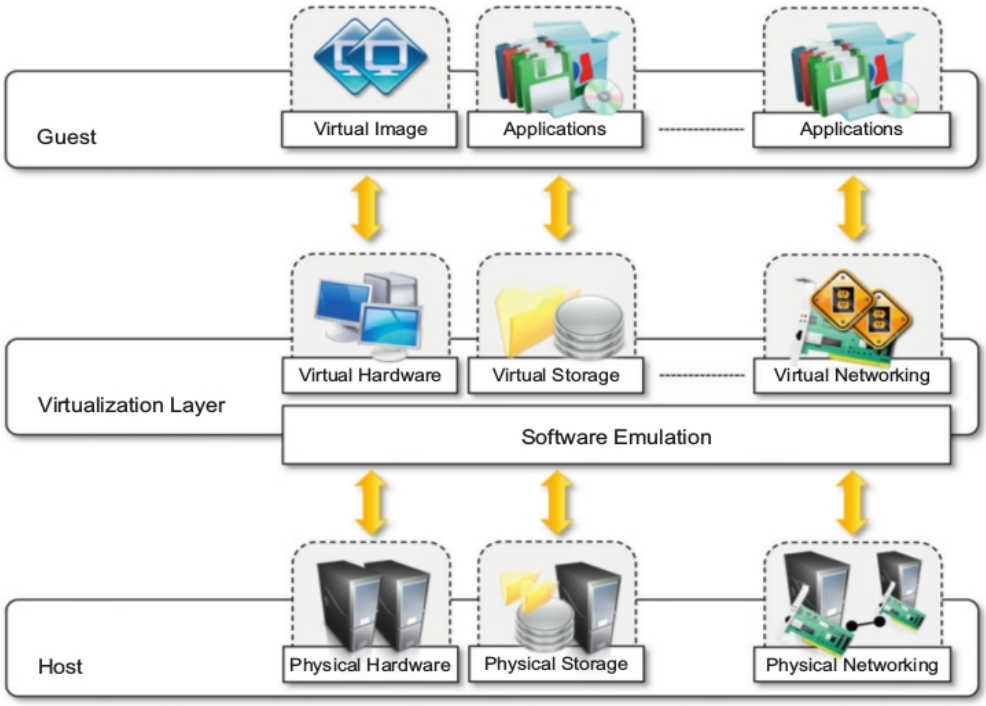


FIGURE 3.1
The virtualization reference model.

The characteristics of virtualized solutions are:
1 Increased security
2 Managed execution
3 Portability

1. Increased security
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. This level of indirection allows the virtual machine manager to control and filter the activity of the guest, thus preventing some harmful operations from being performed. For example, applets downloaded from the Internet run in a sandboxed 3 version of the Java Virtual Machine (JVM), which provides them with limited access to the hosting operating system resources. Both the JVM and the .NET runtime provide extensive security policies for customizing the execution environment of applications.

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 (see Figure 3.2).

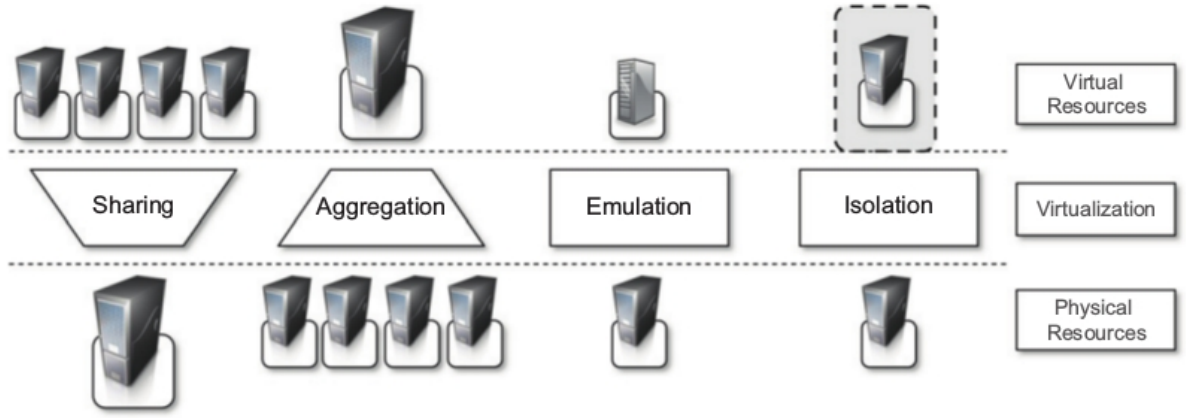


FIGURE 3.2

Functions enabled by managed execution.

Sharing. Virtualization allows the creation of a separate computing environments within the same host. In this way it is possible to fully exploit the capabilities of a powerful guest, which would otherwise be underutilized.

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.

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.

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.

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.

In the case of programming-level virtualization, as implemented by the JVM or the .NET runtime, the binary code representing application components (jars or assemblies) can be run without any recompilation on any implementation of the corresponding virtual machine. This makes the application development cycle more flexible and application deployment very straightforward: One version of the application, in most cases, is able to run on different platforms with no changes.

3.3 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 3.3).

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. Therefore, it deserves major investigation and a further categorization.

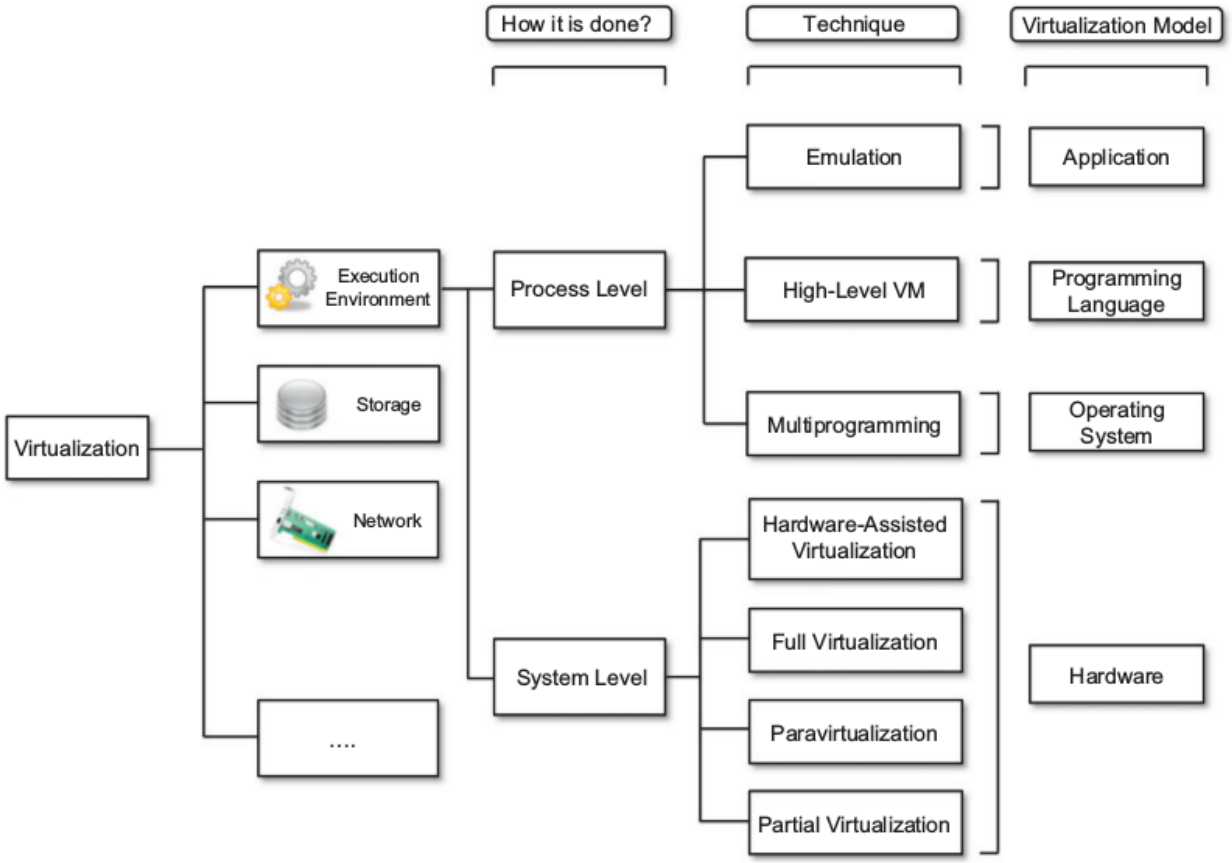


FIGURE 3.3

A taxonomy of virtualization techniques.

3.3.1 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.

1 Machine reference model
2 Hardware-level virtualization
- **a. Hypervisors**
- **b. Hardware virtualization techniques**
- **c. Operating system-level virtualization**

3. Programming language-level virtualization
4. Application-level virtualization

1 Machine reference model

Modern computing systems can be expressed in terms of the reference model described in Figure 3.4. 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 interrupt 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). 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.
The highest level of abstraction is represented by the application programming interface (API), which interfaces applications to libraries and/or the underlying operating system.

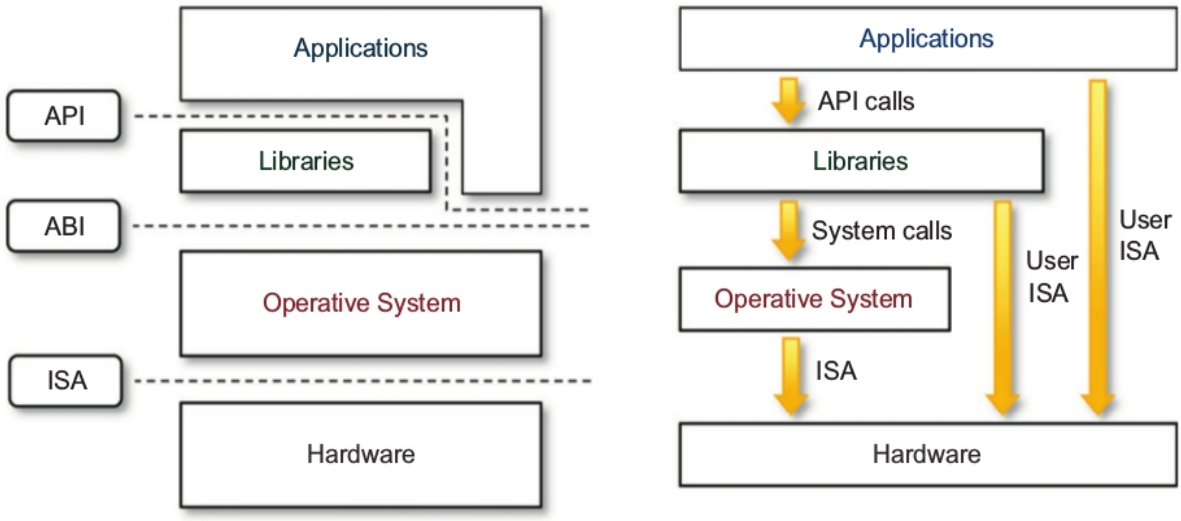


FIGURE 3.4

A machine reference model.

For this purpose, 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, a possible implementation features a hierarchy of privileges (see Figure 3.5) 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.

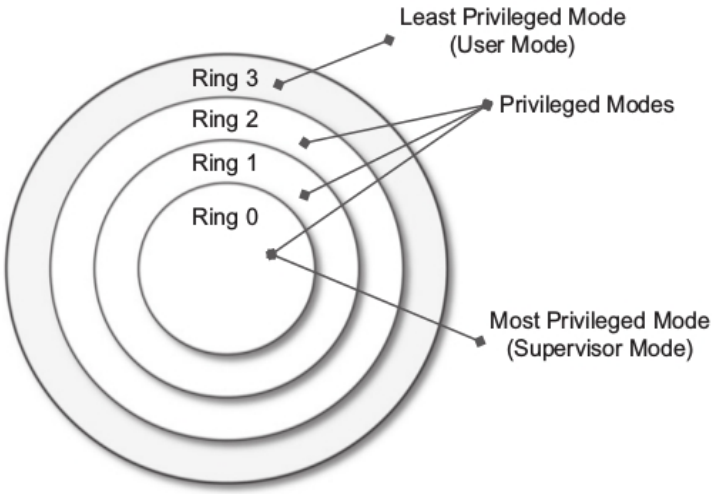


FIGURE 3.5

Security rings and privilege modes.

The distinction between user and supervisor mode allows us to understand the role of the hypervisor and why it is called that. Conceptually, the hypervisor runs above the supervisor mode, and from here the prefix hyper- is used. In reality, hypervisors are run in supervisor mode.

2 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 3.6). 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. This is to differentiate it from process virtual machines, which expose ABI to virtual machines.

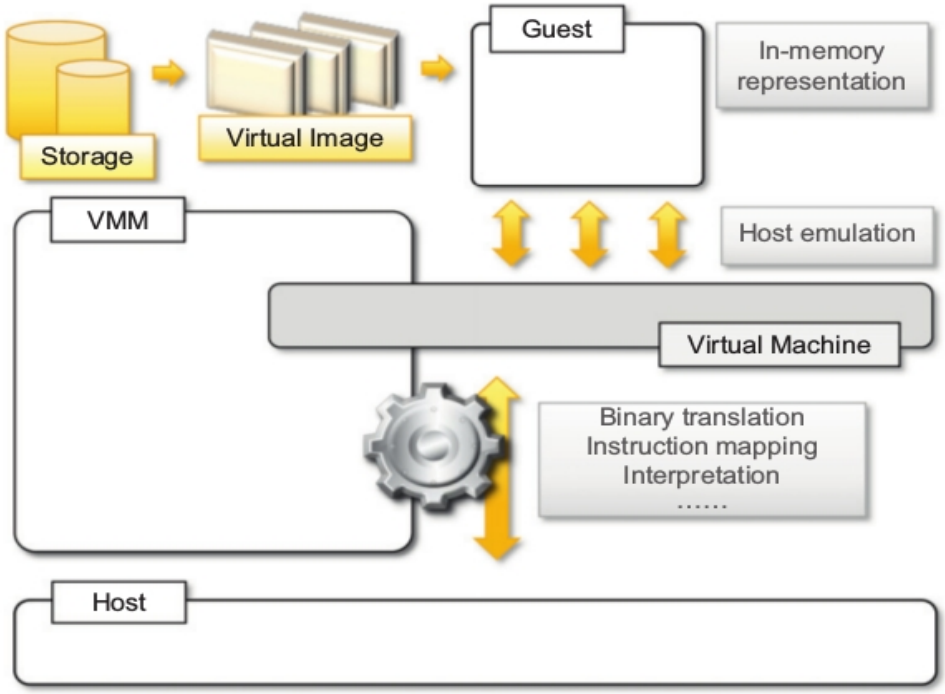


FIGURE 3.6

A hardware virtualization reference model.

a. 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 3.7).

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.

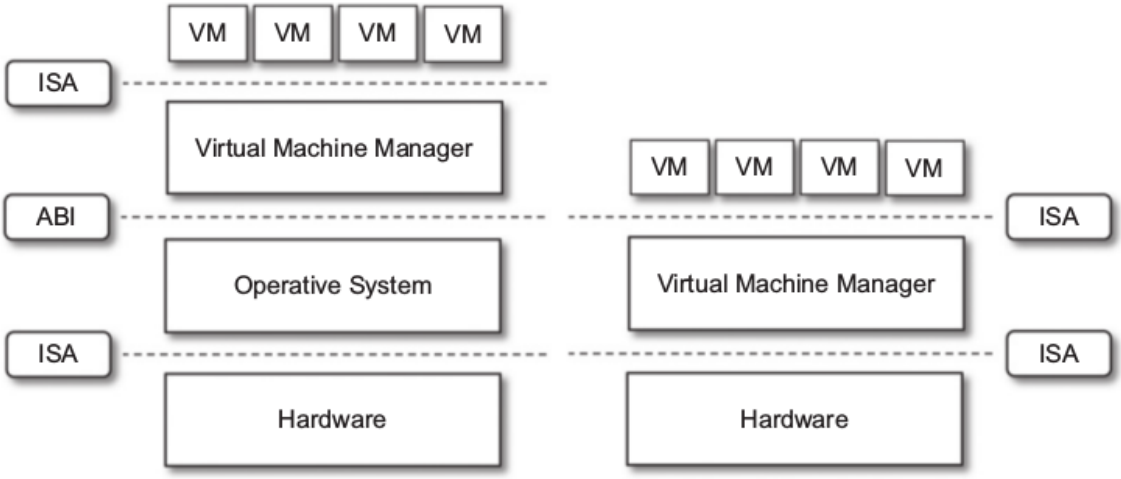


FIGURE 3.7

Hosted (left) and native (right) virtual machines. This figure provides a graphical representation of the two types of hypervisors.

Conceptually, a virtual machine manager is internally organized as described in Figure 3.8. 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.

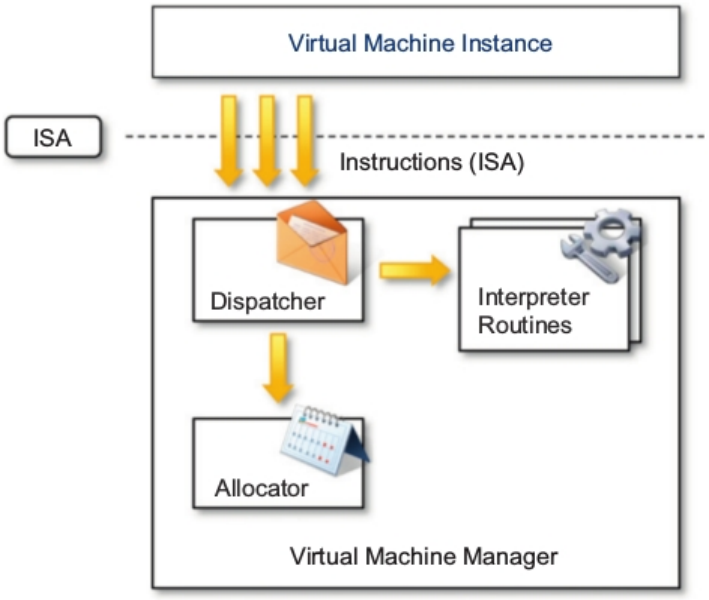


FIGURE 3.8

A hypervisor reference architecture.

The design and architecture of a virtual machine manager, together with the underlying hardware design of the host machine, determine the full realization of hardware virtualization, where a guest operating system can be transparently executed on top of a VMM as though it were run on the underlying hardware. The criteria that need to be met by a virtual machine manager to efficiently support virtualization were established by Goldberg and Popek in 1974 [23]. Three properties have to be satisfied:
1. 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.
2. Resource control. The virtual machine manager should be in complete control of virtualized resources.
3. 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 3.1

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.

THEOREM 3.2

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 3.3

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. Here all sensitive instructions are caught by HVM that are simulated.

b. 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). Intel and AMD introduced processor extensions, and a wide range of virtualization 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.

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.

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.

c. 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.

This virtualization technique can be considered an evolution of the chroot mechanism in Unix systems. The chroot operation changes the file system root directory for a process and its children to a specific directory. As a result, the process and its children cannot have access to other portions of the file system than those accessible under the new root directory.

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).

3 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 sys- tems. 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.

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.

The ability to support multiple programming languages has been one of the key elements of the Common Language Infrastructure (CLI), which is the specification behind .NET Framework. Currently, the Java platform and .NET Framework represent the most popular technologies for enterprise application development. Both Java and the CLI are stack-based virtual machines.
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.
The process virtual machines allow for more control over the execution of programs since they do not provide direct access to the memory. Security is another advantage of managed programming languages; by filtering the I/O operations, the process virtual machine can easily support sandboxing of applications.

4 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.
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:
a. 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.
b. 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.

3.3.2 Other types of virtualization
Other than execution virtualization, other types of virtualization provide an abstract environment to interact with. These mainly cover storage, networking, and client/server interaction.

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.
Storage virtualization allows us to harness a wide range of storage facilities and represent them under a single logical file system. 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).

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.

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.

3.4 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. 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.

Besides being an enabler for computation on demand, virtualization also gives the opportunity to design more efficient computing systems by means of consolidation, which is performed transparently to cloud computing service users.

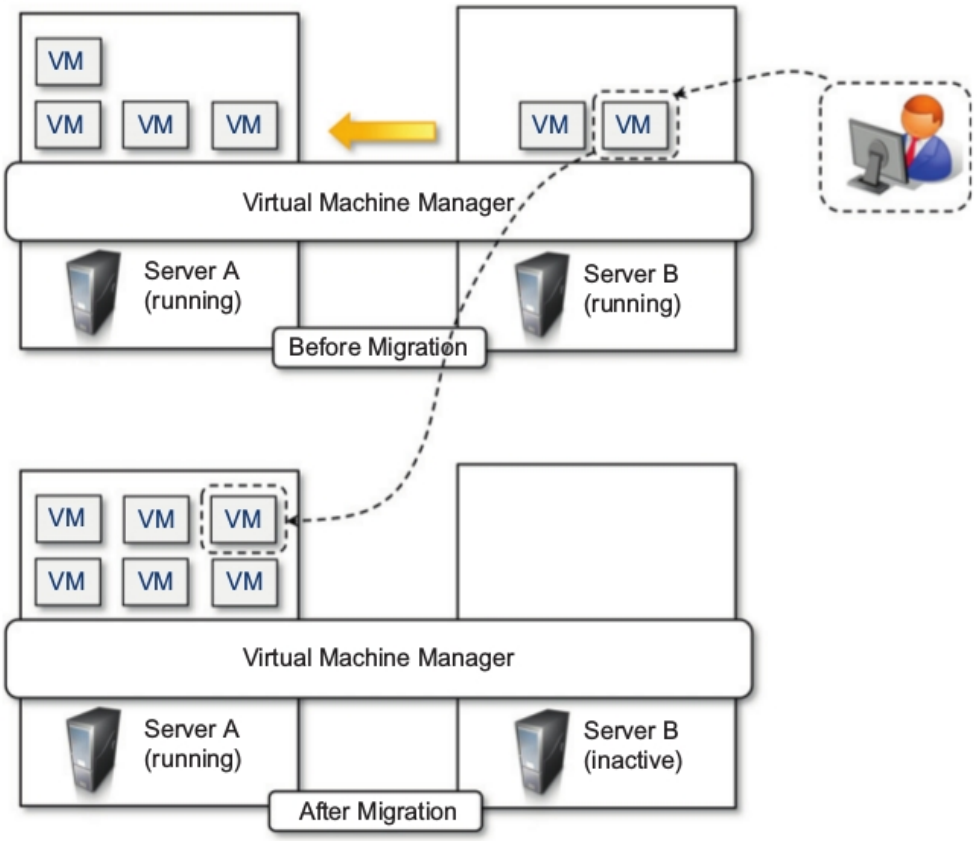


FIGURE 3.10

Live migration and server consolidation.

Since 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. This opportunity is particularly attractive when resources are underutilized, because 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.

3.5 Pros and cons of virtualization

Virtualization has now become extremely popular and widely used, especially in cloud computing. Today, the capillary diffusion of the Internet connection and the advancements in computing technology have made virtualization an interesting opportunity to deliver on-demand IT infrastructure and services.

3.5.1 Advantages of virtualization

1. 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.
2. Portability is another advantage of virtualization, especially for execution virtualization techniques. Virtual machine instances are normally represented by one or more files that can be easily transported with respect to physical systems.
3. 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. Since the guest program is executed in a virtual environment, there is very limited opportunity for the guest program to damage the underlying hardware.
4. Finally, 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.

3.5.2 The other side of the coin: disadvantages

1 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 (delays).

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

2 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.

3 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.

The same considerations can be made for programming-level virtual machines: Modified ver- sions of the runtime environment can access sensitive information or monitor the memory locations utilized by guest applications while these are executed.