CHAPTER 12

WORKFLOW ENGINE FOR CLOUDS

SURAJ PANDEY, DILEBAN KARUNAMOORTHY,
and RAJKUMAR BUYYA

12.1 INTRODUCTION

A workflow models a process as consisting of a series of steps that simplifies the complexity of execution and management of applications. Scientific workflows in domains such as high-energy physics and life sciences utilize distributed resources in order to access, manage, and process a large amount of data from a higher level. Processing and managing such large amounts of data require the use of a distributed collection of computation and storage facilities. These resources are often limited in supply and are shared among many competing users. The recent progress in virtualization technologies and the rapid growth of cloud computing services have opened a new paradigm in distributed computing for utilizing existing (and often cheaper) resource pools for on-demand and scalable scientific computing. Scientific Workflow Management Systems (WfMS) need to adapt to this new paradigm in order to leverage the benefits of cloud services.

Cloud services vary in the levels of abstraction and hence the type of service they present to application users. Infrastructure virtualization enables providers such as Amazon[1] to offer virtual hardware for use in compute- and data-intensive workflow applications. Platform-as-a-Service (PaaS) clouds expose a higher-level development and runtime environment for building and deploying workflow applications on cloud infrastructures. Such services may also expose domain-specific concepts for rapid-application development. Further up in the cloud stack are Software-as-a-Service providers who offer end users with

[1] http://aws.amazon.com

standardized software solutions that could be integrated into existing workflows.

This chapter presents workflow engines and its integration with the cloud computing paradigm. We start by reviewing existing solutions for workflow applications and their limitations with respect to scalability and on-demand access. We then discuss some of the key benefits that cloud services offer workflow applications, compared to traditional grid environments. Next, we give a brief introduction to workflow management systems in order to highlight components that will become an essential part of the discussions in this chapter. We discuss strategies for utilizing cloud resources in workflow applications next, along with architectural changes, useful tools, and services. We then present a case study on the use of cloud services for a scientific workflow application and finally end the chapter with a discussion on visionary thoughts and the key challenges to realize them. In order to aid our discussions, we refer to the workflow management system and cloud middleware developed at CLOUDS Lab, University of Melbourne. These tools, referred to as Cloudbus toolkit [1], henceforth, are mature platforms arising from years of research and development.

12.2 BACKGROUND

Over the recent past, a considerable body of work has been done on the use of workflow systems for scientific applications. Yu and Buyya [2] provide a comprehensive taxonomy of workflow management systems based on workflow design, workflow scheduling, fault management, and data movement. They characterize and classify different approaches for building and executing workflows on Grids. They also study existing grid workflow systems highlighting key features and differences.

Some of the popular workflow systems for scientific applications include DAGMan (Directed Acyclic Graph MANager) [3, 4], Pegasus [5], Kepler [6], and Taverna workbench [7]. DAGMan is a workflow engine under the Pegasus workflow management system. Pegasus uses DAGMan to run the executable workflow. Kepler provides support for Web-service-based workflows. It uses an actor-oriented design approach for composing and executing scientific application workflows. The computational components are called actors, and they are linked together to form a workflow. The Taverna workbench enables the automation of experimental methods through the integration of various services, including WSDL-based single operation Web services, into workflows. For a detailed description of these systems, we refer you to Yu and Buyya [2].

Scientific workflows are commonly executed on shared infrastructure such as Tera-Grid,[2] Open Science Grid,[3] and dedicated clusters [8]. Existing workflow systems tend to utilize these global Grid resources that are made available

[2] http://www.teragrid.org

[3] http://www.opensciencegrid.org

through prior agreements and typically at no cost. The notion of leveraging virtualized resources was new, and the idea of using resources as a utility [9, 10] was limited to academic papers and was not implemented in practice. With the advent of cloud computing paradigm, economy-based utility computing is gaining widespread adoption in the industry.

Deelman et al. [11] presented a simulation-based study on the costs involved when executing scientific application workflows using cloud services. They studied the cost performance trade-offs of different execution and resource provisioning plans, and they also studied the storage and communication fees of Amazon S3 in the context of an astronomy application known as Montage [5, 10]. They conclude that cloud computing is a cost-effective solution for data-intensive applications.

The Cloudbus toolkit [1] is our initiative toward providing viable solutions for using cloud infrastructures. We propose a wider vision that incorporates an inter-cloud architecture and a market-oriented utility computing model. The Cloudbus workflow engine [12], presented in the sections to follow, is a step toward scaling workflow applications on clouds using market-oriented computing.

12.3 WORKFLOW MANAGEMENT SYSTEMS AND CLOUDS

The primary benefit of moving to clouds is application scalability. Unlike grids, scalability of cloud resources allows real-time provisioning of resources to meet application requirements at runtime or prior to execution. The elastic nature of clouds facilitates changing of resource quantities and characteristics to vary at runtime, thus dynamically scaling up when there is a greater need for additional resources and scaling down when the demand is low. This enables workflow management systems to readily meet quality-of-service (QoS) requirements of applications, as opposed to the traditional approach that required advance reservation of resources in global multi-user grid environments. With most cloud computing services coming from large commercial organizations, service-level agreements (SLAs) have been an important concern to both the service providers and consumers. Due to competitions within emerging service providers, greater care is being taken in designing SLAs that seek to offer (a) better QoS guarantees to customers and (b) clear terms for compensation in the event of violation. This allows workflow management systems to provide better end-to-end guarantees when meeting the service requirements of users by mapping them to service providers based on characteristics of SLAs. Economically motivated, commercial cloud providers strive to provide better services guarantees compared to grid service providers. Cloud providers also take advantage of economies of scale, providing compute, storage, and bandwidth resources at substantially lower costs. Thus utilizing public cloud services could be economical and a cheaper alternative (or add-on) to the more expensive dedicated resources. One of the benefits of using virtualized resources for workflow execution, as opposed to having direct access to the physical machine, is the reduced need for securing the physical resource from malicious code using techniques such as sandboxing. However, the long-term effect of using virtualized resources in clouds that effectively share a "slice" of the physical machine, as opposed to using dedicated resources for high-performance applications, is an interesting research question.

12.3.1 Architectural Overview

Figure 12.1 presents a high-level architectural view of a Workflow Management System (WfMS) utilizing cloud resources to drive the execution of a scientific

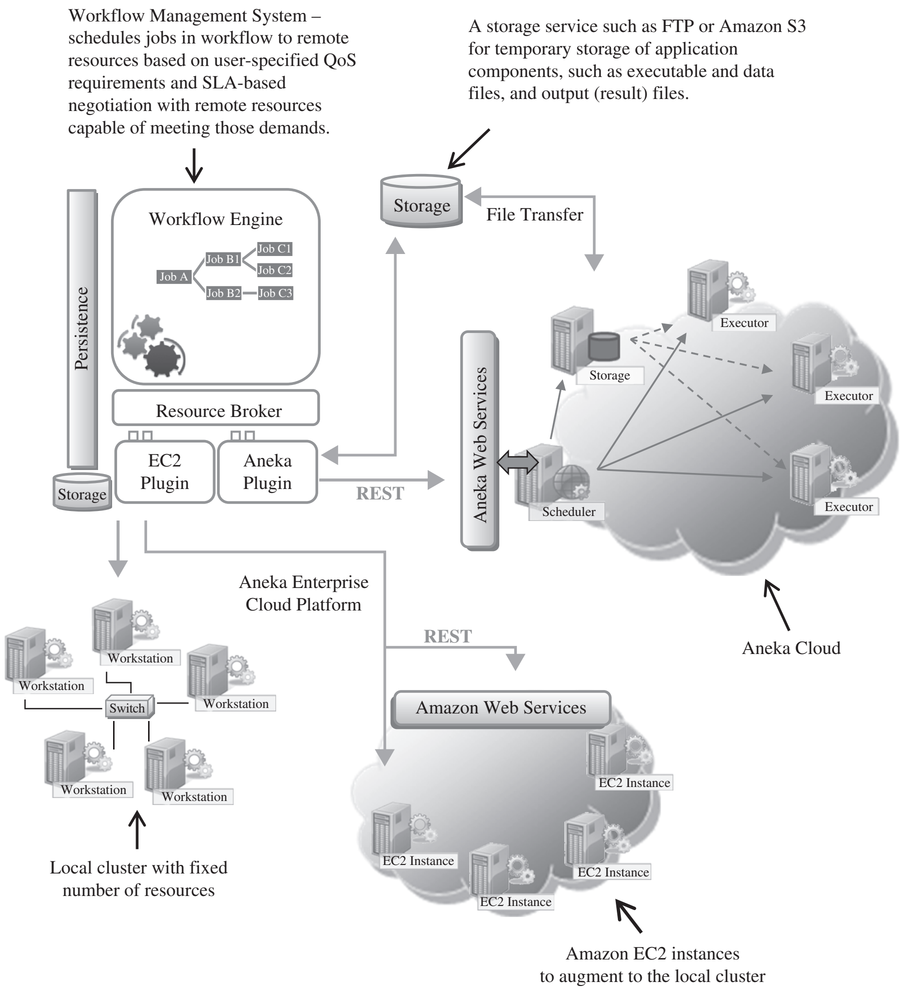

Workflow Management System – schedules jobs in workflow to remote resources based on user-specified QoS requirements and SLA-based negotiation with remote resources capable of meeting those demands.
A storage service such as FTP or Amazon S3 for temporary storage of application components, such as executable and data files, and output (result) files.
Storage
File Transfer
Persistence
Workflow Engine
Job A
Job B1
Job B2
Job C1
Job C2
Job C3
Resource Broker
EC2 Plugin
Aneka Plugin
Storage
REST
Aneka Web Services
Storage
Executor
Executor
Executor
Scheduler
Aneka Cloud
Aneka Enterprise Cloud Platform
REST
Workstation
Workstation
Workstation
Switch
Workstation
Workstation
Local cluster with fixed number of resources
Amazon Web Services
EC2 Instance
EC2 Instance
EC2 Instance
EC2 Instance
Amazon EC2 instances to augment to the local cluster

FIGURE 12.1. Workflow engine in the cloud.

workflow application. The workflow system comprises the workflow engine, a resource broker [13], and plug-ins for communicating with various technological platforms, such as Aneka [14] and Amazon EC2. A detailed architecture describing the components of a WfMS is given in Section 12.4.

User applications could only use cloud services or use cloud together with existing grid/cluster-based solutions. Figure 12.1 depicts two scenarios, one where the Aneka platform is used in its entirety to complete the workflow, and the other where Amazon EC2 is used to supplement a local cluster when there are insufficient resources to meet the QoS requirements of the application. Aneka [13], described in further detail in Section 12.5, is a PaaS cloud and can be run on a corporate network or a dedicated cluster or can be hosted entirely on an IaaS cloud. Given limited resources in local networks, Aneka is capable of transparently provisioning additional resources by acquiring new resources in third-party cloud services such as Amazon EC2 to meet application demands. This relieves the WfMS from the responsibility of managing and allocating resources directly, to simply negotiating the required resources with Aneka.

Aneka also provides a set of Web services for service negotiation, job submission, and job monitoring. The WfMS would orchestrate the workflow execution by scheduling jobs in the right sequence to the Aneka Web Services.

The typical flow of events when executing an application workflow on Aneka would begin with the WfMS staging in all required data for each job onto a remote storage resource, such as Amazon S3 or an FTP server. In this case, the data would take the form of a set of files, including the application binaries. These data can be uploaded by the user prior to execution, and they can be stored in storage facilities offered by cloud services for future use. The WfMS then forwards workflow tasks to Aneka's scheduler via the Web service interface. These tasks are subsequently examined for required files, and the storage service is instructed to stage them in from the remote storage server, so that they are accessible by the internal network of execution nodes. The execution begins by scheduling tasks to available execution nodes (also known as worker nodes). The workers download any required files for each task they execute from the storage server, execute the application, and upload all output files as a result of the execution back to the storage server. These files are then staged out to the remote storage server so that they are accessible by other tasks in the workflow managed by the WfMS. This process continues until the workflow application is complete.

The second scenario describes a situation in which the WfMS has greater control over the compute resources and provisioning policies for executing workflow applications. Based on user-specified QoS requirements, the WfMS schedules workflow tasks to resources that are located at the local cluster and in the cloud. Typical parameters that drive the scheduling decisions in such a scenario include deadline (time) and budget (cost) [15, 16]. For instance, a policy for scheduling an application workflow at minimum execution cost would utilize local resources and then augment them with cheaper cloud resources, if needed, rather than using high-end but more expensive cloud resources. On the contrary, a policy that scheduled workflows to achieve minimum execution time would always use high-end cluster and cloud resources, irrespective of costs. The resource provisioning policy determines the extent of additional resources to be provisioned on the public clouds. In this second scenario, the WfMS interacts directly with the resources provisioned. When using Aneka, however, all interaction takes place via the Web service interface.

The following sections focuses on the integration of workflow management systems and clouds and describes in detail practical issues involved in using clouds for scientific workflow applications.

12.4 ARCHITECTURE OF WORKFLOW MANAGEMENT SYSTEMS

Scientific applications are typically modeled as workflows, consisting of tasks, data elements, control sequences and data dependencies. *Workflow management systems* are responsible for managing and executing these workflows. According to Raicu et al. [17], scientific workflow management systems are engaged and applied to the following aspects of scientific computations: (1) describing complex scientific procedures (using GUI tools, workflow specific languages), (2) automating data derivation processes (data transfer components), (3) high-performance computing (HPC) to improve throughput and performance (distributed resources and their coordination), and (4) provenance management and query (persistence components). The Cloudbus Workflow Management System [12] consists of components that are responsible for handling tasks, data and resources taking into account users' QoS requirements. Its architecture is depicted in Figure 12.2. The architecture consists of three major parts: (a) the user interface, (b) the core, and (c) plug-ins. The user interface allows end users to work with workflow composition, workflow execution planning, submission, and monitoring. These features are delivered through a Web portal or through a stand-alone application that is installed at the user's end. Workflow composition is done using an XML-based Workflow Language (xWFL). Users define task properties and link them based on their data dependencies. Multiple tasks can be constructed using copy-paste functions present in most GUIs.

The components within the core are responsible for managing the execution of workflows. They facilitate in the translation of high-level workflow descriptions (defined at the user interface using XML) to task and data objects. These objects are then used by the execution subsystem. The scheduling component applies user-selected scheduling policies and plans to the workflows at various stages in their execution. The tasks and data dispatchers interact with the resource interface plug-ins to continuously submit and monitor tasks in the workflow. These components form the core part of the workflow engine.

The plug-ins support workflow executions on different environments and platforms. Our system has plug-ins for querying task and data characteristics

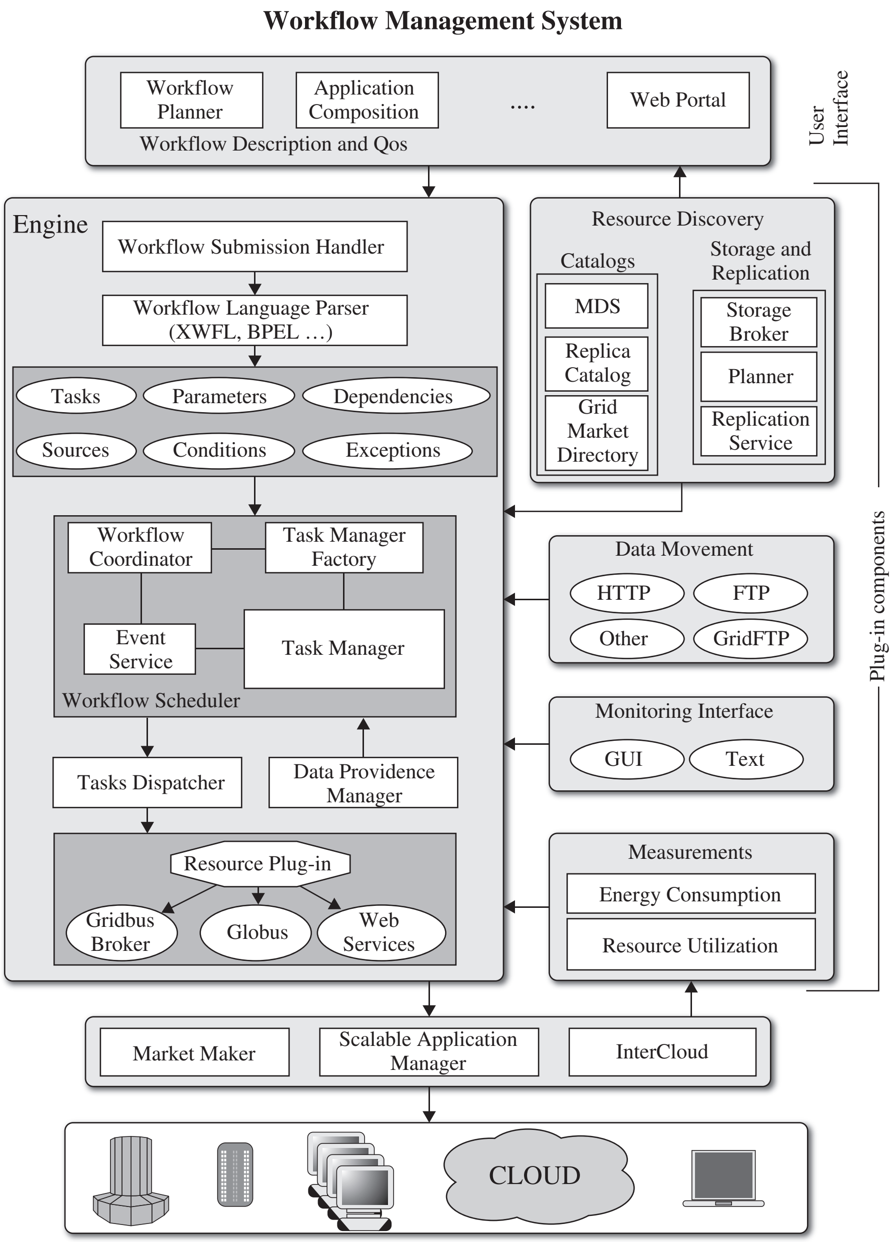


FIGURE 12.2. Architecture of Workflow Management System.

(e.g., querying metadata services, reading from trace files), transferring data to and from resources (e.g., transfer protocol implementations, and storage and replication services), monitoring the execution status of tasks and applications (e.g., real-time monitoring GUIs, logs of execution, and the scheduled retrieval of task status), and measuring energy consumption.

The resources are at the bottom layer of the architecture and include clusters, global grids, and clouds. The WfMS has plug-in components for interacting with various resource management systems present at the front end of distributed resources. Currently, the Cloudbus WfMS supports Aneka, Pbs, Globus, and fork-based middleware. The resource managers may communicate with the market maker, scalable application manager, and InterCloud services for global resource management [18].

12.5 UTILIZING CLOUDS FOR WORKFLOW EXECUTION

Taking the leap to utilizing cloud services for scientific workflow applications requires an understanding of the types of clouds services available, the required component changes in workflow systems for interacting with cloud services, the set of tools available to support development and deployment efforts, the steps involved in deploying workflow systems and services on the cloud, and an appreciation of the key benefits and challenges involved. In the sections to follow, we take a closer look at some of these issues. We begin by introducing the reader to the Aneka Enterprise Cloud service. We do this for two reasons. First, Aneka serves as a useful tool for utilizing clouds, including platform abstraction and dynamic provisioning. Second, we describe later in the chapter a case study detailing the use of Aneka to execute a scientific workflow application on clouds.

12.5.1 Aneka

Aneka is a distributed middleware for deploying platform-as-a-service (PaaS) offerings (Figure 12.3). Developed at CLOUDS Lab, University of Melbourne, Aneka is the result of years of research on cluster, grid, and cloud computing for high-performance computing (HPC) applications. Aneka, which is both a development and runtime environment, is available for public use (for a cost),[4] can be installed on corporate networks, or dedicated clusters, or can be hosted on infrastructure clouds like Amazon EC2. In comparison, similar PaaS services such as Google AppEngine [19] and Windows Azure [20] are in-house platforms hosted on infrastructures owned by the respective companies. Aneka was developed on Microsoft's.NET Framework 2.0 and is compatible with other implementations of the ECMA 335 standard [21], such as Mono. Aneka

[4] http://www.manjrasoft.com

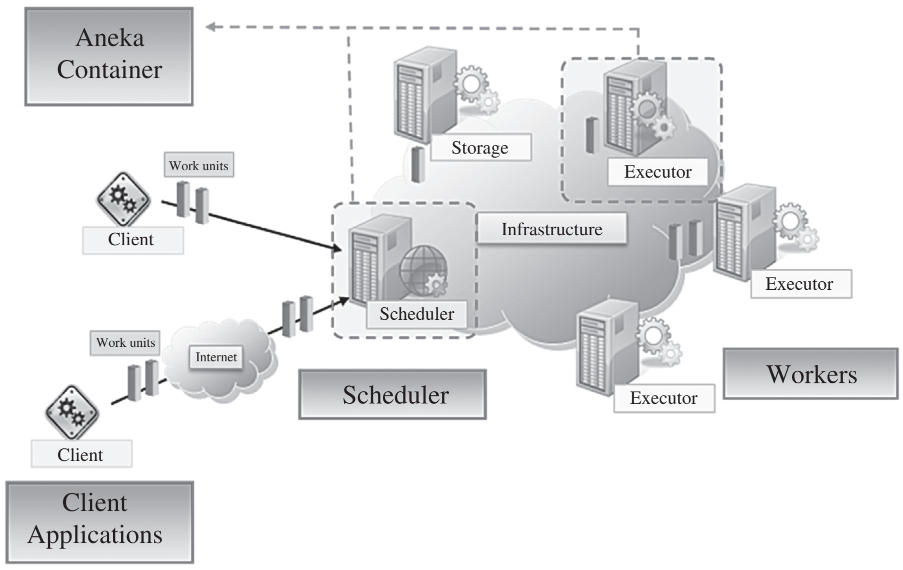


FIGURE 12.3. A deployment of Aneka Enterprise Cloud.

can run on popular platforms such as Microsoft Windows, Linux, and Mac OS X, harnessing the collective computing power of a heterogeneous network.

The runtime environment consists of a collection of Aneka containers running on physical or virtualized nodes. Each of these containers can be configured to play a specific role such as scheduling or execution. The Aneka distribution also provides a set of tools for administrating the cloud, reconfiguring nodes, managing users, and monitoring the execution of applications. The Aneka service stack provides services for infrastructure management, application execution management, accounting, licensing, and security. For more information we refer you to Vecchiola et al. [14].

Aneka's *Dynamic Resource Provisioning* service enables horizontal scaling depending on the overall load in the cloud. The platform is thus elastic in nature and can provision additional resources on-demand from external physical or virtualized resource pools, in order to meet the QoS requirements of applications. In a typical scenario, Aneka would acquire new virtualized resources from external clouds such as Amazon EC2, in order to meet the minimum waiting time of applications submitted to Aneka. Such a scenario would arise when the current load in the cloud is high, and there is a lack of available resources to timely process all jobs.

The development environment provides a rich set of APIs for developing applications that can utilize free resources of the underlying infrastructure. These APIs expose different programming abstractions, such as the *task model*, *thread model,* and *MapReduce* [22]. The *task programming model* is of particular importance to the current discussion. It models "independent bag of tasks" (BoT) applications that are composed of a collection of *work units* independent of each other, and it may be executed in any given order. One of the benefits of the *task programming model* is its simplicity, making it easy to run legacy applications on the cloud. An application using the *task model* composes one or more task instances and forwards them as work units to the scheduler. The scheduling service currently supports the *First-In-First-Out*, *First-In-First-Out with Backfilling, Clock-Rate Priority,* and *Preemption-Based Priority Queue* scheduling algorithms. The runtime environment also provides two specialized services to support this model: the *task scheduling service* and the *task execution service*.

The storage service provides a temporary repository for application files—that is, input files that are required for task execution, and output files that are he result of execution. Prior to dispatching work units, any files required are staged-in to the storage service from the remote location. This remote location can be either the client machine, a remote FTP server, or a cloud storage service such as Amazon S3. The work units are then dispatched to executors, which download the files before execution. Any output files produced as a result of the execution are uploaded back to the storage service. From here they are staged-out to the remote storage location.

12.5.2 Aneka Web Services

Aneka exposes three SOAP Web services for service negotiation, reservation, and task submission, as depicted in Figure 12.4. The negotiation and reservation services work in concert, and they provide interfaces for negotiating

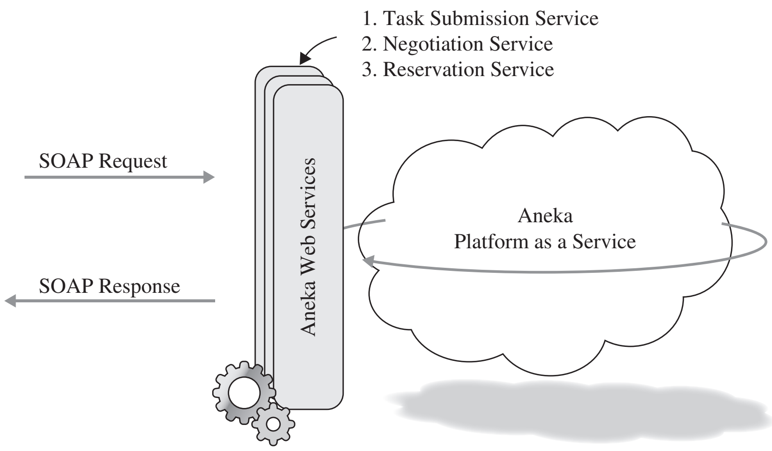


FIGURE 12.4. Aneka Web services interface.

resource use and reserving them in Aneka for predetermined timeslots. As such, these services are only useful when Aneka has limited resources to work with and no opportunities for provisioning additional resources. The *task Web service* provides a SOAP interface for executing jobs on Aneka. Based on the *task programming model*, this service allows remote clients to submit jobs, monitor their status, and abort jobs.

12.5.3 General Approach

Traditional WfMSs were designed with a centralized architecture and were thus tied to a single machine. Moving workflow engines to clouds requires (a) architectural changes and (b) integration of cloud management tools.

Architectural Changes. Most components of a WfMS can be separated from the core engine so that they can be executed on different cloud services. Each separated component could communicate with a centralized or replicated workflow engine using events. The manager is responsible for coordinating the distribution of load to its subcomponents, such as the Web server, persistence, monitoring units, and so forth.

In our WfMS, we have separated the components that form the architecture into the following: user interface, core, and plug-ins. The user interface can now be coupled with a Web server running on a "large" instance of cloud that can handle increasing number of users. The Web request from users accessing the WfMS via a portal is thus offloaded to a different set of resources.

Similarly, the core and plug-in components can be hosted on different types of instances separately. Depending on the size of the workload from users, these components could be migrated or replicated to other resources, or reinforced with additional resources to satisfy the increased load. Thus, employing distributed modules of the WfMS on the basis of application requirements helps scale the architecture.

Integration of Cloud Management Tools. As the WfMS is broken down into components to be hosted across multiple cloud resources, we need a mechanism to (a) access, transfer, and store data and (b) enable and monitor executions that can utilize this approach of scalable distribution of components.

The cloud service provider may provide APIs and tools for discovering the VM instances that are associated to a user's account. Because various types of instances can be dynamically created, their characteristics such as CPU capacity and amount of available memory are a part of the cloud service provider's specifications. Similarly, for data storage and access, a cloud may provide data sharing, data movement, and access rights management capabilities to user's applications. Cloud measurement tools may be in place to account for the amount of data and computing power used, so that users are charged on the pay-per-use basis. A WfMS now needs to access these tools to discover and characterize the resources available in the cloud. It also needs to interpret the access rights (e.g., access control lists provided by Amazon), use the data movement APIs, and share mechanisms between VMs to fully utilize the benefits of moving to clouds. In other words, traditional catalog services such as the Globus Monitoring and Discovery Service (MDS) [23], Replica Location Services, Storage Resource Brokers, Network Weather Service [24], and so on could be easily replaced by more user-friendly and scalable tools and APIs associated with a cloud service provider. We describe some of these tools in the following section.

12.5.4 Tools for Utilizing Clouds in WfMS

The range of tools and services offered by cloud providers play an important role in integrating WfMSs with clouds (Figure 12.5). Such services can facilitate in the deployment, scaling, execution, and monitoring of workflow systems. This section discusses some of the tools and services offered by various service providers that can complement and support WfMSs.

A WfMS manages dynamic provisioning of compute and storage resources in the cloud with the help of tools and APIs provided by service providers. The provisioning is required to dynamically scale up/down according to application requirements. For instance, data-intensive workflow applications may require

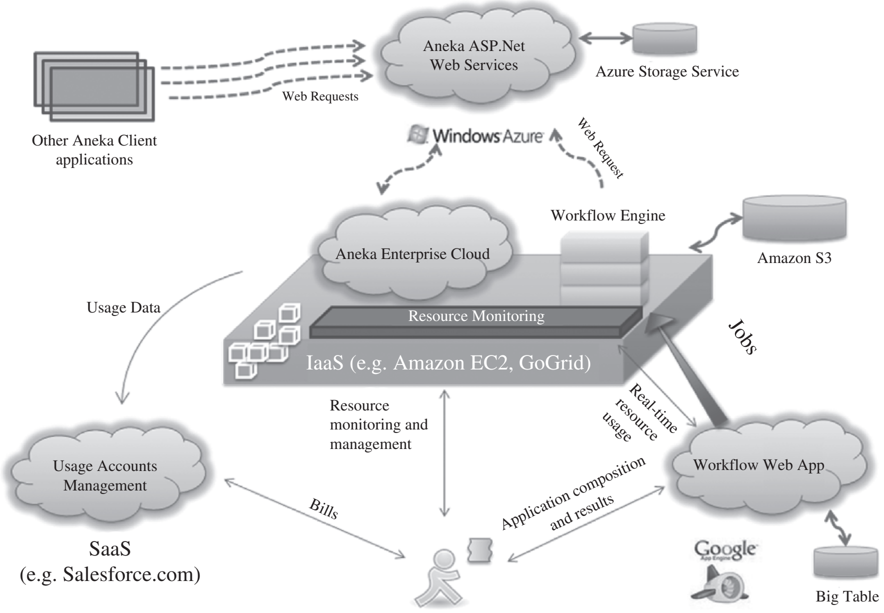

Aneka ASP.Net Web Services
Azure Storage Service
Web Requests
Other Aneka Client applications
Windows Azure
Web Request
Workflow Engine
Amazon S3
Aneka Enterprise Cloud
Usage Data
Resource Monitoring
Jobs
IaaS (e.g. Amazon EC2, GoGrid)
Resource monitoring and management
Real-time resource usage
Usage Accounts Management
Workflow Web App
Application composition and results
Bills
SaaS (e.g. Salesforce.com)
Google App Engine
Big Table

FIGURE 12.5. A workflow utilizing multiple cloud services.

large amount of disk space for storage. A WfMS could provision dynamic volumes of large capacity that could be shared across all instances of VMs (similar to snapshots and volumes provided by Amazon). Similarly, for compute-intensive tasks in an workflow, a WfMS could provision specific instances that would help accelerate the execution of these compute-intensive tasks.

A WfMS implements scheduling policies to assign tasks to resources based on applications' objectives. This task-resource mapping is dependent on several factors: compute resource capacity, application requirements, user's QoS, and so forth. Based on these objectives, a WfMS could also direct a VM provisioning system to consolidate data center loads by migrating VMs so that it could make scheduling decisions based on locality of data and compute resources.

A persistence mechanism is often important in workflow management systems and for managing metadata such as available resources, job queues, job status, and user data including large input and output files. Technologies such as Amazon S3, Google's BigTable, and the Windows Azure Storage Services can support most storage requirements for workflow systems, while also being scalable, reliable, and secure. If large quantities of user data are being dealt with, such as a large number of brain images used in functional magnetic resonance imaging (fMRI) studies [12], transferring them online can be both expensive and time-consuming. In such cases, traditional post can prove to be cheaper and faster. Amazon's AWS Import/Export[5] is one such service that aims to speed up data movement by transferring large amounts of data in portable storage devices. The data are shipped to/from Amazon and offloaded into/from S3 buckets using Amazon's high-speed internal network. The cost savings can be significant when transferring data on the order of terabytes.

Most cloud providers also offer services and APIs for tracking resource usage and the costs incurred. This can complement workflow systems that support budget-based scheduling by utilizing real-time data on the resources used, the duration, and the expenditure. This information can be used both for making scheduling decisions on subsequent jobs and for billing the user at the completion of the workflow application.[6]

Cloud services such as Google App Engine and Windows Azure provide platforms for building scalable interactive Web applications. This makes it relatively easy to port the graphical components of a workflow management system to such platforms while benefiting from their inherent scalability and reduced administration. For instance, such components deployed on Google App Engine can utilize the same scalable systems that drive Google applications, including technologies such as BigTable [25] and GFS [26].

[5] http://aws.amazon.com/importexport/

[6] http://aws.amazon.com/devpay/

12.6 CASE STUDY: EVOLUTIONARY MULTIOBJECTIVE OPTIMIZATIONS

This section presents a scientific application workflow based on an iterative technique for optimizing multiple search objectives, known as evolutionary multiobjective optimization (EMO) [27]. EMO is a technique based on genetic algorithms. Genetic algorithms are search algorithms used for finding optimal solutions in a large space where deterministic or functional approaches are not viable. Genetic algorithms use heuristics to find an optimal solution that is acceptable within a reasonable amount of time. In the presence of many variables and complex heuristic functions, the time consumed in finding even an acceptable solution can be too large. However, when multiple instances are run in parallel in a distributed setting using different variables, the required time for computation can be drastically reduced.

12.6.1 Objectives

The following are the objectives for modeling and executing an EMO workflow on clouds:

- Design an execution model for EMO, expressed in the form of a workflow, such that multiple distributed resources can be utilized.
- Parallelize the execution of EMO tasks for reducing the total completion time.
- Dynamically provision compute resources needed for timely completion of the application when the number of tasks increase.
- Repeatedly carry out similar experiments as and when required.
- Manage application execution, handle faults, and store the final results for analysis.

12.6.2 Workflow Solution

In order to parallelize the execution of EMO, we construct a workflow model for systematically executing the tasks. A typical workflow structure is depicted in Figure 12.6.

In our case study, the EMO application consists of five different topologies, upon which the iteration is done. These topologies are defined in five different binary files. Each file becomes the input files for the top level tasks (A0emo1, A0emo, . . .). We create a separate branch for each topology file. In Figure 12.6, there are two branches, which get merged on level 6. The tasks at the root level operate on the topologies to create new population, which is then merged by the task named "emomerge." In Figure 12.6, we see two "emomerge" tasks in the 2nd level, one task in the 6th level that merges two branches and then splits the population to two branches again, two tasks on the 8th and 10th

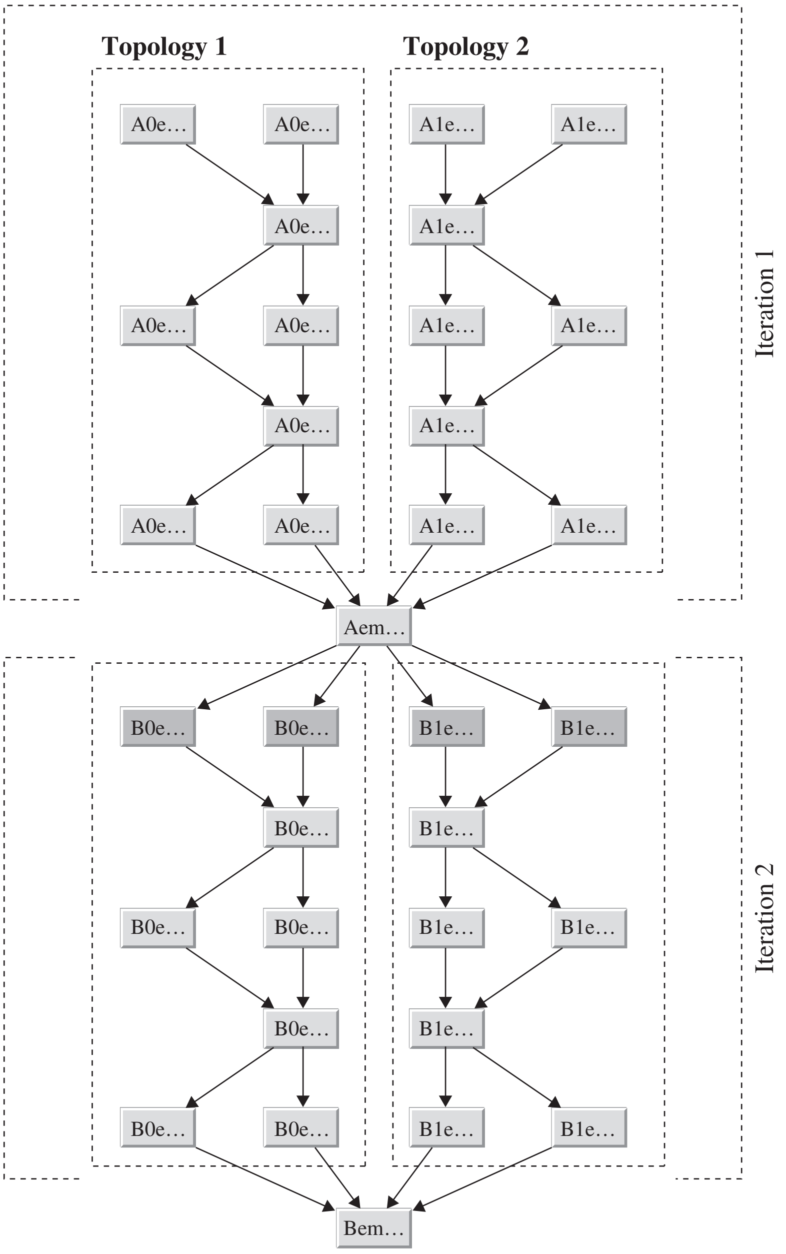


FIGURE 12.6. EMO workflow structure (boxes represent task, arrows represent data-dependencies between tasks).

levels, and the final task on the 12th level. In the example figure, each topology is iterated two times in a branch before getting merged. The merged population is then split. This split is done two times in the figure. The tasks labeled B0e and B1e (depicted as darker shade in Figure 12.6) is the start of second iteration.

12.6.3 Deployment and Results

EMO Application. We use ZDT2 [27] as a test function for the objective function. The workflow for this problem is depicted in Figure 12.6.

In our experiments, we carry out 10 iterations within a branch for 5 different topologies. We merge and split the results of each of these branches 10 times. For this scenario, the workflow constituted of a total of 6010 tasks. We varied the tasks by changing the number of merges from 5 to 10. In doing so, the structure and the characteristics of the tasks in the workflow would remain unchanged. This is necessary for comparing the execution time when the number of task increases from 1600 to 6000 when we alter the number of merges from 5 to 10.

Compute Resource. We used 40 Amazon EC2 compute resources for executing the EMO application. Twenty resources were instantiated at US-east-1a, and 20 were instantiated at US-east-1d. Among these resources, one was used for the workflow engine, one was used for Aneka's master node and the rest were worker nodes. The characteristics of these resources are listed in Table 12.1.

The workflow engine, along with a database for persistence, the IBM TSpace [28] based coordination server, and the Tomcat Web container, was instantiated on a medium instance VM.

Output of EMO Application. After running the EMO workflow, we expect to see optimized values for the two objectives given by the ZDT2 test function. Figure 12.7 depicts the graph that plots the *front* obtained after iterating the EMO workflow depicted in Figure 12.6. The front at Level 2 is not the optimal. After first iteration, the front is optimized. Iteration 2 does not significantly change the front, hence the overlapping of the data for Iteration 1 and 2.

Experimental Results When Using Clouds. Because the EMO workflow is an iterative approach, increasing the number of iterations would increase the quality of optimization in the results. Analogously, the greater the number of tasks completing in the workflow, the greater the number of iterations, hence the better the optimization.

Because the iterations can be carried out for an arbitrarily large number of times, it is usually a best practice to limit the time for the overall calculation. Thus, in our experiment we set the deadline to be 95 minutes. We then analyze the number of tasks completing within the first 95 minutes in two classes of experiments.

TABLE 12.1. Characteristics of Amazon Compute Resources (EC2) Used in Our Experiment

Characteristics	Aneka Master/Worker	Workflow Engine
Platform	Windows 2000 Server	Linux
CPU (type)	1 EC2 Compute Units[a] (small)	5 EC2 Compute Units[b] (medium)
Memory	1.7 GB	1.7 GB
Instance storage	160 GB	350GB
Instance location	US-east-1a (19) US-east-1b(20)	US-east-1a
Number of instances	39	1
Price per hour	$US 0.12	$US 0.17

[a]Small instance (default) 1.7 GB of memory, 1 EC2 compute unit (1 virtual core with 1 EC2 compute unit), 160 GB of instance storage, 32-bit platform.
[b]High-CPU medium instance 1.7 GB of memory, 5 EC2 compute units (2 virtual cores with 2.5 EC2 compute units each), 350 GB of instance storage, 32-bit platform.
Source: Amazon.

Experiment 1: Seven Additional EC2 Instances Were Added. In this experiment, we started executing the tasks in the EMO workflow initially using 20 EC2 compute resources (one node for workflow engine, one node for Aneka master, 18 Aneka worker nodes). We instantiate seven more small instances to increase the total number of resources to 25. They were available for use after 25 minutes of execution. At the end of 95 minutes, a total of 1612 tasks were completed.

Experiment 2: Twenty Additional EC2 Instances Were Added. In this experiment, we started executing the tasks in the EMO workflow using 20

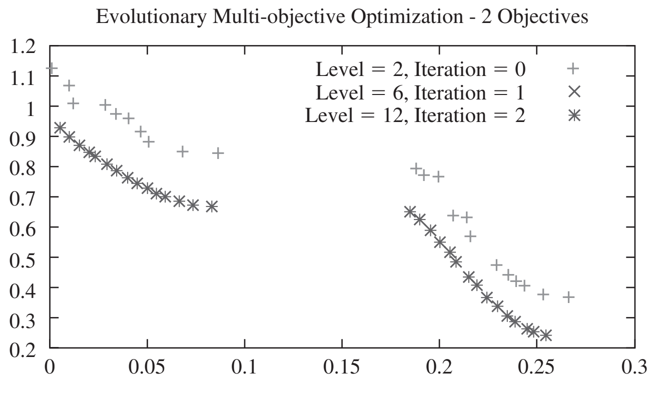


FIGURE 12.7. A graph that plots the pareto-front obtained after executing EMO for ZTD2 test problem.

EC2 compute resources, similar to Experiment 1. We instantiated 20 more EC2 instances after noticing the linear increase in task completion rate. These instances however were available for use after 40 minutes of execution. At the end of 95 minutes, a total of 3221 tasks were completed.

Analysis of the Results. In both experiments, the initial task completion rate increased linearly until we started more instances, as depicted in Figure 12.8. As the number of resources was increased, the rate of task completing increased drastically. This is due to the submission of queued tasks in Aneka to the newly available resources, which would have remained queued if resources were not added.

In the figure, the completion rate curve rises up to a point until all the queued tasks are submitted. The curve then rises gradually because the EMO application is a workflow. Tasks in the workflow get submitted gradually as their parents finish executions. Hence, the completion rate has similar slope

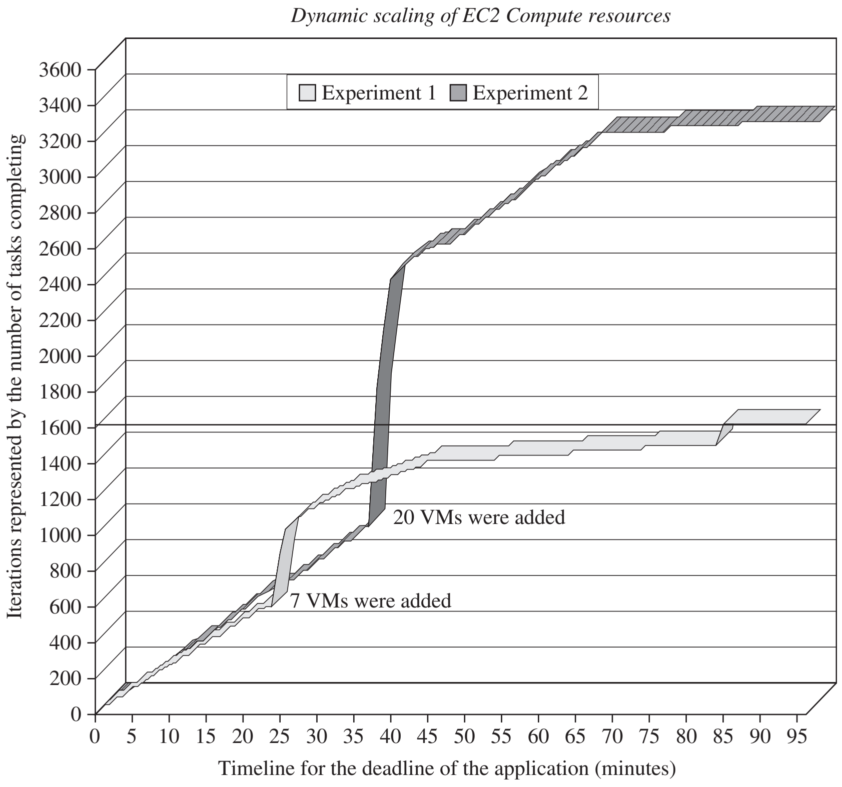
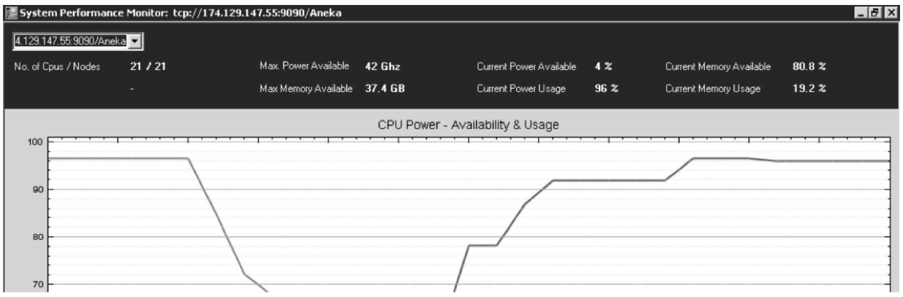


FIGURE 12.8. Number of tasks completing in time as the number of compute resources provisioned were increased at runtime.

as the initial rate, even after increasing the number of resources (30 to 45 minutes for Experiment 1; 45 to 70 minutes for Experiment 2). When more tasks began completing as a result of adding new resources, the workflow engine was able to submit additional tasks for execution. As a result, tasks started competing for resources and hence were being queued by Aneka. Because of this queuing at Aneka's scheduler, the curve flattens after 45 minutes for Experiment 1 and after 70 minutes for Experiment 2.

The most important benefit of increasing the resources dynamically at runtime is the increase in the total number of tasks completing, and hence the quality of final result. This is evident from the two graphs depicted in Figure 12.8. If a total of 25 resources were used, Experiment 1 would complete 1612 tasks by the end of the 95-minute deadline, whereas Experiment 2 would complete executing nearly 3300 tasks within the same deadline if 20 additional resources were added. The quality of results would be twice as good for Experiment 2 as for Experiment 1. However, if a user wants to have the same quality of output as in Experiment 1 but in much shorter time, he should increase the number of resources used well before the deadline. A line just above 1600 in Figure 12.8 depicts the cutoff point where the user could terminate all the VM instances and obtain the same quality of results as Experiment 1 would have obtained by running for 95 minutes. It took ∼45 minutes less time for Experiment 2 to execute the same number of tasks as Experiment 1. This drastic reduction in time was seen even when both experiments initially started with the same number of resources. In terms of cost of provisioning additional resources, Experiment 2 is cheaper because there are fewer overheads in time spent queuing and managing task submissions, since the tasks would be submitted as soon as they arrive at Aneka's master node. If Amazon were to charge EC2 usage cost per minute rather than per hour, Experiment 2 would save 45 minutes of execution time at the cost of 20 more resources.

We also analyzed the utilization of instantiated compute resources by Aneka, as depicted in Figure 12.9. At the time of recording the graph, there were 21 worker nodes in the Aneka cloud, with a combined power of 42 GHz.


System Performance Monitor: tcp://174.129.147.55:9090/Aneka

4.129.147.55:9090/Aneka

No. of Cpus / Nodes	21 / 21	Max. Power Available	42 Ghz	Current Power Available	4 %	Current Memory Available	80.8 %
	-	Max Memory Available	37.4 GB	Current Power Usage	96 %	Current Memory Usage	19.2 %

FIGURE 12.9. Distributed compute resource utilized by Aneka network.

The graph shows a steep rise in the system utilization (labeled as *usage* in the figure) as tasks were submitted for execution. The compute power available (labeled as *available*) decreased to 4% with 80.8% memory available. This decrease in utilization was due to the use of all the available resources for execution of tasks submitted to Aneka by the workflow engine executing EMO workflow.

12.7 VISIONARY THOUGHTS FOR PRACTITIONERS

The cloud computing paradigm is emerging and is being adopted at a rapid rate. Gartner ranks it at the top of the hype cycle for the year 2010 [29]. As the technology is being adopted by practitioners industry-wide, there are numerous challenges to overcome. Moreover, these challenges could be addressed via a realistic vision of the cloud computing models of the near future. This section discusses some of them.

Software and service giants such as Google, Amazon, and Microsoft own large data centers for providing a variety of cloud services to customers. These independent and disparate initiatives would eventually lead to an interconnection model where users can choose a combination of services from different providers in their applications. Our vision provides an entity responsible for brokerage of resources across different cloud providers, termed the market maker [16]. These inter-cloud environments would then facilitate executions of workflow applications at distributed data centers. Large scientific experiments would then be able to use inter-cloud resources, brokered through the market maker.

The essence of using cloud services is to be able to dynamically scale the applications running on top of it. Automating resource provisioning and VM instance management in clouds based on multiobjectives (cost, time, and other QoS parameters) can help achieve this goal. The automation process should be transparent to the end users who would just be interested in running workflow applications under their time and budget constraints. Users would specify either flexible or tight deadline for the cost they pay for using cloud services. It becomes the responsibility of the workflow engine running in the cloud to dynamically scale the application to satisfy multiple users′ request.

In order to facilitate fair but competitive use of cloud resources for workflow applications, a service negotiation module must be in place. This entity would negotiate with multiple service providers to match users′ requirements to a service provider's capabilities. Once a match is found, required resources can then be allocated to the user application. A cloud market directory service is needed to maintain a catalog of services from various cloud service providers. Data and their communication play a vital role in any data-intensive workflow application. When running such applications on clouds, storage and transfer costs need to be taken into account in addition to the execution cost. The right choice of compute location and storage service provider would result in minimizing the total cost billed to a user. A cloud market maker could handle these task and communication issues at the time of negotiation between various cloud service providers.

12.8 FUTURE RESEARCH DIRECTIONS

In Section 12.7, we described some visions and inherent difficulties faced by practitioners when using various cloud services. Drawing upon these visions, we list below some future research directions in the form of broad research directions:

- How to facilitate inter-cloud operations in terms of coherent data exchange, task migration, and load balancing for workflow application.
- When and where to provision cloud resources so that workflow applications can meet their deadline constraints and also remain within their budget.
- How to balance the use of cloud and local resources so that workflow applications can meet their objectives.
- How to match workflow application requirements to any service provider's capabilities when there are numerous vendors with similar capabilities in a cloud.

12.9 SUMMARY AND CONCLUSIONS

To summarize, we have presented a comprehensive description of using workflow engine in cloud computing environments. We discussed the limitations of existing workflow management systems and proposed changes that need to be incorporated when moving to clouds. We also described cloud tools that could help applications use cloud services.

To demonstrate a practical scenario of deploying a workflow engine in clouds, we described in detail our workflow management system and a.NET-based cloud computing platform, Aneka. We presented a case study of an evolutionary multiobjective optimization algorithm. By modeling this application in the form of a workflow, we obtained an order-of-magnitude improvement in the application runtime when compute resources were provisioned at runtime. Thousands of tasks were completed in a short period of time as additional resources were provisioned, eventually decreasing the total runtime of the application.

Based on our experience in using cloud services, we conclude that large applications can certainly benefit by using cloud resources. The key benefits are in terms of decreased runtime, on-demand resource provisioning, and ease of resource management. However, these services come at a price whereby users have to pay cloud service providers on the basis of the resource usage.

Although clouds offer many benefits, they can't and will not replace grids. Clouds will augment grids. Users will use cloud services together with their in-house solutions (cluster/enterprise grids) to enhance the performance of their applications as and when needed.

ACKNOWLEDGMENTS

This work is partially supported through Australian Research Council (ARC) Discovery Project grant. We would like to thank Christian Vecchiola from University of Melbourne for his help in defining the EMO application. We also thank Adam Barker for reviewing and providing feedback on this chapter.

REFERENCES

1. R. Buyya, S. Pandey, and C. Vecchiola, Cloudbus toolkit for market-oriented cloud computing, in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom 2009, Springer, Germany)*, Beijing, China, December 1–4, 2009.
2. J. Yu and R. Buyya, A taxonomy of workflow management systems for grid computing, *Journal of Grid Computing*, **3**(3–4):171–200, 2005.
3. DAGMan Application. http://www.cs.wisc.edu/condor/dagman/ (November 2009).
4. T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor—A distributed job scheduler, in *Beowulf Cluster Computing with Linux*, MIT Press, Cambridge, MA, 2002.
5. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, and K. Vahi. Mapping abstract complex workflows onto grid environments, *Journal of Grid Computing*, **1**: 25–39, 2003.
6. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, Scientific workflow management and the Kepler system, *Concurrency and Computation: Practice and Experience*, **18**(10):1039–1065, 2006.
7. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, in *Bioinformatics*, **20**(17): 3045–3054, 2004.
8. I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, 1999.
9. R. Buyya, *Economic-Based Distributed Resource Management and Scheduling for Grid Computing*, PhD Thesis, Monash University, Melbourne, Australia, April 2002.
10. R. Buyya, D. Abramson, and J. Giddy, An economy driven resource management architecture for global computational power grids, in *Proceedings of the 7th International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, June 26–29, 2000.

11. E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, The cost of doing science on the cloud: the montage example, in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, NJ, 2008, pp. 1−12.

12. S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, and K. Chiu, A grid workflow environment for brain imaging analysis on distributed systems, *Concurrency and Computation: Practice and Experience*, **21**(16):2118−2139, 2009.

13. S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya, Designing a resource broker for heterogeneous grids, *Software: Practice and Experience*, **38**(8): 793−825, 2008.

14. C. Vecchiola, X. Chu, and R. Buyya, Aneka: A software platform for .NET-based cloud computing, in *High Performance and Large Scale Computing*, IOS Press, Amsterdam, Netherlands, 2009.

15. J. Yu and R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, *Scientific Programming Journal*, **14** (3−4): 217−230, 2006.

16. Suraj Pandey, Linlin Wu, Siddeswara Guru, and Rajkumar Buyya, A Particle Swarm Optimization (PSO)-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments, Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, Australia, April 20−23, pp. 400−407, 2010. − Best Paper Award.

17. I. Raicu, Y. Zhao, I. Foster, and A. Szalay, Accelerating large-scale data exploration through data diffusion, in *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing*, ACM, 2008, pp. 9−18.

18. C. Vecchiola, S. Pandey, and R. Buyya, High-performance cloud computing: A view of scientific applications, in *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks* Kaohsiung, Taiwan, December 14−16, 2009.

19. Google AppEngine, http://code.google.com/appengine/ (November 2009).

20. Windows Azure Platform, http://www.microsoft.com/windowsazure/windowsazure/ (November 2009).

21. Standard ECMA-335, http://www.ecma-international.org/publications/standards/Ecma-335.htm (November 2009).

22. C. Jin and R. Buyya, MapReduce programming model for .NET-based cloud computing, in *Proceedings of the 15th International European Parallel Computing Conference*, Delft, The Netherlands, August 25−28, 2009.

23. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, Grid information services for distributed resource sharing, in *10th IEEE International Symposium on High Performance Distributed Computing*, Los Alamitos, CA, 7−9 August 2001.

24. R. Wolski, N. T. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing, *Future Generation Computer Systems*, **15**(5−6):757−768, 1999.

25. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, Bigtable: A distributed storage system for structured data, in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USENIX Association, 2006, p. 15.

26. S. Ghemawat, H. Gobioff, and S. T. Leung, The google file system, *SIGOPS Operating System Review*, **37**(5), 2003, 29−43.
27. C. Vecchiola, M. Kirley, and R. Buyya, Multi-objective problem solving with offspring on enterprise clouds, in *Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region* (HPC Asia 2009), Kaohsiung, Taiwan, March 2−5, 2009.
28. IBM TSpaces, http://www.almaden.ibm.com/cs/tspaces/ (November 2009).
29. Gartner's Hype Cycle for 2009. http://www.gartner.com/technology/research/hype-cycles/index.jsp (Accessed October, 2010).