CHAPTER 13

UNDERSTANDING SCIENTIFIC APPLICATIONS FOR CLOUD ENVIRONMENTS

SHANTENU JHA, DANIEL S. KATZ, ANDRE LUCKOW, ANDRE MERZKY, and KATERINA STAMOU

13.1 INTRODUCTION

Distributed systems and their specific incarnations have evolved significantly over the years. Most often, these evolutionary steps have been a consequence of external technology trends, such as the significant increase in network/bandwidth capabilities that have occurred. It can be argued that the single most important driver for cloud computing environments is the advance in virtualization technology that has taken place. But what implications does this advance, leading to today's cloud environments, have for scientific applications? The aim of this chapter is to explore how clouds can support scientific applications.

Before we can address this important issue, it is imperative to (a) provide a working model and definition of clouds and (b) understand how they differ from other computational platforms such as grids and clusters. At a high level, cloud computing is defined by Mell and Grance [1] as a model for enabling 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.

We view clouds not as a monolithic isolated platform but as part of a large distributed ecosystem. But are clouds a natural evolution of distributed systems, or are they a fundamental new paradigm? Prima facie, cloud concepts are derived from other systems, such as the implicit model of clusters as static

Cloud Computing: Principles and Paradigms, Edited by Rajkumar Buyya, James Broberg and Andrzej Goscinski

bounded sets of resources, which leads to batch-queue extensions to virtualization. Another example is provided by ideas prevalent in grids to address dynamic application requirements and resource capabilities, such as pilot jobs, that are being redesigned and modified for clouds. In either case, clouds are an outgrowth of the systems and ideas that have come before them, and we want to consciously consider our underlying assumptions, to make sure we are not blindly carrying over assumptions about previous types of parallel and distributed computing.

We believe that there is novelty in the resource management and capacity planning capabilities for clouds. Thanks to their ability to provide an illusion of unlimited and/or *immediately available* resources, as currently provisioned, clouds in conjunction with traditional HPC and HTC grids provide a balanced infrastructure supporting scale-out and scale-up, as well as capability (HPC) and quick turn-around (HTC) computing for a range of application (model) sizes and requirements. The novelty in resource management and capacity planning capabilities is likely to influence changes in the usage mode, as well deployment and execution management/planning. The ability to exploit these attributes could lead to applications with new and interesting usage modes and dynamic execution on clouds and therefore new application capabilities. Additionally, clouds are suitable infrastructure for dynamic applications—that is, those with execution time resource requirements that cannot be determined exactly in advance, either due to changes in runtime requirements or due to interesting changes in application structure (e.g., different solver with different resource requirement).

Clouds will have a broad impact on legacy scientific applications, because we anticipate that many existing legacy applications will adapt to and take advantage of new capabilities. However, it is unclear if clouds as currently presented are likely to change (many of) the fundamental reformulation of the development of scientific applications. In this chapter, we will thus focus on scientific applications that can benefit from a dynamic execution model that we believe can be facilitated by clouds. Not surprisingly, and in common with many distributed applications, coarse-grained or task-level parallelism is going to be the basis of many programming models aimed at data-intensive science executing in cloud environments. However, even for common programming approaches such as MapReduce (based on task-level parallelism), the ability to incorporate dynamic resource placement and management as well as dynamic datasets is an important requirement with concomitant performance advantages. For example, the Map and Reduce phases involve different computations, thus different loads and resources; dynamical formulations of applications are better suited to supporting such load-balancing. Clouds are thus emerging as an important class of distributed computational resource, for both data-intensive and compute-intensive applications.

There are novel usage modes that can be supported when grids and clouds are used concurrently. For example, the usage of clouds as the computational equivalent of a heat bath establishes determinism—that is, well-bounded time-to-completion with concomitant advantages that will accrue as a consequence.

But to support such advanced usage modes, there is a requirement for programming systems, models, and abstractions that enable application developers to express decompositions and which support dynamic execution. Many early cloud applications employ ad hoc solutions, which results in a lack of generality and the inability of programs to be extensible and independent of infrastructure details. The IDEAS design objectives—**I**nteroperability, **D**istributed scale-out, **E**xtensibility, **A**daptivity, and **S**implicity—summarize the design goals for distributed applications. In this chapter we demonstrate several examples of how these objectives can be accomplished using several cloud applications that use SAGA.

13.1.1 Fundamental Issues

In this chapter, we want to consider a set of fundamental questions about scientific applications on clouds, such as: What kind of scientific applications are suitable for clouds? Are there assumptions that were made in developing applications for grids that should consciously be thrown out, when developing applications for clouds? In other words, from an application's perspective, how is a cloud different from a traditional grid? What kind of scientific applications can utilize both clouds and grids, and under what conditions? The issue of how applications and environments are developed is a chicken-and-egg situation. One might ask which applications are suitable for a given environment. Similarly, one might ask which environment can support a given application. Applications are developed to run in specific environments, while environments are developed to run specific applications. This coupling is a Zen-like paradox.

Clouds as a Type of Distributed Infrastructure. Before we can analyze if there is a fundamentally different class of applications that can be supported on cloud systems, it is imperative to ask, What is the difference between clouds and other distributed infrastructure?

To structure the differences between grid and cloud applications, if any, let us use the three phases of an applications life cycle: (i) development, (ii) deployment, and (iii) execution [2]. In development, if we think of the three vectors (execution unit, communication, and coordination) aiding our analysis, then neither resource management or scheduling influence the above three vector values. In deployment, clouds can be clearly differentiated from clusters and grids. Specifically, the runtime environment [as defined by the virtual machine (VM)] is controlled by the user/application and can be set up as such; this is in contrast to traditional computational environments. By providing simplicity and ease of management, it is hoped that the changes at the execution level may feed back to the application development level.

Some uncertainty lies in the fact that there are some things we understand, while there are some things that are dependent on evolving technologies and are thus unclear. For example, at the execution level, clouds differ from clusters/

grids in at least a couple of different ways. In cloud environments, user-level jobs are not typically exposed to a scheduling system; a user-level job consists of requesting the instantiation of a VM. Virtual machines are either assigned to the user or not (this is an important attribute that provides the illusion of infinite resources). The assignment of a job to a VM must be done by the user (or a middleware layer). In contrast, user-level jobs on grids and clusters are exposed to a scheduling system and are assigned to execute at a later stage. Also a description of a grid/cluster job typically contains an explicit workload description. In contrast, for clouds, a user-level job typically contains the container (a description of the resource requested) but does not necessarily contain the workload itself. In other words, the physical resources are not provisioned to the workload but are provisioned to the container. This model is quite similar to resource reservations where one can obtain a "container" of resources to which jobs can be later be bound. Interestingly, at this level of formulation, pilot jobs can be considered to provide a model of resource provisioning similar to the one that clouds natively provide.

An additional issue is compositional and deployment flexibility. A number of applications are difficult to build, due to runtime dependencies or complicated nonportable build systems. There is often a need to control the runtime environment at a fine-grained level, which is often difficult with grids; this often provides a rationale for using cloud environments. Clouds offer an opportunity to build virtual machines once, then to load them on various systems, working around issues related to portability on the physical systems, because the VM images can be static, while real systems (both hardware and software) are often changing.

A third issue is scheduling flexibility. Clouds offer the ability to create usage modes for applications to support the situation where, when the set of resources needed to run an application changes (perhaps rapidly), the resources can actually be changed (new resources can be added, or existing resources can be removed from the pool used by the job).

Scientific Cloud Applications as Distributed Applications. We have previously [2] introduced the concept of *Distributed Application Vectors* to structure the analysis and understanding of the main characteristics with a view to understanding the primary design requirements and constraints. Specifically, we determined that understanding the execution units, communication requirements, coordination mechanisms, and execution environment of a distributed application was a necessary (minimally complete) set of requirements. We will argue that both the vectors and the abstractions (patterns) for cloud-based applications are essentially the same as those for grid-based applications, further lending credibility to the claim that cloud-based applications are of the broader distributed applications class.

Most applications have been modified to utilize clouds. Usually, the modifications have not been at the application level, but more at the point at which the application uses the infrastructure. It appears that there is not a

major distinction between a classic grid application or a scientific cloud application; they are both incarnations of distributed applications—with the same development concerns and requirements, but with different deployment and execution contexts. In other words: Cloud applications are essentially a type of distributed applications, but with different infrastructure usage than grid applications. Due to a better control on the software environment, there is the ability to do some things better on clouds; thus, some types of applications are better suited/adapted to clouds. Programming models, such as MapReduce, that support data-intensive applications are not exclusively cloud-based, but due to the programming systems and tools as well as other elements of the ecosystem, they are likely to find increased utilization. Thus, at this level, there are no fundamental new development paradigms for cloud-based applications *a priori*.

We also formally characterized [2] patterns that can be used to capture aspects of distributed coordination, communication, and execution. Specifically, we identified three important elements ("vectors") influencing the overall development of distributed applications, coordination, communication, and execution and showed how these and data access patterns can be associated with a primary distributed application concern (reproduced and extended in Table 13.1).

We will discuss how using cloud capabilities will enable applications to exploit new scenarios, for example, the dynamic adjustment of application parameters (such as the accuracy) or the dynamic addition of new resources to an application. In order to motivate and structure these applications and their usage modes, we will provide a brief overview of a classification of scientific cloud applications in the next section. We will then discuss SAGA, which is an API for distributed applications as a viable programming system for clouds. We establish this with three distinct applications that have been developed for clouds using SAGA, further bolstering the connection between cloud applications and distributed applications. We end this chapter with a discussion of issues of relevance to scientific applications on clouds—including design objectives, interoperability with grids, and application performance considerations.

TABLE 13.1. A Classification of Some Commonly Occurring Patterns in Distributed Computing.[a]

Coordination	Communication	Deployment	Data Access
Client-server	Pub-sub	Replication	Co-access
P2P	Stream	At-home	One-to-one
Master-worker (TF, BoT)	Point-to-point	Brokering	One-to-many
Consensus	Broadcast	Co-allocation	Scatter-gather
Data processing pipeline			All-to-all

[a] The patterns are placed into a category that represents the predominant context in which they appear and address; this is not to imply that each pattern addresses only one issue exclusively. *Source*: Adapted from Jha et al. [2].

13.2 A CLASSIFICATION OF SCIENTIFIC APPLICATIONS AND SERVICES IN THE CLOUD

Common models of clouds [1,3,4] introduce composite hierarchies of different layers, each implementing a different service model (see Figure 13.1). The services of each layer can be composed from the services of the layer underneath, and each layer may include one or more services that share the same or equivalent levels of abstraction. The proposed layers consist of the following: the Software as a Service (SaaS) layer, the platform as a service (PaaS) layer, and the Infrastructure as a Service (IaaS) layer. The IaaS layer can be further divided into the computational resources, storage, and communications sublayers, the software kernel layer, and the hardware/firmware layer that consists of the actual physical system components. As shown in Figure 13.1, clouds can also be classified according to their deployment model into public and private clouds. A public cloud is generally available on pay-per-use basis. Several infrastructures have emerged that enable the creation of so-called private clouds—that is, clouds that are only accessible from within an organization.

Based on the proposed service layers, we will derive a classification from the application's perspective, with our aim to provide suggestions and raise further discussions on how scientific applications could possibly foster in the cloud

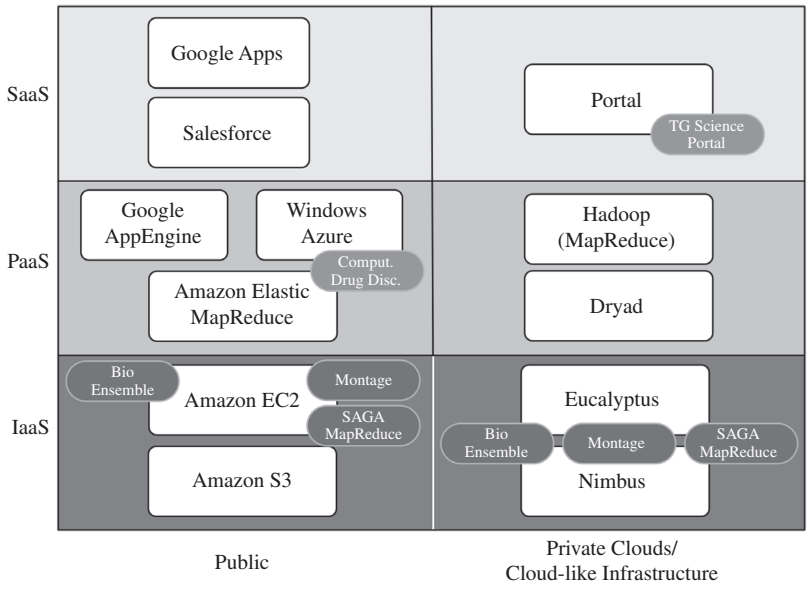

FIGURE 13.1. Cloud taxonomy and application examples: Clouds provide services at different levels (IaaS, PaaS, SaaS). The amount of control available to users and developers decreases with the level of abstraction. According to their deployment model, clouds can be categorized into public and private clouds.

environment. Although our taxonomy is targeted toward specific cloud environments, we strongly believe that a scientific application should and must remain interoperable regardless of the execution backend or the initial development infrastructure.

The identification of how cloud application services fit into the layers may allow software developers to better comprehend the nature of parameters introduced in each layer. Such an assumption could lead into easier and more efficient implementation of cloud-operable scientific applications. Research work from the traditional cluster/grid era systems has already determined important features like scalability, extensibility, and high availability that should play an integral role in a distributed application's core functionality.

Before we discuss scientific cloud applications in Section 13.3, here we will explain the details of the layers in the cloud model.

13.2.1 Software as a Service (SaaS) Layer

The software as a service layer is the highest layer in the proposed model. SaaS provides ready-to-run services that are deployed and configured for the user. In general, the user has no control over the underlying cloud infrastructure with the exception of limited configuration settings. Regarding scientific applications, such a layer may represent an access point for the end user to reach a service, like a portal or a visualization tool. Scientific portals have been used by many grid services.

A strong characteristic of SaaS services is that there is no client side software requirement. All data manipulated in such systems are held in remote infrastructures where all the processing takes place. One of the most prominent advantages of applications that are presented in this layer is universal accessibility regardless of the client system's software availability. This scheme provides flexibility to the end user and transparency of any complex mechanism involved. Some widely used examples of services that belong to this category are Google Apps and Salesforce. A prominent example from the science community is the TeraGrid Science Gateways [5].

These gateways provide among other things several domain specific web portals, which can be used to access computational and data services.

13.2.2 Platform as a Service (PaaS) Layer

The Platform as a Service (PaaS) layer provides the capability to deploy custom applications on the cloud providers infrastructure. These applications are developed using the programming languages and APIs defined by the cloud provider. Similar to SaaS, the user has only limited control over the underlying cloud infrastructures: He can deploy and configure applications created using the vendor's programming environment. The process of implementing and deploying a cloud application becomes more accessible while allowing the programmer to focus on important issues like the formulation of the scientific

algorithm. A developer does not have to worry about complex programming details, scalability, load balancing, or other system issues that may hinder the overall process of building an application. All such criteria are already specified by the given API that abstracts underlying architectural parameters.

A well-known PaaS example is the Google App Engine [6] that equips developers with a Python and Java API and runtime environment for the implementation of web applications. Windows Azure [7] is Microsoft's PaaS platform and offers different types of runtime environments and storage services for applications. While, in particular, Google App Engine is primarily geared toward Web applications (such as science portals), Windows Azure is also well-suited for compute- and data-intensive applications. Watson et al. [8] use Windows Azure—in particular the data storage and VM execution environment—to conduct data mining for computational drug discovery.

Another PaaS abstraction that is used for parallel processing of large amounts of data is MapReduce (MR) [9]. The framework solely requires the user to define two functions: the *map* and the *reduce* function. Both functions operate on key/value pairs: The map function transforms an input key/value pair representing a data row to an output key/value pair; the reduce function is used to merge all outputs of the map functions. Generally, the MapReduce framework handles all complexities and orchestrates the distribution of the the data as well as of the map and reduce tasks. Hadoop [10] is a well-known example of an open-source MapReduce framework. Amazon's Elastic MapReduce [11] provides a hosted MapReduce service.

Another example of an environment for data-intensive computing is Microsoft Dryad [12]. The framework allows the programmer to efficiently use resources for running data parallel applications. In Dryad a computation has the form of a directed graph (DAG), where the program instances that compose the computation are represented as graph vertices and the one-way communication channels between the instances are represented as graph edges. The Dryad infrastructure includes computational frameworks like Google's MapReduce. A port of Dryad to Windows Azure is planned, but at the time of writing is not available.

PaaS clouds provider higher-level abstractions for cloud applications, which usually simplifies the application development process and removes the need to manage the underlying software and hardware infrastructure. PaaS offers automatic scalability, load balancing, and failure tolerance. However, the benefits are also associated with some drawbacks: Generally, PaaS services usually provide highly proprietary environments with only limited standard support. App Engine, for example, supports parts of the Java Enterprise API, but uses a custom BigTable-based [13] data store.

13.2.3 Infrastructure-as-a-Service Layer

The infrastructure-as-a-service (Iaas) layer provides low-level, virtualized resources, such as storage, networks, and other fundamental computing

resources via self-services to the user. In general, the user can deploy and run arbitrary software, which usually includes operating systems as well as applications. However, the user has no knowledge of the exact location and specifics of the underlying physical resources. Cloud providers usually offer instant elasticity; that is, new resources can be rapidly and elastically provisioned to scale-up or scale-out applications dynamically.

Computational cloud resources are represented through virtual machine instances (VMs), where the user is usually granted full administrative access and has the ability to build and deploy any kind of service infrastructure. Such VMs usually come with an OS already installed. The developer may choose a VM to rent that has the OS she wants. Amazon EC2 [14] is the prime example of such a service and currently offers a variety of VM images, where one may choose to work on a Windows platform or on some Linux-based platforms. The developer can further configure and add extra libraries to the selected OS to accommodate an application. Rackspace [15] and GoGrid [16] provide similar services. Eucalyptus [17] and Nimbus [18] offer EC2 compatible infrastructures, which can be deployed in-house in a private cloud. Several scientific clouds utilize these frameworks—for example, Science Cloud [19] and Future Grid [20].

VMs are provided to the user under SLAs, where the cloud provider guarantees a certain level of system's performance to their clients. They usually involve fees on behalf of the user utilizing the leased computational resources, while open source/research cloud infrastructures don't include any financial requirement. When a team of scientists rents some virtual resources to run their experiments, they usually also lease data storage to store their data/results remotely and access them within the time limits of their agreement with the service provider. Examples of public cloud storage service are Amazon S3 [21] and Rackspace Cloud Files [22]. Walrus [23] is a S3 interface compatible service, which can be deployed on private cloud infrastructures. Another common cloud-like infrastructure is distributed file systems, such as the Google File System (GFS) [24] and the Hadoop File System (HDFS) [25]. Both systems are optimized for storing and retrieving large amounts of data.

13.2.4 Discussion of Cloud Models

Several scientific applications from different domains (e.g., life sciences, high-energy physics, astrophysics, computational chemistry) have been ported to cloud environments (see references 26–28 for examples). The majority of these applications rely on IaaS cloud services and solely utilize static execution modes: A scientist leases some virtual resources in order to deploy their testing services. One may select different number of instances to run their tests on. An instance of a VM is perceived as a node or a processing unit. There can be a multiple number of instances under the same VM, depending on the SLA one has agreed on. Once the service is deployed, a scientist can begin testing on the virtual nodes; this is similar to how one would use a traditional set of local clusters.

Furthermore, most of this research has solely attempted to manually customize legacy scientific applications in order to accommodate them into a cloud infrastructure. Benchmark tests on both EC2 virtual instances and conventional computational clusters indicated no significant difference in the results with respect to total running time (wall clock) and number of processors used. So far, there hasn't been much discussions on implementing scientific applications targeted to a cloud infrastructure. Such first-principle applications require programatic access to cloud capabilities as dynamic provisioning in an infrastructure-independent way to support dynamic execution modes.

In summary, clouds provide services at different levels (IaaS, PaaS, SaaS). In general, the amount of control available to users and developers decreases with the level of abstraction. Only IaaS provides sufficient programmatic control to express decompositions and dynamic execution modes, which seems central to many scientific applications.

13.3 SAGA-BASED SCIENTIFIC APPLICATIONS THAT UTILIZE CLOUDS

In this chapter we take the scope of "cloud applications" to be those distributed applications that are able to explicitly benefit from the cloud's inherent elasticity—where elasticity is a kind of dynamic execution mode—and from the usage modes provided by clouds. This excludes those applications that are trivially mapped to a small static set of small resources, which can of course be provided by clouds but do not really capture the predominant advantages and features of clouds.

Earlier work of the chapter authors [28] has shown that the Simple API for Grid Applications (SAGA) [29] provides a means to implement first-principle distributed applications. Both the SAGA standard [30] and the various SAGA implementations [31,32] ultimately strive to provide higher-level programming abstractions to developers, while at the same time shielding them from the heterogeneity and dynamics of the underlying infrastructure. The low-level decomposition of distributed applications can thus be expressed via the relatively high-level SAGA API.

SAGA has been used to develop scientific applications that can utilize an ever-increasing set of infrastructure, ranging from vanilla clouds such as EC2, to "open source" clouds based upon Eucalyptus, to regular HPC and HTC grids, as well to a proposed set of emerging "special-purpose" clouds. SAGA has also been used in conjunction with multiple VM management systems such as OpenNebula (work in progress) and Condor (established). In those cases where the application decomposition properties can be well-mapped to the respective underlying cloud and its usage usage modes (as discussed before), the resulting applications are fit to utilize cloud environments. In other words, if clouds can be defined as elastic distributed systems that support specific usage modes, then it seems viable to expect explicit application level support for those

usage modes, in order to allow applications to express that usage mode in the first place.

If we now consider the variety of scientific applications (see reference 2), it seems clear that (i) no single usage mode will be able to accommodate them all and (ii) no single programming abstraction will be able to cover their full scope. Instead, we see a continuum of requirements and solutions that try to map the application structure to the specific distributed runtime environment. This is exactly where SAGA tries to contribute: It provides a *framework* for implementing higher-level programming abstractions (where it does not provide those abstractions itself), each expressing or demanding a certain usage mode. The SAGA layer allows to abstract the specific way in which that usage mode is provided—either implicitly by adding additional structure to the distributed environment, or explicitly by exploiting support for that usage mode, for example, the elasticity in a specific cloud.

This section will discuss several SAGA-based scientific cloud applications, but we assert that the discussion holds just as well for applications that express their decomposition in other ways programatically. We do not claim that SAGA is the ultimate approach to develop cloud applications, but given our experience so far, it at least seems to be a viable approach that allows applications to directly benefit from the features that clouds, as specific distributed environments, provide: (a) support for specific usage modes and (b) elasticity of resources. Below we will present a number of examples that illustrate and verify that approach.

13.3.1 MapReduce

As discussed in Section 13.2, MapReduce (MR) is a prominent example for a PaaS: The MR framework allows users to (a) define their own specific map and reduce algorithms and (b) utilize the respective PaaS infrastructure with its MR supporting usage modes (elasticity, communication, etc.). With the emergence of the currently observed broad spectrum of cloud infrastructures, it became, however, necessary to implement the MR framework for each of them. Furthermore, MR has traditionally not been heavily used by the scientific computation community, so that efficient implementations on the "legacy" grid and cluster platforms have been largely missing, which raises the barrier for adoption of MR for scientific applications.

The SAGA MapReduce [33] provides a MR development and runtime environment that is implemented using the SAGA. The main advantage of a SAGA-based approach is that it is infrastructure-independent while still providing a maximum of control over the deployment, distribution, and runtime decomposition. In particular, the ability to control the distribution and placement of the computation units (workers) is critical in order to implement the ability to move computational work to the data. This is required to keep data network transfer low and, in the case of commercial clouds, the monetary cost of computing the solution low.

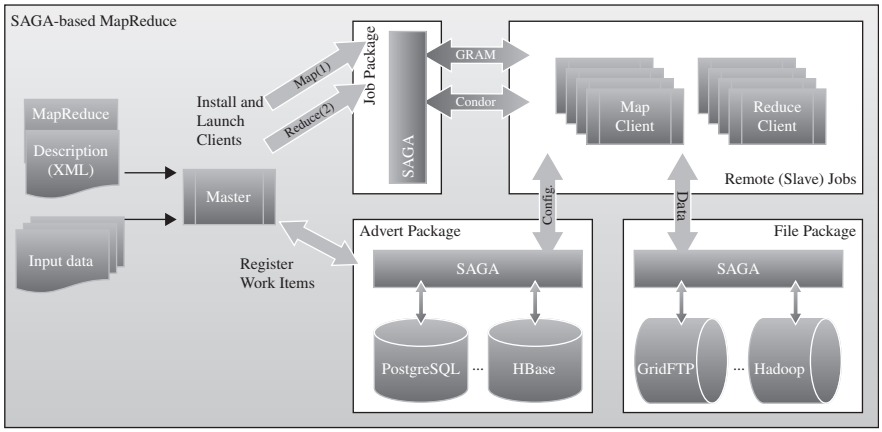

FIGURE 13.2. SAGA MapReduce framework. A master-worker paradigm is used to implement the MapReduce pattern. The diagram shows several different infrastructure options that can be utilized by the application.

Figure 13.2 show the architecture of the SAGA MR framework. Several SAGA adaptors have been developed to utilize SAGA MapReduce seamlessly on different grid and cloud infrastructures [28]. For this purpose, adaptors for the SAGA job and file package are provided. The SAGA job API is used to orchestrate mapping and reduction tasks, while the file API is utilized to access data. In addition to the local adaptors for testing, we use the Globus adaptors for grids and the AWS adaptors for cloud environments. Furthermore, we provide various adaptors for cloud-like infrastructure, such as different open-source distributed file systems (e.g., HDFS [25] and CloudStore [34]), and key/value stores (e.g., HBase [35]).

Tables 13.2 and 13.3 show some selected performance data for SAGA MapReduce; further details can be found in references 28 and 33. These tests established interoperability across a range of distinct infrastructure concurrently. Ongoing work is currently adding dynamic resource placement and job management to the framework, and it is also experimenting with automated data/compute colocation. The SAGA-based MapReduce implementation has shown to be easily applicable to sequence search applications, which in turn can make excellent use of the MapReduce algorithm and of a variety of middleware backends.

13.3.2 SAGA Montage

Montage [36, 37], an astronomical image mosaicking application that is also one of the most commonly studied workflow applications, has also been studied [38] with SAGA. Montage is designed to take multiple astronomical

TABLE 13.2. Performance Data for Different Configurations of Worker Placements[a]

Number of workers		Data	T_S	T_{Spawn}	$T_S - T_{\text{Spawn}}$
TeraGrid	AWS	Size (MB)	(sec)	(sec)	(sec)
4	—	10	8.8	6.8	2.0
—	1	10	4.3	2.8	1.5
—	2	10	7.8	5.3	2.5
—	3	10	8.7	7.7	1.0
—	4	10	13.0	10.3	2.7
—	4 (1)	10	11.3	8.6	2.7
—	4 (2)	10	11.6	9.5	2.1
—	2	100	7.9	5.3	2.6
—	4	100	12.4	9.2	3.2
—	10	100	29.0	25.1	3.9
—	4 (1)	100	16.2	8.7	7.5
—	4 (2)	100	12.3	8.5	3.8
—	6 (3)	100	18.7	13.5	5.2
—	8 (1)	100	31.1	18.3	12.8
—	8 (2)	100	27.9	19.8	8.1
—	8 (4)	100	27.4	19.9	7.5

[a]The master places the workers either on clouds or on the TeraGrid (TG). The configurations, separated by horizontal lines, are classified as either all workers on the TG or having all workers on EC2. For the latter, unless otherwise explicitly indicated by a number in parentheses, every worker is assigned to a unique VM. In the final set of rows, the number in parentheses indicates the number of VMs used. It is interesting to note the significant spawning times, and its dependence on the number of VM, which typically increase with the number of VMs. T_{Spawn} does not include instantiation of the VM.

TABLE 13.3. Performance Data for Different Configurations of Worker Placements on TG, Eucalyptus–Cloud, and EC2.[a]

Number of Workers			Size	T_S	T_{Spawn}	$T_S - T_{\text{Spawn}}$
TG	AWS	Eucalyptus	(MB)	(sec)	(sec)	(sec)
—	1	1	10	5.3	3.8	1.5
—	2	2	10	10.7	8.8	1.9
—	1	1	100	6.7	3.8	2.9
—	2	2	100	10.3	7.3	3.0
1	—	1	10	4.7	3.3	1.4
1	—	1	100	6.4	3.4	3.0
2	2	—	10	7.4	5.9	1.5
3	3	—	10	11.6	10.3	1.6
4	4	—	10	13.7	11.6	2.1
5	5	—	10	33.2	29.4	3.8
10	10	—	10	33.2	28.8	2.4

[a]The first set of data establishes cloud–cloud interoperability. The second set (rows 5–11) shows interoperability between grids and clouds (EC2). The experimental conditions and measurements are similar to those in Table 13.2.

images (from telescopes or other instruments) and stitch them together into a mosaic that appears to be from a single instrument.

Montage initially focused on being scientifically accurate and useful to astronomers, without being concerned about computational efficiency, and it is being used by many production science instruments and astronomy projects [39]. Montage was envisioned to be customizable, so that different astronomers could choose to use all, much, or some of the functionality, and so that they could add their own code if so desired. For this reason, Montage is a set of modules or tools, each an executable program, that can run on a single computer, a parallel system, or a distributed system. The first version of Montage used a script to run a series of these modules on a single processor, with some modules being executed multiple times on different data. A Montage run is a set of tasks, each having input and output data, and many of the tasks are the same executable run on different data, referred to as a stage.

Later Montage releases delivered two new execution modes, suitable for grid and also cloud environments [40], in addition to sequentially execution. First, each stage can be wrapped by an MPI executable that calls the tasks in that stage in a round-robin manner across the available processors. Second, the Montage workflow can be described as a directed acyclic graph (DAG), and this DAG can be executed on a grid. In the released version of Montage, this is done by mDAG, a Montage module that produces an abstract DAG (or A-DAG, where abstract means that no specific resources are assigned to execute the DAG), Pegasus [41, 42], which communicates with grid information systems and maps the abstract DAG to a concrete resource assignment, creating a concrete DAG (or C-DAG), and DAGMan [43], which executes C-DAG nodes on their internally specified resources.

The generality of Montage as a workflow application has led it to become an exemplar for those in the computer science workflow community, such as those working on: Pegasus, ASKALON [44], quality-of-service (QoS)-enabled GridFTP [45], SWIFT [46], SCALEA-G [47], VGrADS [48], and so on.

A lot of interesting work has been done around the accommodation of workflow and generally data-intensive applications into the cloud. Such applications have a large amount and number of data dependencies, which are usually represented using a DAG to define the sequence of those dependencies. Different approaches have been used to test how well a traditional application like Montage could fit in and utilize virtual resources without compromising any of its functionality or performance [49], including a SAGA-based workflow system, called "digedag," has been developed. This allows one to run Montage applications on a heterogeneous set of backends, with acceptable performance penalties [38]. Individual nodes of Montage workflows are usually sequential (i.e., nonparallel) computations, with moderate data input and output rates. Those nodes thus map very well to resources that are usually available in today's IaaS clouds, such as AWS/EC2 or Eucalyptus. SAGA-based Montage workflows can thus seamlessly scale out, and simultaneously span grid, cloud, and cluster environments. It must be noted that workflows with other

TABLE 13.4. Execution Measurements

#	Resources	Middleware	Walltime (sec)	Standard Deviation (sec)	Difference from Local (sec)
1	L	F	68.7	9.4	—
2	L	S	131.3	8.7	62.6
3	L	C	155.0	16.6	86.3
4	L	F, S	89.8	5.7	21.1
5	L	F, C	117.7	17.7	49.0
6	L	F, C	133.5	32.5	64.8
7	L	F, S, C	144.8	18.3	76.1
8	Q	S	491.6	50.6	422.9
9	E	A	354.2	23.3	285.5
10	E, Q	S, A	363.6	60.9	294.0
11	L, Q, E	F, S, A	409.6	60.9	340.9
12	L	D	168.8	5.3	100.1
13	P	D	309.7	41.5	241.0

Resources: L, local; P, Purdue; Q, Queen Bee; E, AWS/EC2
Middleware: F, FORK/SAGA; S, SSH/SAGA; A, AWS/SAGA; C, Condor/SAGA; D, Condor/DAGMan.

compute/data characteristics could not be mapped onto cloud resources prevalent today: The usage modes supported by AWS/EC2 and the like do not, at the moment, cover massive parallel applications, low-latency pipeline, and so on.

Table 13.4 gives the results (mean + standard deviation) for several SAGA Montage experiments. The AWS/EC2 times (#9, #10, #11) are cleared of the EC2 startup times—those are discussed in detail in reference 28. If multiple resources are specified, the individual DAG nodes are mapped to the respective resources in round-robin fashion. Note that the table also gives the times for the traditional DAGMan execution to a local and a remote Condor pool (#12, #13).

13.3.3 Ensemble of Biomolecular Simulations

Several classes of applications are well-suited for distributed environments. Probably the best-known and most powerful examples are those that involve an ensemble of decoupled tasks, such as simple parameter sweep applications [50]. In the following we investigate an ensemble of (parallel HPC) MD simulations. Ensemble-based approaches represent an important and promising attempt to overcome the general limitations of insufficient timescales, as well as specific limitations of inadequate conformational sampling arising from kinetic trappings. The fact that one single long-running simulation can be substituted for an ensemble of simulations makes these ideal candidates for distributed environments. This provides an important general motivation for researching

ways to support scale-out and thus enhance sampling and to thereby increase "effective" timescales studied.

The physical system we investigate is the HCV internal ribosome entry site and is recognized specifically by the small ribosomal subunit and eukaryotic initiation factor 3 (eIF3) before viral translation initiation. This makes it a good candidate for new drugs targeting HCV. The initial conformation of the RNA is taken from the NMR structure (PDB ID: 1PK7). By using multiple replicas, the aim is to enhance the sampling of the conformational flexibility of the molecule as well as the equilibrium energetics.

To efficiently execute the ensemble of batch jobs without the necessity to queue each individual job, the application utilizes the SAGA BigJob framework [51]. BigJob is a Pilot Job framework that provides the user a uniform abstraction to grids and clouds independent of any particular cloud or grid provider that can be instantiated dynamically. Pilot Jobs are an execution abstraction that have been used by many communities to increase the predictability and time-to-solution of such applications. Pilot Jobs have been used to (i) improve the utilization of resources, (ii) reduce the net wait time of a collection of tasks, (iii) facilitate bulk or high-throughput simulations where multiple jobs need to be submitted which would otherwise saturate the queuing system, and (iv) implement application-specific scheduling decisions and policy decisions.

As shown in Figure 13.3, BigJob currently provides an abstraction to grids, Condor pools, and clouds. Using the same API, applications can

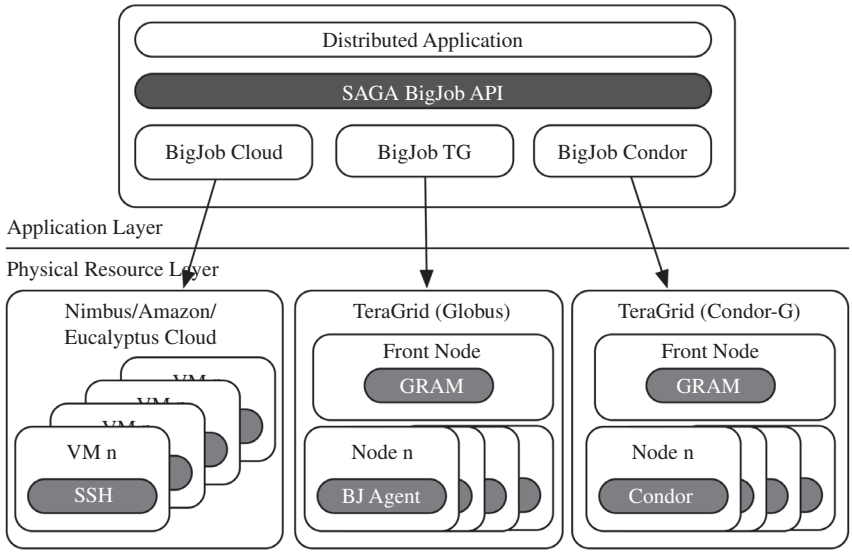

FIGURE 13.3. An overview of the SAGA-based Pilot Job: The SAGA Pilot-Job API is currently implemented by three different back-ends: one for grids, one for Condor, and one for clouds.

dynamically allocate resources via the big-job interface and bind sub-jobs to these resources.

In the following, we use an ensemble of MD simulations to investigate different BigJob usage modes and analyze the time-to-completion, T_C, in different scenarios.

Scenario A: T_C for Workload for Different Resource Configurations. In this scenario and as proof of scale-out capabilities, we use SAGA BigJob to run replicas across different types of infrastructures. At the beginning of the experiment a particular set of Pilot Jobs is started in each environment. Once a Pilot Job becomes active, the application assigns replicas to this job. We measure T_C for different resource configurations using a workload of eight replicas each running on eight cores. The following setups have been used:

Scenario A1: Resource I and III—Clouds and GT2-based grids.

Scenario A2: Resource II and III—Clouds and Condor grids.

Scenario A3: Resource I, II, and III—Clouds, GT2, and Condor grids.

For this experiment, the LONI clusters Poseidon and Oliver are used as grid and Condor resources, and Nimbus is used as a cloud resource.

Figure 13.4 shows the results. For the first three bars, only one infrastructure was used to complete the eight-replica workload. Running the whole scenario in the Science Cloud resulted in a quite poor but predictable performance; the standard deviation for this scenario is very low. The LONI resources are about three times faster than the Science Cloud, which corresponds to our earlier findings. The performance of the Condor and grid BigJob is similar, which can be expected since the underlying physical LONI resources are the same. Solely, a slightly higher startup overhead can be observed in the Condor runtimes.

In the next set of three experiments, multiple resources were used. For Scenario A1 (the fourth bar from left), two replicas were executed on the Science Cloud. The offloading of two replicas to an additional cloud resource resulted in a light improvement of T_C compared to using just LONI resources. Thus, the usage of cloud resources must be carefully considered since T_C is determined by the slowest resource, that is, Nimbus. As described earlier, the startup time for Nimbus images is, particularly for such short runs, significant. Also, NAMD performs significantly worse in the Nimbus cloud than on Poseidon or Oliver. Since the startup time on Nimbus averages to 357 sec and each eight-core replica runs for about 363 sec, at least 720 sec must be allowed for running a single replica on Nimbus. Thus, it can be concluded that if resources in the grids or Condor pool are instantly available, it is not reasonable to start additional cloud resources. However, it must be noted that there are virtual machines types with a better performance available—for example, in the Amazon cloud. These VMs are usually associated with higher costs (up to $2.40 per CPU hour) than the Science Cloud VMs. For a further

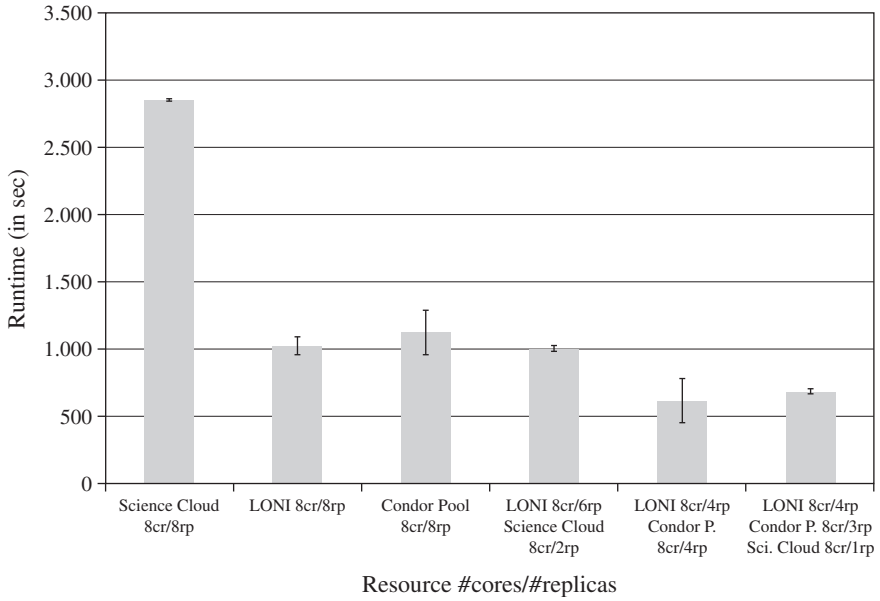

FIGURE 13.4. Collective usage of grid, Condor, and cloud resources for workload of eight replicas. The experiments showed that if the grid and Condor resource Poseidon has only a light load, no benefits for using additional cloud resources exist. However, the introduction of an additional Condor or grid resource significantly decreases T_C.

discussion of cost trade-offs for scientific computations in clouds, see Deelman et al. [52].

Scenario B:* T_C *for Workload for Different Resource Configurations. Given that clouds provide the illusion of infinite capacity, or at least queue wait-times are nonexistent, it is likely that when using multiple resource types and with loaded grids/clusters (e.g., TeraGrid is currently over-subscribed and typical queue wait-times often exceed 24 hours), most sub-jobs will end up on the cloud infrastructure. Thus, in Scenario B, the resource assignment algorithm we use is as follows: We submit tasks to non-cloud resources first and periodically monitor the progress of the tasks. If insufficient jobs have finished when time equal to T_X has elapsed (determined per criteria outlined below), then we move the workload to utilize clouds. The underlying basis is that clouds have an explicit cost associated with them; and if jobs can be completed on the TeraGrid/Condor-pool while preserving the performance constraints, we opt for such a solution. However, if queue loads prevent the performance requirements from being met, we move the jobs to a cloud resource, which we have shown has less fluctuation in T_C of the workload.

For this experiment we integrated a progress manager that implements the described algorithm into the replica application. The user has the possibility to

TABLE 13.5. Usage of Cloud Pilot Jobs to Ensure Deadline

Result	Number of Occurrences	Average T_C (minutes)
No VM started	6	7.8
1 VM started	1	36.4
2 VMs started	1	47.3
3 VMs started	2	44.2

specify a maximum runtime and a check interval. At the beginning of each check interval, the progress manager compares the number of jobs done with the total number of jobs and estimates the total number of jobs that can be completed within the requested timeframe. If the total number of jobs is higher than this estimate, the progress monitor instantiates another BigJob object request additional cloud resources for a single replica. In this scenario, each time an intermediate target is not met, four additional Nimbus VMs sufficient for running another eight core replica are instantiated. Table 13.5 summarizes the results.

In the investigated scenario, we configured a maximum runtime of 45 min and a progress check interval of 4 min. We repeated the same experiment 10 times at different times of the day. In 6 out of 10 cases the scenario was completed in about 8 minutes. However, the fluctuation in particular in the waiting time on typical grid resources can be very high. Thus, in four cases, it was necessary to start additional VMs to meet the application deadline. In two cases, three Pilot Jobs each with eight cores had to be started, and in one case a single Pilot Job was sufficient. In a single case the deadline was missed solely because not enough cloud resources were available; that is, we were only able to start two instead of three Pilot Jobs.

13.4 DISCUSSION

It is still unclear what the predominant usage mode of cloud infrastructures will be. As shown, there are a large number of applications that are able to utilize clouds, including both data-intensive applications (i.e., those that require data-compute affinity) and compute-intensive applications. While clouds can support different compute-intensive usage modes (e.g., distributed, tightly coupled and loosely coupled applications), tightly coupled applications are less well suited for clouds because current cloud infrastructures lack high-end, low-latency interconnects. Another interesting type of application includes programs that are able to utilize clouds in addition to traditional grids in a hybrid mode. Using dynamic and adaptive execution modes, the time-to-solution for many applications can be reduced and exceptional runtime situations (e.g., failures or scheduling delays) can be handled.

Developing and running applications on dynamic computational infrastructures such as clouds presents new and significant challenges. This includes the need for programming systems such as SAGA, which is able to express the different usage modes, associated runtime trade-offs, and adaptations. Other issues include: decomposing applications, components and workflows; determining and provisioning the appropriate mix of grid/cloud resources, and dynamically scheduling them across the hybrid execution environment while satisfying/balancing multiple possibly changing objectives for performance, resilience, budgets, and so on.

13.4.1 IDEAS Revisited

In computational science applications that utilize distributed infrastructure (such as computational grids and clouds), dealing with heterogeneity and scale of the underlying infrastructure remains a challenge. As shown in Table 13.6, SAGA and SAGA-based abstractions help to advance the IDEAS design objectives: **I**nteroperability, **D**istributed scale-out, **E**xtensibilty, **A**daptivity and **S**implicity:

- **Interoperability.** In all three examples, application-level interoperability is provided by the SAGA programming system. SAGA decouples applications from the underlying physical resources and provides infrastructure-independent control over the application deployment, decomposition, and runtime execution.
- **Distributed Scale-Out.** SAGA-based applications and frameworks, such as SAGA BigJob and Digedag, support the distributed scale-out of applications to multiple and possibly heterogeneous infrastructures—for example, different types of clouds and grids.
- **Extensibility.** The example clouds applications are extensible in several directions; new functionality and usage modes can simply be incorporated using SAGA. Additional distributed cloud and grid infrastructures can be included by configuration using a different middleware adaptor.

TABLE 13.6. Design objectives addressed by the different applications: Interoperability, infrastructure independence; Distributed Scale-Out, ability to use multiple distributed resources concurrently; Extensibility, extensibility and general purpose uptake; Adaptivity, ability to respond to changes; and Simplicity, greater simplicity without sacrificing functionality and performance.

Application	Interoperability	Distr. Scale-Out	Extensibility	Adaptivity	Simplicity
SAGA MapReduce	Y	Y	Y		Y
SAGA Montage	Y	Y	Y		Y
Biomolecular Ensemble	Y	Y	Y	Y	Y

- **Adaptivity.** Distributed applications that utilize SAGA are able to explicitly benefit from the cloud properties such as elasticity and to pursue dynamic execution modes. Examples of such usage mode include the usage of additional resources to meet a deadline or to meet an increased resource demand due to a certain runtime condition.
- **Simplicity.** SAGA provides a simple, high-level programming abstraction to express core distributed functionality. Simplicity arises from the fact that the API is very focused and reduced to the most essential functionalities.

13.4.2 Interoperability of Scientific Applications across Clouds and HPC/Grids

It is still unclear what kind of programming models and programming systems will emerge for clouds. It has been shown that traditional distributed applications can be easily ported to IaaS environments. The nature of applications as well as the provided system-level interfaces will play an important role for interoperability. While several technical infrastructure features, as well as economical policies, influence the design of programming models for the cloud era, it is important for effective scientific application development that any such system should not be constrained to a specific infrastructure—that is, it should support infrastructure interoperability at the application-level.

The SAGA programming system provides a standard interface and can support powerful programming models. SAGA allows application developers to implement common and basic distributed functionality, such as application decomposition, distributed job submission, and distributed file movement/management, independently of the underlying infrastructure. The SAGA cloud adaptors provide the foundation for accessing cloud storage and compute resource via the SAGA API. The ability to design and develop applications in an infrastructure-independent way leads to new kinds of application, such as dynamic applications. Such applications have dynamic runtime requirements and are able to adapt to changing runtime environments and resource availabilities. SAGA provides developers with new capability while introducing a new set of challenges and trade-offs. Application developers are, for example, able to utilize new execution modes in conjunction with "traditional" distributed applications but must, however, consider new trade-offs, for example, when selecting a resource.

The MapReduce programming model has exemplified a novel way to construct distributed applications for the cloud. It has been perceived as a programming pattern to lead the implementation of some future scientific applications. There has been a lot of testing on simple applications performing map and reduce computations on VMs as well as on traditional local clusters in order to first verify the scalability of performance that the proposed model successfully offers and then, most importantly, guarantee interoperability

between VMs and local clusters for a given application. As shown, SAGA MapReduce is able to run across different cloud and cloud-like back-end infrastructures.

As highlighted earlier, SAGA provides the basis for dynamic applications. Such applications greatly benefit from the ability of clouds to dynamically provision resources. The biomolecular ensemble application, for example, easily scales out to cloud and grid infrastructures and is able to utilize additional cloud resources to ensure the progress toward a deadline. Furthermore, SAGA enables applications and higher-level frameworks such as BigJob to deploy dynamic schedulers that determine the appropriate mix of cloud/grid resources and are able to adaptively respond to special runtime situations, such as faults.

Similarly, the development of workflow applications such as SAGA Montage can be both simple and efficient using the right tools. While SAGA Montage can easily be run across grid and clouds, the current version follows a traditional static execution model. In the future, the decision of where to run Montage components should be made at runtime, taking into account the current system and network utilization. Furthermore, capabilities, such as the ability to dynamically reschedule tasks, should be considered.

13.4.3 Application Performance Considerations

Undoubtedly, the most important characteristic for the establishment of a scientific application is its overall performance. There are proposals on including HPC tools and scientific libraries in EC2 AMIs and have them ready to run on request. This might lead to re-implementing some HPC tools and deploying public images on Amazon or other vendors specifically for scientific purposes (e.g., the SGI Cyclone Cloud [53]). Still, in order to include ready-to-use MPI clusters on EC2, there are several challenges to be met: The machine images must be manually prepared, which involves setting up the operating system, the application's software environment and the security credentials. However, this step is only initially required and comparable with moving an application to a new grid resource. Furthermore, the virtual machines must be started and managed by the application. As shown, several middleware frameworks, such as BigJob, are already able to utilize and manage cloud resources taking the burden off the application. Depending on the cloud infrastructure used, the spawning of VMs usually involves some overhead for resource allocation and for staging the VM to the target machine. At the end of the run, the results must be obtained and stored persistently, and the cluster must be terminated.

Another concern that scientists have to deal with in a cloud environment are different computational overheads as well as high and sometimes unpredictable communication latencies and limited bandwidths. For applications that are HPC applications, where the coupling of communication and computation is

relatively tight and where there is relatively frequent communication including global communication, clouds can be used, but with added performance overhead, at least on today's clouds. These overheads have various sources, some of which can be reduced. How much of this overhead must exist and will exist in the future is unclear.

There are two types of overhead: (i) added computational overhead of a VM and (ii) communication overhead when communicating between VMs. The first type of overhead results from the use of VMs and the fact that the underlying hardware is shared. While clouds nowadays deploy highly efficient virtualization solutions that impose very low overheads on applications (see reference 51), unanticipated load increases on the cloud providers infrastructure can affect the runtime of scientific applications. The communication overhead mainly results from the fact that most clouds do not use networking hardware that is as low-overhead as that of dedicated HPC systems. There are at least two routes to parallelism in VMs. The first is a single VM across multiple cores; the second is parallelism across VMs. The latter type is especially affected from these communication overheads; that is, tightly coupled workloads (e.g., MPI jobs) are likely to see a degraded performance if they run across multiple VMs.

Also, the common perception of clouds does not include the ability to co-locate different parts of a single application on a single physical cluster. Again, some of this network-related overhead can be reduced. At the time of writing this chapter, it is unclear to the authors if there is community consensus on what the performance of HPC applications on clouds is expected to be compared to bare-metal, whether the future model is that of a single VM over multiple-cores, or if there will be an aggregation of multiple VMs to form a single application, and thus importantly it is unclear what the current limitations on performance are. Additionally, there is also work in progress to develop pass-through communication and I/O, where the VM would not add overhead, though this is not yet mature.

13.5 CONCLUSIONS

As established earlier, both cloud and grid applications are incarnations of distributed applications. Applications require only small modifications to run on clouds, even if most of them only utilize "legacy" modes; that is, they usually run on a set of static resources [54]. Additionally, cloud applications are generally able to take advantage of existing abstractions and interfaces.

With the emergence of clouds and a general increase in the importance of data-intensive applications, programming models for data-intensive applications have gained significant attention; a prominent example is MapReduce. It is important to remember that these are not grid- or cloud-specific programming models; they can be used in either or both contexts. Most applications can in principle use either a grid or a cloud; whether they use a grid or a cloud is

dependent upon the level of control and decomposition that needs to be asserted and/or retained. Additional factors that determine this decision include the offering of the programming model, as well as a mapping to the capabilities of infrastructure that addresses the desired affinities, such as compute–communication and compute–data affinities [54].

The usability and effectiveness of a programming model is dependent upon the desired degree of control in the application development, deployment, and execution. To efficiently support coordinated execution across heterogeneous grid and cloud infrastructures, programming tools and systems are required. It is important to ensure that such programming systems and tools provide open interfaces and support the IDEAS design objectives. Furthermore, these tools must address the cloud's inherent elasticity and support applications with dynamic resource requirements and execution modes. Programming systems such as SAGA provide developers with ability to express application decompositions and coordinations via a simple, high-level API. Having established that cloud applications are conceptually akin to grid applications, we have shown, via several scientific applications, how SAGA has proven to be a programming system to develop applications that can utilize grids and clouds effectively.

REFERENCES

1. P. Mell and T. Grance, *The NIST definition of cloud computing*.
2. S. Jha et al., Programming Abstractions for Large-scale Distributed Applications, submitted to *ACM Computing Surveys*; draft at http://www.cct.lsu.edu/~sjha/publications/dpa_surveypaper.pdf.
3. L. Youseff, M. Butrico, and D. Da Silva, Toward a unified ontology of cloud computing, in *Proceedings of the Grid Computing Environments Workshop*, GCE '08, November 2008, 1–10.
4. M. Armbrust et al., Above the clouds: A Berkeley View of Cloud Computing, *Technical Report UCB/EECS-2009–28*, EECS Department, University of California, Berkeley, February 2009.
5. N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam, TeraGrid science gateways and their impact on science. *Computer*, **41**(11):32–41, 2008.
6. Google App Engine, http://code.google.com/appengine/.
7. Windows Azure, http://www.microsoft.com/windowsazure/.
8. P. Watson, D. Leahy, H. Hiden, S. Woodman, and J. Berry, *An Azure Science Cloud for Drug Discovery*, Microsoft External Research Symposium, 2009.
9. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, Berkeley, CA, USENIX Association, 2004, pp. 137–150.
10. Hadoop: Open Source Implementation of MapReduce, http://hadoop.apache.org/.
11. Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.

12. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks, *SIGOPS Operating System Review*, **41**(3):59–72, 2007.
13. F. Chang et al., Bigtable: A distributed storage system for structured data, in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association,
14. Amazon EC2 Web Service, http://ec2.amazonaws.com.
15. Rackspace Cloud, http://www.rackspacecloud.com/.
16. GoGrid Cloud Hosting, http://www.gogrid.com/.
17. Eucalyptus, http://open.eucalyptus.com/.
18. K. Keahey, I. Foster, T. Freeman, and X. Zhang, Virtual workspaces: Achieving quality of service and quality of life in the grid, *Scientific Programming*, **13** (4):265–275, 2005.
19. Science Cloud. http://scienceclouds.org/.
20. Future Grid. http://www.futuregrid.org/.
21. Amazon S3 Web Service. http://s3.amazonaws.com.
22. Rackspace Cloud Files. http://www.rackspacecloud.com/cloud_hosting_ products/files/.
23. Eucalyptus Walrus. http://open.eucalyptus.com/wiki/EucalyptusStorage_ v1.4.
24. S. Ghemawat, H. Gobioff, and S. Leung, The Google File System, *SIGOPS Operating System Reviews*, **37**(5):29–43, 2003.
25. HDFS. http://hadoop.apache.org/common/docs/current/hdfs_design. html.
26. C. Evangelinos and C. Hill, Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2, *Cloud Computing and Its Applications* (CCA-08), 2008.
27. M.-E. Bégin, Grids and Clouds—Evolution or Revolution, https://edms.cern. ch/file/925013/3/EGEE-Grid-Cloud.pdf, 2008.
28. A. Merzky, K. Stamou, and S. Jha, Application level interoperability between clouds and grids, in *Proceedings of the Grid and Pervasive Computing Conference*, May 2009, pp. 143–150.
29. T. Goodale et al., SAGA: A simple API for grid applications, high-level application programming on the grid, *Computational Methods in Science and Technology*, **12**(1):7–20, 2006.
30. T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith, A Simple API for Grid Applications (SAGA), *OGF Document Series 90*, http://www.ogf.org/documents/GFD.90.pdf.
31. H. Kaiser, A. Merzky, S. Hirmer, and G. Allen, The SAGA C++ Reference Implementation, in *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'06)—Library-Centric Software Design (LCSD'06)*, Portland, OR, USA, October 22–26 2006.
32. JSaga. http://grid.in2p3.fr/jsaga/index.html.
33. C. Miceli, M. Miceli, S. Jha, H. Kaiser, and A. Merzky, Programming abstractions for data intensive computing on clouds and grids, in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2009, pp. 478–483.

34. CloudStore, http://kosmosfs.sourceforge.net.
35. HBase, http://hadoop.apache.org/hbase/.
36. The Montage project, http://montage.ipac.caltech.edu/.
37. G. B. Berriman, J. C. Good, D. Curkendall, J. Jacob, D. S. Katz, T. A. Prince, and R. Williams. Montage: An on-demand image mosaic service for the NVO, Astronomical Data Analysis Software and Systems (ADASS) XII, 2002.
38. A Merzky, K Stamou, S Jha, and D Katz, A fresh perspective on developing and executing DAG-based distributed applications: A case-study of SAGA-based Montage, in *Proceedings of the IEEE Conference on eScience 2009*, Oxford.
39. G. B. Berriman, J. C. Good, A. C. Laity, J. C. Jacob, D. S. Katz, E. Deelman, G. Singh, M.-H. Su, R. Williams, and T. Prince, Science applications of Montage: An astronomical image mosaic engine, presented at IAU XXXVI General Assembly, 2006.
40. E. Deelman et al., The cost of doing science on the cloud: The Montage example, *Proceedings of SC08*, Austin, Texas, 2008.
41. 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**(1):25–39, 2003.
42. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, Pegasus: Mapping scientific workflows onto the grid, in *Proceedings of the Across Grids Conference*, 2004.
43. Condor DAGMAn, http://www.cs.wisc.edu/condor/dagman/.
44. M. Wieczorek, R. Prodan, and T. Fahringer, Scheduling of scientific workflows in the askalon grid environment, *ACM SIGMOD Record*, **34**(3):52–62, 2005.
45. M. Humphrey and S. Park, Data throttling for data-intensive workflows, in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 1–11, 2008.
46. Y. Zhao et al., Swift: Fast, reliable, loosely coupled parallel computation, *IEEE Congress on Services*, pages 199–206, 2007.
47. H. Truong, T. Fahringer, and S. Dustda, Dynamic instrumentation, performance monitoring and analysis of grid scientific workflows, *Journal of Grid Computing*, **2005**(3):1–18, 2005.
48. VGRaDS: Montage, a project providing a portable, compute-intensive service delivering custom mosaics on demand.
49. G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling, Scientific workflow applications on Amazon EC2, in *Proceedings of the Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science 2009)*, 2009.
50. H. Casanova, G. Obertelli, F. Berman, and R. Wolski, The apples parameter sweep template: User-level middleware for the grid, *Scientific Programming*, **8** (3):111–126, 2000.
51. A. Luckow, L. Lacinski, and S. Jha. Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems, in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010.

52. 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*, IEEE Press, New York, 2008, pp. 1–12.

53. SGI Cyclone–HPC Cloud, http://www.sgi.com/products/hpc_cloud/cyclone/index.html.

54. S. Jha, A. Merzky, and G. Fox, Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes, *Concurrency and Computation: Practice & Experience*, **21**(8):1087–1108, 2009.