Chapter 1

Understanding the Cloud Computing Landscape

Lamia Youseff, Dilma M. Da Silva,
Maria Butrico, and Jonathan Appavoo

Contents

1.1 Introduction

The goal of this chapter is to present an overview of three different structured views of the cloud computing landscape. These three views are the *SPI* cloud classification, the *UCSB-IBM* cloud ontology, and *Hoff's* cloud model. Each one of these three cloud models strives to present a comprehension of the interdependency between the different cloud systems as well as to show their potential and limitations. Furthermore, these models vary in the degree of simplicity and comprehensiveness in describing the cloud computing landscape. We find that these models are complementary and that by studying the three structured views, we get a general overview of the landscape of this evolving computing field.

1.2 Cloud Systems Classifications

The three cloud classification models present different levels of details of the cloud computing landscape, since they emerged in different times of evolution of this computing field. Although they have different objectives—some are for academic understanding of the novel research area, while others target identifying and analyzing commercial and market opportunities—they collectively expedite comprehending some of the interrelations between cloud computing systems. Although we present them in this chapter in a chronological order of their emergence—which also happens to reflect the degree of details of each model—this order does not reflect the relative importance or acceptance of one model over the other. On the other hand, the three models and their extensions are complementary, reflecting different views of the cloud. We first present the *SPI* model in Section 1.2, which is the oldest of the three models. The second classification is the *UCSB-IBM* ontology, which we detail in Section 1.3. We also present a discussion of a recent extension to this ontology in Section 1.4. The third classification is *Hoff*'s cloud model, which we present in Section 1.5. We discuss the importance of these classifications and their potential impact on this emerging computing field in Section 1.6.

1.3 SPI Cloud Classification

As the area of cloud computing was emerging, the systems developed for the cloud were quickly stratified into three main subsets of systems: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Early on, these three subsets of the cloud were discussed by several cloud computing experts, such as in [24,30,31]. Based on this general classification of cloud systems, the SPI model was formed and denotes the Software, Platform, and Infrastructure systems of the cloud, respectively.

1.3.1 Cloud Software Systems

This subset of cloud systems represents applications built for and deployed for the cloud on the Internet, which are commonly referred to as Software as a Service (SaaS). The target user of this subset of systems is the end user. These applications, which we shall refer to as cloud applications, are normally browser based with predefined functionality and scope, and they are accessed, sometimes, for a fee per a particular usage metric predefined by the cloud SaaS provider. Some examples of SaaS are salesforce customer relationships management (CRM) system [33], and Google Apps [20] like Google Docs and Google SpreadSheets.

SaaS is considered by end users to be an attractive alternative to desktop applications for several reasons. For example, having the application deployed at the provider's data center lessens the hardware and maintenance requirements on the users' side. Moreover, it simplifies the software maintenance process, as it enables the software developers to apply subsequent frequent upgrades and fixes to their applications as they retain access to their software service deployed at the provider's data center.

1.3.2 Cloud Platform Systems

The second subset of this classification features the cloud platform systems. In this class of systems, denoted as Platform as a Service (PaaS), the provider supplies a platform of software environments and application programming interfaces (APIs) that can be utilized in developing cloud applications. Naturally, the users of this class of systems are developers who use specific APIs to build, test, deploy, and tune their applications on the cloud platform. One example of systems in this category is Google's App Engine [19], which provides Python and Java runtime environments and APIs for applications to interact with Google's runtime environment. Arguably, Microsoft Azure [26] can also be considered a platform service that provides an API and allows developers to run their application in the Microsoft Azure environment.

Developing an application for a cloud platform is analogous to some extent to developing a web application for the old web servers model, in the sense that developers write codes and deploy them in a remote server. For end users, the final result is a browser-based application. However, the PaaS model is different in that it can provide additional services to simplify application development, deployment, and execution, such as automatic scalability, monitoring, and load balancing. Furthermore, the application developers can integrate other services provided by the PaaS system to their application, such as authentication services, e-mail services, and user interface components. All that is provided through a set of APIs is supplied by the platform. As a result, the PaaS class is generally regarded to accelerate the software development and deployment time. In turn, the cloud software built for the cloud platform normally has a shorter time-to-market. Some academic projects have also emerged to support a more thorough understanding of PaaS, such as AppScale [5].

Another feature that typifies PaaS services is the provision of APIs for metering and billing information. Metering and billing permits application developers to more readily develop a consumption-based business model around their application. Such a support helps integrate and enforce the relationships between end users, developers, PaaS, and any lower-level providers, while enabling the economic value of the developers and providers.

1.3.3 Cloud Infrastructure Systems

The third class of systems, according to the SPI classification model, provides infrastructure resources, such as compute, storage, and communication services, in a flexible manner. These systems are denoted as Infrastructure as a Service (IaaS). Amazon's Elastic Compute Cloud (EC2 [8]) and Enomalism elastic computing infrastructure [10] are arguably the two most popular examples of commercial systems available in this cloud category.

Recent advances in operating system (OS) Virtualization have facilitated the implementation of IaaS and made it plausible on existing hardware. In this regard, OS Virtualization technology enables a level of indirection with respect to direct hardware usage. It allows direct computer usage to be encapsulated and isolated in the container of a virtual machine (VM) instance. As a result, OS Virtualization enables all software and associated resource usage of an individual hardware user to be treated as a schedulable entity that is agnostic to the underlying physical resources that it is scheduled to use. Therefore, OS Virtualization allows IaaS providers to control and manage efficient utilization of the physical resources by enabling the exploitation of both time division and statistical multiplexing, while maintaining the familiar and flexible interface of individual standard hardware computers and networks for the construction of services using existing practices and software. This approach is particularly attractive to IaaS providers given the underutilization of the energy-hungry, high-speed processors that constitute the infrastructure of data centers. Amazon's infrastructure service, EC2, is one example of IaaS systems, where users can rent computing power on their infrastructure by the hour. In this space, there are also several academic open-source cloud projects, such as Eucalyptus [14] and Virtual Workspaces [38].

1.4 UCSB-IBM Cloud Ontology

The UCSB-IBM cloud ontology emerged through a collaboration effort between academia (University of California, Santa Barbara) and industry (IBM T.J. Watson Research Center) in an attempt to understand the cloud computing landscape. The end goal of this effort was to facilitate the exploration of the cloud computing area as well as to advance the educational efforts in teaching and adopting the cloud computing area.

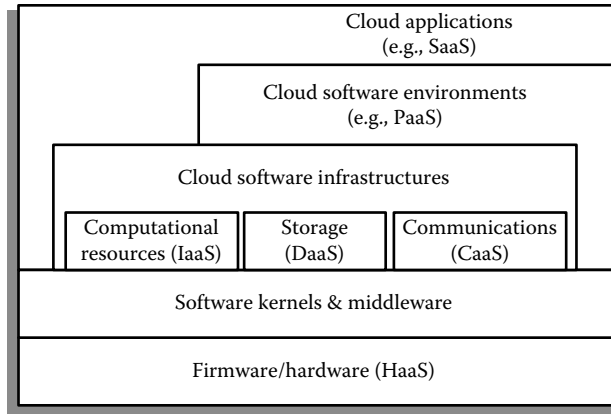

Figure 1.1 UCSB-IBM Cloud Computing Classification Model depicted as five layers, with three constituents to the cloud infrastructure layer.

In this classification, the authors used the principle of composability from a Service-Oriented Architecture (SOA) to classify the different layers of the cloud. Composability in SOA is the ability to coordinate and assemble a collection of services to form composite services. In this sense, cloud services can also be composed of one or more of other cloud services.

By the principle of composability, the UCSB-IBM model classified the cloud in five layers. Each layer encompasses one or more cloud services. Cloud services belong to the same layer if they have an equivalent level of abstraction, as evident by their targeted users. For example, all cloud software environments (also known cloud platforms) target programmers, while cloud applications target end users. Therefore, cloud software environments would be classified in a different layer than cloud applications. In the UCSB-IBM model, the five layers compose a cloud stack, where one cloud layer is considered higher in the cloud stack if the services it provides can be composed from the services that belong to the underlying layer. The UCSB-IBM cloud model is depicted in Figure 1.1.

The first three layers of the UCSB-IBM cloud are similar to the SPI classification, except that the authors break the infrastructure layer into three components. The three components that compose the UCSB-IBM infrastructure layer are computational resources, storage, and communications. In the rest of this section, we explain in more detail this ontology's components.

1.4.1 Applications (SaaS)

Similar to the SPI model, the first layer is the cloud application layer. The cloud application layer is the most visible layer to the end users of the cloud. Normally, users access the services provided by this layer through the browser via web

portals, and are sometimes required to pay fees to use them. This model has been recently proven to be attractive to many users, as it alleviates the burden of software maintenance and the ongoing operation and support costs. Furthermore, it exports the computational work from the users' terminal to the data centers where the cloud applications are deployed. This in turn lessens the hardware requirements needed at the users' end, and allows them to obtain superb performance for some of their CPU-intensive and memory-intensive workloads without necessitating large capital investments in their local machines. Arguably, the cloud application layer has enabled the growth of a new class of end-user devices in the form of "netbook" computers, which are less expensive end-user devices that rely on network connectivity and cloud applications for functionality. Netbook computers often have limited processing capability with little or no disk drive-based storage, relying on cloud applications to meet the needs for both.

As for the providers of cloud applications, this model simplifies their work with respect to upgrading and testing the code, while protecting their intellectual property. Since a cloud application is deployed at the provider's computing infrastructure (rather than at the users' desktop machines), the developers of the application are able to roll smaller patches to the system and add new features without disturbing the users with requests to install updates or service packs. The configuration and testing of the application in this model is arguably less complicated, since the deployment environment, i.e., the provider's data center becomes restricted. Even with respect to the provider's profit margin, this model supplies the software provider with a continuous flow of revenue, which might be even more profitable on the long run. This *SaaS* model conveys several favorable benefits for the users and providers of the cloud application. The body of research on SOA has numerous studies on composable IT services, which have a direct application to providing and composing *SaaS*.

The UCSB-IBM ontology illustrates that the cloud applications can be developed on the cloud software environments or infrastructure components (as discussed in Sections 1.3.2 and 1.3.3). In addition, cloud applications can be composed as a service from other services, using the concepts of SOA. For example, a payroll application might use another accounting system's SaaS to calculate the tax deductibles for each employee in its system without having to implement this service within the payroll software. In this respect, the cloud applications targeted for higher layers in the stack are simpler to develop and have a shorter time-to-market. Furthermore, they become less error prone, since all their interactions with the cloud are through pretested APIs. However, being developed for a higher stack layer limits the flexibility of the application and restricts the developers' ability to optimize its performance.

Despite all the advantageous benefits of this model, several deployment issues hinder its wide adoption. Specifically, the security and availability of the cloud

applications are two of the major issues in this model, and they are currently addressed by the use of lenient service-level agreements (SLAs). Furthermore, coping with outages is a realm that users and providers of *SaaS* have to tackle, especially with possible network outage and system failures. Additionally, the integration of legacy applications and the migration of the users' data to the cloud are slowing the adoption of *SaaS*. Before they can persuade users to migrate from desktop applications to cloud applications, cloud applications' providers need to address end-users' concerns about security and safety of storing confidential data on the cloud, users' authentication and authorization, uptime and performance, as well as data backup and disaster recovery.

1.4.2 Cloud Software Environment (PaaS)

The second layer in the UCSB-IBM cloud ontology is the cloud software environment layer (also dubbed the software platform layer). The users of this layer are cloud applications' developers, implementing their applications and deploying them on the cloud. The providers of the cloud software environments supply the developers with a programming-language-level environment of well-defined APIs to facilitate the interaction between the environments and the cloud applications, as well as to accelerate the deployment and support the scalability needed by cloud applications. The service provided by cloud systems in this layer is commonly referred to as *Platform as a Service* (*PaaS*). Section 1.2 mentioned Google's App Engine and Microsoft Azure as examples of this category. Another example is SalesForce's Apex language [2] that allows the developers of the cloud applications to design, along with their applications' logic, their page layout, workflow, and customer reports.

Developers reap several benefits from developing their cloud application for a cloud programming environment, including automatic scaling and load balancing, as well as integration with other services (e.g., authentication services, e-mail services, and user interface) supplied to them by the *PaaS* provider. In such a way, much of the overhead of developing cloud applications is alleviated and is handled at the environment level. Furthermore, developers have the ability to integrate other services to their applications on demand. This makes the development of cloud applications a less complicated task, accelerates the deployment time, and minimizes the logic faults in the application. In this respect, a Hadoop [21] deployment on the cloud would be considered a cloud software environment, as it provides its applications' developers with a programming environment, namely, the Map Reduce [7] framework for the cloud. Yahoo Research's Pig [28] project, a high-level language to enable processing of very large files in the Hadoop environment, may be viewed as an open-source implementation of the cloud platform layer. As such, cloud software environments facilitate the development process of cloud applications.

1.4.3 Cloud Software Infrastructure

The third layer in the USCB-IBM ontology is the cloud software infrastructure layer. It is here that this ontology more distinctly departs from the SPI ontology. The USCB-IBM ontology takes a finer-grain approach to distinguishing the roles and components that provide the infrastructure to support SPI ontology's PaaS layer. Specifically, it breaks the infrastructure layer down into a software layer that is composed of three distinct parts and places these on top of two additional layers. The three components, *computational resources, storage,* and *communications*, composing the cloud software infrastructure layer are described below.

a. *Computational resources:* VMs are the most common form for providing computational resources to cloud users at this layer. OS Virtualization is the enabler technology for this cloud component, which allows the users unprecedented flexibility in configuring their settings while protecting the physical infrastructure of the provider's data center. The users get a higher degree of flexibility since they normally get super-user access to their VMs that they can use to customize the software stack on their VM for performance and efficiency. Often, such services are dubbed *IaaS*.
b. *Storage:* The second infrastructure resource is data storage, which allows users to store their data at remote disks and access them anytime from any place. This service is commonly known as *D*ata-Storage as a *S*ervice (*DaaS*), and it facilitates cloud applications to scale beyond their limited servers. Examples of commercial cloud *DaaS* systems are Amazon's S3 [32] and EMC Storage Managed Service [9].
c. *Communication:* As the need for guaranteed quality of service (QoS) for network communication grows for cloud systems, communication becomes a vital component of the cloud infrastructure. Consequently, cloud systems are obliged to provide some communication capability that is service oriented, configurable, schedulable, predictable, and reliable. Toward this goal, the concept of Communication as a Service (CaaS) emerged to support such requirements, as well as network security, dynamic provisioning of virtual overlays for traffic isolation or dedicated bandwidth, guaranteed message delay limits, communication encryption, and network monitoring. Although this model is currently the least discussed and adopted cloud service in the commercial cloud systems, several research papers and articles [1,11,13] have investigated the various architectural design decisions, protocols, and solutions needed to provide QoS communication as a service. One recent example of systems that belong to CaaS is the Microsoft Connected Service Framework (CSF) [25]. Voice over IP (VoIP) telephone systems, audio and video conferencing, as well as instant messaging are candidate cloud applications that can be composed of CaaS and can in turn provide composable cloud solutions to other common applications.

In addition to the three main layers of the cloud, the UCSB-IBM model includes two more layers: the software kernel and the firmware/hardware layer.

1.4.4 Software Kernel Layer

It provides the basic software management for the physical servers that compose the cloud. Unlike the SPI ontology, the UCSB-IBM ontology explicitly identifies the software used to manage the hardware resources and its existing choices instead of focusing solely on VM instances and how they are used. Here, a software kernel layer is used to identify the systems software that can be used to construct, manage, and schedule the virtual containers onto the hardware resources. At this level, a software kernel can be implemented as an OS kernel, hypervisor, VM monitor, and/or clustering middleware. Customarily, grid computing applications were deployed and run on this layer on several interconnected clusters of machines. However, due to the absence of a virtualization abstraction in grid computing, jobs were closely tied to the actual hardware infrastructure, and providing migration, check-pointing, and load balancing to the applications at this level was always a complicated task.

The two most successful grid middleware systems that harness the physical resources to provide a successful deployment environment for grid applications are, arguably, Globus [15] and Condor [36]. The body of research in grid computing is large, and several grid-developed concepts are realized today in cloud computing. However, additional grid computing research can potentially be integrated to cloud research efforts. For example, grid computing microeconomics models [12] are possible initial models to study the issues of pricing, metering, and supply–demand equilibrium of the computing resources in the realm of cloud computing. The scientific community has also addressed the quest of building grid portals and gateways for grid environments through several approaches [4,6,16,17,34,35]. Such approaches and portal design experiences may be very useful to the development of usable portals and interfaces for the cloud at different software layers. In this respect, cloud computing can benefit from the different research directions that the grid community has embarked for almost a decade of grid computing research.

1.4.5 Cloud Hardware/Firmware

The bottom layer of the cloud stack in the UCSB-IBM ontology is the actual physical hardware and switches that form the backbone of the cloud. In this regard, users of this cloud layer are normally big enterprises with large IT requirements in need of subleasing *H*ardware as a *S*ervice (*HaaS*). For this, the *HaaS* provider operates, manages, and upgrades the hardware on behalf of its consumers for the lifetime of the sublease. This model is advantageous to the enterprise users, since often they do not need to invest in building and managing data centers. Meanwhile, *HaaS* providers have the technical expertise as well as the cost-effective infrastructure to host the systems. One of the early *HaaS* examples is Morgan Stanley's sublease

contract with IBM in 2004 [27]. SLAs in this model are stricter, since enterprise users have predefined business workloads with strict performance requirements. The margin benefit for *HaaS* providers materializes from the economy of scale of building data-centers with huge floor space, power, cooling costs, as well as operation and management expertise.

HaaS providers have to address a number of technical challenges in operating their services. Some major challenges for such large-scale systems are efficiency, ease, and speed of provisioning. Remote, scriptable boot loaders is one solution to remotely boot and deploy a complete software stack on the data centers. PXE [29] and UBoot [37] are examples of remote bootstrap execution environments that allow the system administrator to stream a binary image to multiple remote machines at boot time. Other examples of challenges that arise at this cloud layer include data center management, scheduling, and power and cooling optimization. IBM Kittyhawk [3] is an example of a research project that targets the hardware cloud layer. This project exploits novel integrated scalable hardware to address the challenges of cloud computing at the hardware level. Furthermore, the project attempts to support many of the software infrastructure features at the hardware layer, thus permitting a more direct service model of the hardware. Specifically, it provides an environment in which external users can obtain exclusive access to raw metered hardware nodes in an on-demand fashion, similar to obtaining VMs from an IaaS provider. The system allows the software to be loaded and network connectivity to be under user control. Additionally, the prototype Kittyhawk system provides users with UBoot access, allowing them to script the boot sequence of the potentially thousands of Blue Gene/P nodes they may have allocated.

1.5 Jackson's Expansion on the UCSB-IBM Ontology

The UCSB-IBM model was adapted by several computing experts to facilitate the discussions and conversations about other aspects of the cloud. One of these aspects was the cloud security. With a focus on supporting cloud computing for governmental agencies, Jackson [23] adapted the original UCSB-IBM model and extended on it with the goal of supporting a more detailed view of the security aspects of the cloud computing field. By adding several additional layers to support cloud access management, workflow orchestration, application security, service management, and an explicit connectivity layer, Jackson highlighted several particulars of the security challenges for this emerging computing field. Specifically, he modified the original ontology to add the following three sets of layers:

1. *Access management layer:* This new layer is added above the cloud application layer and is intended to provide access management to the cloud applications implementing SaaS. In the form of different authentication techniques, this layer can provide a simplified and unified, yet efficient, form of protection. In

turn, this can simplify the development and usage of the SaaS applications while addressing the security concerns for these systems. In this way, one of the security risks in the cloud could simply be contained and addressed in one high-level layer, thereby confining one of the main risk factors in the cloud applications.

2. *Explicit SOA-related layers:* This set of layers offers several SOA features in a more explicit form that simplifies their utilization. Jackson added this set of layers between the application (Saas) and platform (Paas) layers in the original UCSB-IBM ontology. For example, one of the layers in this set is the workflow orchestration layer, which provides services for managing and orchestrating business-workflow applications in the cloud. Another layer in this set is the service discovery layer, which also facilitates the discovery of services available to an application and potentially simplifies its operation and composition of other services.
3. *Explicit connectivity Layers:* The third set of layers in this extension was mainly added to support explicit networking capability in the cloud. Realizing that network connectivity in the cloud is an important factor in addressing the security of data, Jackson extended the model by adding extra network security layers. These additional layers were placed between the cloud software infrastructure layers and their components. By analyzing the security of the "*data in motion*" and "*data at rest*," Jackson's model covered the security aspects of the data in the cloud at the network level as well.

1.6 Hoff's Cloud Model

Inspired by the SPI model and the UCSB-IBM cloud ontology, Christofer Hoff [22] organized an online collaboration and discussion between several cloud computing experts to build an ontology upon the earlier models. Hoff's Model, as shown in Figure 1.2, presented a new cloud ontology in more detail.

This model focused on analyzing the three main cloud services: IaaS, PaaS, and SaaS. The model dissects the *IaaS* layer to several other components. Data center facilities, which include power and space, is the first component. Hardware is the second component in the IaaS layer, which consists of compute node, data storage, and network subcomponents. Abstraction is the next component, which abridges the hardware through systems like VM monitors, grid, and cluster utilities. The next component is the core connectivity and delivery, which provides the various services supporting the systems utilizing the IaaS layer, such as authentication services and DNS services. In this model, the abstraction component and the connectivity and delivery component are interleaving, since they are closely interdependent on each other's services. The API component presents the management services as well as a simplified interface to the next layer in the cloud. One system, for example, that implements this API sub-layer is the *GoGrid CloudCenter*

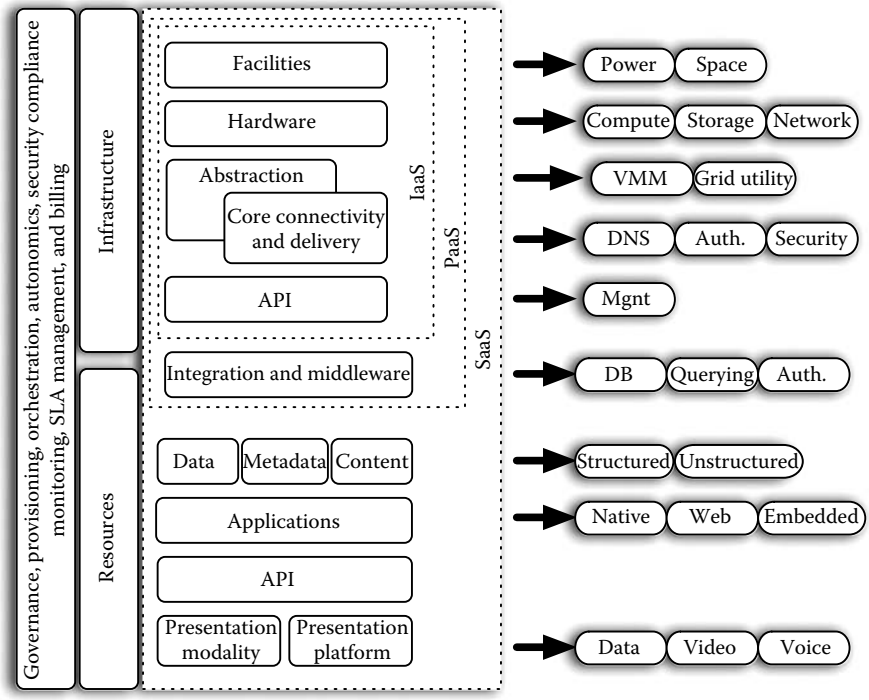

Figure 1.2 Hoff's cloud ontology, which emerged as an online collaboration and discussion between different cloud computing experts to further analyze the cloud components.

API [18]. This next layer in Hoff's model, which is the *PaaS*, is composed of one sub-layer that provides the integration services in the cloud. This sub-layer provides several services, such as authentication, database, and querying services.

The *SaaS* layer in Hoff's model is also further broken down into several sub-layers and components. The cloud application *data sub-layer* is shown to consist of the actual data, the metadata describing the real data, and its content, which can be in a structured or unstructured form. The application component in the SaaS layer is categorized into three categories: native applications, web applications, and embedded applications. A native application can be a desktop application that uses a cloud service. A web application is a cloud application that is accessed via the web browser. Finally, an embedded application is a cloud application that is embedded into another application. The final two sub-layers in the *SaaS* layer in Hoff's model are the applications' API and the presentation sub-layers. Hoff's model further decomposed the presentation sub-layers into data presentation, video presentation, and voice presentation, recognizing the different forms of cloud data presentations.

As portrayed in Figure 1.2, Hoff's model addresses more details of the composition of the cloud. The increased detail reveals additional aspects and challenges to cloud computing; however, it comes at the cost of simplicity. Nevertheless, the three cloud models presented in this chapter are regarded complementary and represent different viewpoints of the new emerging cloud computing field.

1.7 Discussion

As the cloud computing technology continues to emerge, more cloud systems are developed and new concepts are introduced. In this respect, a fundamental understanding of the extent to which cloud computing inherits its concepts from various computing areas and models is important to understand the landscape of this novel computing field and to define its potentials and limitations. Such comprehension will facilitate further maturation of the area by enabling novel systems to be put in context and evaluated in the light of existing systems. Particularly, an ontological, model-based approach encourages new systems to be compared and contrasted with existing ones, thus identifying more effectively their novel aspects. We contend that this approach will lead to more creative and effective cloud systems and novel usage scenarios of the cloud. With this in mind, our approach has been to determine the different layers and components that constitute the cloud, and study their characteristics in light of their dependency on other computing fields and models.

An ontology of cloud computing allows better understanding of the interrelations between the different cloud components, enabling the composition of new systems from existing components and further recomposition of current systems from other cloud components for desirable features like extensibility, flexibility, availability, or merely optimization and better cost efficiency. We as well postulate that understanding the different components of the cloud allows system engineers and researchers to deal with hard technological challenges. For example, comprehending the relationship between different cloud systems can accentuate opportunities to design interoperable systems between different cloud offerings that provide higher-availability guarantees. Although high availability is one of the fundamental design features of every cloud offering, failures are not uncommon. Highly available cloud applications can be constructed, for example, by deploying them on two competitive cloud offerings, e.g., Google's App Engine [19] and Amazon's EC2 [8]. Even in the case that one of the two clouds fails, the other cloud will continue to support the availability of the applications. In brief, understanding the cloud components may enable creative solutions to common cloud system problems, such as availability, application migration between cloud offerings, and system resilience. Furthermore, it will convey the potential of meeting higher-level implementation concepts through interoperability between different systems. For example, the high-availability requirement may be met by formulating an inter-cloud protocol,

which enables migration and load balancing between cloud systems. Resilience in the cloud, for example, can also be met through concepts of self-healing and autonomic computing. The broad objective of this classification is to attain a better understanding of cloud computing and define key issues in current systems as well as accentuate some of the research topics that need to be addressed in such systems.

Not only can an ontology impact the research community, but it also can simplify the educational efforts in teaching cloud computing concepts to students and new cloud applications' developers. Understanding the implications of developing cloud applications against one cloud layer versus another will equip developers with the knowledge to make informed decisions about their applications' expected time-to-market, programming productivity, scaling flexibility, as well as performance bottlenecks. In this regard, an ontology can facilitate the adoption of cloud computing and its evolution. Toward the end goal of a thorough comprehension of the field of cloud computing, we have introduced in this chapter three contemporary cloud computing classifications that present cloud systems and their organization at different levels of detail.

References

1. J. Hofstader. Communications as a service. http://msdn.microsoft.com/en-us/library/bb896003.aspx
2. Apex: Salesforce on-demand programming language and framework. http://developer.force.com/
3. J. Appavoo, V. Uhlig, and A. Waterland. Project kittyhawk: Building a global-scale computer: Blue Gene/P as a generic computing platform. *SIGOPS Oper. Syst. Rev.*, 42(1):77–84, 2008.
4. M. Chau, Z. Huang, J. Qin, Y. Zhou, and H. Chen. Building a scientific knowledge web portal: The nanoport experience. *Decis. Support Syst.*, 42(2):1216–1238, 2006.
5. N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine application development and deployment. Technical Report TR-2009-02, University of California, Santa Barbara, CA, 2009.
6. M. Christie and S. Marru. The LEAD portal: A teragrid gateway and application service architecture: Research articles. *Concurr. Comput. Pract. Exp.*, 19(6):767–781, 2007.
7. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (*OSDI*), San Francisco, CA, pp. 137–150, 2004.
8. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/
9. EMC Managed Storage Service. http://www.emc.com/
10. Enomalism elastic computing infrastructure. http://www.enomaly.com
11. A. Hanemann et al. PerfSONAR: A service oriented architecture for multi-domain network monitoring. In B. Benatallah et al., editors, *ICSOC*, Amsterdam, the Netherlands, *Lecture Notes in Computer Science*, vol. 3826, pp. 241–254. Springer, Berlin, Germany, 2005.

12. R. Wolski et al. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. J. G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pp. 747–772. John Wiley & Sons, Chichester, U.K., 2003.
13. W. Johnston et al. Network communication as a service-oriented capability. In L. Grandinetti, editor, *High Performance Computing and Grids in Action, Advances in Parallel Computing*, vol. 16, IOS Press, Amsterdam, the Netherlands, March 2008.
14. Eucalyptus. http://eucalyptus.cs.ucsb.edu/
15. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl.*, 11(2):115–128, 1997.
16. D. Gannon et al. Building grid portal applications from a web-service component architecture. *Proc. IEEE* (Special Issue on Grid Computing), 93(3):551–563, March 2005.
17. D. Gannon, B. Plale, M. Christie, Y. Huang, S. Jensen, N. Liu, S. Marru, S. Pallickara, S. Perera, and S. Shirasuna. Building grid portals for e-science: A service oriented architecture. *High Performance Computing and Grids in Action*. IOS Press, Amsterdam, the Netherlands, 2007.
18. GoGrid Cloud Center API. http://www.gogrid.com/how-it-works/gogrid-API.php
19. Google App Engine. http://code.google.com/appengine
20. Google Apps. http://www.google.com/apps/business/index.html
21. Hadoop. http://hadoop.apache.org/
22. C. Hoff. Christofer hoff blog: Rational survivability. http://rationalsecurity.typepad.com/blog/
23. K. L. Jackson. An ontology for tactical cloud computing. http://kevinljackson.blogspot.com/
24. M. Crandell. Defogging cloud computing: A taxonomy, June 16, 2008. http://refresh.gigaom.com/2008/06/16/defogging-cloud-computing-a-taxonomy/
25. Microsoft Connected Service Framework. http://www.microsoft.com/serviceproviders/solutions/connectedservicesframework.mspx
26. Microsoft Azure. http://www.microsoft.com/azure
27. M. Stanley. IBM ink utility computing deal. http://news.cnet.com/2100-7339-5200970.html
28. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, Vancouver, Canada, pp. 1099–1110, 2008. ACM, New York.
29. Preboot Execution Environment (PXE) Specifications, Intel Technical Report, September 1999.
30. R. W. Anderson. Cloud services continuum, July 3; 2008. http://et.cairenenet/2008/07/03/cloud-services-continuum/
31. R. W. Anderson. The cloud services stack and infrastructure, July 28, 2008. http://et.cairene.net/2008/07/28/the-cloud-services-stack-infrastructure/
32. Amazon Simple Storage Service. http://aws.amazon.com/s3/
33. Salesforce Customer Relationships Management (CRM) system. http://www.salesforce.com/
34. T. Severiens. Physics portals basing on distributed databases. In *IuK*, Trier, Germany, 2001.

35. P. Smr and V. Novek. Ontology acquisition for automatic building of scientific portals. In J. Wiedermann, G. Tel, J. Pokorný, M. Bieliková, and J. Stuller, editors, *SOFSEM 2006: Theory and Practice of Computer Science: 32nd Conference on Current Trends in Theory and Practice of Computer Science*, pp. 493–500. Springer Verlag, Berlin/Heidelberg, Germany, 2006.
36. D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, 2005.
37. Das U-Boot: The universal boot loader. http://www.denx.de/wiki/U-Boot/WebHome
38. Virtual Workspaces Science Clouds. http://workspace.globus.org/clouds/