Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2013, Article ID 782174, 31 pages
http://dx.doi.org/10.1155/2013/782174

Review Article

Foundations and Technological Landscape of Cloud Computing

Nattakarn Phaphoom, Xiaofeng Wang, and Pekka Abrahamsson

Faculty of Computer Science, Free University of Bolzano-Bozen, Piazza Domenicani 3, 39100 Bolzano, Italy

Correspondence should be addressed to Nattakarn Phaphoom; phaphoom@inf.unibz.it

Received 31 October 2012; Accepted 12 December 2012

Academic Editors: M. Meringer and R. J. Walker

The cloud computing paradigm has brought the benefits of utility computing to a global scale. It has gained paramount attention in recent years. Companies are seriously considering to adopt this new paradigm and expecting to receive significant benefits. In fact, the concept of cloud computing is not a revolution in terms of technology; it has been established based on the solid ground of virtualization, distributed system, and web services. To comprehend cloud computing, its foundations and technological landscape need to be adequately understood. This paper provides a comprehensive review on the building blocks of cloud computing and relevant technological aspects. It focuses on four key areas including architecture, virtualization, data management, and security issues.

1. Introduction

Cloud computing technology has attracted significant attention from both academic and industry in recent years. It is perceived as a shift in computing paradigm in which computing services are offered and acquired on demand over the global-scaled network [1]. In this paradigm, cloud service providers manage a pool of computing resources, generally by means of virtualization, and offer services in terms of infrastructure, platform, and software to consumers using a multitenancy model [2]. Consumers can provision and release such computing capabilities as needed through self-service interfaces. Service usages are automatically metered, allowing consumers to pay for the services only for what they use. In this ecosystem, cloud service providers gain an opportunity to maximize their profit through the economies of scale, while consumers gain access to (seemingly) unlimited resources to address their changing demands without requiring an upfront investment on cost and effort to set up an IT infrastructure.

Cloud computing has the potential to transform IT industry and change the way Enterprise IT is operated [3]. The adoption of cloud services, that is, provisions of data center, hosted deployment environment or on demand software, leads to different degrees of impacts on an enterprise. Focusing on a technical perspective, the potential benefits of cloud services include (a) an ability to shorten the cycle from ideas to profitable products; (b) an ability to address volatile workload without service interruptions or slowing down system performance; (c) simplified processes of establishing environments for application development and deployment; (d) decreased run time for backend jobs by using temporarily acquired resource; (e) an efficient solution for business continuity; (f) minimized software maintenance activities due to a shift of work to the cloud provider; and (g) centralized quality assurance as responsibility on quality control such as security and performance are transferred to the provider [4–7].

However, cloud services and security concerns inherited from their underlying technologies might negatively impact an enterprise if they are not properly managed. Technical and security risks identified in this context include (a) data lock-in and system lock-in; (b) unreliable system performance due to many uncontrollable factors such as network traffic, load balancing, and context switching cost; (c) decreased performance due to virtualization; (d) complex integration between legacy and cloud-based systems; (e) incompatibility of user behaviors and enterprise process over a new version of cloud software, as software upgrade is controlled by the provider; (f) information leakage in a multitenant model; (g) data interception during the transfer over public networks; and (h) security breach in a virtualization monitoring layer [3, 8–10].

One of the core aspects of cloud computing is that it hides IT complexity under service offering models. While it is evident that components in the building blocks of cloud services introduce a certain degree of impacts on service characteristics, it is less evident how different the impacts would be in different configurations. This knowledge is essential for service consumers. Software developers working on cloud platforms need to understand it to apply appropriate software designs and properly configure their deployment environments, in order to ensure certain characteristics of resulting software. Enterprise consumers need this knowledge during service level agreement (SLA) negotiation and to determine the line of responsibility. End users also need it to adjust their usage behavior.

This paper aims to provide a better understanding over cloud computing technology as well as its associated foundations. The knowledge serves as a basis for the in-depth analysis and assessment of cloud services. For software developers, this paper adds new aspects to consider when developing software in-the-cloud and for-the-cloud. For researchers, it identifies the landscape of the underlying technology of cloud computing, especially virtualization, data management, and security.

The remainder of the paper is organized into five sections. In the next section, we explain what cloud computing is and what it is not through reviewing various definitions, service models, deployment models, and relevant concepts. An in-depth comparison between grid and cloud computing is also presented in this section. Section 3 provides a review on existing cloud computing reference architectures and relevant quality attributes of cloud services. Virtualization technology is captured in Section 4, with a focus on the approaches to hardware system virtualization and its use scenarios. Section 5 is focused on data management in distributed environments and the design considerations of selected services from leading providers. Cloud security issues are discussed in Section 6 by considering the vulnerabilities associated to the key architectural components of cloud computing. The paper ends with the conclusions in the last section.

2. Cloud Computing Basics

The semantic of cloud computing is identified by its definition, service models, and deployment models.

2.1. Definition. The standard definition of "cloud computing" is on the way of reaching its consensus [11]. Among many interpretations of this term, its general objective and view are agreeable. The term "cloud computing" refers to the fact that computing services of any form—IT infrastructure, platforms, or applications—could be provisioned and used through the Internet. Cloud is built upon a large-scaled distributed infrastructure in which a pool of resources is generally virtualized, and offered services are distributed to clients in terms of a virtual machine, deployment environment, or software. In this way cloud services could be scaled dynamically according to requirements and current workloads. The usage of resources is measured, and the payment is made on a consumption basis.

Foster et al. provide a definition of cloud computing as they compare cloud with grid computing. According to their definition [12], cloud computing is "*a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.*"

Vaquero et al. identify more than 20 existing definitions of cloud computing and propose their own definition. As it is perceived in the end of 2008 [13], cloud computing is "*a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay- per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.*"

The most cited definition of cloud computing is the one proposed by The US National Institute of Standards and Technology (NIST). NIST provides the following definition [2]: "*Cloud computing is 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.*"

These definitions reveal three characteristics of the clouds. First of all, cloud services are massively scalable, and the acquisition and release of these services could be done dynamically with minimum operational supports required. Secondly, the cost is charged on a usage basis and the quality of services is guaranteed by a providers based on a service level agreement. Lastly, the quality of cloud services, such as security and performance, relies primarily on availability of Internet and how underlying resources are managed and distributed to clients.

2.2. Service Models. A service model determines the types of computer resources offered to consumers. Three main types of cloud services are infrastructure (IaaS), platform (PaaS), and software (SaaS). However, new service models are continuously emerging.

2.2.1. Infrastructure-as-a-Service (IaaS). A provider provides a virtual infrastructure where computing resources including processing units, storage, and network could be provisioned in order to set up a deployment environment for their software system. A customer has flexibility to manage and control a software stack to be deployed ranging from an operating system, middleware, and applications. Examples of IaaS are Amazon Elastic Compute Cloud (EC2) (http://aws.amazon.com/), Eucalyptus (http://open.eucalyptus.com/), Openstack (http://openstack.org/projects/compute/), and OpenNebula (http://www.opennebula.org/).

2.2.2. Platform-as-a-Service (PaaS). PaaS provides customers with the capability to develop and deploy applications based on tools and programming languages supported by the providers. This hosted platform is configurable in a limited manner based on a provided set of APIs. Examples of this class of services include Google AppEngine (http://code.google.com/appengine/), Windows Azure Platform (http://www.microsoft.com/windowsazure/), Force.com (http://developer.force.com/), and Rackspace (http://www.rackspace.com/).

2.2.3. Software-as-a-Service (SaaS). SaaS provides the capability to use the applications which run on cloud infrastructure. These applications are accessible through standard interfaces such as a web browser or an email client. SaaS offers the experience of getting to work on applications and data from anywhere at any time by using various form of devices. Examples of widely used SaaS are Facebook, Gmail, and OfficeLive (http://www.officelive.com/en-us/). Enterprise SaaS exist in many domains such as accounting, customer relationship management, content management, and enterprise resource planning.

2.2.4. Human-as-a-Service (HuaaS). HuaaS relies on information aggregation techniques to extract meaningful information or prediction from massive-scale data [24]. The services make use of information provided by large cyber communities, such as Digg (http://digg.com/), which aims to be the first source of news in any topic. A concept of Crowdsourcing [25, 26] and Crowdservicing [27], which gathers a group of people to solve complex problems or to contribute with innovative ideas, belongs to this model. Examples of Crowdsourcing include community-based design and human-based computation.

2.2.5. Everything-as-a-Service (XaaS). A computing paradigm is moving toward a XaaS concept in which everything could be acquired as a service. Cloud computing and aforementioned service models are in support of XaaS [24]. XaaS implies a model of dynamic environments in which clients have a full control to customize the computing environment to best fit their unique demands by composing varieties of cloud-based services.

These examples are fairly well-known cloud services. OpenCrowd taxonomy [16] presents a collection of cloud service providers for each of the defined models.

2.3. Deployment Models. Different deployment models are designed to support a variation of consumers' privacy requirements for cloud adoption. NIST defines cloud deployment models as public, private, community, and hybrid [2]. Virtual private cloud is introduced by Amazon as an alternative solution that balances flexibility of public clouds and security of private clouds [28, 29].

(i) *Public cloud*. The cloud infrastructure is owned and managed by a provider who offers its services to public.

(ii) *Private cloud*. The cloud infrastructure is built for a specific organization, but might be managed by a third party under a service level agreement.

(iii) *Virtual private cloud (VPC)*. Virtual private cloud removes security issues caused by resource sharing of public clouds by adding a security platform on top of the public clouds. It leverages virtual private network (VPN) technology and provides some dedicated resources allowing consumers to customize their network topology and security settings.

(iv) *Community cloud*. The infrastructure is shared among several organizations that have common requirements or concerns.

(v) *Hybrid cloud*. Several types of clouds are composed together through data portability technology and federated authentication mechanism.

2.4. History of Computing Paradigms. Cloud computing introduces a shift in a computing paradigm. Voas and Zhang in their article [1] illustrate the computing evolution through six distinct phases. In the first phase people use a powerful *mainframe* which is designed to multiplex its computing cycle to support multiple applications. A *personal computer* has become a convenient mean to perform a daily work in the next phase, responding to a drop of hardware costs and its sufficient computational power. A *computer network* in a form of local area network (LAN) or wide area network (WAN) is flourished in the third phases as a mean for resource sharing. The *Internet* has introduced after that as a global network that allows people to utilize remote services. The fifth phase has brought a concept of *grid computing* which utilizes distributed systems and parallel computing to serve high throughput and high performance computing. Finally, in a current phase *cloud computing* provides a convenient way to acquire any form of computing services through the Internet as another utility.

2.5. Relevant Concepts. The concept of cloud computing is not a revolution. In fact, it introduces an overlap with many concepts and technologies, such as grid computing, utility computing, and virtualization. This subsection gives an overview to these concepts and points out common angles that each of these concepts share with cloud computing.

2.5.1. Grid Computing. Grid computing is a distributed computing paradigm that enables resource sharing among multivirtual organizations in order to solve a common computational problem [30]. Grid has been developed originally to serve scientific applications that require tremendous computational power. The grid composes of heterogeneous and physically distributed resources. To unify a computing environment from such diversity, grid frameworks provide standard protocols and middleware that mediate accesses to a range of physical resources, as well as organizing basic functionalities such as resource provisioning, catalogue, job scheduling, monitoring, and security assurance. Compared to cloud computing, grid shares a common objective of achieving optimization of resource usage. The difference is that cloud infrastructure is owned and managed by a single organization, resulting in a homogenous platform by nature. The management of cloud, as a consequence, focuses on utilizing a resource pool, rather than coordinating resources. An example of grid framework is Globus (http://www.globus.org/).

2.5.2. Utility Computing. Utility computing presents a model that enables the sharing of computing infrastructure in which the resources from a shared pool are distributed to clients upon requests, and the cost of acquiring services are charged based on the usage [31]. Cloud computing realizes the concept of utility computing by offering computing power which could be acquired over the Internet under a fixed or on demand pricing model. In addition to resource optimization, achieving a satisfactory level of service quality is a main goal for both environments.

2.5.3. Virtualization. Virtualization is an abstraction of a computing system that provides interfaces to hardware including a processing unit and its register, storages, and I/O devices [32–34]. These physical resources are visible to processes or devices in a higher abstraction level as virtualized resources. Virtualization at a system level allows multiple virtual machines to operate on the same physical platform. Virtualization at a process level is managed by an operating system, multiplexing processing cycles and resources to support multiple processes. Cloud computing providers in general use system virtualization to provide the capability of resource pooling by allocating and deallocating virtualized resources in terms of a virtual machine to a client's system on demand. Examples of virtualization solutions are VMWare (http://www.vmware.com/) (commercial), Xen (http://www.citrix.com/) (open source), and KVM (http://www.linux-kvm.org/) (open source).

2.5.4. Service Oriented Architecture. Architectural patterns are used to create designs that are standardized, well understood, and predictable. These patterns are proven solutions to recurring common problems in software design [35]. Service oriented architecture (SOA) is a widely adopted architectural pattern for designing distributed and loosely couple systems. It is designed to help organizations to achieve business goals including easy integration with legacy systems, remove inefficient business processes, and reduce cost, agile adaptation, and fast responses in competitive markets [36].

Varieties of architectures could be derived from SOA. Erl, in his book "SOA design patterns" [37] and a community site (http://www.soapatterns.org/), introduces more than eighty patterns for service oriented systems. The following are examples of categories and corresponding designs.

(i) *Patterns for creating service inventory*. Enterprise Inventory (to maximize recomposition), Service Normalization (to avoid redundant service logic), and Service Layer (to organize logics based on a common functional type).

(ii) *Patterns for organizing logical service layers*. Utility Abstraction (to govern common nonbusiness centric logics), Entity Abstraction (to organize agonistic business processes), and Process Abstraction (to organize nonagonistic business processes).

(iii) *Patterns for enhancing interoperability*. Data Model Transformation (to convert data of different schemas), Data Format Transformation (to allow services to interact with programs that use different data formats), and Protocol Bridging (to allow services that use different communication protocol to exchange data).

(iv) *Patterns for infrastructure*. Enterprise Service Bus, Orchestration, and Service Broker.

(v) *Patterns for enhancing security*. Exception Shielding (to prevent disclosure of internal implementation when exceptions occur), Message Screening (to protect a service for malformed and malicious input), and Service Perimeter Guard (to make internal services available to external users without exposing other internal resources).

SOA is considered as a basic for cloud computing, as it provides software architecture that addresses many quality attributes required for cloud services, such as component composibility, reusability, and scalability. The concept of SOA is leveraged to construct extensible cloud solution architecture, standard interfaces, and reusable components [22].

2.6. A Comparison of Grid and Cloud Computing. Grid and cloud computing have been established based on a particular common ground of distributed and parallel computing, targeting at a common goal of resource sharing. Both technologies offer a similar set of advantages such as flexibility of acquiring additional resources on demand and optimizing the usage of infrastructure. However, they pose differences especially from a point of view of resource management. This section provides such comparisons based on the work of Foster et al. presented in [12]. In their work cloud and grid are compared in terms of business models, architecture, resource management, and programming models.

2.6.1. Business Models. A business model captures a flow in which services are created and delivered to clients in order to generate incomes to a service provider and to serve clients' requirements. In cloud computing a role of providers is to provide computing services as a basic utility. Cloud generally offers a fixed or on demand pricing model. In either case, clients benefit from a reduction on upfront IT investments and flexibility in scaling its applications. In grid environments, an incentive to join the community is to get access to additional computing utilities, with the price of sharing one's own resources. The whole community benefits from optimization.

2.6.2. Architecture. In terms of infrastructure cloud is designed to serve as an internet-scale pool of resources. The whole infrastructure is managed by a single provider. Thus, a single unit of resource is conformed to a common governance

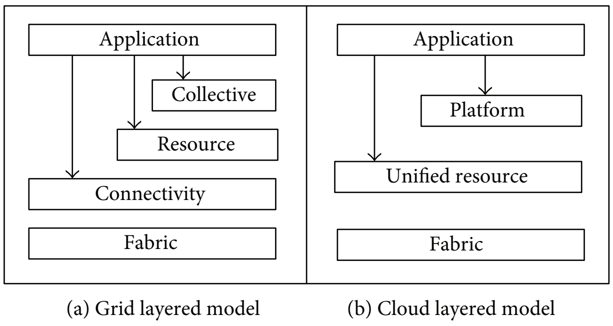

FIGURE 1: An architecture comparison: (a) grid model that provides connectivity to heterogeneous resources and (b) cloud model that manages a pool of resources by means of virtualization [12].

model. In contrast, grid has been developed to integrate distributed, dynamic, and heterogeneous resources. A set of open standard protocols and facilities is established to allow those resources to be interoperated. The differences are reflected in their reference models (Figure 1). Composing of four layers, distributed resources residing at a cloud fabric layer are encapsulated (generally by means of virtualization), so that they can be used by platform and application layers as integrated resources. Grid composes of five layers. Instead of depending on virtualization, grid defines standard protocols at a connectivity layer which allows communications among distributed nodes. A resource layer defines protocols for publication, discovery, negotiation, and payment. A collective layer controls interactions across collections of resources such as scheduling and monitoring. Applications exploit services provided at the lower layer through APIs.

2.6.3. Resource Management. Resource management targets at the mechanisms to control resource pooling, to achieve effective resource allocation and a satisfactory level of service quality. It covers four main areas including computing model, data model and locality, virtualization, and monitoring.

Computing Model. A computing model concerns with how resources are distributed for computational work. Cloud resources are distributed in terms of a virtual machine. A new instance of a virtual machine is created and placed to a physical location which is unknown to clients. The placement algorithm is customized to maintain a balance of platform utilization, relevant costs, and a guaranteed quality of services. In contrast, grid uses queuing system to manage jobs and resource allocations. Job stays in queue until the required amount of resources are available. Once allocated, resources are dedicated only for that job. Due to such scheduling policy, interactive applications which require short latency time could not operate natively on the grid.

Data Model and Locality. In both grid and cloud, data are distributed and replicated into a number of nodes to minimize the cost of communication between data and processors. Cloud uses a MapReduce framework to handle data locality. MapReduce runs on top of the file system, where data files are partitioned into chunks and replicated in many nodes. When a file needs to be processed, the storage service schedules a processor at the node hosting each chunk of the data to process the job. However, data locality cannot be easily exploited in grid, as the resource is allocated based on availability. One technique to tackle this issue is to consider data locality information, while a processor is schedule for computation. This approach is implemented in a data-aware scheduler.

Virtualization. Virtualization is a mechanism to provide abstraction to resources in the fabric layer, allowing administrative work (e.g., configuring, monitoring) to be performed more effectively in the cloud. Grid does not rely on virtualization as much as cloud does. This is due to the scale and the fact that each organization in grid community has ultimate control over their resources.

Monitoring. In cloud environment, client's capability for monitoring is restricted to a type of services employed. A model that provides infrastructure-as-a-service (IaaS) gives more flexibility for clients to monitor and configure lower level resources and middleware. Monitoring inside grid environment could be done in a more straightforward manner through user's credential which defines the right of users to access resources at different grid sites.

2.6.4. Programming Model. On top of Google's MapReduce, a number of programming models have been created to facilitate the development of distributed and parallel programming [38]. These include Sawzall (http://code.google.com/p/szl/), Hadoop (http://hadoop.apache.org/), and Pig (http://pig.apache.org/). Microsoft provides DryadLINQ framework to serve the similar purpose [39]. A main concern of programming model in grid is due to a large number of heterogeneous resources. Programming models which are generally used are Message Passing Interfaces (MPI), Grid Remote Procedural Call, and Web Service Resource Framework which allows applications to be stateful.

In brief, cloud and grid share similarity in terms of their goal and underlining technologies that serve as a building block. They are different from a point of view of resource management. These differences are caused by the fact that (1) clouds and grids build upon resources of different nature;

(2) clouds are operated for larger target group and serve a wider range of applications which put emphasis on different aspects of service quality.

2.7. Summary. The most respected definition of cloud is the one given by NIST. Cloud embraces the following characteristics: (a) it provides on demand computing capability which is accessible through the internet; (b) the computing capability could be provisioned and scaled with a minimum operational effort; (c) the usage of resources is metered and charged accordingly; (d) provider's resources are pooled to serve multiple clients; (e) it inherits benefits and risks from IT outsourcing. In terms of computing paradigm, clouds are considered as an evolution of grid computing. Both technologies share a common goal of optimizing resource usage and offer a similar set of advantages. However, clouds and grid are significantly different from a perspective of resource management.

3. Architecture and Quality of Cloud Services

Cloud computing architecture is partly represented through service and deployment models described in the previous section. Its complete architecture must capture relationship and dependency among relevant entities and activities in that environment.

3.1. Existing Cloud Computing Reference Architecture. A reference model is an abstract view of an environment of interest. It presents relationships and dependencies among entities in that environment, while abstracting away the standard, technology, and implementation underneath. It is particularly useful to identify an abstract solution to a given issue and to determine the scope of influence.

A report by the US National Institute of Standards and Technology (NIST) gathers cloud reference architecture models proposed by known organizations [40]. Architectural elements that are consistently presented in these models are (a) *a layered model*, that combines key components and their relationship; (b) *actors*, including their role and responsibilities; and (c) *management domains*, which facilitate basic operations of data centers and services on top of it.

This subsection summarizes main components of reference models list in the NIST report and proposed models found in the literatures.

3.1.1. Distributed Management Task Force, Inc. DMTF (http://dmtf.org/) proposes the cloud conceptual architecture integrating actors, interfaces, data artifacts, and profiles [14]. It focuses on a provider interface layer which offers specific services, standards, and environments to different users according to their registered profiles.

3.1.2. IBM. IBM's view on cloud management architecture combines actor's roles, services, virtualized infrastructure, and provider management platforms [15]. It offers Business-Process-as-a-Service (BPaaS) on to top of software capability which allows an automated customization of business workflows. The management platforms are customized for business-related services and technical-related services.

3.1.3. Cloud Security Alliance. CSA (https://cloudsecurityalliance.org/) introduces a seven-layer stack model that captures relationship and dependency of resources and services [16]. The model is customized for security analysis by separating a layer of resource management to an abstraction sublayer and a core connectivity and delivery sublayer. Elements of cloud applications (or SaaS) are presented through four sublayers comprising data, metadata, contents, applications, APIs, modality, and a presentation platform.

3.1.4. Cisco. Instead of focusing on a service model representation, Cisco explicitly puts security architecture and service orchestration into the frame [17]. Cisco framework consists of five layers: data center architecture, security, service orchestration, service delivery and management, and service customer. A security framework is built across the whole infrastructure. An orchestration layer maps a group of technological components to a services component for delivery.

3.1.5. Open Security Architecture. OSA (http://www.opensecurityarchitecture.org/) publishes a collection of cloud security patterns for more than 25 usage contexts such as client-server modules, identity management patterns, and SOA internal service usages [18]. These patterns combine actors, systems, activities, and features of related entities. They could be used as a high level use cases that capture cloud service interfaces for each of the actor's activities.

3.1.6. The Federal Cloud Computing Initiative. FCCI (http://www.info.apps.gov/) targets at government-wide adoption of cloud computing [19]. FCCI defines eight service components for the government to be addressed to deliver online user interfaces. These components consist of customizable user pages, application library, online storage, collaboration enabler widgets, connectivity, interoperability, provisioning and administrative tools, and the security mechanism that apply for the entire system. FCCI also provides a drafted layered service framework that outlines main service components for each layer of the government cloud.

3.1.7. The Storage Networking Industry Association. SNIA (http://www.snia.org/) proposes Cloud Data Management Interface (CDMI) as standard interfaces to access cloud storage and to manage the data stored [20]. CDMI comprises three main interfaces to increase interoperability among cloud storages: (1) data management interfaces that allow application to add, retrieve, update, and delete data elements stored in the cloud; (2) storage management interfaces that support legacy system, scalable nonrelational database (NoSQL), and object storages; and (3) the Resource Domain Model which describes how requirements and quality of services could be achieved.

3.1.8. Youseff et al. The authors propose cloud ontology based on composibility of service [21]. A cloud layer is higher

Table 1: Cloud computing reference models.

Name	Objective	Key components	References
Distributed management task force	To achieve interoperability	Actors, service interfaces, and profile	[14]
IBM	General purpose	Actors and roles, cloud services, and management activities to support business-related services and technical-related services	[15]
Cloud security alliance	Security assessment	Stack model, cloud services	[16]
Cisco	General purpose	Stack model for service composition	[17]
Open security architecture	Security assessment	Actors, flow of traffic and information in the cloud, security policy implemented by each actors, and servers to secure cloud operations	[18]
Federal cloud computing initiative	Standard for government clouds	Stack model representing cloud core capabilities and their associated management domain, actors, and cloud services	[19]
Cloud data management interfaces	Standard interfaces for cloud storage	Interfaces to data storage and associated metadata	[20]
Cloud ontology	General purpose	Stack model representing basic cloud resources and services	[21]
Cloud computing open architecture	Open standard and cloud ecosystem	Stack model integrating cloud virtual resources, common reusable services, core services, offerings, unified architecture interfaces, quality and governance, and ecosystem management	[22]

in stack if its services are composed of other services of underlying layers. Services belong to the same layer if they have the same level of abstraction. The concept results in a simple five-layered cloud ontology, consisting of hardware, software kernel, cloud software infrastructure, cloud software environment, and cloud applications.

3.1.9. Zhang and Zhou. The authors present a cloud open architecture (CCOA) aiming to assist strategic planning and consultancy of cloud computing services [22]. Their architecture illustrates seven principles of cloud and their correlations. These principles include virtualization, service orientation for reusable services, provision and subscription, configurable cloud services, unified information representation and exchange framework, quality and governance, and ecosystem management. Cloud actors are integrated to related principles.

Table 1 summarizes key details of aforementioned models.

The existence of multiple cloud computing architectures, even though serving different purposes, reflects a lack of standardization and interoperability in this field. In fact, the views of cloud represented by each model are not disrupted. They rather reflect cloud environments at different levels of abstraction and put a focus on different aspects. However, having a uniformed model would enhance collaborations among stakeholders and help to prevent a vendor lock-in problem.

3.2. Cloud Computing Layered Models. The objective of the first focus area is to understand the relationship and dependency of basic cloud components, actors, and management activities. A multilayer stack model allows us to put in place the technology associated to each layer, without being interfered by management activities. Figure 2 compares three models which depict cloud components at the different levels of abstraction.

Model (a) is introduced by Foster et al. to identify the differences between grid and cloud computing [12]. This four-layer model separates the fabric layer and the unified resource layer to present the distributed and resource-sharing nature of cloud, as well as to identify the need of a virtualization tool to simulate isolated environment for each consumer. In Figure 2(b), Youseff et al. design a five-layer model to capture different types of cloud services [21]. It distinguishes three types of service models (i.e., IaaS, PaaS, and SaaS) which are built upon one another, as well as three types of virtualized resources (computing, storage, and network) under cloud infrastructure. Interfaces to virtual resources and operating system are not explicit in this model. Model (c) proposed by CSA explicitly defines APIs layer which mediates communication between an operation system and integrated virtual resources. It also illustrates dependency of relevant SaaS components. This model is probably most appropriate for a security assessment.

3.2.1. Facilities and Hardware Layer. The bottom layer consists of physical computing resources, storages, network devices, data centers, and a mean to provide access to physical resources from other networks. CSA separates the hardware and facility layer to identify different kinds of security concerns associated to hardware and data centers. Relevant technologies in this layer include green data center, distributed system, cluster system, and firewall.

3.2.2. Abstraction Layer. The abstraction layer provides a unified view of distributed and heterogeneous physical resources generally by mean of virtualization. The abstract infrastructure composes of the view of servers (processor, memory, and node), storages, network, and other facilities. Relevant technologies include virtualization and virtual machine monitor.

3.2.3. Core Connectivity and Delivery Layer. This layer provides necessary tools to perform basic cloud operations

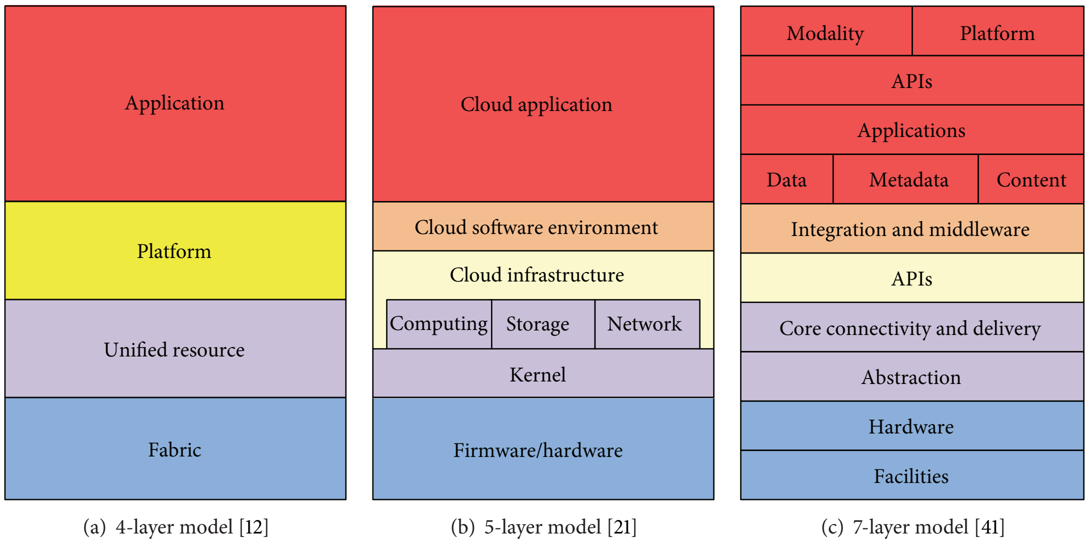

FIGURE 2: Cloud computing stacked models.

such as resource provision, orchestra, utilization, monitoring, and backup. It allows providers to manage load balancing, optimization of resource, and security in multitenant environments. Relevant technologies include resource pooling, multitenancy, distributed storages, NoSQL, virtual machine, virtual network, load balancing, cloud service bus, and MapReduce.

3.2.4. APIs Layer. The API layer provides interfaces for consumers to access, manage, and control their provision resources. Relevant technologies include web services, virtual machine, virtual data center, authentication and authorization mechanisms, multitenancy, and Infrastructure-as-a-Service.

3.2.5. Integration and Middleware Layer. This layer provides a customizable development environment on top of a virtualized platform for the development and deployment of cloud software. Relevant technologies include hardened pared-down operating system, development environment, deployment environment, and Platform-as-a-Service.

3.2.6. Application Layer. This layer offers web applications and services running on cloud infrastructure which are accessible through standard interfaces and devices. Relevant technologies include web services (e.g., WSDL, SOAP, and REST), web technology (e.g., HTML, CSS, JavaScript, DOM, AJAX, and mash-up), authentication and authorization (public-key cryptography), federated identity management (OpenID, Oauth), secured web browsers, data format (e.g., XML, HTML, and JSON), and Software-as-a-Service.

3.3. Cloud Actors and Roles. Four types of cloud actors are defined in the reference models and literature. These include *consumer, provider, facilitator, and developer*. A consumer refers to an end-user or an organization that use cloud services. It could be further categorized into three subclasses including end users of SaaS, users of PaaS, and users of IaaS. A provider offers services to consumers at agreed quality levels and prices. SaaS providers maintain cloud software which is offered as a web application or a service. PaaS providers maintain virtualized platforms for development and deployment of cloud software. IaaS providers maintain hosted data centers. A facilitator interacts with consumers, providers, and other facilitators to provide a requirement-specific cloud solution by integrating and customizing standard cloud services. Facilitators could be seen as a cloud carrier or broker. A developer develops, tests, deploys, maintains, and monitors cloud services.

3.4. Cloud Management Domain. Several models (i.e., IBM, GSA, NIS, and GSA) identify groups of management activities required to maintain cloud production environments. We derive five cloud management domains based on management activities outlined in reviewed reference models.

3.4.1. Management of Physical Resources and Virtualization. This domain is primarily used by providers to maintain physical and virtualized cloud infrastructure. It allows basic operation including resource monitoring, optimization, load balancing, metering the usage, and providing isolation over multitenancy environment.

3.4.2. Management of Service Catalogues. The objective of this domain is to make cloud services and applications available to consumers. It allows services to be found, requested, acquired, managed, and tested.

3.4.3. Management of Operational Supports. This domain concerns with technical-related services. The responsibilities are threefold. First of all, it provides management of service instances. This includes deployment, configure, testing, debugging, and performance monitoring. Second, it provides transparency and control over an isolated deployment environment. From consumer perspective, especially for IaaS and PaaS, the provided information is sufficient for SLA management, capacity planning, and analysis security concerns. Lastly, it provides management over resources. This includes provisioning, configuration management, backup, and recovery.

3.4.4. Management of Business Supports. This domain concerns with business-related services, for instance, invoice, billing, and customer management.

3.4.5. Security Management. Every layer of cloud stack needs different security mechanisms. The service model determines an actor who is responsible for maintaining security concerns for each layer. Generally the security of virtualized infrastructure (facility, hardware, abstraction, and connectivity) is managed by the provider. Consumers of IaaS have to manage the integration of an operating system and virtualized infrastructure. Consumers of PaaS have to manage the configuration of deployment environments and application security. Security concerns, including user authentication and authorization, are all handled by the providers in SaaS.

3.5. Quality of Cloud Computing Services. Cloud computing is considered as a form of outsourcing where the ultimate management and control over acquired services are delegated to an external provider [6, 42]. A needed level for services is defined through a formal contract between the provider and its consumers. This contract is known as service level agreement (SLA). For the consumers, it is important to ensure that the agreed level of service is respected, and any violation is reported accordingly. For the providers, it is important to manage dynamic infrastructure to meet SLA and to maximize the profit and resource utilization [43].

SLA is defined in terms of quality of services such as performance and availability. Dynamic nature of clouds caused by virtualization, resource pooling, and network directly impacts service characteristics. Quality attributes relevant to cloud services are given in this subsection.

3.5.1. Scalability and Elasticity. Scalability refers to capability to scale up or down the computing resources including processing units, memory, storages, and network to response to volatile resource requirements [2]. Elasticity refers to ability to scale with minimum overheads in terms of time and operation supports [2, 38]. Providers offer this characteristic to IaaS and PaaS through automatic resource provisions. For SaaS, scalability means ability to address changing workload without significantly downgrading other relevant quality attributes [42, 44].

3.5.2. Time Behavior. Time behaviors (e.g., performance, response time) are critical of latency sensitive applications and introduce high impact for user experiences. As cloud applications are operated on virtual distributed platform, time behaviors touch upon various area such as a quality of network, virtualization, distributed storage, and computing model. Aforementioned factors cause unreliable time behavior for cloud services [3, 9, 45].

3.5.3. Security. Security and trust issues are early challenges to the introduction of a new technology. As cloud infrastructure is built upon several core technologies, the security relies on every of these components. Trust requires portions of positive experiences and provider's reputation. Common threats to cloud security include abuse of cloud services, insecure APIs, malicious insiders, shared technology vulnerabilities, data loss and leakage, and service hijacking [41]. Mechanisms to handle such issues and other cloud vulnerabilities should be explicitly clarified prior to the service adoption.

3.5.4. Availability. Availability refers to a percentage of time that the services are up and available for use. SLA contracts might use a more strict definition of availability by counting on uptime that respects at the quality level specified in the SLA [46].

3.5.5. Reliability. Reliability is capability of services to maintain a specific level of performance overtime (adapted from ISO/IEC 9126 [ISO/IEC 9126-1]). Reliability is influenced directly by a number of existing faults, a degree of fault tolerance, and recoverable capability of services in case of failures. Cloud infrastructure is built upon a number of clusters of commodity servers, and it is operated on the internet scale. Partial network failures and system malfunction need to be taken as a norm, and such failures should not impact availability of services.

3.5.6. Portability. Portability is an ability to move cloud artifacts from one provider to another [47]. Migration to the cloud introduces a certain degree of dependency between client systems and the service providers. For instance, clients might rely on proprietary features and versions of hardware supported by a certain provider. Such dependency needs to be minimized to facilitate future migrations and to reduce a risk of system lock-in and data lock-in. A lack of standardization for virtual machines, hypervisors, and storage APIs causes similar issue [3].

3.5.7. Usability. Usability refers to capability of services to be understood, learned, used, and attractive to users (adapted from ISO/IEC 9126 [ISO/IEC 9126-1]). IaaS and PaaS providers should offer sufficient APIs to support resource provisions, management, and monitoring activities. Usability is particularly important for SaaS to retain customers due to a low cost of switching.

3.5.8. Customizability. Capability of services to be customized to address individual user preferences is important for services that serve internet-scale users [48]. As it is impossible for providers to offer unique solution for each user, the service should be designed in a way that it allows a sufficient degree of customizability.

3.5.9. Reusability. Reusability is capability of software service components to serve for construction of other software.

Cloud computing amplifies the possibility of service reuse through broad Internet access.

3.5.10. Data Consistency. Consistency characteristic is relevant for SaaS. If the data is consistent, then all clients will always get the same data regardless of which replicas they read from. However, it is costly to maintain strong consistency in distributed environments [49], and this is the case for clouds. Many cloud storage solutions compromise a strong degree of consistency for higher availability and network partition tolerance [38, 50, 51].

3.6. Summary. In this section we reviewed existing cloud computing reference architectures proposed by different enterprises and researchers for different purposes. Some intend to initiate an open standard to enhance interoperability among providers; some aim to understand the dependencies of relevant components for effective security assessment; others are for general proposes. Three entities are generally presented in the reference models. These include a stack model, actors, and management domains. The most abstract model of cloud consists of four-layers fabric, unified resource, platform, and application. One of the most descriptive models is proposed by CSA. It separates the fabric layer into facility and hardware sublayers. The unified resource layer is separated into abstraction and core connectivity sublayers; the platform layer is separated into infrastructure's APIs and middleware sublayers; the application layer is further divided into a number application components. Cloud actors might take more than one role at a time; for instance, an enterprise could be a customer of IaaS provided by Amazon and develop SaaS for its customers. The responsibilities of each actor are explicitly defined in service level agreement. Management and operational activities to maintain cloud production environments could be grouped into five domains including physical resources and virtualization, service catalogues, operational supports, business supports, and security. Essential quality attributes relevant to cloud service are availability, security, scalability, portability, and performance. Cloud underlying technologies, such as virtualization, distributed storages, and web services, directly affect these quality attributes. Fault tolerance is taken as a mandatory requirement as partial network failures, and occasional crashes of commodity servers are common for systems of the internet scale. Usability is one of the most important characteristics for SaaS due to a low cost of switching.

4. Virtualization

Cloud computing services offer different types of computing resources over the internet as a utility service. A cloud service provider manages clusters of hardware resources and dynamically allocates these resources for consumers in terms of a virtual machine. Consumers acquire and release these virtual resources according to current workloads of their applications. The provider ensures a secure compartment for each of the consumer's environments, while trying to utilize the entire system at the lowest cost. Virtualization is an enabler technology behind this scenario.

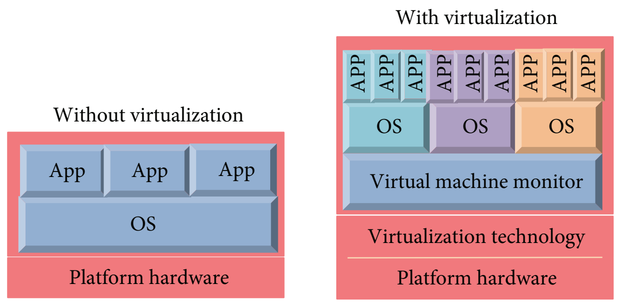

Figure 3: A comparison of a computer system with and without virtualization [53].

4.1. Virtualization Basics. Virtualization is an abstraction of a computer system which allows multiple guest systems to run on a single physical platform [52]. Host's physical resources (e.g., processor, registers, memory, and I/O devices) are shared and accessible through standard virtualization interfaces. Virtualization software creates an isolated environment for each of the guest systems by multiplexing host's computing cycles and virtualizing hardware resources. It mediates communications between guest and host systems and manipulates their messages and instructions if necessary to maintain the guest's illusion of having an ultimate control over the system.

Virtualization could be done at a process or system level. The software for system virtualization is broadly known as a hypervisor or a virtual machine monitor (VMM). The term VMM is used in this document, but they are semantically interchangeable.

Figure 3 compares a computing system with and without virtualization. Virtualized systems have a layer of VMM running on top of the hardware. A VMM allows multiple and possible different guest VMs to run simultaneously on the same physical system, while maintaining their illusions of having a sole control over the resources. To achieve this transparency the VMM operates in a higher privilege level than guest VMs. It intercepts guest's privilege instructions and executes them differently at the hardware level when necessary. It also maintains a secure confinement for each of the VM instances. Thus, resources allocated to one VM could not be interfered by other VMs running on the same physical block. This mechanism helps to promote security, as any compromise to system security is confined within an original VM.

Three immediate advantages of virtualization that could be derived from this model include: (1) *hardware independence*, as it allows different operating systems to run on the same platform; (2) *easy migration*, as the state of the system and applications is kept inside the VM; and (3) *resource optimization*, as the physical resources are shared to serve multiple virtual systems.

4.2. History of Virtualization Technology. The evolution of virtualization technology is explained by Rosenblum and Garfinkel [34]. Virtualization has been a solution to different issues during the evolution of computing trend. Its introduction dates back in '60 in the age of mainframe computing.

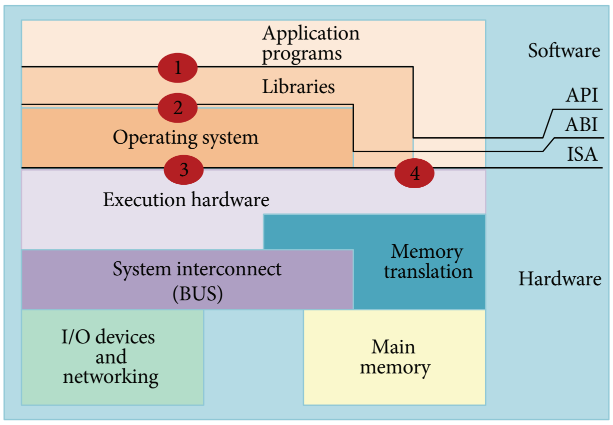

Figure 4: Interfaces of a computer system at different levels of abstraction [32].

IBM found a way to multiplex the usage of expensive hardware resources to support multiple applications in the same time. VMM was implemented as an abstraction layer that partitions a mainframe system to one or more virtual machines. Each of the machines held similar architecture as the hardware platform. A decrease of hardware cost and an emergence of multitasking OS during '80 caused in a drop of VMM necessity, until the point that the design of hardware no longer supported an efficient implementation of VMM.

The rule of polarity took place at the flourish age of microcomputers and complex OS. The drop in hardware cost has led to a variation of machines. Over-complex and large operating systems compromised its reliability. The OS became fragile and vulnerable, that the system administrator deployed one application per machine. Clearly, these machines were underutilized and resulted in maintenance overheads. In '05, virtualization became a solution to this problem. It was used as a mean to consolidate servers. Nowadays, virtualization is used for security and reliability enhancements.

4.3. Computer Architecture. Understanding of virtualization requires knowledge on computer architecture. Architecture could be seen as a formal specification of interfaces at a given abstraction level. A set of the interfaces provides a control over the behaviors of resources implemented at that level. Implementation complexity is hidden underneath. Smith and Nair describe the computer architecture at three abstraction levels: a hardware level, an operating system level, and an operating system library level [32]. Figure 4 illustrates the dependency among each level of interfaces. Virtualization exposes these interfaces and interacts with virtual resources they provide.

The interfaces at three abstraction level of computer systems are defined as follows.

4.3.1. Instruction Set Architecture (ISA). As an interface to hardware, ISA is a set of low level instructions that interact directly with hardware resources. It describes a specification of instructions supported by a certain model of processor. This includes an instruction's format, input, output, and the semantic of how the instruction is interpreted. While most of these instructions are only visible to the OS, some can be called directly by applications.

4.3.2. Application Binary Interface (ABI). At a higher level ABI provides indirect accesses to hardware and I/O devices through OS system calls. The OS executes necessary validation and perform that operation on behalf of the caller. In contrast to ISA ABI is platform independent, as the OS handles the actual implementation on different platforms.

4.3.3. Application Programming Interface (API). At the application level functionality is provided to application programs in terms of libraries. API is independent from the model of platform and OS, given that the variations are implemented at the lower abstraction levels (ABI and ISA).

4.4. Hardware Virtualization Approaches. As mentioned, a virtualized system contains an addition software layer called VMM or hypervisor. The main functionality of VMM is to multiplex hardware resources to support multiple guest VMs and to maintain its transparency. To achieve this VMM needs to handle the virtualization of a processor unit and registers, memory space, and I/O devices [34]. This subsection summarizes an implementation approach for virtualizing these resources.

4.4.1. CPU Virtualization. There are several techniques to allow multiple guest systems to share similar processing units. We summarize three main approaches for CPU virtualization that are largely discussed in the literature. The first technique requires a hardware support; the second relies on a support from an operating system; the last uses a hybrid approach.

Direct Execution. CPU is virtualizable if it supports VMM's direct execution [34]. Through direct execution, guest's privileged and unprivileged instructions are executed in a CPU's unprivileged mode, while VMM's instructions are executed in a privileged mode. This architecture allows a VMM to trap guest's privileged instructions (kernel instructions) and CPU's responses and emulate them in order to let VMM run transparently.
Figure 5 compares a CPU privileged ring that supports the direct execution with a tradition system. In tradition system (Figure 5(a)) operating system has the highest privilege. VMM uses a deprivileging technique (Figures 5(b) and 5(c)) to facilitate its operations. For example, through a 0/3/3 model (Figure 5(c)) guest's privileged and unprivileged instructions are executed in a CPU's unprivileged mode, while VMM's instructions are executed in a CPU's privileged mode.

Paravirtualization. Paravirtualization technique is introduced by Denali [54, 55] and is used by Xen [56]. It is one of the most common techniques to support an implementation of VMM on nonvirtualizable CPUs [34]. Through paravirtualization, the operating systems are ported to a specific CPU architecture. A nonvirtualizable part of ISA is replaced with virtualized and efficient code. This allows most of typical applications to run unmodified. This technique results in a

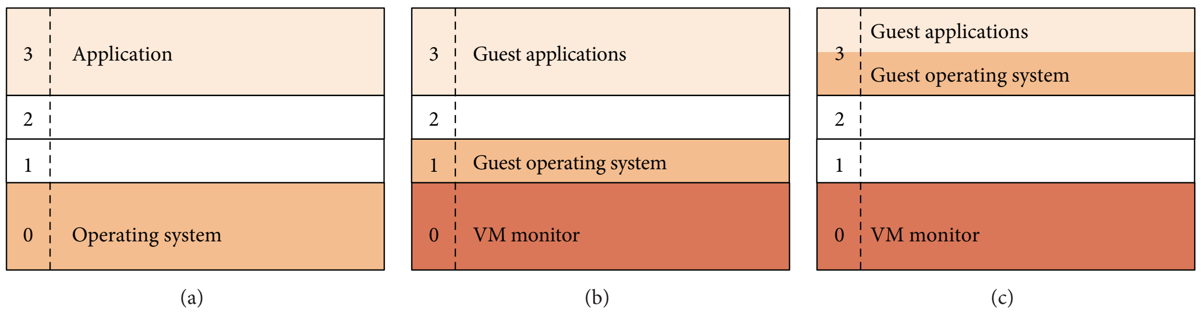

Figure 5: Privileged ring options for virtualizing CPU [33].

better system performance due to a reduction of trapping and instruction emulating overheads. However, it requires support from operating system vendors.

Binary Translation. This technique combines direct execution with on-the-fly binary translation. Typical applications running under CPU unprivileged mode can run using direct execution, while nonvirtualizable privileged code is run under control of the binary translator. The traditional binary translator was later improved for better performance. Instead of using a line-based translation, this translator translates the privileged code into an equivalent block with a replacement of problematic code and stores the translated block in the cache for future uses.

4.4.2. Memory Virtualization. A software technique for virtualizing memory which is to have VMM maintains a shadow version of a guest's page table and to force CPU to use the shadow page table for address translation [34]. The page table contains mappings of virtual addresses (used by processes) and physical addresses of the hardware (RAM). When a virtual address is translated to a physical address, CPU first searches for a corresponding address in a translation lookaside buffer (TLB). If a match is not found, it then searches in a page table. If a match is found, the mapping is written into TLB for a future translation. In case the lookup fail, a page fault interruption is generated.

To maintain a valid shadow page table, VMM must keep track of the guest's page table and update corresponding changes to the shadow page table. Several mechanisms are designed to ensure the consistency of page tables [57].

Write Protect. One technique is to write protect the physical memory of the guest's page table. Any modification by the guest to add or remove a mapping, thus, generates a page fault exception, and the control is transferred to VMM. The VMM then emulates the operation and updates the shadow page table accordingly.

Virtual TLB. Another technique is Virtual TLB. It relies on CPU's page fault interruptions to maintain the validity of shadow page table. VMM allows new mapping to be added to the guest's page table without any intervention. When the guest tries to access the address using that mapping, the page fault interruption is generated (as that mapping does not exist in the shadow page table). The interruption allows VMM to add new mapping to the shadow page. In the case that the mapping is removed from the guest' page table, VMM intercepts this operation and removes the similar mapping from the shadow page table.

In addition, VMM needs to distinguish the general page fault scenario from the one associated to inconsistency of the shadow page table. This could result in significant VMM overhead.

4.4.3. I/O Virtualization. The goal of virtualizing I/O devices is to allow multiples VM to share a single host's hardware. Challenges in this area include scalability, performance, and hardware independence [58]. Existing approaches are as follows.

Device Emulation. Through device emulation, VMM intercepts an I/O operation and performs it at the hardware devices [59]. This technique does not require changes in an operating system and device drivers, but it generates significant overhead of context switching between VM and VMM.

Paravirtualization. Paravirtualization is introduced to reduce the limitation of device emulation. To achieve better performance, a guest operating system or device drivers is modified to support VMM interfaces and to speed up I/O operations [60, 61]. The performance of paravirtualization is better than the pure emulation approach, but it is significantly slower as compared to the direct access to the device.

Direct Access. An intermediate access through VMM is bypassed when using a direct access. VMs and I/O devices communicate directly through a separate channel processor. It results in significant elimination of virtualization overhead. However, the advantage of hardware independence is lost by having VMs tie with a specific hardware.

4.5. System Virtualization. When considering computing architecture, virtualization is generally implemented at two levels: at the application level (process virtualization) and at the bottom of software stack (system virtualization). Smith and Nair summarize the mechanism behind these basic types of virtualization [32].

Process virtualization is a basic mechanism used in multitasking operating systems. The OS virtualizes processing unit, registers, memory address space, and I/O resources for each process, so that multiple processes can run simultaneous without intervening each other. The OS maintains the isolation for each process instance (no intervention), maintains the state and context for each process instance, and ensures that each process receives a fair share of processing cycles.

This process is executed through scheduling and context switching. Through context switching, the OS switch in the value of CPU registers for the current process, so that it starts from the previous processing state. CPU is virtualized by scheduling algorithm and context switching. The memory is virtualized by giving an individual memory page table for each process. Interactions between a process and virtual resources are through ABI and API. In short, the goal of process virtualization is to allow multiple processes to run in the same hardware, while getting a fair share of CPU times and preventing intervention, such as access to memory, from other processes.

In *system virtualization* the whole computer system is virtualized. This enables multiple virtual machines to run isolated on the same physical platform. Each virtual machine can be either different or similar to the real machine. System virtualization is known as virtual machine monitor (VMM) or hypervisor. VMM operates above the physical resource layer. VMM divides the resources among VMs using static or dynamic allocation. In a static allocation, a portion of resources allocated to a specific VM is solely dedicated for that VM. For instance, each core of CPU might be fixedly allocated to each of the client VMs. Dynamic allocation manages entire resource as a pool. A portion of resources is dynamically allocated to a VM when needed and is deallocated to the pool when the job is done. Static allocation results in higher degree of isolation and security, while dynamic allocation helps to achieve better performance and utilization.

As mentioned, VMM provides system virtualization and facilitates the operations of VMs running above it. Several types of VMM could be found in the literature. In general they provide similar functionalities, but implementation details underneath are different, for instance, how the I/O resources are shared or how ISA translations are performed. Different approaches to system virtualization are as follows.

4.5.1. Bare-Metal or Native VMM. Native VMM runs on bare hardware and provides device drivers to support multiple VMs placed on top. Guest VMs might use similar or different ISA as an underlying platform. VMM runs in the highest privileged mode and gains an ultimate control over the resources. VMM needs to maintain its transparency to guest VMs, while providing them secured compartment. To achieve this, it intercepts and emulates guest privileged instructions (kernel related). Most of the guest applications could be run unmodified under this architecture. Examples of traditional VMM include XEN (http://xen.org/) and VMWare ESX (http://www.vmware.com/products/vsphere/esxi-and-esx/index.html).

Traditional VMM may virtualize a complete ISA to support guest VMs that use different ISA than the host platform. XEN uses paravirtualization, while VMWare ESX uses binary translation. Other VMMs might use a combination of both techniques or their improvement or neither. This depends on the hardware support, the collaboration from OS vendors, and system requirements [52].

4.5.2. Hosted VMM. An alternative to native VMM places VMM on top of the host operating system. The hosted VMM could be installed as an application. Another advantage of hosted VMM is that it relies on components and services provided by the host OS and virtualizes them to support multiple guest VMs. For instance, in contrast to traditional VMMs, hosted VMM uses device driver from host OS. This results in a smaller-size and less complex VMM. However, host VMM does not support different ISA guest VMs. Example of this type of VMM are Oracle VirtualBox (https://www.virtualbox.org/) and VMWare workstation (http://www.vmware.com/products/workstation/).

4.5.3. Codesigned VMM. Codesigned VMMs target at improving performance by compromising portability. It implements a proprietary ISA that might be completely new or is an extension of an existing ISA. To achieve performance, a binary translator translates guest's instruction to an optimized sequence of host ISA and caches the translation for future use. Codesigned VMM is placed in a hidden part of memory inaccessible by guest systems. Examples include Transmeta Crusoe and IBM iSeries.

4.5.4. Microkernel. Microkernel is a thin layer over the hardware that provides basic system services, for instance isolated address space to support multiple processes [52]. Microkernel serves as a base for virtualization, in which provisioning application could be deployed upon to provide a complete system virtualization. The OS could also be paravirtualized and run directly on the microkernel to increase the performance.

4.6. Use Scenarios and Benefits. The benefits of virtualization could be derived from its usage model. Uhlig et al. identify three main use scenarios of virtualization and their respective benefits as follows.

Workload Isolation. Main benefits as virtualization is used for workload isolation (Figure 6(a)) are as follows: (a) security enhancement, as compromise to security is limited to a specific compartment; (b) reliability enhancement, a fault in one module that might generate a system failure in one VM does not affect others VM running on the same physical block; (c) fault tolerance, as virtualization allows control over the state of VM through suspend, resume, mark a checkpoint, and roll back.

Workload Consolidation. In many cases virtualization is used for server consolidation (Figure 6(b)). Its benefits for this use case are (a) *reduction of server maintenance cost*. One solution to increase system reliability is to run single-OS and single application on a server. It leads to a situation that a company needs to maintain a proliferation of underutilized servers to support different types of applications. Virtualization could be used to consolidate individual server into a single platform, increasing utilization and reducing maintenance cost.

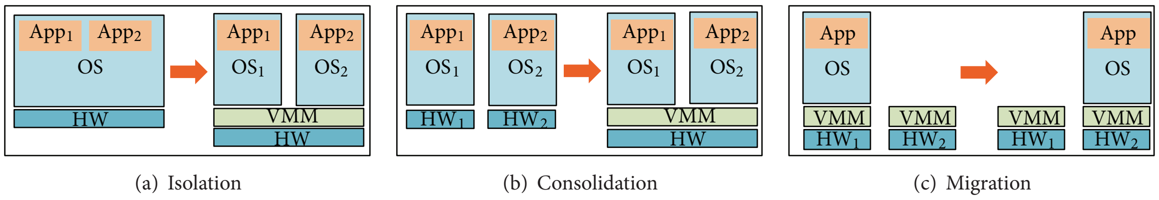

FIGURE 6: Three usage models of virtualization [33].

(b) *Supporting incompatible legacy systems*, as virtualization enable legacy and updated OS to run concurrently on the same platform.

Workload Migration. Figure 6(c) illustrates a case that virtualization supports system migration. It allows hot migrations and load balancing to be performed easily and effectively, as the state of the entire system is encapsulated within the VM and decoupled from the physical platform.

4.7. Summary. This section gives a review on virtualization technology, especially on system-level virtualization which is used to manage a resource pool in cloud computing environments. Virtualization has been used since 1960 in mainframe environments for a system isolation purpose. Before the introduction of cloud computing, virtualization provides an efficient solution to consolidate underutilized servers. The benefits of server consolidation, including resource optimization, a reduction of energy consumption, and a maintenance cost reduction, drive its large adoption. System virtualization software, known as a virtual machine monitor (VMM), allows multiple guest systems to share a similar hardware system. To enable this sharing, three main hardware elements, that is, CPU, memory, and I/O devices, are virtualized. Based on architecture and supports from operating system vendors, CPU virtualization could be done in several ways. Its goal is to multiplex CPU cycles and to remain invisible to guest systems. VMM direct execution requires VMM to run in a higher privileged level than the guest's OS and applications, in order to trap guest's instructions and emulate CPU responses when necessary. Paravirtualization ports an OS directly to a specific CPU architecture, resulting in better performance. The direct execution could be combined with line-based or block-based binary translation to remove the dependency on OS vendors. I/O virtualization appears to be the most problematic one that causes significant performance overhead.

5. Cloud Data Management

The goal of this section is to explore cloud storage solutions and understand impacts of their design decisions to system characteristics. Scalable storages known as NoSQL are becoming popular, as it solves performance problems of relation databases when dealing with big data. The differences between NoSQL and relational databases including their implementation details are analyzed in this section.

5.1. Data Management Basis. For several decades traditional ways of storing data persistently have been through relational databases or file systems [38]. An alternative to the traditional approaches has been introduced in recent years under the name of NoSQL [62]. A number of cloud storage solutions, for instance, Google Bigtable [63], Yahoo's PNutts [50], Amazon's SimpleDB (http://aws.amazon.com/simpledb/), Cassandra [64], and CouchDB (http://couchdb.apache.org/) belong to this category. This section summarizes the characteristics of relational and NoSQL databases.

5.1.1. Relational Databases. Relational database is a common term for relational database management systems (RDBMS). RDBMS represents a collection of relations and mechanisms that force a database to conform to a set of policies such as constrains, keys, and indices. A relation (table) consists a number of tuples (rows) that have a similar set of attributes (columns). In other words, a relation represents a class and tuples represent a set of objects which belong to the same class. The relation is defined using data schema which describes a name and a data type for each of the attributes.

RDBMS provides operations to define and adjust a schema for table and enforces that data stored in the relation are strictly conformed to the schema. Relations can be modified through *insert*, *delete*, and *update* operations. Operations across relations are provided based on set operations including *union*, *intersection*, and *difference*. *Selections* of tuples from relations with a specific criterion, *projection*, *join* of multiple relations could be done through query languages. RDBMS supports *data indexing* to improve query performance. It also provides a concept of *foreign keys* to maintain data integrity.

Characteristics and Limitations. RDBMS was designed primarily for business data processing [65]. It supports a concept of *transactional operations* to serve this purpose. This support guarantees that a unit of work (transaction) performed in a database must be done successfully, or otherwise all the executed operations in the same unit of work must be cancelled [66]. This proposition is also known as all-or-nothing. RDBMS supports flexible query languages including expensive operations such as multiple joins and range queries.

However, strong data consistency and complex queries supported by RDBMS cause several limitations in terms of scalability, performance, and inflexibility of data schema.

(a) *Scalability.* Automatic partitioning which is a key for performance scalability could not be done naturally in RDBMS due to the guarantee on transactional consistency and complex data access operations it provides [67, 68].

(b) *Performance.* As RDBMS was not originally designed for distributed environments, the way to improve its performance is through architectural changes and reducing operational overheads on a single server [62]. The performance issue of RDBMS is caused by the fact that its core architecture was designed for more than 30 years when hardware characteristics, including processor performance, memory and disk space, were much different than today [65].

(c) *Inflexibility of Schema Changes.* Current markets for data storages have evolved from business data processing to other areas, such as stream processing and text management, in which the data do not necessarily conform to a strict database schema.

5.1.2. NoSQL Databases. As an alternative to RDBMS, NoSQL databases offer scalable distributed data tier for large scale data management [38]. The term NoSQL was firstly used in 1998 as a name for an open-source database that does not offer SQL interfaces. It was reintroduced in 2009 referring to nonrelational distributed database model [69].

There are two categories of NoSQL databases [62, 69], as follows.

Key-Value Stores. Data are stored as an array of entries, where a single entry is identified through a unique key. Common operations are deletion, modification or read an entry of a given key, and insertion of a new key with associated data. Key-value stores are implemented by distributed hash tables. Examples include Amazon Dynamo and MemcacheDB.

Document-Oriented Stores. Document-oriented storages represent loosely structured data storages where there is no predefined schema or constrains limiting databases to conform to a set of requirements. Records can contain any number of attributes. Attributes could be added, removed, and modified in a flexible manner without interrupting ongoing operations. Examples include MongoDB and CouchDB.

NoSQL databases differ significantly at the implementation level, for instance, data models, update propagation mechanisms, and consistency scheme. However, several characteristics and features are common for NoSQL systems [62, 67, 70]. First of all, it is designed for distributed environments, where data are horizontally partitioned and replicated across multiple sites. Second, the system uses nonrelational data models, allowing flexibility over schema changes. Third, it provides a restrictive set of queries and data operations, most of which are based on a single row. Fourth, on the contrary to a strong consistency guaranteed by relation databases, NoSQL systems often tradeoff the consistency to yield higher availability and better response time.

5.2. Foundation. This section covers a concept of database transactions which is guaranteed through ACID properties and mechanisms to support ACID in distributed systems.

5.2.1. ACID Properties. The ACID is fundamental principle of database system [66]. It contains the following properties.

(i) *Atomicity*. All actions of a transaction is executed and reflected in the database, or the entire transaction is rolled back (all or nothing).

(ii) *Consistency*. A transaction reaches its normal state, committing only legal results and preserving the consistency of the database.

(iii) *Isolation*. Events within a transaction are hidden from other transactions running concurrently, allowing the transaction to be reset to the beginning state if necessary.

(iv) *Durability*. Once a transaction has been completed, results have been committed to database, and the system must guarantee that the modification is permanent even in the case of subsequent failures. The durability is ensured by the use of transaction logs that facilitate the restoration process of committed transaction if any failure occurs.

Based on the ACID properties, a transaction can be terminated in three ways. First, it successfully reaches its commit point, holding all properties true. Second, in a case that bad input or violations that prevent a normal termination has been detected, all the operations that have been executed are reset. Finally, the transaction is aborted by the DBMS in the case of session time-out or deadlock.

5.2.2. CAP Theorem. Web services are expected to provide strongly consistent data and to be highly available. To preserve consistency they need to behave in a transactional manner; that is, ACID properties are respected [49]. The strong consistency is particularly important for critical and financial systems. Similarly, the service should be available whenever it is needed, as long as the network on which it runs is available. For distributed network, however, it is desirable that the services could sustain through a certain level of network failures.

It is challenging in general to maintain the ACID properties for distributed storage systems, as the data are replicated over geographic distances. In practice, it is impossible to achieve three desired properties in the same time [49, 71]. We could get only two out of the three. CAP theorem describes trade-off decisions needed to be made when designing highly scalable systems. It is related to three core system requirements as follows.

(i) *Consistency (atomic consistency)*. The notion of consistent services is somewhat different than consistent property of database systems, as it combines the database notion of atomicity and consistence. The consistency enforces that multiple values of the same data is not allowed.

(ii) *Availability*. Requests to a nonfailure node must result in a response, instead of a message about a service being unavailable.

(iii) *Partition tolerance*. When data and logic are distributed to different nodes, there is a chance (which is not rare) that a part of network becomes unavailable.

This property guarantees that "*no set of failures less than total network failure is allowed to cause the system to respond incorrectly.*"

To deal with CAP, the designer has an option of dropping one of three properties from system requirements or improving an architectural design. The scenarios of dropping one of the CAP are, for instance, running all components related to the services on one machine (i.e., dropping partition tolerance); waiting until data of every replicated node become consistent before continuing to provide the services (i.e., dropping availability); or accepting a notion of weaker consistency.

CAP implies that if we want a service to be highly available (minimal latency) and we want the system to be tolerant to network partition (e.g., messages lost, hardware outages), then sometimes there will be a case that the values of the data at different nodes are not consistent.

5.2.3. Consistency Scheme. As mentioned, dealing with the consistency across replicas is one of challenges of distributed services. A variety of consistency models have been proposed to support applications that can tolerate different levels of relaxed consistency [51]. The following defines well-known classes of consistency.

(i) *Strong consistency*. After the update is complete, subsequent accesses to any replicas will return the updated value.

(ii) *Eventual consistency*. The system guarantees that if there is no new update to the object, the subsequent accesses will eventually return the updated value. The degree of inconsistency depends on communication delay, system workload, and the number of replicas.

(iii) *Timeline consistency*. Timeline refers to a scenario in which all the replicas of a given record apply all updates to the record in the same order. This is done by using a per record mastership mechanism. With this mechanism the replica that receives the most frequent updates for each record is set as a master for that record [50]. As the updates are performed in an asynchronized manner, the system provides various data retrieval APIs that support different consistency levels. This mechanism is introduced by Yahoo!.

(iv) *Optimistic consistency (weak consistency)*. The system does not guarantee that the subsequence access will return the updated value.

5.2.4. Architectural Tradeoffs. A number of cloud storage systems have emerged in the recent years. A lack of a standard benchmark makes it difficult to understand the design tradeoffs and quality of services; each of them provides to support different workloads. To this end, Cooper et al. summarize main tradeoffs the providers face during the architectural design which impact the CAP property of the system and applications relying on it [72].

Read Performance versus Write Performance. Different types of applications (i.e., latency sensitive applications at one end and throughput oriented applications at another end) needs different tradeoffs between optimizing for read and write operations. Several design decisions for these operation exists. An *update* could be written to a target file for each single operation or could be written later as a group update. A *log* could be recorded on a row basis where a complete row is written, or it could be stored as a log-structured system where only an update delta is recorded. The structured log can be inefficient for reads as all the updated must be merged to form a consistent record but it provides a lower cost on updates. An *access mechanism*, that is, sequential and random access, should also be suitable for a specific nature of applications.

Latency versus Durability. Updates could be written to disk before it returns success to users, or it could be buffered for a group write. In cases that multiple updates could be merge to a single I/O operation, the group write results in a lower latency and higher throughput. This advantage comes with a risk of losing recent updates when a server crashes.

Synchronous versus Asynchronous Replication. The purpose of synchronization is to ensure that data stored in all replicas are updated and consistent. An algorithm for synchronizing replicas determines a level of consistency, availability (through a locking mechanism), and response time.

Data Partitioning. A storage could be strictly row-based structured or allows for a certain degree of column storage. Row-based storage is efficient for accessing a few records for their entirely content, while column-based storage is efficient for accessing multiple records for their certain details.

The concept of "one-architecture-fits-all" does not suit for distributed storage systems. However, the ultimate goal for each design is to maximize key properties of storage system: performance, availability, consistency, and durability. Different design decisions reflect in the features and quality of services provided by different storage providers which are discussed in the next section.

5.3. Selected Cloud-Based Data Management Services. Cloud providers offer a number of solutions for very large data storages. It is necessary to understand the mechanisms that each solution applies to enforce system requirements. This includes, for instance, how the data are partitioned across machines (elasticity), how the updates are routed (consistency), how the updates are made persistent (durability), and what and how failures are handled (availability).

5.3.1. Google Bigtable. Google Bigtable is a distributed data storage designed to scale to a very large size, to be fault tolerant, and to support a wide range of applications. It is designed to support applications which require different workload and performance. Bigtable is used by more than 60 products of Google, such as search engine, Google Docs, and Google Earth. Chang et al. describe architecture of Bigtable as summarized in this subsection [63].

(1) *Data Model*. Bigtable is a sparse, distributed, and multidimensional sorted map. The map is indexed by a combination

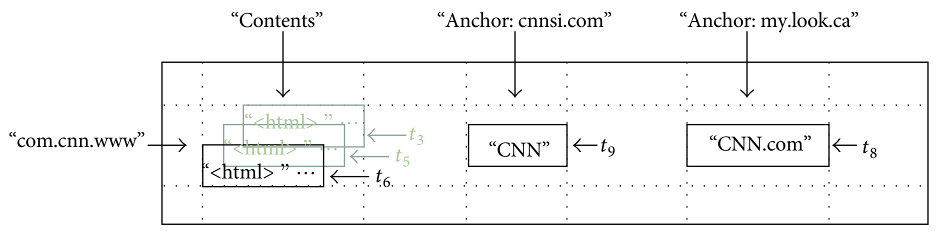

Figure 7: An example a Bigtable that stores a webpage [63].

of row key, column key, and timestamp. This design makes it convenient to store a copy of a large collection of web pages into a single table Figure 7.

Row. Read or write of data is atomic under a single row key disregard of a number of columns involved. A row range of the table is partitioned into a *tablet* which represents a unit of distribution and load balancing. A read of short row ranges is sufficient and requires communications only with few machines.

Column Family. A column key is formed by a syntax *family (qualifier)*. A column family is a group of associated column keys. A table can have unbounded number of columns, but generally a number of column families are in hundreds at most. An access control is performed at the column family level.

Timestamp. Each cell of Bigtable can contain multiple versions of the same data, indexed by a timestamp.

(2) *System Model*. Bigtable is built on top of several components of Google infrastructure.

Distributed File System (DFS). Bigtable uses DFS to store logs and data files. A cluster of Bigtable is controlled by a cluster management system which performs job scheduling, failure handlings, consistency controls, and machine monitoring.

Google SSTable. SSTable is an immutable sorted map from keys to values used by DFS to store chucks of data files and its index. It provides look up services for key/value pairs of a specific key range. Figure 8 depicts how a tablet is chucked under SSTable.

Chubby Distributed Lock Service. Chubby is a highly available persistent distributed lock service. It consists of five active replicas, one of which is assigned to be a master to handle service requests. The main task of Chubby is to ensure that there is at most at a time only one active master which is responsible for storing a bootstrap location of Bigtable data, discovering tablet servers and storing Bigtable schemas and access control lists.

A structure of Bigtable consists three main components: a master server, tablet servers, and a library that is stored at the client-side. The master server assigns a range of tablets to be stored at each tablet server, monitors servers' status, controls load balancing, and handles changes of the table schema. The tablet servers could be added or removed with response to the current workload under the control of the master. Read and write of records are performed by the tablet server. Thus, clients communicate directly to a tablet server to access their data. This architecture eliminates a bottleneck at the master server, as clients do not rely on it for tablet location information. Instead, this information is cached in a library at the client side.

(3) *Consistency and Replication Mechanism*. A large number of Google products rely on Google File System (GFS) for storing data and replication. While sharing the same goals as other distributed storage systems (such as fault tolerance, performance, and availability), GFS are optimized for the following use scenarios [23]: (a) component failures are treated as a norm rather than exceptions; (b) files are large and consist in a large number of application objects; (c) in most occasions files are changed by appending new data rather than changing existing details; and (d) GFS APIs are accustomed to suit application requirements.

GFS uses a relaxed consistency model which implies the following characteristics [23].

Atomicity and Correctness. File namespace mutations are atomic and are exclusively handled by the master server. A global order of operations is recorded in the master's operation log, enforcing the correct execution order for concurrent operations. Possible states of mutated file are summarized in Table 2. A file region is *consistent* if clients get the same data regardless of which replica has been accessed. A file region is *defined* if after the mutation it is consistent and clients see the changes from all replicas. A file region is *inconsistent* if different clients see different data when reading from different locations.

A mutation could be a write or a record append. A write mutation writes data at an application-specific file offset. An append mutation appends the data atomically at least once at the offset determined by GFS and returns that offset to the client. When the write mutation succeeds without interferences from concurrent writes, the file region is defined. In existence of concurrent writes, successful write mutations leave the region consistent but undefined; that is, data are consistent across replicas, but they might not contain the data written by the last mutation. Failed write mutations result in an inconsistent file region. With regards to an append mutation, GFS guarantees that the data must have been written at the same offset on all replicas before the operation reports success. The region in which the data

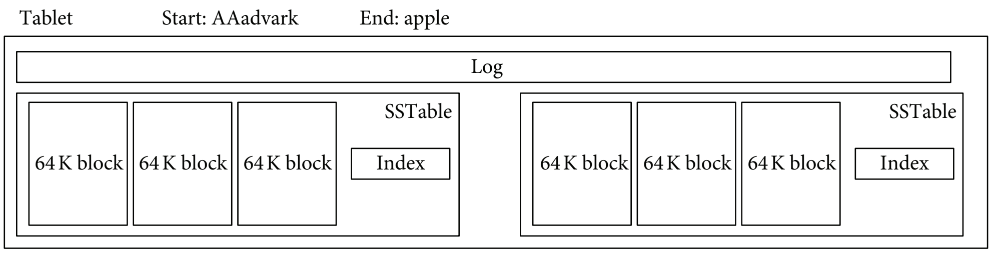

Figure 8: Tablet.

Table 2: File region state after mutation [23].

	Write	Record append
Serial success	Defined	Defined interspersed with inconsistent
Concurrent success	Consistent but undefined	
Failure	Inconsistent	

have been written successfully is defined, while intervening regions are inconsistent.

Updates for Replicas. GFS guarantees that a file region is defined after successful mutations by using the following update sequence: (1) mutations are applied to a file region in the same order on all replicas; (2) stale replicas (the ones have missed mutation updates during the time that chunkserver is down) is not involved during the update, thus their locations are not given to clients.

5.3.2. Yahoo PNUTS. Yahoo PNUTS is a massively parallel distributed storage system that serves Yahoo's web applications. The data storage is organized as an ordered table or hash table. PNUTS is designed based on an observation that a web application mostly manipulates a single data record at a time, and activities on a particular record are initiated mostly at a same replica. This design decision reflects on its guaranteed consistency and replication process. Cooper et al. describe the architecture of PNUTS as explained in this subsection [50].

(1) *Data Model*. PNUTS presents a simplified relational model in which data are organized into a table of records (row) with multiple attributes (column). In addition to typical data types, it allows arbitrary and possibly large structured data, which is called *bulb*, to be stored in a record. Examples of bulb objects include pictures and audio files. Schema changes are flexible, such that new attributes can be added without interrupting ongoing activities. The system does not enforce referential constrains, and some of the attributes could be left unfilled.

The query language of PNUTS is more restrictive than those supported by relation models. It allows selections and projections over a single table. A primary key is required for updates and deletes of records. In addition, it provides a multiget operation which retrieves up to a few thousand records in parallel based on a given set of primary keys. Currently the system does not support complex and multi-table queries such as join and group by operations.

(2) *System Model*. PNUTS system is divided into multiple regions, each of which represents a complete system and contain a complete copy of tables. It relies on a pub/submechanism for replication and update propagations. The architecture of PNUT with two regions is illustrated in Figure 9. Each region contains three main components: storage units, tablet controller, and routers. Message brokers control replications and consistency among different regions.

Storage Unit. In terms of storage, tables are partitioned horizontally into a number of tablets. A tablet size varies from hundred megabytes to few gigabytes. A server contains thousands of tablets. The assignments of tablets to servers are optimized for load balancing. PNUT supports two types of storage structures: ordered tables and hash tables.

Router. In order to localize a record to be read or written, a router determines a tablet that contains the requested record and a server that stores that tablet. For ordered tables, the router stores an interval mapping which defines boundaries of tablets and a map from tablets to a storage unit. For hash tables, the hash space is divided into intervals, each of which corresponds to a single tablet. Tablet boundaries are defined by a hash function of a primary key.

Tablet Controller. Although the process of record localization is performed by the router, it stores only a cached copy of the mapping. The whole mappings are maintained by the tablet controller. It controls load balancing and division of records over tablets. Changes of record locations cause the router to misroute the requests and trigger the new mapping retrieval.

(3) *Consistency and Replication Mechanism*. A consistency model of PNUTS comes from an observation that web applications often changes one record at a time, and different records have activities with different locality. PNUTS proposes *per record timeline consistency*, in which all replicas of a given record apply a series of update to that record in the same order. It also supports a range of APIs for various degrees of consistency guarantees. To implement this mechanism, one of the replicas is appointed as the master independently for each record. The updates to that record are sent to the master. The record master is automatically adjusted to the replica that receives the majority of write requests.

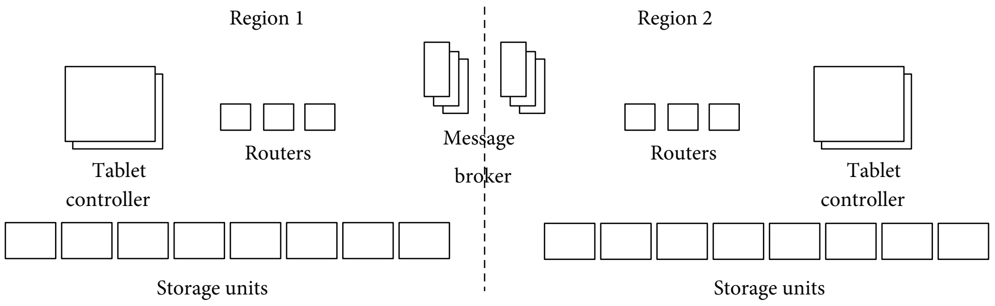

Figure 9: PNUTS's architecture.

The system currently provides three variations of data accesses: *read any* (no consistency guaranteed), *read critical* (guarantee that the returned record is at least as new as a given version), and *read latest* (strong consistency guaranteed). It provides two variations of write operations: *write* (transactional ACID guaranteed), *test-and-set-write* (the write is performed if and only if the record holds the same version as the given one).

5.3.3. Amazon Dynamo. Amazon has developed a number of solutions for large scale data storages, such as Dynamo, Simple Data Storage S3, SimpleDB, and Relational Database Service (RDS). We focus on the architecture of Dynamo because it has served a number of core e-commerce services of Amazon that need a tight control over the trade-offs between availability, consistency, performance, and cost effectiveness. The architecture of Dynamo is described by DeCandia et al. [73].

(1) *Data Model*. Dynamo is classified as a distributed key-value storage optimized for high availability for write operations. It provides read and write operations to an object which is treated as an array of bytes uniquely identified by a primary key. The write operation requires that a context of the object is specified. The object's context encodes system metadata, such as a version which is necessary for validity checks before write requests could be performed.

(2) *System Model*. Dynamo uses consistent hashing to distribute workloads across multiple storage hosts. Consistent hashing is a solution for distributed systems in which multiple machines must agree on a storage location for an object without communication [74]. Through this partitioning mechanism, the hash space is treated as a ring in which a largest hash value is connected to the smallest one. Each storage node receives a random value which determines its position on a ring. An assignment of an object to a storage node is done by hashing an object's key which results in a position in the ring and walking the ring clockwise to the first larger node than the hash value.

Dynamo replicates objects into multiple nodes. Once a storage node is determined, that node (known as *coordinator*) is responsible for replicating all data items that fall into its range to the successor nodes in the ring. Figure 10 illustrates a storage ring in which workloads are partitioned among node A to G. Suppose that the hash function of key K of an object falls between a location of node A and B, leading the object to be stored at node B. Node B is in charge of replicating this object to node C, D. In other words, node D will store copies of data that their key falls between A and D.

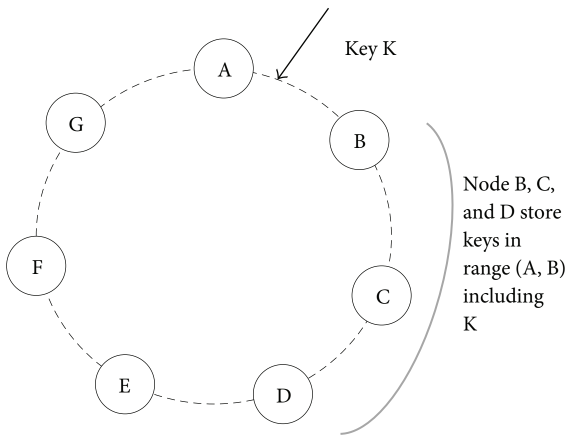

Figure 10: Partitioning and replication model of Amazon Dynamo.

(3) *Consistency and Replication Mechanism*. Dynamo provides eventual consistency, as updates are propagated to replicas in an asynchronous manner. A write request responses to the caller before the update is successfully performed at all the replicas. As a result, a read request might not return the latest update. However, applications that rely on this system can tolerate the cost of relaxed consistency for the better response time and higher availability.

Dynamo uses a vector clock to handle different versions of the object during the update reconciliation. A write request is attached with object's context metadata which contains the vector clock information. This data is received from the earlier read. Read and write operations are handles by the coordinator node. To complete a write operation, the coordinator generates the vector clock for the new version, writes the object locally, and propagates the update to highest-ranked nodes which are preferable locations for storing that object. In addition, the system allows clients to define a minimum number of nodes that must participate in a successful write. Each write request is successful when the coordinator receives at least that minimum number of responses from the target nodes.

5.4. Summary. Cloud storage solutions are reviewed in this section. We begin the section by clarifying the fundamental differences between relational and scalable NoSQL databases. The name NoSQL comes from its lack of supports on complex query languages and data integrity checks. NoSQL works well with big data because the data are partitioned and stored in distributed nodes. It generally maintains better performance as compared to relation databases by compromising strong data consistency and complex query supports. CAP theorem explains the dependencies between three desired properties-consistency, availability, and partition tolerance. To deal with CAP, one property needs to be dropped from a system property, resulting in having a CA, CP, or AP as a main architectural driver. Consistency schemes ordered from the strongest to and weakest one are strong, eventual, timeline, and optimistic accordingly. While strong consistency guarantees that accesses to any replica return a single most updated value, optimistic consistency does not guarantee whether or not an access to a certain replica returns an updated value at all. Three scalable storage models used by leading cloud providers, that is, Google Bigtable, Yahoo PNUTS, and Amazon Dynamo, are distributed key value stores that supports relaxed consistency and provide low level APIs as query interfaces. They are in fact significantly different in architecture and implementation details. The diversity of data management interfaces among providers might result in data lock-in which is one of the primary concern of its adoption.

6. Security

The objective of this section is to provide an insight to security concerns associated to cloud computing. To achieve it, first of all, we point out security vulnerabilities associated to the key architectural components of cloud computing. Guidance to reduce the probability that vulnerabilities would be exploited is described. In the end we present a mapping of cloud vulnerabilities and prevention mechanisms.

6.1. Vulnerabilities to Cloud Computing. The architecture of cloud computing comprises an infrastructure and a software operating on the cloud. Physical locations of the infrastructure (computing resources, storages, and networks) and its operating protocol are managed by a service provider. A virtual machine is served as a unit of application deployment. Underneath a virtual machine lies an additional software layer, that is, virtualized hypervisor, which manages a pool of physical resources and provides isolated environments for clients' virtual machines.

Vulnerability, threat, and risk are common terms in the context of security which are used interchangeably, regardless of their definitions. To remain consistent we provide the definition of these terms as follows: (i) Vulnerability is defined by NIST as "*a flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised and result in security breach or a violation of the system' security policy.*" (ii) Threat is defined by ENISA as "*a circumstance or event with potential to adversely impact an asset through unauthorized access, destruction, disclosure, modification, and/or denial of services.*" (iii) Risk is defined in ISO27005 based on the previous terms as "*the potential that a given threat will exploit vulnerability of an asset or group of assets and thereby cause harm to an organization.*"

Vulnerabilities to cloud computing arise from flaws or weaknesses of its enabling technologies and architectural components [10]. A simplified cloud stack model provides a good basic for vulnerability identifications. As Section 2 explains, clouds are abstracted into five layers including: a software layer, a platform layer, an infrastructure layer, a unified resource layer, and a fabric layer. We explain cloud security by discussing the vulnerabilities associated to system procedure, design and implementation, and controls of each component at each layer.

Vulnerabilities and security requirements of cloud services are identified in a number of literatures. In the work of Grobauer et al. the influences of cloud computing on established security issues are analyzed [10]. Cloud specific vulnerabilities are identified in association to the core technologies, essential characteristics, and architecture of cloud computing. The discussion on the last area is based on a reference architecture proposed by University of California and IBM [21]. The mapping to those architectural components, in turn, points out the relevant vulnerabilities to specific cloud services' consumers.

Through his work "Monitoring Cloud Computing by Layer," Spring presents a set of restrictions and audits to facilitate cloud security [75, 76]. The discussion is organized around a seven-layer cloud model proposed by the cloud security alliance. These layers are either controlled by cloud providers or consumers. The line of responsibility is generally clarified in terms of SLA. The list of security controls by layer assist SLA formation and security monitoring.

Iankoulova and Daneva performed a systematic review to address requirements of SaaS security and propose solutions to deal with it [77]. Their selection criterion leads to identifying 55 papers for a detailed analysis. The discussion is organized into nine areas following the taxonomy of security quality factors. These include access control, attack/harm detection, nonrepudiation, integrity, security auditing, physical protection, privacy, recovery, and prosecution. Top areas for cloud security research include integrity, access control, and security auditing, while the issues related to physical protection, nonrepudiation, recovery, and prosecution are not broadly discussed.

In this subsection we shortly review the functionalities and components at each cloud layer and point out a number of vulnerabilities and subvulnerabilities related to them.

6.1.1. Application Layer. At the highest layer, SaaS is offered to consumers based on web technologies. The reliability of web services (backend), web browsers (frontend), and authentication and authorization mechanisms influences service security.

Vulnerabilities of Web Services (v1). The vulnerabilities of web services are associated to the implementation of a session handler and the mechanism to detect inputs which creates erroneous executions.

(i) *Session-handling vulnerability (v1a).* Http is a stateless protocol, while a majority of nontrivial applications and transactions require the notion of state. The state is handled by a session-handling mechanism. An inappropriate implementation of the session-handler might introduce vulnerabilities on session riding and session hijacking, where the credential of the message originator is stolen and misused [10].

(ii) *Injection vulnerability (v1b).* This vulnerability is exploited by manipulating service requests or input to an application with an intension to create an erroneous execution at the backend. Injections occur in three forms: SQL injection (targeting at backend database), command injection (targeting at backend operating system), and cross-site scripting (targeting at a victim's web browser) [10].

Vulnerabilities of Client-Side Data Manipulation (v2). Client-side data manipulation vulnerability is caused by inappropriate permissions given to web browser components such as plug-ins and mash-ups, allowing these components to read and modify data sent from the web application to the server [10, 78]. Attacks targeting at this vulnerability impact information confidentiality and integrity.

Vulnerabilities of Authentication and Authorization (v3). The vulnerability of authentication and authorization refers to the weaknesses in a credential verification process and the management of credential information from both provider and user sides. A variety of consumer population increases the complexity in managing the controls in this area [79]. Moreover, an enterprise authentication and authorization framework does not generally cover the integration of SaaS [80].

(i) *Authentication vulnerability (v3a).* It includes insecure user behaviors (for instance, reused credentials, weak password), the usage of one-factor authentication, and a weak credential changing and resetting process.

(ii) *User authorization scheme vulnerabilities (v3b).* It is caused by insufficient authorization checks on programming interfaces, coarse authorization scheme which does not support delegation of user privileges (to access only required resources).

Vulnerabilities of Encryption and Keys (v4). The use of a weak encryption algorithm, an insufficient key length, or an inappropriate key management process introduces encryption vulnerability.

(i) *Weak cryptography vulnerability (v4a).* Keys and encryption are crucial to protect confidentiality and integrity of data in shared environment. The use of insecure and obsolete cryptography is vulnerable in cloud environments especially for public clouds.

(ii) *Key management vulnerability (v4b).* Key management concerns how the encryption keys are created, stored, transferred, backed up, and recovered. Compromise in this process results in information leakage and a loss of data those keys protect.

6.1.2. Platform Layer. A virtualized cloud platform is composed of a layer of development and deployment tools, middleware, and operating system (OS) which are installed on a virtual machine. The security of platforms relies on the robustness of each component, their integration, the platform management portal, and provided management APIs.

Vulnerabilities of Cloud Operating Systems (v5). An OS arranges communications (system calls) between applications and hardware, therefore it has access to all data stored in the virtual machine. Data leakage could occur when an OS contains malicious services running on the background. For PaaS, a common practice of providers to secure cloud OS is to deploy single, harden, pared-down OS, and monitor binary changes on the OS image [75]. However, this mechanism is not applied to IaaS where an OS monitoring process is performed individually by each client.

Vulnerabilities of Access to Platform Administrative and Management Interfaces (v6). PaaS providers must provide a portal which allows system administrators to manage and control their environments. These administrative and management accesses share a similar set of web services vulnerabilities and the vulnerabilities of identity authentication and authorization control [10].

6.1.3. Infrastructure Layer. Cloud infrastructure is transferred to consumers in terms of a virtual machine (VM) that provides computing capability, memory, storages, and connectivity to a global network. Vulnerabilities to cloud infrastructure are caused by traditional VM weaknesses, management of VM images, untraceable virtual network communications, data sanitization in shared environments, and access to infrastructure administrative and management APIs.

Vulnerabilities of a Virtual Machine and Its Interfaces (v7). Apart from the vulnerability residing in a VM image itself, this issue is concerned with the common practice of a provider to offer cloned VM images for IaaS and PaaS consumers and the management of these images.

(i) *Traditional VM vulnerability (v7a).* Data centers are traditionally protected by perimeter-security measures such as firewalls, security zone, intrusion prevention and detection tools, and network monitoring. In contrast, security controls in virtual environments must be implemented also at the VM level. Vulnerable areas of VM include the remote access to administrative interfaces, the ease of reconfiguration that might propagate unknown configuration errors, and *patch* *management* which is difficult, if not impossible, to maintain by the providers to ensure the compatibility for all VMs' configuration [81].

(ii) *Cloned VM image vulnerability (v7b).* This vulnerability is concerned with how VM images are handled. The usage of cloned VM and OS images as an abstract platform is a common practice for PaaS consumers [78]. As attackers could easily register for IaaS and gain access to the image, they could track useful knowledge related to the drawbacks and limitations of that type of VM such as determining the channel of data leakage and might be able to attack consumers using the same product [10, 81].

(iii) *Insecure VM image vulnerability (v7c).* Another issue is that an IaaS consumer might obtain and use manipulated VM images from a notorious source. In this case, attackers gain a back-door access to the victim's platform to conduct their malicious activities [10].

Vulnerabilities of Virtual Network Communications (v8). Virtual communication channel vulnerability is concerned with the communications between VMs on the same physical resources. Generally a VM provides the capability for users to configure virtual network channels in order to directly communicate with another VM running on the same physical platform. The messages sent through such channels are invisible to network monitoring devices and therefore untraceable [78].

Vulnerabilities of Data Sanitization (v9). As previously mentioned, data in use, in transit and at rest must be secured. Access control and encryption are employed to protect data at rest. Secure network protocol, encryption, and public key are used to protect data in transit. However, the mechanism of freeing resources and removing sensitive data is overlooked, even though it is equally important in a shared environment. *Data sanitization vulnerability* is related to deletion of data, applications, and platform instances in the end of their life cycle, which is more difficult to perform when the physical resources are shared with other tenants [10, 78]. Flaws in data sanitization could result in data leakage.

Vulnerabilities of Access to Infrastructure Administrative and Management Interfaces (v10). IaaS must provide interfaces allowing administrators to perform their administrative and management activities. This includes remote accesses to utilize and configure computing services, storages, and network. These accesses share a similar set of web services vulnerabilities and the vulnerabilities of identity authentication and authorization control [10].

6.1.4. Unified Resource Layer. Multitenancy is a core mechanism of cloud computing to achieve the economies of scale. A virtualized hypervisor is designed to operate multiple VMs on a shared platform and therefore leads to resource optimization, decreased energy consumption, and cost reduction. There are, however, a number of vulnerabilities associated to hypervisors and the multitenancy architecture.

Vulnerabilities of a Virtualized Hypervisor and Its Interfaces (v11). An insecure implementation of a hypervisor causes several vulnerabilities as follows.

(i) *Vulnerabilities of a complex hypervisor (v11a).* A simple, small-sized, and feature-limited hypervisor is generally more robust and easier to maintain. However, several hypervisors are adapted from an operating system and combined with advanced features. A proper tradeoff between simplicity and functionality is essential in order to maintain its robustness [78].

(ii) *Vulnerabilities of access to administrative and management interface (v11b).* A hypervisor provides APIs and web portals, allowing cloud administrators to perform their activities. These accesses share a similar set of web service vulnerabilities.

Vulnerabilities of Multitenant Environments (v12). The concept of multitenancy holds different definitions based on an abstraction to which it is applied [16]. It generally refers to an architectural approach that leverages shared infrastructure, data, services, and applications to serve different consumers. Achieving secure multitenancy thus requires policy driven enforcement, segmentation, service level agreement, and billing models for different use scenarios. The followings point out its vulnerabilities.

(i) *Vulnerabilities of data leakage in multitenant environments (v12a).* The leakage could occur when the process of data sanitization does not completely remove the data stored in the physical resources before returning them to the share pool.

(ii) *Vulnerabilities of cross-tenant access (v12b).* Malicious cross-tenant access refers to an attempt of malicious code to escape from its compartment and to interfere with processes of other VMs running on the same tenant.

Vulnerabilities of Sharing Network Components (v13). In IaaS, it is likely that several VM instances share network infrastructure components such as DNS servers and DHCP servers [10]. Attacks to such components would create a cross-tenant impact.

6.1.5. Fabric Layer. This layer is concerned with physical security including servers, processors, storages, and network devices hosted in a data center.

Vulnerabilities of Physical Resources (v14). The physical security of cloud data centers is concerned with the following issues [78].

(i) *Malicious insider vulnerability.* Malicious insiders could introduce severe security threats, as they might gain physical access to the datacenter if a provider neglects in applying an appropriate privilege and access control.

Table 3: Cloud computing core technologies and associated vulnerabilities.

Layer	Functionality	Vulnerabilities
(1) Application	Provide services through web applications and web services.	(v1) Vulnerabilities of web services (v2) Vulnerabilities of client-side environments (v3) Vulnerabilities of authentication and authorization (v4) Vulnerabilities of encryption mechanisms and keys
(2) Platform	Provide programming interfaces and mediate communications between software and the underlining platform.	(v5) Vulnerabilities of a cloud platform (v6) Vulnerabilities of access to platform administrative and management interfaces
(3) Infrastructure	Provide computing and storage capabilities and connectivity to a global network.	(v7) Vulnerabilities of a virtual machine (v8) Vulnerabilities of virtual network communications (v9) Vulnerabilities of data sanitization (v10) Vulnerabilities of access to infrastructure administrative and management interfaces
(4) Unified resources	Three main features of hypervisors: operate multitenant virtual machine and application built up on it; provide isolation to multiple guest VMs; support administrative work to create, migrate, and terminate virtual machine instances.	(v11) Vulnerabilities of a virtualized hypervisors and its interfaces (v12) Vulnerabilities of multi-tenant environments (v13) Vulnerabilities of shared network components
(5) Fabric	Cloud physical infrastructure including servers, processors, storages, and network devices hosted in the data center.	(v14) Vulnerability of physical resources

(ii) *Natural disaster vulnerability.* Environmental concerns impact the availability of data and the continuity of services running on the cloud. A data center must have a comprehensive continuity of services plan in place, preferably conforming to an accepted standard.

(iii) *Internet access vulnerability.* This refers to an attempt for an unauthorized outsider to gain access to a datacenter through network channels.

A summary of the vulnerabilities associated to each of the cloud enabling technologies is presented in Table 3.

6.2. Common Threats to Cloud Security. Having identified the associated vulnerabilities to cloud computing, we are able to analyze to which extent they cover security breaches occurred and the concerns raised by current and prospect cloud consumers. In this section we present a list of threats to cloud security based on the cloud security alliance "*Top threats to cloud computing*" report [41].

Abuse of Cloud Computing. Based on its simple registration process and flexible usage model, clouds attract a number of attackers to host and conduct their malicious activities. Several malicious cloud adoptions have been found in recent years. Examples include *Zeus Botnet* which steals victims' credential and credit card information, *InfoStealer Trojan horses* designed to steal personal information, and *download for Microsoft Office and Adobe exploits*. It could be prevented by enforcing a strict initial registration which allows for sufficient identity validation.

Insecure Application Programming Interfaces. Clouds provide interfaces for consumers to use services and perform administrative and management activities such as provision, monitoring, and controlling VM instances. These features are generally designed as web services and thus inherit their vulnerabilities. Robust designs must ensure appropriate authentication and authorization mechanisms, a secure key management process, strong encryption, and sufficient monitoring of intrusion attempts.

Malicious Insiders. This threat is concerned with considerable damage that malicious insiders could create by getting an access or manipulating data in a data center. Centralization of data inside cloud servers itself creates an attractive condition for an adversary to try out fraud attempts. To reduce the risk, cloud providers must ensure strong and sufficient physical access controls, perform employee's background checks, make security process transparent to consumers, and allow for external audits.

Shared Technology Vulnerabilities. A virtualized hypervisor provides secure compartments and mediates communication between guest systems and physical resources underneath. Flaws in hypervisors might grant an inappropriate permission for an operating system to access or modify physical resources which belong to other tenants. The risk could be reduced by monitoring physical resources for unauthorized activities, implementing best practices for deployment and configuration, performing a vulnerability scan and configuration audits on a hypervisor, and following best practices during its installation and configuration.

Data Loss and Data Leakage. Data loss and leakage could be a result of unauthorized access, data manipulation, or physical damage. To protect unexpected loss and leakage, providers must implement robust access controls and secure key management process. Backup and recovery mechanisms should be put in place.

Account and Service Hijacking. Cloud service models put paramount importance to authentication and authorization, as the security of every service depends on its accountability and robustness. The use of two-factor authentication, secure key management process, and a careful design of web services help to reduce the damage of this attack.

While most of the common threats to cloud security are associated to its underlining technologies (web services and virtualization), some of them (abusing and malicious insiders) could be solved by better governances. This list is expected to be updated as the usage of cloud computing grows in popularity.

6.3. Recommended Practices to Enhance Cloud Security. This subsection presents a list of best practices to enhance cloud security as proposed by leading cloud providers and researchers. The practices are organized into five main areas including: identity and access control, data security, application security, security of virtualized environment, and physical and network security.

6.3.1. Identity and Access Control. The provisions of IT infrastructure offered by cloud computing extend the focus of identity and access control from the application level to the platform and virtual machine level. The goal of identity and access control is to ensure that accesses to data and applications are given only to authorized users. Associated areas include strong authentication, delegated authentication and profile management, notification, and identity federation. A set of recommended practices is relevant to both cloud providers and consumers.

Authentication. The challenges of authentication include the use of strong authentication methods, identity management, and federation.

(i) *Verification of users identities and access rights* is securely performed before the access to data and services is granted [82].

(ii) *Multifactor authentication* is applied before the access to highly sensitive and critical data (e.g., customer's identity, financial information) and processes (e.g., an administrative portal) is granted [82, 83]. Cloud providers should provide various authentication methods, including, for instance, biometrics, certificates, and one-time password, to support different security requirements.

(iii) *A secure communication tunnel* (e.g., VPN connections using SSL, TSL, or IPSEC) and a valid certificate are required to access highly sensitive and critical assets [16, 82].

(iv) *The use of federated identity*, that is, authenticating users through identity providers, enables a single sign-on or a single set of identities which are valid among multiple systems [82]. A privilege granted to an external user authenticated through identity federation should be limited appropriately [16].

(v) *Delegated authentication capability* is provided for enterprise consumers. Several current standards for exchanging authentication (and authorization) are Security Assertion Markup Language (SAML) and WS-federation [16].

Authorization. The goal of authorization is to ensure adequacy and appropriateness of the access control to data and services. It could be achieved by matching a user's privileges to his/her job responsibilities and to the characteristics of assets to be accessed and maintaining information necessary for audits.

(i) *Least-privileged scheme.* Users are granted the access only to the data and services necessary for their job responsibilities [82–84]. A least-privileged model for authorization and role-based access model support this practice. Access to additional data and services requires a formal approval and an audit trail is recorded [84].

(ii) *Asset classification.* Assets are classified according to their sensitivity levels and criticality levels [82, 83]. This information is used to determine the strength of authentication methods and the access granted.

(iii) *Regular review and audit trails.* User access lists and granted authority need to be regularly updated and reviewed. Audit trails are maintained [16, 82–84].

Identity Management. Strong identity management includes the restriction of a strong password, password expiration, secure password changing, and resetting mechanisms [82–84]. A full list is provided in [82].

Notification. Cloud providers should implement a notification process to inform users of security breaches that (might) happen [82, 84]. Logs of activities on users' behavior and access patterns, such as the one implemented by Facebook and Google to monitor users' log-in behaviors, can be monitored for malicious attempts [76]. Unexpected behaviors are notified for user considerations.

6.3.2. Data Security. The security of data and information on the cloud is one of the main hindrances for prospective cloud consumers. ISO 27001 defines information security through the following six perspectives: *confidentiality*—ensuring that the information is prevented from disclosure to unauthorized parties; *integrity*—ensuring that the information is not maliciously modified during the transit; *availability*—ensuring that the denial of service is prevented so the information is available when needed; *authenticity*—ensuring that the retrieved information is genuine; *authorization*—ensuring that the information could be accessed only by authorized parties; and *nonrepudiation*—ensuring that each party could not deny their action.

The responsibility of cloud providers is to guarantee that the security is satisfied at every state of the data life cycle, that is, created, stored, used, transferred, archived, and destroyed. Data security policy and practices adopted by a provider should be explicitly declared and adhered to the quality of services agreed in the SLA.

Data Classification and Access Control. This practice area supports authorization. Its challenge is to define appropriate and practical data classification scheme and access controls for different classes of data.

(i) *Data owner's practices.* The data owner is responsible for determining the data category and the access control. The classification scheme should at least consider the data sensitivity, criticality, and the frequency of use [82–84]. The access control defines who (individual person, role) get access to the data, under which permission (read, write, modify, and delete) and under which circumstances. It is equally important to keep the classification scheme and the access control updated. Tools such as Digital Right Management and Content Discovery are designed to facilitate this process [16].

(ii) *Provider's support.* The providers should support different levels of protection according to the classification scheme. Necessary information is logged for audit.

Encryption and Key Management. Encryption is used to ensure confidentiality, integrity, authenticity, and nonrepudiation. It should be applied at all states of the data life cycle when operating in shared environments.

(i) *Strong encryption.* Encryption is done by using a trustworthy encryption algorithm and an appropriate key length. At minimum, the 128-bit key length is required for symmetric encryption, and the 2048-bit key length is required for asymmetric encryption [83].

(ii) *Key management.* A key management process ensures the protection of key storages, the appropriateness of access controls, and key backup and recoverability. IEEE1619.3 is designed to support this process [16].

(iii) *Understanding security mechanisms used by the provider.* Consumers should understand the encryption mechanisms used in the cloud storage and apply additional mechanisms when the provided features are not sufficient for a security level required.

Backup and Recovery. Data availability is guaranteed by an appropriate backup and recovery plan. Its challenge is to ensure that the backup is performed regularly, and data encryption is done when it is required. In addition, it is equally important to perform the recovery test. In Google, the backup is supported by a distributed file system, in which the data are segregated into chunks and replicated over many places [84].

Data Sanitization. Data sanitization is the process of removing data stored in memory and storage before returning such resources to the shared pool. Sanitization is more difficult in a multi-tenancy environment where physical resources are shared [75]. The challenge is to ensure that the data is completely deleted and unrecoverable, and the sanitization does not impact the availability of data resided in other tenants. Several techniques such as crypto shredding, disk wiping, and degaussing facilitate this process [16].

6.3.3. Application Security. People and processes are the main factors to achieve secured software. It is essential to have motivated and knowledgeable team members to adopt a development process which suits to the context and to have sufficient resources. This practice area aims to ensure that all of these factors are in place to support the development of secured software. We set the primary focus on web applications which is a general form of SaaS.

Software Development Life Cycle. Software quality is influenced by the whole software development life cycle. The development team ought to set quality as a priority, apart from scheduling and financial concerns. Compromise on quality impacts a company's sustainability in a long run, if not worse. The challenge in this area is to put appropriate and sufficient practices in place to ensure their compliance, to implement an appraisal program, and to provide the relevant knowledge to associated people. A list of best practices to ensure security of developed software adopted in Google [84] and Microsoft [83] is presented in this subsection.

(i) *Team and software development process.* A development team should be well trained and understand products and the context of the project. The team is equipped with sufficient knowledge on design patterns, web services, vulnerability patterns, protection strategies, and cloud computing. Quality attributes on focus are identified as a basis for design decisions. A development model is selected and customized to suit the nature of the project and the resources.

(ii) *Practices for design.* For the design, software design best practices are adopted when appropriate. The related practices are formal architecture reviews, identifications of high-level security risks, development of sufficient detailed designs, measurement programs for software security tracking, identification alternative implementation solutions, and taking associated risks to project-level decision making process, obtaining an architecture review from an external third party to assess the possible limitations of an internal review.

(iii) *Practice for coding.* For code artifacts, the related best practices are conformance to the coding standard and guidance, pair programming, test first, code inspections, peer review on critical modules, and development of reusable components which are proved to be secure for certain types of vulnerabilities for common usages. Google has implemented a database access layer which is robust for SQL injection vulnerability, and a HTML template framework designed to prevent cross-site-scripting vulnerability.

(iv) *Practices for testing.* For testing, automate tests help to detect common coding errors, especially in interfaces, application logics, and workflows. In addition to the normal test practices, fuzzing tools could be used to detect vulnerabilities. This tool generates erroneous inputs to the program and records crashes and errors that occur, so that developers could correct them before deployment [76].

Web Services Implementation. A web service holds certain classes of vulnerabilities when it is not appropriately implemented. In cloud, web services are one of the core technologies in which cloud applications and operations (for instance, administration, provision, and service orchestration) rely on. It is a challenge for a development team to understand vulnerability and possible attack scenarios and to adopt counter mechanisms to ensure all perspectives of security. An example of standard counter mechanisms is *WS-Security* which combines three elements to secure service communications: *security tokens*—to verify a user's identity and his/her access rights; *encryption*—to ensure confidentiality and integrity; and *signature*—to ensure integrity and non-repudiation.

Frontend Environment. A frontend environment, including web browsers and front front-end systems, is one area of the attack surfaces [8]. The vulnerability of web browsers mainly results from an inappropriate level of permission given to its plug-ins. To ensure the front-end security, users should allow only necessary and trustable plug-ins to be installed in web browsers, avoid saving their identity in the web browsers, and use antivirus and antispyware with updated profiles.

6.3.4. Virtualized Environment. Virtualization covers the layers of a virtualized hypervisor which handles compartmentalization and resource provisions, a virtual machine, and an operating system.

Virtualized Hypervisor. A special care should be given to a hypervisor, as it introduces a new attack surface, and its erroneous executions generate a cross-tenant impact [8]. Consumers should understand the supported features and limitations of the virtualized hypervisor used by providers, including the mechanism they employ to perform compartmentalization and provide security controls. When necessary, additional security mechanisms should be integrated to reduce the dependency on the provider [16]. A hypervisor should provide an interface to monitor the traffic crossing VM backbones through virtualized communication channels (between VMs resided in the same tenant) which are invisible to traditional network monitoring devices [16, 75].

Virtual Machine. A VM runs on a compartment managed by a hypervisor. The challenge for securing a VM is to verify the provided security controls, the robustness and the authenticity of an acquired VM image.

(i) *Secure image life cycle.* Providers and consumers maintain a secured life cycle of VM images. IaaS and PaaS consumers should acquire a VM image from a reliable source. Even in the case that the image is taken from the provider, the authenticity of the acquired image needs to be verified. The provider maintains a secure VM image management process from creating, storing, and cataloguing to distributing it.

(ii) *Understanding security mechanisms used by the provider.* Consumers should identify the security controls which are in place for acquired VM images and their dependencies. VM access channels and types of connection should be also identified.

(iii) *Various security zones for VM instances.* The provider segregates VMs of different classes into different security zones. VM classification considers the criticality of hosted data and processes, types of usage, and the production stage for, for example, development and production [16].

(iv) *Using tools designed for multi-tenancy.* There are several techniques to enhance VM security such as the use of bidirectional firewalls on each VM instantiation and the hypervisor-level security controls [81]. Concurrent applications of protection tools, designed for stand-alone systems on multi-tenants, might affect the performance of the whole physical platform. Cloud providers should consider performing this work at the hypervisor level.

Cloud Operating System and Middleware. Compromises to the security of an OS affect the security of the whole data and processes resided in the systems, therefore an OS should be highly robust and secure. The following practices are recommended to secure an OS.

(i) *Robust OS.* Rich-featured OS and complex configuration could compromise system robustness. Consumers should consider an OS which contains only necessary features and has been scanned for vulnerabilities [75]. Architecture flaws could be a result of misconfiguration of resources and policies. Maintaining a simple architecture prevent erroneous settings that might threaten security controls [76]. A strong authentication mechanism is required for an administrative access to the OS [16].

(ii) *Single hardened OS deployment.* Best practices for providers are the use of a single hardened OS and white list process monitoring. A common technique for PaaS providers to enhance security is to deploy a single hardened OS throughout the cloud and scan for its binary changes to detect security breaches [75]. Questionable changes result in a system rollback, thus keeping the system in a known good state. Walters and Petroni demonstrate this process in their paper [85]. A process-monitoring feature should run against a list of allowed processes, instead of a black list [76].

6.3.5. Network and Physical Security

Robust Physical Access Control. The centralization of data residing in a data center poses a significant security concern. Cloud consumers should be aware of a malicious insider

Table 4: Mapping of cloud vulnerabilities (column) and recommended security practices (row).

	Application				Platform		Infrastructure				Unified resource			Fabric
	Web service v.	Client-side data manipulation v.	Authentication & Authorization v.	Encryption & key management v.	Cloud platform v.	Access to platform admin interface v.	Traditional virtual machine v.	Virtual network communication v.	Data sanitization v.	Access to VM admin interface v.	Virtualized hypervisor & interfaces v.	Multi-tenancy v.	Shared network component v.	Physical v.
Identity and access control														
Authentication			×											
Authorization			×											
Identity management			×											
Notification			×											
Data security														
Data classification			×											
Encryption and key	×		×	×		×				×	×			
Backup and recovery														×
Data sanitization									×			×		
Application security														
Dev. life cycle	×		×	×	×	×	×			×				
Web service imp.	×													
Frontend environment	×	×												
Virtualized environment														
Virtualized hypervisor	×		×					×		×		×		
Virtual machine	×		×				×			×				
Operating system	×		×		×									
Network and physical security														
Robust physical access														×
BCP														×
Network control														×

threat in a cloud environment, where centralized data amplifies the motivation of malicious attempts [41]. The goal of having a robust and appropriate physical access control is to prevent insider abuse within a cloud provider.

(i) *Various security zones.* The data center should be separated to different security zones for hosting data and services which require different levels of protection. Control mechanisms are applied according to the criticality of different security zones.

(ii) *Using a least privilege policy for access controls.* A least privilege policy is applied to give only necessary permission to limited users. Employees should have limited knowledge on customers. Additional access requires a formal approval, and audit trail is always respected. Employee privilege is up-to-date and regularly reviewed. Background check on employees is performed.

Business Continuity Plan (BCP). The objective of this regulation is to ensure the availability of data and services hosted in cloud infrastructures. Consumers should ensure that the provider implements and respects standard Business Continuity Plan. Examples of such standard are BS25999 and ISO22301 [41].

Network Control. In cloud environments a provider is responsible for protecting a customer's data from accesses across the Internet. Network borders should be protected by robust mechanisms or devices. Firewalls are in place to protect each external interface, only necessary ports are open, and the default setting is denial. Intrusion detection and prevention mechanisms should be employed and kept up-to-date [41].

The mapping between the cloud vulnerabilities identified in Section 6.1 and the recommended security practices is presented in Table 4.

6.4. Summary. Security is one of the most critical hindrances to nontrivial adoptions of new technologies. Cloud computing inherits security issues from its enabling technologies such as web services, virtualization, multi-tenancy, and cryptography. A virtualized hypervisor which enables resource sharing leads to the economies of scale but introduces a new attack surface. We analyze the security in the cloud by understanding the weaknesses associated to system procedures, design, and implementation of the components of cloud infrastructure. These weaknesses, known as *vulnerability*, could be exploited and result in security breaches, thus requiring stakeholders to be attentive and adopt appropriate counteractions. At the application layer (SaaS), programmers need to handle the vulnerabilities of web services, authentication and authorization, and encryption and keys. End users need to prevent their frontend environments from malicious plug-ins and malware. At the platform layer in which a cloud OS and middleware are offered as a ready-made solution for development and deployment platforms, security mechanisms are generally integrated to the solution. PaaS consumers should understand which mechanisms are in place and impacts they might have on different system configurations. The vulnerabilities at the infrastructure level (IaaS) are associated to virtual machines and virtual network communications. IaaS consumers are responsible for their own security controls from this layer up to the application layer. The vulnerabilities to the unified resource layer related to multi-tenancy and virtualized hypervisors are critical as their compromises would affect the whole cloud services. The security mechanisms and processes adopted by a provider at this level should be explicitly clarified as a part of the SLA. Physical security deals with malicious insiders, business continuity plans for natural disaster, and the preventions of unauthorized accesses through network channels. Having vulnerabilities identified, we gathered the security practices recommended by leading cloud providers, and experts and classified them into 17 practice areas.

7. Conclusions

In this paper we depicted a comprehensive scenery of cloud computing technology. The view of cloud computing is shaped by its definition, service model, and deployment model. The characteristic of cloud services is shaped by its architecture, service management approach, and underlying technologies. Based on its solid foundations, cloud promises significant benefits on enterprise-level cost reduction, increased agility, and better business-IT alignment. Many more benefits arise when other levels or perspectives are taken into consideration. Apart from these benefits, however, cloud services still pose a number of limitations that hinder their wider adoption. Such limitations originate in the very nature of its building blocks. An example is the instability of large-scaled network. A limited capability of network has become a main obstacle for services that require reliable response time, such as high performance computing and latency sensitive applications, to be deployed in the clouds. Security is another major hindrance of cloud adoption that touches upon many technological factors including multi-tenancy, virtualization, and federated identity management. It is further aggravated by the fact that service level agreements are generally explicit about placing security risks on consumers. As cloud computing is not the silver bullet for all circumstances, it is necessary for technology adopters to sufficiently understand its concerns and properly handle them. Without comprehending its underlying technology, finding the most appropriate adoption approach seems to be an impossible mission. Our paper can assist this endeavor by offering a comprehensive review of the fundamentals and relevant knowledge areas of cloud computing.

References

[1] J. Voas and J. Zhang, "Cloud computing: new wine or just a new bottle?" *IT Professional*, vol. 11, no. 2, pp. 15–17, 2009.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[3] M. Armbrust, A. Fox, R. Griffith et al., "A Berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, 2009.

[4] M. Creeger, "CTO roundtable: cloud computing," *Communications of the ACM*, vol. 52, no. 8, pp. 50–56, 2009.

[5] L. Herbert and J. Erickson, *The ROI of Software-As-A-Service. White Paper*, Forrester Research, Inc., 2009.

[6] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda, "Decision support tools for cloud migration in the enterprise," in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD '11)*, pp. 541–548, 2011.

[7] N. Phaphoom, N. Oza, X. Wang, and P. Abrahamsson, "Does cloud computing deliver the promised benefits for IT industry?" in *Proceedings of the WICSA/ECSA Companion Volume(WICSA/ECSA '12)*, pp. 45–52, ACM, 2012.

[8] N. Gruschka and M. Jensen, "Attack surfaces: a taxonomy for attacks on cloud services," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD '10)*, pp. 276–279, July 2010.

[9] P. Hofmann and D. Woods, "Cloud computing: the limits of public clouds for business applications," *IEEE Internet Computing*, vol. 14, no. 6, pp. 90–93, 2010.

[10] B. Grobauer, T. Walloschek, and E. Stöcker, "Understanding cloud computing vulnerabilities," *IEEE Security and Privacy*, vol. 9, no. 2, pp. 50–57, 2011.

[11] C. Baun, M. Kunze, J. Nimin, and S. Tai, *Cloud Computing: Web-Based Dynamic IT Services*, Springer, 2011.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, November 2008.

[13] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Computer Communication Review*, vol. 39, pp. 50–55, 2008.

[14] DMTF, "Interoperable Clouds a white paper from the Open Cloud Standards Incubator," 2009.

[15] IBM, "IBM Cloud Computing—Service Management," 2009.

[16] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing V2.1," 2009.

[17] Cisco, "Cisco Cloud Computing—Data Center Strategy, Architecture, and Solutions," 2009.

[18] Open Security Architecture, "SP-011: Cloud Computing Pattern," 2011.

[19] US General Service Administration, "Cloud Computing Initiative Vision and Strategy Document (DRAFT)," 2010.

[20] Storage Networking Industry Association, "5. Overview of Cloud Storage," 2011.

[21] 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)*, pp. 1–10, November 2008.

[22] L. J. Zhang and Q. Zhou, "CCOA: cloud computing open architecture," in *Proceedings of the IEEE International Conference on Web Services (ICWS '09)*, pp. 607–616, July 2009.

[23] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, 2003.

[24] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? An architectural map of the cloud landscape," in *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09)*, pp. 23–31, May 2009.

[25] J. Howe, "The rise of crowdsourcing," *Wired Magazine*, vol. 14, pp. 1–5, 2006.

[26] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with Mechanical Turk," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 453–456, ACM, New York, NY, USA, April 2008.

[27] J. G. Davis, "From crowdsourcing to crowdservicing," *IEEE Internet Computing*, vol. 15, no. 3, pp. 92–94, 2011.

[28] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10)*, pp. 27–33, April 2010.

[29] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[30] I. Foster, "The anatomy of the grid: enabling scalable virtual organizations," in *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 6–7, 2001.

[31] A. Andrzejak, M. Arlitt, and J. Rolia, "Bounding the resource savings of utility computing models," Tech. Rep. HPL-2002-339, Hewlett Packard Labs, 2002.

[32] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[33] R. Uhlig, G. Neiger, D. Rodgers et al., "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[34] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[36] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a Service-Oriented Architecture," Tech. Rep. CMU/SEI-2007-TR-015, Software Engineering Institute of Carnegie Mellon University, 2007.

[37] T. Erl, *SOA Design Patterns*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2009.

[38] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys & Tutorials*, no. 99, pp. 1–26, 2011.

[39] Y. Yu, M. Isard, D. Fetterly et al., "DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language," in *Proceedings of the 8th USENIX Conference on Operating systems design and implementation (OSDI '08)*, pp. 1–14, USENIX Association, Berkeley, Calif, USA, 2008.

[40] The US National Institute of Standards, "Cloud Architecture Reference Models," 2011.

[41] Cloud Security Alliance, "Top threats to cloud computing V1.0," 2010.

[42] J. Oriol Fitó, I. Goiri, and J. Guitart, "SLA-driven elastic cloud hosting provider," in *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP '10)*, pp. 111–118, February 2010.

[43] F. Faniyi and R. Bahsoon, "Engineering proprioception in SLA management for cloud architectures," in *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA '11)*, pp. 336–340, 2011.

[44] J. Y. Lee, J. W. Lee, D. W. Cheun, and S. D. Kim, "A quality model for evaluating software-as-a-service in cloud computing," in *Proceedings of the 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA '09)*, pp. 261–266, December 2009.

[45] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, and C. Pu, “Variations in performance and scalability when migrating n-tier applications to different clouds,” in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD ’11)*, pp. 73–80, 2011.

[46] D. Bao, Z. Xiao, Y. Sun, and J. Zhao, “A method and framework for quality of cloud services measurement,” in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE ’10)*, pp. V5358–V5362, August 2010.

[47] S. Dowell, A. Barreto, J. B. Michael, and M.-T. Shing, “Cloud to cloud interoperability,” in *Proceedings of the 6th International Conference on System of Systems Engineering (SoSE ’11)*, pp. 258–263, 2011.

[48] M. Alhamad, T. Dillon, and E. Chang, “Conceptual SLA framework for cloud computing,” in *Proceedings of the 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST ’10)*, pp. 606–610, April 2010.

[49] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *SIGACT News*, vol. 33, pp. 51–59, 2002.

[50] B. F. Cooper, R. Ramakrishnan, and U. Srivastava, “PNUTS: Yahoo!’s hosted data serving platform,” in *Proceedings of the VLDB Endowment*, vol. 2, pp. 1277–1288, 2008.

[51] W. Vogels, “Eventually consistent,” *Communications of the ACM*, vol. 52, pp. 40–44, 2009.

[52] H. Douglas and C. Gehrmann, “Secure Virtualization and Multicore Platforms State-of-the-Art report,” Swedish Institute of Computer Science, 2009.

[53] S. J. Vaughan-Nichols, “New approach to virtualization is a lightweight,” *Computer*, vol. 39, no. 11, pp. 12–14, 2006.

[54] A. Whitaker, M. Shaw, and S. D. Gribble, “Denali: lightweight virtual machines for distributed and networked applications,” Tech. Rep., The University of Washington, 2002.

[55] W. Chen, H. Lu, L. Shen, Z. Wang, N. Xiao, and D. Chen, “A novel hardware assisted full virualization technique,” in *Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS ’08)*, pp. 1292–1297, November 2008.

[56] I. Pratt, K. Fraser, S. Hand, C. Limpach, and A. Warfield, “Xen 3.0 and the art of virtulization,” in *Proceedings of the Ottawa Linux Symposium*, pp. 65–77, 2005.

[57] Advanced Micro Devices Inc., “AMD-V Nested Paging revision 1.0,” Whitepaper, 2008.

[58] G. P. Chen and J. S. Bozman, “Optimizing I/O Virtualization: preparing the datacenter for next-generation applications,” White Paper sponsored by Intel Corporation, 2009.

[59] J. Sugerman, G. Venkitachalam, and B. Lim, “Virtualizing I/O devices on VMware workstation’s hosted virtual machine monitor,” in *Proceedings of the General Track, USENIX Annual Technical Conference*, pp. 1–14, 2001.

[60] P. Barham, B. Dragovic, K. Fraser et al., “Xen and the art of virtualization,” in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP ’03)*, pp. 164–177, ACM, New York, NY, USA, October 2003.

[61] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm:thelinux virtual machine monitor,” in *Proceedings of the Ottawa Linux Symposium*, pp. 225–230, 2007.

[62] M. Stonebraker, “SQL databases v. NoSQL databases,” *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.

[63] F. Chang, J. Dean, S. Ghemawat et al., “Bigtable: a distributed storage system for structured data,” *ACM Transactions on Computer Systems*, vol. 26, no. 2, article 4, 2008.

[64] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 35–40, 2010.

[65] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, “The end of an architectural era: (it’s time for a complete rewrite),” in *Proceedings of the 33rd international conference on Very large data bases (VLDB ’07)*, pp. 1150–1160, VLDB Endowment, 2007.

[66] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983.

[67] Z. Wei, G. Pierre, and C.-H. Chi, “CloudTPS: scalable transactions for web applications in the cloud,” *IEEE Transactions on Services Computing*, vol. 99, p. 1, 2011.

[68] B. Kemme and G. Alonso, “Don’t be lazy, be consistent: postgres-r, a new way to implement database replication,” in *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB ’00)*, pp. 134–143, Morgan Kaufmann Publishers, San Francisco, Calif, USA, 2007.

[69] A. Lith and J. Mattsson, *Investigating storage solutions for large data—a comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data [M.S. thesis]*, Chalmers Tekniska Hogskola, Sweden, 2010.

[70] B. G. Tudorica and C. Bucur, “A comparison between several NoSQL databases with comments and notes,” in *Proceedings of the 10th Roedunet International Conference (RoEduNet ’11)*, pp. 1–5, 2011.

[71] E. A. Brewer, “Toward robust distributed systems,” Principles of distributed computing, Portland, Ore, USA, 2000.

[72] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with YCSB,” in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC ’10)*, pp. 143–154, ACM, New York, NY, USA, June 2010.

[73] G. DeCandia, D. Hastorun, M. Jampani et al., “Dynamo: amazon’s highly available key-value store,” in *Proceedings of 21st ACM SIGOPS symposium on Operating systems principles (SOSP ’07)*, pp. 205–220, ACM, New York, NY, USA, 2007.

[74] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, “Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web,” in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC ’97)*, pp. 654–663, ACM, New York, NY, USA, May 1997.

[75] J. Spring, “Monitoring cloud computing by layer—part 1,” *IEEE Security and Privacy*, vol. 9, no. 2, pp. 66–68, 2011.

[76] J. Spring, “Monitoring cloud computing by layer—part 2,” *IEEE Security and Privacy*, vol. 9, no. 3, pp. 52–55, 2011.

[77] I. Iankoulova and M. Daneva, “Cloud computing security requirements: a systematic review,” in *Proceedings of the 6th International Conference on Research Challenges in Information Science (RCIS ’12)*, pp. 1–7, 2012.

[78] W. A. Jansen, “Cloud hooks: security and privacy issues in cloud computing,” in *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS ’10)*, pp. 1–10, January 2011.

[79] S. A. Almulla and C. Y. Yeun, “Cloud computing security management,” in *Proceedings of the 2nd International Conference on Engineering System Management and Applications (ICESMA ’10)*, pp. 1–7, April 2010.

[80] R. Chow, P. Golle, M. Jakobsson et al., “Controlling data in the cloud: outsourcing computation without outsourcing control,”

in *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW '09)*, pp. 85–90, November 2009.

[81] L. M. Kaufman, "Can public-cloud security meet its unique challenges?" *IEEE Security and Privacy*, vol. 8, no. 4, pp. 55–57, 2010.

[82] IBM, "Cloud Security Guidance IBM Recommendations for the Implementation of Cloud Security," 2009.

[83] Microsoft Global Foundation Services, "Securing Microsoft's Cloud Infrastructure," 2009.

[84] Google, "Security Whitepaper: Google Apps Messaging and Collaboration Products," 2010.

[85] A. Walters and N. L. Petroni Jr., "Volatools: integrating volatile memory forensics into the digital investigation process," in *presented at Black Hat DC*, 2007.