Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking

Steffen Kächele
Institute of Distributed Systems
University of Ulm, Germany
Email: steffen.kaechele@uni-ulm.de

Christian Spann
Institute of Distributed Systems
University of Ulm, Germany
Email: christian.spann@uni-ulm.de

Franz J. Hauck
Institute of Distributed Systems
University of Ulm, Germany
Email: franz.hauck@uni-ulm.de

Jörg Domaschka
Institute of Information Resource Management
University of Ulm, Germany
Email: joerg.domaschka@uni-ulm.de

***Abstract*—Cloud computing is currently classified by the generally accepted terms *IaaS*, *PaaS* and *SaaS*. These are not precise enough to give users and providers a common terminology to differentiate between existing cloud offers. Furthermore, there is no common and accepted terminology for storage and network cloud services. Terms like *StaaS* and *NaaS* have been coined but lack a clear definition. This paper contributes a fine-grained taxonomy for computation, storage and network services which is still compliant with existing terminology. Our taxonomy is based on abstractions cloud services provide to tenants; it does not consider implementation aspects. In addition to our taxonomy, we provide a classification of many existing services and simple decision trees that allow an easy classification of any cloud offer of interest. Our comprehensive and fine-grained taxonomy is a new basis for the determination of existing and future cloud offerings.**

I. Introduction

The cloud computing paradigm proposes the on-demand usage of provided and maintained resources on hardware and software level. The terms Infrastructure, Platform and Software as a Service (*IaaS, PaaS* and *SaaS*) classify three different service models and are widely used and commonly accepted in literature [1], [2]. In principle, they describe different layers of abstraction at which cloud resources are offered at. Yet, different systems categorised as IaaS, and as PaaS respectively, do provide fundamentally distinct services to their tenants. Hence, using the existing terminology, it is impossible to exactly classify current and future cloud offerings.

The similarity between all cloud offerings is the provisioning of resources in a flexible and abstracted way. We identify three important types of resource domains. Most prominent, *computational* resources allow the deployment, execution and use of software. Besides, cloud systems may provide *storage* and *network* services usable either stand-alone or in conjunction with computational resources. The terms *StaaS (Storage as a Service)* and *NaaS (Network as a Service)* were coined for them, yet in a very unspecific way [3], [4].

This paper presents a unique taxonomy of cloud systems for all three types of resources. Our first contribution is the first usage-oriented and fine-grained terminology for network and storage services. Furthermore, our taxonomy goes beyond the established terms for computation clouds and allows their classification in a more detailed manner. As a second contribution, we clarify the consequences of choosing or providing services of a particular type and on a particular abstraction level. This greatly helps system administrators, software developers and system architects when specifying requirements and approaches to cloud-based software and systems.

In this document, Section II introduces the methodology we use. Sections III–V consider the three resource types computation, network and storage. For each type, we revisit existing abstraction layers and derive a cloud taxonomy We also present real-world examples for each type demonstrating the relevance of our classification. Section VI discusses the relationship of offerings from the three domains. Section VII presents a discussion of related work, before we conclude.

II. Methodology

In contrast to existing terminology, we use a two-level approach to classification. The first-level classification criterion is the *type of resource* offered to the cloud tenant. The second level comprises the *abstractions* of a dedicated type. We determine the abstraction of a cloud offering via a *classification rule*. The *offering effect* considers the consequences for both cloud providers and tenants when operating with a cloud of a certain type and on a certain abstraction.

Resource type: We consider three different types: *Computing* services provide mechanisms to run applications. *Storage* services offer a way to store data persistently, whereas *network* services comprise any mechanism used to communicate between (virtual) machines, applications and users.

Abstraction levels: In order to put our taxonomy on a solid foundation, we revisit well-known and accepted abstraction levels for each resource type. We particularly target abstractions that are already commonly known and used outside cloud environments. Finally, we focus on a precise definition of abstraction levels in order to minimise ambiguities.

Classification rule: Having such levels of abstraction, we define a cloud service c to be offered on layer l with l being

this work was partially founded by the EU FP7, GA No. 317715.

the topmost abstraction level that is fully managed by the cloud provider. Hence, our taxonomy focuses on the view offered to the cloud tenant (i.e. customer). Note that we consider both stand-alone and combined services.

Offering effect: The taxonomy itself does not consider underlying technological implementation details such as virtualisation techniques for establishing its terminology. Yet, in order to illustrate the consequences of operating with a certain kind of cloud service, this paper also lists consequences for tenants and cloud providers regarding possibilities and duties.

III. Computational Resources

The provisioning of computational resources constitutes the predominant part of cloud offerings. These are commonly subdivided into the three categories IaaS, PaaS and SaaS. For we consider this separation as too coarse, this section introduces a more fine-grained taxonomy of computation clouds based on the abstraction layers found in typical business applications. We first collect these abstraction layers, then introduce our new terminology and finally use it to categorise real systems.

A. Software Abstraction Layers

To illustrate the abstraction layers of a software system, we use a typical web application where clients access a service via HTTP. Our example comprises a Servlet, a Servlet container, the Java runtime environment, an operating system and the hardware. We discuss the relevant abstractions bottom-up. The higher the layer the more abstract and advanced the provided means are to use computational resources. At the same time, higher layers often induce more extensive restrictions than lower ones, e.g. by determining the programming language.

Hardware (HW) forms the least abstract layer in our hierarchy. It provides bare metal resources such as CPUs, computing cores, the amount and type of RAM, co-processors and other hardware devices. The latter may include network interface cards and storage controllers. In our example, HW may consist of a standard x86 architecture, two CPUs with four cores each and four gigabytes of memory.

Operating systems (OS) reside on top of physical or virtual hardware. They provide isolation features resulting in the ability to execute multiple processes from multiple users and to share resources. The example OS may be GNU/Linux.

From an operating system point of view any application being executed is a process. Yet, with respect to abstraction and programming, multiple types of applications exist. They may directly make use of OS functionality or apply further software components with a possibly different abstraction.

A *runtime environment (RE)* realises a container for the execution of applications. The OS provides a basic runtime for all processes running in a system. Higher level runtime environments may significantly enrich the OS runtime and further intermediate runtime environments. Typical features of that layer are the execution and interpretation of intermediate code and sophisticated libraries that applications can use. The example RE is the Java Virtual Machine whose execution environment significantly differs from the OS environment.

The *framework* layer *(FW)* uses mechanisms provided by RE and OS. The distinguishing feature of frameworks is *inversion of control (IoC)* [5]. IoC demands that the application be passive and all activity be triggered by the FW. Having full control of activity enables the FW to provide rich implicit application support such as transaction management, persistence of data, access control and protocol handling. The example FW is a Tomcat Servlet container.

TABLE I. Types of computation-oriented cloud services

abstr. layer/ cloud term	offered service (examples)	classification criteria
APP/SaaS	application (Web shop)	no way for deploying own logic
FW/FaaS	appl.type-specific software container (Servlet, Bean)	deployment of code via cloud API; no view of single nodes; externally triggered activity
RE/RaaS	runtime environment (Java SE)	deployment of code via cloud API; no view of single nodes; full control of activity
OS/OSaaS	operating system (Linux)	SW installation/deployment via OS; perform administration tasks; unchangeable OS core; view of single nodes
HW/HWaaS	HW platform (x86)	select HW platform; manual updates of OS required

The *application (APP)* contains the business logic. It is located on top of the software stack and can be run on any layer above HW. Yet, it is commonly deployed on RE or FW layer. An APP may or may not be accessible for clients (Web APPs vs. HPC). In our example, the APP has a Web interface.

B. Terminology of the Computational Cloud

In the previous section, we identified five layers of abstraction. Our taxonomy defines terms for cloud systems that reflect these abstractions. According to the *classification rule* only the highest abstraction level fully managed by the cloud provider is of importance for the classification. In relation to computation this implicitly covers the granularity of a deployment unit as well as the power of the user interface available to the tenant. Table I summarises the layers, lists the service they offer to tenants and also contains classification criteria that help identify the layer of a particular cloud offering.

Infrastructure as a Service

IaaS is commonly perceived as providing resources on hardware level. Yet, a closer look at existing offers reveils that the common denominator for IaaS is the provision of a node-based infrastructure. We identified two different layers of abstraction to offer such services.

HWaaS: Hardware as a Service offers bare HW resources to the *tenant*. As access is granted at a low level the tenant is free to install and configure arbitrary software including the OS. Yet, this means that the tenant is fully responsible for running and managing the entire software stack starting from the boot loader and the OS kernel. The *provider* has to maintain the hardware. He may support scalability by offering mechanisms to spawn new machine instances and thus extend the resource pool of the tenant. Yet, the tenant remains responsible for its application exploiting the larger pool.

OSaaS: Operating System as a Service offers *tenants* a fully managed OS including underlying HW resources. A tenant perceives this environment (i.e. OS) as a single computing node. He composes his cloud application from the interplay of

OS *(daemon) processes*. The installation and deployment of these processes occurs solely via OS mechanisms. Due to the single-node view, it is possible to use OS features such as IPC between multiple processes. As the OS is entirely managed, the tenant may not install its own OS and has no influence on the core elements of the OS such as drivers and kernel modules. This is the elementary difference from HWaaS where the tenant may run arbitrary OSs and also configure the kernel. In addition to HWaaS the *provider* of OSaaS also takes care of maintaining the OS, e.g. applying security patches to the kernel and restarting the system if needed. Although tenants will have less control over the OS, we consider OSaaS as *providing fundamental resources*, which makes it compliant to the NIST definition of IaaS [2].

Platform as a Service

PaaS allows tenants to deploy applications in a cloud environment [2]. In contrast to OSaaS and HWaaS, PaaS environments no longer provide the perception of a computing node to the tenant any more. At most, the tenant may observe multiple instances of his application. We see the need to differentiate between two types of PaaS services.

RaaS: Runtime Environment as a Service offers the *tenant* an environment ready to run his applications. The deployment of these applications is his only responsibility. The *provider* manages all layers up to and including the runtime. This management includes applying bug fixes. For the perception of distinct nodes has vanished, the provider may offer better support for automatic scalability compared to HWaaS and OSaaS for instance through distribution-aware implementations of the RE such as a distributed Java Virtual Machine [6] for Java applications.

FaaS: Framework as a Service constitutes the highest layer that still allows tenants to deploy application logic, e.g. in form of Servlets. Typically, it is tailored towards a dedicated application type or use case. As with RaaS, *tenants* receive a fully managed software platform. Yet, it is provided as a framework so that applications remain passive and are invoked by the cloud (IoC). Tenants may define scalability-related non-functional properties such as the desired quality of service or the maximum allowed operating costs. The *provider* has to maintain the environment and to manage the underlying infrastructure including FW implementation. Providers may support automatic scalability and high availability by exploiting FaaS's high abstraction level (e.g. distributed services, messaging/eventing, session replication and load balancing).

Software as a Service

SaaS is the provisioning of entire applications as a resource [2]. *Tenants* select pre-defined applications, while *providers* manage the hardware and software infrastructure including aspects such as scalability.

C. Examples

Table II presents real-world examples for the computation abstractions of our terminology. Not all systems shown are cloud offers. Instead, we also include technologies that may serve as a basis for future clouds. Fig. 1 contains a decision diagram showing how to derive the classification of a single system. We discuss some of the systems in detail.

TABLE II. EXAMPLES OF COMPUTATION CLOUDS & TECHNOLOGIES(*)

SaaS	
Google Calendar	salesforce.com
Microsoft Office 365	jira.com
RaaS (PaaS)	**FaaS (PaaS)**
Microsoft Azure	Cloud Foundry
COSCA	Heroku Dynos
ProActive	Jestalic
GridKa	Google App Engine
XtreemOS*	OpenShift
nOStrum*	Cumulogic
Microsoft SoftGrid*	
HWaaS (IaaS)	**OSaaS (IaaS)**
OpenCirrus	Joyent SmartOS
GoGrid Cloud / Dedicated Servers	Solaris Zones*
Amazon EC2 / Eucalyptus	FreeBSD Jail*
OpenStack	Linux VServer*
Flexiant Flexiscale	OpenVZ*
Rackspace	Alcatraz*
Nimbus Nimbula	LXC*
Cloud Central	

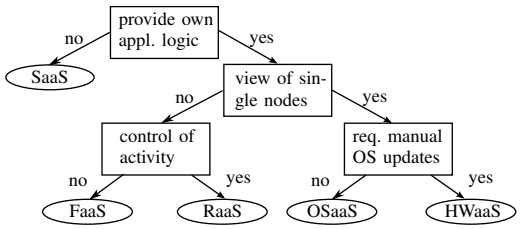

Fig. 1. Decision diagram for determining the type of a computation offering

HWaaS: Providers may implement HWaaS by offering access to non-virtualised physical nodes such as OpenCirrus.org [7] and GoGrid Dedicated Servers[1]. Other realisations apply virtualisation or hardware emulation. Amazon EC2[2] is the most prominent example amongst many similar offerings such as Eucalyptus [8] and others[3,4,5,6,7,8].

OSaaS: Joyent SmartOS[9] offers virtual operating system instances by separating a single operating system into multiple partitions. Such offerings may use virtualisation techniques on OS level such as Solaris Zones, FreeBSD Jail [9] and Linux VServer [10]. OpenVZ [11] provides isolated Linux containers on a single physical server that enables dynamic real-time partitioning and resource management. Alcatraz [12] is a safe execution environment isolating potentially malicious processes. LXC[10] adds resource management and isolation mechanisms to Linux' process management infrastructure.

RaaS: Microsoft Azure [13] provides a fully managed environment for .NET applications. Our COSCA system [14] realises distributed OSGi environments in the cloud[11] and allows file access and socket communication. ProActive [15] is an environment to run Java byte code. Other containers

1 http://www.gogrid.com/cloud-hosting/dedicated-servers.php
2 for information on Amazon services see http://aws.amazon.com/products
3 http://www.openstack.org
4 http://www.cloudcentral.com.au/cloud/server
5 http://www.flexiscale.com
6 http://www.gogrid.com/cloud-hosting/cloud-servers.php
7 http://www.nimbula.com
8 http://www.rackspace.com
9 http://www.joyent.com/technology/smartos
10 http://lxc.sourceforge.net
11 Note that this is RaaS as there is no inversion of control.

TABLE III. TYPES OF CLOUD-BASED STORAGE SERVICES

abstr. view/ cloud term	offered storage (examples)	classification criteria
mapping/ DMaaS	semantic mappings (JPA)	mapping between storage and application domain
database/ DBaaS	typed data store (DBMS, key-value store)	structured, typed data
file system/ FSaaS	files (ext3, S3)	structured, typeless data
raw/RStaaS	raw storage (block device, EBS)	unstructured, typeless data

directly base on the OS such as *nOStrum*[12], our native OSGi implementation [16]. Grid computing and job submission systems constitute further instances of this layer. Examples are GridKa [17], the XtreemOS Grid OS [18] and Microsoft SoftGrid [19] for job management and isolation.

FaaS: Many FaaS systems offer JavaEE-based Servlet containers. VMware's vFabric[13] provides a Tomcat-driven container environment. The Google App Engine[14] uses a Jetty-based Web Servlet container. Cumulogic[15] builds IaaS images for Amazon EC2 and Eucalyptus starting from Servlets. Both OpenShift Express and OpenShift Flex[16] are based on the JBoss application server. In addition, they provide a Ruby environment. Also other systems support multiple FWs. Cloud Foundry[17] amongst others supports Spring and Rails. Heroku's[18] Dynos support web applications written in various languages. Jelastic[19] offers Tomcat, Jetty and Glassfish as application servers. Finally, Map/Reduce [20] FWs available in multiple languages support the processing of large data sets.

SaaS: SaaS offerings include Google Calender[20], Mircrosoft Office 365[21], salesforce.com and many others.

IV. STORAGE RESOURCES

Beside computation, cloud providers commonly offer storage services. The term *Storage as a Service (StaaS)* covers aspects of such systems including databases and file systems. *StaaS* is used in the literature [3], [4], but still lacks a precise definition.

For our terminology of storage clouds, we start with abstraction layers found in storage terminology. From them, we derive a classification of cloud-based storage services. Finally, we bring existing offerings into line with our classification.

A. Storage Abstraction Layers

The abstraction with respect to data access is mostly defined by the view an accessor has on the data.

The lowest level is the *raw view*. Here, the accessor only sees a device with a dedicated size, an access mode and an addressing mode. The *file system view* abstracts from the raw view in the sense that it allows accessing data in units of files and directories that may also be nested within each other. The next higher view adds type information to the managed chunks of data. Such information is mostly stored in databases [21] which is why we call it *database view*. An even higher abstraction level appears when the data representation of one environment (e.g. the programming language) is automatically transferred to the data representation of another environment (e.g. the database). We refer to this as the *data mapping view*.

B. Terminology of the Storage Cloud

The used data abstraction layers can be re-discovered in cloud offerings. As presented in Table III, we distinguish four categories of cloud-based storage services.

RStaaS: Raw Storage as a Service offers storage, but by definition does not pre-set how the data shall be organised thereon. Interfaces for data access can range from bus-level (IDE, SCSI) to Internet protocols (iSCSI, HTTP). In any case, however, the data representation is raw binary data. Thus, *tenants* are responsible for any aspect of storage management such as adding file systems on top of a raw storage. Besides their task of managing and provisioning storage, *providers* can offer reliability, e.g. by replicating the data on volume level via RAID and by executing automatic backups.

FSaaS: File System as a Service allows access to binary records of data structured in storage containers, e.g. files and directories. *Tenants* get fully maintained and managed structured storage (i.e. a file system such as ext3, NTFS and XtreemFS [22]). The API may be OS-like (e.g. POSIX), but also network-oriented (e.g. HTTP). FSaaS tenants are fully responsible for managing and organising the content of files as well as the file structure. FSaaS *providers* define the file system type and provide means of access control. In order to establish scalability they may use distributed file systems.

DBaaS: Database as a Service, similar to FSaaS, provides structured storage. Unlike FSaaS, the content of a storage container offers semantic information such as the types of the stored records. The existence of semantic information is often coupled with the presence of a querying mechanism and a high-level query language. It is the *tenant's* task to map the stored data to the proper in-memory representation required by the application using the data (e.g. programming language objects). The *provider* is responsible for managing the underlying storage system that may be accessible via various interfaces (e.g. JDBC, HTTP). Besides the mechanisms available for FSaaS and RStaaS, providers can apply techniques on database level to ensure scalability and fault tolerance such as distributed databases and sharding.

DMaaS: Data Mapping as a Service is capable of mapping two different data representations to each other, where only the application's view is provided by the tenant. Its distinctive feature is a direct, widely transparent embedding in the domain of the data source. A DMaaS system may, for example, provide a mapping mechanism that is fully integrated with the programming model of the computation layer. Due to this automatic mapping, there is no immediate programming interface to DMaaS systems. It is the task of the *tenant* to provide an instance of the source domain (e.g. application logic). He may also have to supply meta-data information to help the mapper. *Providers* can establish scalability and

12 formerly named nOSGi
13 http://www.vmware.com/vfabric
14 http://code.google.com/appengine
15 http://www.cumulogic.com
16 http://www.openshift.com
17 http://www.cloudfoundry.com
18 http://www.heroku.com
19 http://www.jelastic.com
20 http://www.google.com/calendar
21 http://www.office365.com

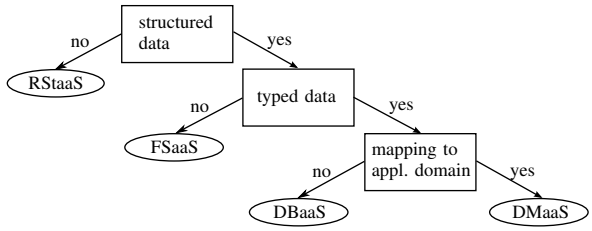

Fig. 2. Decision diagram for determining the type of a storage offering

TABLE IV. EXAMPLES OF STORAGE CLOUDS & TECHNOLOGIES(*)

DBaaS	**DMaaS**
Amazon RDS Google Datastore Cloud Foundry backends	JPA in Google App Engine Mongo Mapper*
RStaaS	**FSaaS**
Amazon EBS Amazon Storage Gateway	Amazon S3 Google Blobstore Google FS/Hadoop FS* Dropbox

fault-tolerance mechanisms using means on any lower view. They can also integrate appropriate mechanisms in the data mapper. Furthermore, they can offer higher order services such as application-level transactions or managed entity beans.

C. Examples

Table IV lists real-world examples of storage services. Fig. 2 depicts how to categorise an existing system. As with computation, the table not only lists existing cloud offerings, but also includes enabling technologies for future offerings.

RStaaS: Amazon Elastic Block Storage (EBS)[2] is an in-cloud storage. Amazon Storage Gateway[2] in addition redirects virtual volumes to on-premise devices and to Amazon S3[2].

FSaaS: Hadoop File System (HDFS)[22] and the Google File System (gFS) [23] are distributed scalable cloud file systems with a general purpose API. In contrast, Amazon S3[2] and the Google Blobstore[14] provide access to named blobs. Dropbox[23] provides web-based storage including synchronization of local files.

DBaaS: Examples for DBaaS range from relational databases such as Amazon RDS[2] down to systems with weaker consistency guarantees and less features. The Google App Engine Datastore[14] provides a schemaless datastore focused on high availability. Cloud Foundry[17] offers various database interfaces as backends including Postgres and MongoDB.

DMaaS: The Java Persistence API (JPA) transparently maps objects to database content and vice versa. The Google App Engine[14] provides such a JPA-like mapping to the Google Datastore[14]. MongoMapper[24] realises a mapping of Ruby data to MongoDB instances. It may, e.g., be used in Ruby on Rails applications running on Cloud Foundry[17].

V. NETWORKING RESOURCES

In addition to computation and storage, cloud systems also offer network services. Even though the term *Network as a Service (NaaS)* is used in the literature [4], [24], it lacks a clear definition. Networking clouds provide means for communication, but may also offer value-added services, e.g. load balancers. In the following, we discuss widely used abstraction layers for networking. Before presenting examples, we classify network services in clouds according to these abstractions.

TABLE V. TYPES OF NETWORK CLOUD SERVICES

OSI layer/ cloud term	**offered network service (examples)**	**classification criteria**
application/ ALaaS	appl. protocol (HTTP); load balancing (for HTTP)	appl.-specific protocol; all protocols from cloud
transport/ TLaaS	appl. to appl. com (TCP socket); load balancing (for TCP)	fixed transport protocol; appl.-based addressing
network/ NLaaS	routing/firewall (iptables); host to host com (raw socket)	multi-hop messages; host-based communication
link/ LLaaS	network access (Ethernet)	configurable topology; no routing support

A. Network Abstraction Layers

The OSI/ISO layering with its seven layers is a widely accepted standard for network stacks. Yet, Layers 5 and 6 are hardly ever separated from Layer 7 in Internet-based systems, and Layer 1 cannot be directly addressed by software. This leaves us four layers of abstraction to consider.

The *link layer* is the lowest considered network layer. Classically, it provides packet-based access to a physical network infrastructure and topology. It defines the type of network interface cards used and decides on issues such as directionality of the link and its bandwidth.

The *network layer* is concerned with sending and routing packages from sender to receiver. It offers multi-hop, host-to-host capabilities and defines the transitive closure of reachable nodes. Moreover, it provides an addressing scheme on host level such as IP addresses.

The *transport layer* enriches the network layer with multiplexing capabilities, which allows addressing applications instead of hosts. It also enables application-to-application communication. In addition, the transport layer may add further capabilities such as reliable message transfer and flow control.

The *application layer* abstracts even further in the sense that it defines application-level interaction patterns and message formats. For instance, it may only allow request-response interaction via HTTP messages.

B. Terminology of the Network Cloud

Naturally, the abstractions can be identified in network-related cloud offers. Hence, we introduce four kinds of network clouds and define their distinctive features (cf. Table V).

LLaaS: With *Link Layer as a Service*, *tenants* can select network interfaces on device level (e.g. linked to virtual machines when using HWaaS). They thus decide on underlying network type, bandwidth and topology; and may run their own Layer 3 protocol on top (e.g, IPv4, IPv6, IPX). When using LLaaS from an application it is essential that the network can be accessed on raw packet level (Layer 2). This is, for instance, not possible using the Java networking API. The cloud *provider* has to connect LLaaS interfaces to (virtual) networks as configured by tenants. He also has to isolate traffic

22 http://hadoop.apache.org/
23 http://www.dropbox.com
24 http://mongomapper.com

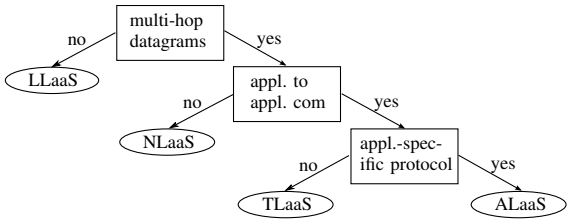

Fig. 3. Decision diagram for determining the type of a network offering

of different tenants, e.g. by the use of VLANs. Value-added services include virtual bridges between different networks, e.g. connecting distant locations and even the Internet.

NLaaS: Network Layer as a Service supports routing and abstracts from the underlying network topology. When using an IP-based NLaaS, *tenants* may have to stick to predefined IP addresses or restricted DNS names that are automatically mapped to IP addresses. Tenants are free to use arbitrary Layer 4 protocols on top. *Providers* offer fully managed Layer 3 multi-hop network access. This covers network management up to host level. Packet filtering, i.e. restricting allowed ports and protocols, as well as load balancers on Layer 3 are examples of value-added services. Typically NLaaS includes a connection to the Internet.

TLaaS: With *Transport Layer as a Service*, *tenants* get fully managed Layer 4 flows bound to a dedicated transport protocol with its respective properties, e.g. reliability and message ordering for TCP. A tenant can neither switch between arbitrary transport protocols nor use its own one. In contrast to NLaaS, TLaaS offerings are not necessarily bound to hosts, but can be bound to applications identified for instance using port numbers. Tenants have to implement the application protocol on top. The cloud *provider* is responsible for providing access to flows between abstract endpoints, e.g. transport layer sockets. Value-added services on this level include Internet connection, (virtual) connection management, firewalls and transport-level load balancers.

ALaaS: Application Layer as a Service offers communication protocols for a specific application type. It may comprise object-oriented and message-oriented middleware as well as streaming and publish-subscribe services. The only task for the *tenant* is to provide information for the communication such as headers and payload. The cloud *provider* is responsible for all communication issues up to the application protocol. By making the tenant adhere to the required interaction pattern providers may bind ALaaS offerings to a programming model that is specific to the application type. Value-added services include Internet connection, deep-packet inspection and scalability support by application-level load balancers.

C. Examples

This section presents real-world examples for network cloud services. Table VI summarises all examples. Fig. 3 shows the decision diagram used for the classification.

LLaaS: Amazon EC2[2] instances make use of virtualised network devices. Amazon Direct Connect[2] realises a direct link between Amazon cloud routers and a tenant's network router. In their Flexiscale utility computing[5], Flexiant manage VLAN tags to separate traffic of different tenants.

TABLE VI. EXAMPLES OF NETWORK CLOUDS & TECHNOLOGIES(*)

TLaaS	**ALaaS**
Inlab Balance*	Cloud Foundry frontend
COSCA network layer	Google App Engine load balancer
Amazon Elastic Load Balancing	HaProxy*/Nginx*
	Amazon Elastic Load Balancing
	LTN SmartCloud
LLaaS	**NLaaS**
Amazon Direct Connect	Amazon VPC
EC2 & Flexiscale virtualised devices	

NLaaS: Amazon VPC[2] manages personal IP address spaces. It further allows integration of Amazon cloud resources with private networks via VPN.

TLaaS: The COSCA network layer [25] introduces virtualised socket access and transparently routes packets to application nodes. Other examples include TCP-level load balancers such as Inlab Balance running in TCP mode or Amazon Elastic Load Balancing[2].

ALaaS: ALaaS offerings include Servlet-based FaaS systems and HTTP-level load balancers that redirect requests to one of many application instances, e.g. in Cloud Foundry[17]. Amazon ELB[2] can also be run on HTTP-level. Other providers use HTTP reverse proxies such as HaProxy[25] and Nginx[26]. LTN SmartCloud[27] provides a video delivery service.

VI. DISCUSSION

In the previous sections, we have presented a cloud taxonomy for computation, storage and network resources. As the different types are not fully orthogonal, we first consider the computation view on storage and network. Afterwards, we discuss the interplay between types. Finally, we present a full classification of Amazon's cloud services. Due to space limitations an indepth discussion and the classification of other providers' offers have to remain subject of future work.

The computation view on network and storage: Most storage and network services cannot be offered without additional computation resources. From a computation point of view, this allows their classification as SaaS which is exactly what happened prior to our classification. Solely RStaaS and LLaaS may be considered as IaaS as they are fundamental resources that do not necessarily need computation resources.

Interplay: Basically, stand-alone services are conceivable in each of our categories. Computation services can be independently combined with both network and storage. This combination can be either explicit or implicit. An explicit combination requires tenant-provided logic to make direct use of the offered resources. In consequence, this logic has to be changed when the cloud provider changes the respective programming interface. In contrary, when the combination is implicit, application logic can consider the resource as *just there* and does not need to execute binding or configuration steps. As an example, the EBS storage device in Amazon EC2 can be automatically linked to EC2 instances.

Apparently, on a technical level, it is always possible to explicitly integrate computation with any type of storage and

[25] http://haproxy.1wt.eu
[26] http://nginx.org
[27] http://www.ltnglobal.com/

TABLE VII. CLASSIFICATION OF AMAZON CLOUD SERVICES

computing service	class
Elastic Compute Cloud	HWaaS
Elastic Beanstalk	FaaS
Elastic MapReduce	FaaS
Simple Workflow Service	FaaS
Data Pipeline	FaaS
Flexible Payments Service	SaaS
Alexa Web Info. Service	SaaS
Alexa Top Sites	SaaS
DevPay	SaaS
Identity/Access Mgmt	SaaS
Marketplace	SaaS
Auto Scaling	SaaS
CloudWatch	SaaS
CloudFormation	SaaS

storage service	class
Elastic Block Store	RStaaS
Storage Gateway	RStaaS

storage service	class
Simple Storage Service	FSaaS
Glacier	FSaaS
CloudFront	FSaaS
Relational DB Service	DBaaS
DynamoDB	DBaaS
SimpleDB	DBaaS
ElastiCache	DBaaS
Redshift	DBaaS
CloudSearch	DBaaS

network service	class
Direct Connect	LLaaS
Virtual Private Cloud	NLaaS
Elastic Load Balancing	TLaaS
Elastic Load Balancing	ALaaS
Route 53	ALaaS
Simple Email Service	ALaaS
Simple Queue Service	ALaaS
Simple Notific. Service	ALaaS

network service. A discussion of the feasibility of implicit integration, in constrast, has to be left open for future work.

Amazon's Cloud Offerings: Table VII lists almost all cloud services offered by Amazon[2]. The table omits services that require explicit human interaction. Even though such services are sometimes labelled as *Human as a Service* [26], this term is subject to on-going controversy and not considered here.

We recognise that Amazon does not offer OSaaS and RaaS as computation service. The absence of OSaaS is probably caused by the fact that it requires a sophisticated mechanism to update an OS transparently. The absence of RaaS makes it impossible to deploy custom server applications without also deploying an EC2 instance. It is also noticeable that there are at least two different kinds of SaaS services: Those that offer a cloud-related service such as Cloud Formation and hence, constitute a meta service; and others that may be directly used such as Identity and Access Management. We consider this difference an indicator that, in addition to network and storage, there may be other service types that are worth being separated from SaaS. Such considerations are orthogonal to our taxonomy and we leave them as future work.

Amazon offers all storage levels except DMaaS. Several FaaS offerings, e.g. the Simple Workflow Service, come with an implicit storage whose presence supersedes the need for data mapping functionality. Nevertheless, DMaaS might be beneficial for other computation services such as Beanstalk.

Amazon provides network services on all layers. It is worth noticing that the Elastic Load Balancer operates on both TLaaS and ALaaS, as it balances TCP and HTTP load. ALaaS services illustrate very well the Janus-faced nature that services may have. For instance, Route 53 is an entire application to control DNS behaviour (hence SaaS), while it also influences communication on application layer (hence ALaaS).

VII. RELATED WORK

The classification of cloud systems has also been targeted by other authors. Early classifications distinguish applications, platforms and infrastructure[28,29] that widely match the established definitions of SaaS, PaaS, IaaS (SPI model) [2].

Several taxonomies extend the SPI model by introducing additional service levels. The UCSB-IBM terminology establishes software infrastructure as a new level that involve IaaS and additionally includes storage (Data as a Service) and communication (Communication as a Service) aspects [27]. PaaS is located above software infrastructure. Moreover, UCSB-IBM also introduces kernel level and firmware level that both reside below software infrastructure level and that both comprise enabling technology such as hypervisors, clustering mechanisms and programmable network hardware. In contrast to our work, UCSB-IBM clearly targets cloud providers and does not provide relevant terminology for tenants.

Rimal et al. [28] list Hardware as a Service in addition to IaaS, but do not clearly distinguish between the two. Prodan et al. [29] add file hosting as a category, yet without considering other storage levels. Zhou et al. [24] consider hardware, system and application level. According to them, the hardware level comprises IaaS and NaaS while the system level is equal to PaaS. The application level comprises SaaS, but also Data as a Service, and Identity and Policy Management as a Service.

Other authors have proposed multi-dimensional taxonomies many of which use the SPI service type or an extension thereof as one dimension. The other dimensions often cover non-functional properties. Security aspects are considered by several authors as a cross-cutting concern [29], [28], [27]. The Use-case Discussion Group [30] adds a dimension that considers the APIs that service provider, service user and service developer use to access the SPI levels. Oliveira et al. [31] classify scientific workflow applications. They extend the SPI categories to include StaaS, which they define to provide *structured ways to access and maintain a storage facility that is remotely hosted*. They define StaaS as Database as a Service when a remotely hosted database is used. Networking aspects, in contrast, are not considered by any of these authors.

Youseff et al. [27] present a model of cloud systems (called Huff's model) that uses the SPI categories for establishing a more fine-grained taxonomy. It separates IaaS into facilities (e.g. buildings), hardware (compute, storage, network), abstraction (e.g. VM monitoring), connectivity (e.g. DNS, security) and management API. In this model, PaaS deals with system integration and middleware support. It realises DB access and additional security support. Finally, the model divides SaaS into data aspects, applications, API and presentation aspects; hence, an application is not necessarily bound to a user interface. According to Youseff et al., data aspects comprise the actual data, meta-data describing the data and the data content, which may be either structured or unstructured.

Strach et al. [32] exclusively consider cloud data hosting and introduce six distinguishing properties. Yet, they only focus on the database level and omit other storage types.

While all those models and classifications describe diverse aspects of cloud resources, none of them takes a tenant-oriented view. Instead, many of them do not have a clear view, focus on the provider view or on technical aspects ignoring the tenant perspective. Moreover, only few consider network services; and none of them really differentiates between different kinds of them. Many of these authors provide a set of classification types that only reflects current offerings. They do,

[28] http://gigaom.com/2008/06/16/defogging-cloud-computing-a-taxonomy/

[29] http://blogs.forrester.com/09-04-20-yet_another_cloud_%E2%80%93_how_many_clouds_do_we_need

however, not define a clear separation between their categories and, even worse, of the types within a category.

Our classification, in contrast, focuses on a high-level view of the resource types computation, storage and network. It exploits existing, widely-known abstraction layers, which eases the differentiation between the layers for a dedicated resource type. Hence, it provides a precise view on different types of computation services and further defines a strong and comprehensive terminology for both network and storage services. By omitting technical aspects, our taxonomy offers a clear terminology that serves both cloud tenants and cloud providers for understanding each other's needs and services.

VIII. Conclusions

Current classifications reduce cloud systems basically to IaaS, PaaS and SaaS [33]. This paper goes beyond that terminology and classifies cloud offerings in a precise and definite way.

Our first contribution is the differentiation between the three resource types computation, storage and network. We refine PaaS and IaaS systems with respect to computation. Moreover, we present the first comprehensive taxonomy of network and storage services in clouds. Our taxonomy is usage-oriented and is based on wide-spread, accepted and highly mature abstractions for each of the resource types. The use of these well-known abstractions makes it easy to adopt our terminology. The real-world examples we presented for all terms of our taxonomy demonstrate its usability and relevance.

Our second contribution is a detailed discussion of the impact of acquiring and offering cloud services of a particular type and abstraction level. The usage-oriented view of the taxonomy helps system administrators, software developers and system architects when specifying requirements for and architectures of cloud-based infrastructures. It further enables the re-use of legacy specifications based on existing abstractions when moving to the cloud.

Concluding, the taxonomy introduced in this paper can certainly help to avoid confusion about provided cloud services, and is a leap forward towards a fine-grained classification of cloud systems. For the future, we envision further extensions by integrating non-functional properties, such as security and identity management, and the provider view on cloud systems.

References

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *GCE '08*, Nov. 2008, pp. 1–10.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," 2009.

[3] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," 2009.

[4] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *SIGCOMM Comp. Commun. Rev.*, vol. 41, pp. 45–52, 2011.

[5] R. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.

[6] W. Zhu, C.-L. Wang, and F. C. M. Lau, "Jessica2: a distributed Java Virtual Machine with transparent thread migration support," in *Cluster Comp., 2002. Proc. 2002 IEEE Int. Conf. on*, 2002, pp. 381–388.

[7] A. I. Avetisyan et al., "Open Cirrus: a global cloud computing testbed," *IEEE Computer*, vol. 43, pp. 35–43, 2010.

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *CCGRID '09*, 2009, pp. 124–131.

[9] R. Watson, "Introducing supporting infrastructure for trusted operating system support in FreeBSD," in *Proc. of the BSDCon 2000*, 2000.

[10] B. des Ligneris, "Virtualization of Linux based computers: the Linux-VServer project," in *HPCS '05*, 2005, pp. 340–346.

[11] K. Kolyshkin, "OpenVZ: virtualization in Linux," 2006.

[12] Z. Liang, W. Sun, V. N. Venkatakrishnan, and R. Sekar, "Alcatraz: an isolated environment for experimenting with untrusted software," *ACM Trans. Inf. Syst. Secur.*, vol. 12, pp. 14:1–14:37, January 2009.

[13] R. Brunetti, *Windows Azure step by step*. Microsoft Press, 2011.

[14] S. Kächele, J. Domaschka, and F. J. Hauck, "COSCA: an easy-to-use component-based PaaS cloud system for common applications," in *CloudCP '11*. New York, NY, USA: ACM, 2011, pp. 4:1–4:6.

[15] D. Caromel, C. Delbe, A. Di Costanzo, and M. Leyton, "ProActive: an integrated platform for programming and running applications on grids and P2P systems," *Computational Methods in Sci. and Techn.*, vol. 12, no. 1, pp. 69–77, 2006. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00125034/

[16] S. Kächele, J. Domaschka, H. Schmidt, and F. J. Hauck, "nOSGi: a POSIX-compliant native OSGi framework," in *COMSWARE '11*. New York, NY, USA: ACM, 2011, pp. 4:1–4:2.

[17] J. Kennedy, C. Serfon, G. Duckeck, R. Walker, A. Olszewski, and S. N. et al., "ATLAS computing operations within the GridKa cloud," *J. of Physics: Conf. Series*, vol. 219, no. 7, p. 072039, 2010.

[18] C. Morin, "XtreemOS: a grid operating system making your computer ready for participating in virtual organizations," in *ISORC '07*, 2007, pp. 393–402.

[19] Microsoft Cooperation, "SoftGrid application virtualization: the next frontier," Whitepaper, 2007, 2nd Edition.

[20] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *OSDI '04*, 2004.

[21] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Boston, MA: Addison-Wesley, 1987.

[22] Hupfeld et al., "The XtreemFS architecture—a case for object-based file systems in grids," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 17, pp. 2049–2060, Dec. 2008.

[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003.

[24] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *Universal Communication Symp. (IUCS), 2010 4th Int.*, 2010, pp. 40–46.

[25] S. Kächele and F. J. Hauck, "COSCAnet: virtualized sockets for scalable and flexible PaaS applications," in *UCC'13: Proc. of the 6th IEEE/ACM Int. Conf. Utility and Cloud Computing*. IEEE, 2013.

[26] K. Petruch, V. Stantchev, and G. Tamm, "Cloud computing governance aspects," SRH Hochschule Berlin, Whitepaper, 2011.

[27] L. Youseff, D. M. Da Silva, M. Butrico, and J. Appavoo, "Understanding the cloud computing landscape," *Cloud Computing and Software Services*, pp. 1–16, 2010.

[28] B. Rimal and E. Choi, "A conceptual approach for taxonomical spectrum of cloud computing," in *Ubiquitous Information Technologies Appl. ICUT '09. Proc. of the 4th Int. Conf. on*, 2009, pp. 1–6.

[29] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *10th IEEE/ACM Int. Conf. on Grid Comp.*, Oct. 2009, pp. 17–25.

[30] Cloud Computing Use Cases Discussion Group, "Cloud computing use cases v4," Whitepaper, 2010.

[31] D. Oliveira, F. Baião, and M. Mattoso, "Towards a taxonomy for cloud computing from an e-science perspective," in *Cloud Computing*, ser. Comp. Comm. and Netw., N. Antonopoulos and L. Gillam, Eds. Springer London, 2010, pp. 47–62.

[32] S. Strauch, O. Kopp, F. Leymann, and T. Unger, "A taxonomy for cloud data hosting solutions," in *CGC 2011*, 2011, pp. 577–584.

[33] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? An architectural map of the cloud landscape," in *CLOUD '09*, 2009, pp. 23–31.