Developing Cloud Software

Algorithms, Applications, and Tools

Edited by

Ivan Porres
Tommi Mikkonen
Adnan Ashraf

TUCS General Publication
No 60, October 2013
ISBN 978-952-12-2952-7
ISSN 1239-1905

1 Introduction to Cloud Computing Technologies

Adnan Ashraf[1], Mikko Hartikainen[2], Usman Hassan[3], Keijo Heljanko[3], Johan Lilius[1], Tommi Mikkonen[2], Ivan Porres[1], Mahbubul Syeed[2], and Sasu Tarkoma[4]

[1]Åbo Akademi University, Turku, Finland
Email: firstname.lastname@abo.fi

[2]Tampere University of Technology, Tampere, Finland
Email: firstname.lastname@tut.fi

[3]Aalto University, Espoo, Finland
Email: firstname.lastname@aalto.fi

[4]University of Helsinki, Helsinki, Finland
Email: firstname.lastname@helsinki.fi

Abstract–This chapter presents the main technologies currently used in cloud computing, what are the main commercial offerings and what are their programming models. We discuss hardware virtualization technologies used in datacenters, three different service abstraction levels: infrastructure, platform and application and the main driver and adoption problems in cloud computing.

Keywords-Cloud computing, virtualization, scalability, IaaS, PaaS, SaaS.

The authors are listed in alphabetical order.

1.1 Technology Drivers and Adoption Problems

The cloud computing paradigm is a new framework for purchasing computing as a utility (Utility Computing) instead of using traditional datacenters to provide data processing capability. Perhaps the best overview of the technology drivers behind cloud computing is given in the report "Above the Clouds" [7, 6] from Berkeley, which also discusses the main obstacles and opportunities in cloud computing.

Cloud computing contains a number of technologies that are required to realize the "computing as utility" promise made by cloud computing vendors. Many of the key technologies have existed already before the term "cloud computing" was invented, while others have been born out of the Internet-scale deployment of computing in distributed data centers by several big companies such as Google, Amazon, Yahoo, and Salesforce.com. Consequently there are many definitions of cloud computing but the main distinguishing features of cloud computing are:

- Computing resources can be purchased *on-demand* from a virtually unlimited supply.
- The capital expenses needed to purchase computing resources up-front are changed to operational expenses, shifting the capital investment risk for under/overprovisioning to the cloud computing vendor.
- Computing is priced with a *pay-as-you-go* pricing model where capacity can be scaled up and down on a short term basis.

The National Institute of Standards (NIST) in the US has also decided to emphasize the elasticity of computing resources in their definition of cloud computing [40], which is a definition we can largely agree with. One frequently cited article defines the cloud as follows:

> Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured 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 [54].

One of the main drivers for cloud computing are the economics of scale in datacenter building and operations costs. The pricing of hardware, electricity, cooling, and especially global network capacity is much more competitive

when bought for data centers with tens or hundreds of thousands of servers instead of small scale datacenter operations with maybe a hundred to a thousand servers.

This chapter focuses on the technical aspects of the available cloud computing platforms and mostly overlooks financial and business aspects of cloud computing. The key technologies we are discussing in this chapter are:

1. *Hardware virtualization*: In order to allow for maximum flexibility of offering customers the illusion of dedicated computing, storage, and networking infrastructure on a computing infrastructure shared with other clients, virtualization technologies are heavily used. Section 1.2.1 discusses the commonly used virtualization technologies, for example Xen is used by the Amazon cloud offering.

2. *Sandboxing*: Sometimes the overhead per (Linux or Windows) virtual machine can be quite significant, as typically each virtual machine is running its own kernel instance. Another commonly used approach is to use high-level programming languages with sandboxing (Python, Java, etc.) virtual machines to do the isolation between clients on a multi-tenant cloud infrastructure. Examples of this are the Google App Engine that uses Python and Java runtime environments that have been sandboxed to disallow things such as writing to files and opening of network sockets. Sandboxing is discussed in more detail in Section 1.2.2.

3. *Virtualized network block storage*: This is similar to having a virtualized network file server with redundancy available to mount storage to a virtual machine. A typical product is the Amazon Elastic Block Store, which can be used to mount network attached storage to a cloud server instance. The technologies employed here are similar to traditional file serving technology with a virtualization layer on top. This technology is discussed in more detail in Section 1.2.3.

4. *Scalable datastore*: One of the key technologies in scalable web services is the scalable datastore, a database component to manage the data behind the web application. There is a large interest in *NoSQL* (for *No SQL* or *Not Only SQL*) datastores. This is a quite central topic that we will discuss at length in Section 1.3.1.

5. *Scalable file storage*: Another basic cloud building blocks is that of geographically replicated hashed file storage. Examples of this service include the Amazon S3 storage system, and the recently announced

Google storage system. The data in replicated hashed file storage is accessed through a REST (via HTTP) based interface, and can thus be directly linked into in web sites. Replicated hashed file storage is often used to store the virtual machine images that are loaded to virtual machines at startup, to store backups of block storage, as well as to store large binary blobs (images, software, etc.) of data served through Web servers, as well as the seeds of data to be distributed through content distribution networks. These will be discussed in Section 1.3.2

6. *Scalable batch processing*: One of the key new technologies used in the cloud are scalable batch processing systems. The main reference implementation here is Google MapReduce and its open source implementation Apache Hadoop. This will be discussed in detail in Section 1.3.3.

7. *Cloud controller*: All of the cloud offering provide either a command line or a Web based interface to deploy and administer cloud computing, including not only computing but also storage and networking. Examples include the Eucalyptus Cloud Controller (CLC), the OpenNebula Virtual Infrastructure Manager, and the Web based Google App Engine Administration Console. These technologies will be discussed in Chapter 1.4. The different types of approaches (Infrastructure as a Service, Platform as a Service, Software as a Service) are addressed in individual Sections 1.4.1, 1.4.2, and 1.4.3.

In addition to technologies listed above most of the traditional Web application and web content serving technologies such as web services using load balancing and caching are quite predominant in Cloud application development, as one of the main drivers of scalable technologies are Web applications deployed at the massive Internet scale. For example, clouds are used to do the distribution of static Web content globally. Several providers such as Akamai and Amazon with its Cloudfront service offer caching services for static content using globally distributed network of datacenters to minimize the Internet data transfer fees for providing Web-based services. As these are basically Web serving content delivery networks, they will not be discussed further in this chapter.

Figure 1.1 presents an overview of the central elements in cloud computing. The lowest layer pertains to datacenters, clusters, and networking. Flexibility is achieved by using virtualization, and creating and moving platform instances at runtime. On the client-side, the Web browser is becoming a key platform for applications. Various Web application frameworks are then used through the browser. On a higher level, the aim of the cloud infra-

structure is to support on-demand service creation, management, and access. Open APIs are key components for interoperable cloud-based systems.

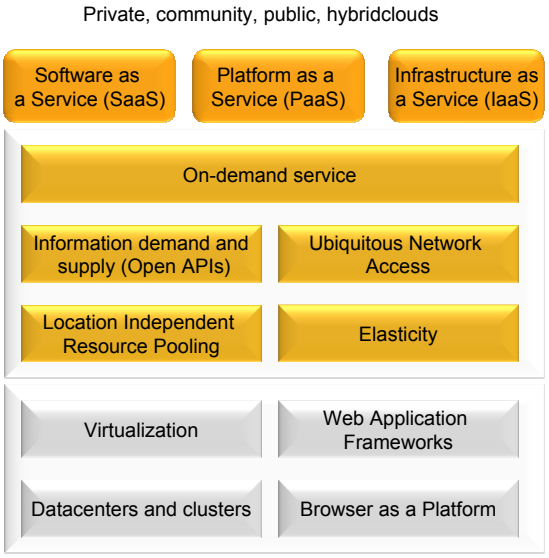

Figure 1.1: Overview of cloud services

The figure highlights the different roles found in cloud computing, namely the service consumer, provider, and developer. The service provider is a central element of cloud computing, because it facilitates the deployment and execution of various building blocks. The service provider achieves flexibility through virtualization and dynamic configuration of the runtime system. This flexibility that takes the supply and demand of content into account can be seen as a central feature of clouds. Virtualized resources include networks, CPUs, and storage. In order to implement and manage a cloud platform, a number of management features are needed. The necessary management features include reporting, Service Level Agreements, capacity planning, and billing. The software layer providing the virtualization is called a virtual machine monitor or hypervisor. A hypervisor can run on bare hardware (Type 1 or native VM) or on top of an operating system (Type 2 or hosted VM). The service developer uses APIs exposed by the cloud platform and the software it is executing. The service developer needs to have tools for service creation, deployment and publishing, and analyzing the service at runtime.

The service consumer uses various APIs and user interfaces to access the deployed services.

The services of Cloud computing can be divided into three categories: Software-as-a-Service (SaaS), in which a vendor supplies the hardware infrastructure, the software product, and interacts with the user using a portal. Platform-as-a-Service (PaaS), in which a set of software and development tools are hosted by a provider on the provider's infrastructure, for example, Google's AppEngine. Infrastructure-as-a-Service (IaaS), which involves virtual server instances with unique IP addresses and blocks of on-demand storage, for example, Amazon's Web services infrastructure. Figure 1.2 shows layer architecture of cloud computing.

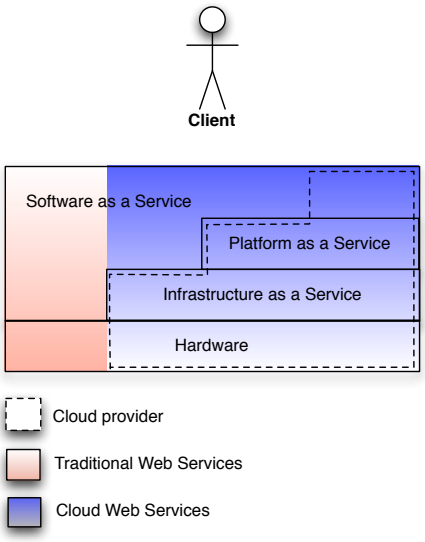

Figure 1.2: Layer architecture of cloud software

This chapter focuses on the cloud computing technologies from two different perspectives. The first one is the view of the application programmer: What kinds of application frameworks do the different cloud providers provide and what are the benefits and drawbacks on the frameworks in question. The second perspective is that of a cloud service provider: What technologies are employed in the cloud platforms, especially concerning the scalability of services. Many of the available implementations share similar (or even the same) components, and they are often composed by mixing and matching

proprietary and open source components.

1.2 Background Technologies

There are some commonly used technologies that have already existed before the introduction of cloud computing, but which have lately experienced a renaissance due to the introduction of cloud computing. Such technologies include in particular the following:

- hardware virtualization technologies
- virtualized network block storage
- sandboxing

These technologies play such a big role in current approaches to cloud computing that we have decided to address them separately.

1.2.1 Cloud Hardware Virtualization Technologies

Virtualization, or the capability to make one computer appear as several computers or a totally different computer, is a 4 decades old idea, introduced by IBM in its 7044 computer together with the *Compatible Time Sharing System* (CTSS) developed by MIT.

Virtualization is key component in Cloud computing as it allows one to distinguish the underlying hardware from the operating system, and allows the cloud hardware provider to easily let the client run any operating system that is needed.

The report *Secure Virtualization and Multicore Platforms State-of-the-Art report*, by Heradon Douglas and Christian Gehrmann [20] provides a good overview of the underlying techniques and issues.

Virtualization techniques can be split into two main approaches:

1. *System virtualization* in which the entire system is virtualized. This enables multiple virtual systems to run concurrently totally isolated from each other. The *hypervisor* or *virtual machine monitor* provides access to memory, devices, network, including the CPU. As a consequence, the Guest operating system thinks it has the machine for itself.

2. *Para-virtualization* in which the guest operating system is modified to cooperate with the hypervisor. The guest is modified to use interfaces that are safer or more efficient to use than the original guest operating system interfaces.

The two main hypervisor types are the following:

- Type 1 runs directly on the host's hardware and executes the guest operating system.
- Type 2 (or hosted) runs within an operating system.

System virtualization typically requires hardware support from the processor. Such support is provided e.g. by Intel Virtualization technology Intel-VT [33], AMD's support for virtualisation AMD-V [4], or ARM's TrustZone [53] . Thus, system virtualization is typically only supported for newer IA-32, Xeon, Itanium, Athlon, and Phenom families of processors.

System virtualization can also be achieved by *pre-virtualization*, in which the guest operating system code is scanned before execution by the hypervisor and then modified at run-time, thus resembling a Just-In-Time compiler. This approach naturally comes at a premium performance wise.

A comprehensive list of different virtualization solutions is available on wikipedia [15]. The list contains over 70 different solutions, with either open-source, or commercial licensing. Below, we focus on 3 solutions, KVM, XEN, and VMWare. This choice is motivated by the fact that XEN is the open-source market leader, while VMWare can be considered the commercial market leader. KVM has been included in the list as it is gaining quite a lot of interest in the Linux community.

XEN

The Xen hypervisor was created at the University of Cambridge at the end of the 1990's as part of the Xenoserver research project. The first open-source release of the Xen hypervisor was done 2002, and the current version of the hypervisor is 4.0. It can thus be considered a mature and stable product. Commercial support is provided by XenSource Inc. Xen is also provided as the virtualization solution by solution providers like Red Hat, Novell, and Sun. Xen is currently marketed by Citrix (`http://www.citrix.com/`).

Xen is usually considered a para-virtualization solution, although version 4.0 adds capabilities for system virtualization. Xen handles device drivers by running a special operating system in a special high-privilege Xen domain (dom0). This operating system handles all device driver requests and has optimized device drivers for the available hardware. The guest operating system then has to be modified to work against these interfaces.

VMWare

VMware (`http://www.vmware.com/`) offers a commercial hypervisor ESX [58]. ESX runs on "bare metal" and does not require a separate operating system. Instead it comes with an included Linux kernel that is booted first and used to load special device drivers and other features required by the hypervisor. The Linux kernel provides access to all devices of the system to the guest operating system using these drivers. In principle, VMWare is thus a para-virtualization solution. However "Scan-Before-Execution" the VMWare marketing term for its run-time pre-virtualization technology, allows the guest operating system to run unmodified on VMWare.

KVM

KVM is a newcomer among virtualization solutions. What KVM provides is a solution to make a Linux kernel into a hypervisor by loading a module. Each guest operating system is now a process in user-mode of the KVM hypervisor. KVM assumes that it is running on a processor with hardware support for virtualization. Thus it is not possible to run it on older processors, nor is any such support planned.

KVM consists of two parts: the KVM module that is used to virtualize memory and QEMU [46], an emulator, for virtualization of I/O.

Summary

Figure 1.3 presents a summary of well-known hypervisors including the three hypervisors mentioned above. The key properties of hypervisors include whether or not the system is open source, on what level does it operate (type 1 or 2), what hardware is supported and what hardware can be virtualized, and additional features such as live nested virtualization and live migration.

1.2.2 Sandboxing

Sandboxing is a commonly used security mechanism that separates programs and resurces from one another. By including the different applications in separate sandboxes, the infrastructure used to host them can be shared by numerous applications, some of which may have different trust levels. Moreover, it is easy to use experimental software in the same infrastructure as the production software, since by encapsulating the different systems into sandboxes of their own they can not cause harm to each other.

	Xen	KVM	VMWare	Hyper-V
Open source	Yes	Yes	No	No
Type 1/2	1.5 (Dom0 privileged guest)	1.5 (Linux kernel, Qemu)	2	2
Hardware	x86 / x86_64	x86 / x86_64	x86 / x86_64	x86_64
Virtual hardware	Qemu: x86 / x86_64	Qemu: X86 / x86_64	x86 / x86_64	x86 / x86_64
Features	Nested virtualization, live migration	Nested virtualization, live migration		
Association	Citrix	Red Hat / Intel	VmWare	Microsoft

Figure 1.3: Comparison of hypervisors

In the simplest form, sandbox systems are really isolating applications from each other and the hosting operating system in full. They have no way to interact, exept indirectly in terms of processor time, which they must share, provided that the same computing infrastructure is used. However, it is also common that not everything is isolated to such a degree, but different privileges can be offered to applications in exchange for e.g. providing reasonable evidence that the developer is authorized to use some services. Such a fine-grained sandboxing system can be implemented using so-called capabilities, which can be used to provide an access to different resources based on more detailed definitions.

Since sandboxing has been proven a really useful technology in many fields of computing, it is not uncommon to find different implementations that have been geared towards some particular area of application. In the realm of cloud computing, the most common use of sandboxing is together with a virtualization system, where the goal of sandboxing is to provide an illusion of a single computer, dedicated to the developer, which is isolated from the rest of the applications run in the same server farm or datacenter.

1.2.3 Virtualized Network Block Storage

The goal of storage virtualization is to abstract the physical location of the data from users and developers. Provided with adequate implementation,

this leads to location independence. The role of the virtualization storage is to provide a mapping from the perceived data to the actual physical location.

For obvious reasons, the actual form of the mapping is implementation dependent. For example, there may be limitations on granularity of the mapping, where different implementations provide a scale from a full, physical single disk residing physically in a certain computer to small subsets of the disk, provided in e.g. megabytes or gigabytes.

Commonly available implementations allow heterogeneous management of multi-vendor storage systems. Consequently it is possible to build a virtualized system out of best-suited component subsystems, which may provide different quality of service in terms of e.g. access speed.

The benefits of virtualized storage systems in general are many. They include at least the following:

- *Non-distruptive data migration.* With the virtualized system, data can be migrated to different locations even if it is being used. Consequently, there is more freedom on organizing the data in the network in accordance to the services that are being provided and computers and data storages that are currently available.

- *Improved utilization.* As is general with cloud computing, one of the gains of using a virtualized storage is the ability to use the available resources in a more optimized fashion.

- *Simplified management.* Another common gain of cloud computing is the fact that one needs less management points when relying on virtualized storage. This in turn simplifies management.

1.2.4 Network Virtualization

OpenFlow is an open standard that allows to run experimental protocols in production networks [39]. OpenFlow is a feature added to switches, routers, access points (APs) and basestations, allowing these datapath devices to be controlled through an external, standardized API. Major switch vendors are now implementing the system and it is used by universities to deploy innovative networking technology. Basically, OpenFlow is a software-defined Ethernet switch. Software-defined networking is expected to be one of the new emerging research topics in computer networking.

Figure 1.4 presents an overview of the OpenFlow protocol. Routers and switches implement the open API that allows administrators and control components to create, modify, and remove flows in the flow table. The protocol is a step towards software-defined networks.

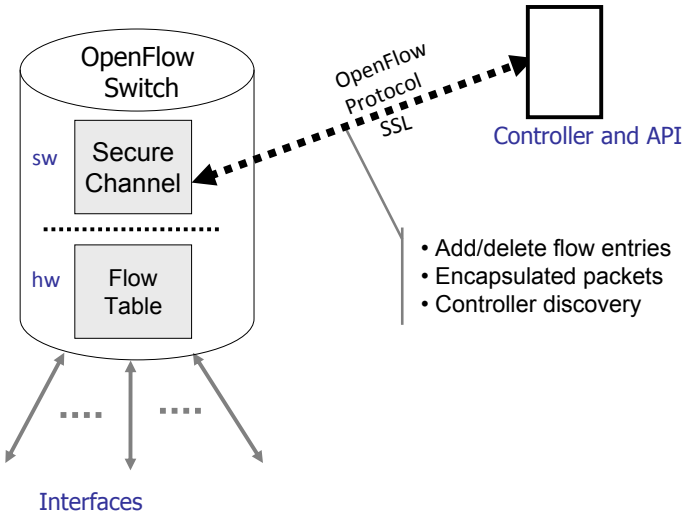

Figure 1.4: The OpenFlow protocol

NOX is an open source network control platform that can control all connectivity on the network including forwarding, routing, which hosts and users are allowed. NOX is a control plane element that can be used with OpenFlow switches [28].

1.3 Scalable Cloud Technologies

A key driver behind cloud computing are Web applications deployed at the Internet scale. One of the problems with these applications is the variability of demand in the capacity needed to serve users. An application that proves successful on the Internet, for example a game, can have its load increased dramatically in a very short time period. For example, when released as a Facebook plugin, Animoto (`http://animoto.com/`) traffic doubled every 12 hours for 3 days [6]. If the application serving the load is not constructed from scalable building blocks, such scaling to heavy Internet scale loads is not possible. Another way to see these scalable cloud computing technologies is that they are software based approaches to horizontally scale data processing to be run on a large number of small machines instead of a few very powerful machines. This can have potential in cost reduction for hardware as well as potential for energy consumption reduction. As an example, Google discusses

the high energy consumption of typical servers at idle, and are suggesting architectures that are more energy-proportional, that is, the server power should be proportional to its application performance level [5, 8]. Thus the cloud should also be able to scale not only up but also down: when application load decreases, the number of active servers needed to serve the load should also be automatically decreased as well. Another example is the paper [52], which discusses the use of virtualization technology to power off idle machines for better power efficiency during hours of low load.

The scalable cloud computing technologies presented here can also be seen as a preview of technologies to be employed on a smaller (company private datacenter) scale in the future, as the need of scaling of systems to a "private cloud" of small commodity servers in an economical fashion becomes an issue.

1.3.1 Scalable Datastore

One of the key components to scalability on the Internet scale is that of a scalable datastore. The datastore is needed as a backend to Web applications serving dynamic Web pages built on top of the Representational State Transfer (REST) architectural style [23]. These datastores can be seen as databases with often very limited functionality but they are able to scale to even tens of thousands of servers in a single datastore instance. For example, all of the Google App Engine applications are sharing a single datastore instance making up a massively distributed datastore system.

Example scalable cloud datastores include Google Datastore (based on Google Bigtable [12, 13]), Amazon SimpleDB (`http://aws.amazon.com/simpledb/`), Amazon Dynamo [19] (based on Chord [48, 49]), the Yahoo! PNUTS (also called Sherpa) [16] system (which internally uses MySQL (`http://www.mysql.com/`) for ordered storage), the open source Apache Cassandra [36] (`http://cassandra.apache.org/`) system, the open source Tokyo Tyrant (`http://1978th.net/tokyotyrant/`) system, and the open source Project Voldemort (`http://project-voldemort.com/`), just to name a few examples. As can be seen from the list of systems above the field is still quite fragmented but also in very active development.

NoSQL Datastores

One of the current hot topics is *NoSQL* (for *No SQL* or *Not Only SQL*) datastores. A good short overview of the topic can be found in [50, 51]. In traditional database systems literature [29] the guarantees given to the user by most traditional database systems are denoted with the term ACID

(Atomicity, Consistency, Integrity, Durability). A part of the NoSQL movement is instead insisting on that a much more limited datastore functionality called BASE [24] (Basically available, soft state, eventually consistent) is to be adopted. The main idea of datastores implementing BASE is to favor availability (and often also low write latency) over consistency.

An example given by the Amazon Dynamo paper [19] is that of a shopping cart: A customer would prefer to have the shopping cart application available even though some datacenters of the Amazon infrastructure would not be currently available due to e.g., disk failures or network connection problems to other datacenters which might also manipulate the same shopping cart. In the rare occasion that a shopping cart update should manage to write its data only to a subset of the servers storing the database, and another update for the cart comes in, the database would contain two conflicting versions of the shopping cart data. This is conceptually very similar to merge conflicts in distributed revision control systems. When such inconsistency is noticed, the shopping cart application would eventually be presented with two conflicting versions of the contents of the shopping cart.

The approach taken by Amazon was to use the union of the two shopping carts as the "true contents" of the shopping cart in the case of such rare failure, and to handle the exceptional cases of missing deletes of items from the shopping cart by the user being presented with a slightly wrong contents of the shopping cart. This allowed the Amazon system to scale easier but at the expense of having each of the applications handle the inconsistent versions of the datastore contents in an application dependent fashion. Amazon also employs a lot of ACID systems in billing and order manipulation, but these are often not customer facing applications, and thus can tolerate the longer latencies and lower availability than the applications the customers are directly interacting with. So a good system design might employ a mixture of BASE and ACID datastores, depending on the application and its inherent requirements.

So one of the main debates in scalable datastores is BASE vs. ACID [45]. The debate is based on a theoretical result on distributed databases: Brewer argued in his CAP conjecture that it is impossible to build a database system that is:

- *Consistent*: The client perceives that a set of operations has occurred all at once.
- *Available*: Every operation must terminate in an intended response.
- *Partition tolerant*: Operations will complete, even if individual components are unavailable.

This was later proved to indeed be true [26], and without additional assumptions it is indeed impossible to create a distributed database system satisfying all the three properties at the same time. To overcome this impossibility result, one has to drop one of these three guarantees by either:

- Discarding partitions: CA: The database has to be centralized which immediately leads to scalability problems.
- Discarding availability: CP: A database server must disallow writes in case the update can not be written to a majority of the database servers due to network partition.
- Discarding consistency: AP: The database must allow for inconsistency of writes in case of network partition of the database servers.

Brewer gives in his PODC 2000 keynote some examples of systems in the different classes:

- CA systems include: Single-site databases, LDAP, and NFS. These systems are often based on two-phase commit algorithms, or cache invalidation algorithms.
- CP systems include: Distributed databases, distributed locking systems, and majority protocols. The systems often use pessimistic locking or majority (aka quorum) algorithms such as Paxos [37].
- AP systems include: Filesystems allowing updates while disconnected from the main server such as Coda, Web caching, and DNS. The employed mechanisms include cache expiration times and leases.

One of the key observations here is that caching of Web applications have already discarded consistency as caches might contain stale data not available anymore in the database. So in a way data staleness of data already exists in Web applications to some extent.

When we look at the BASE vs. ACID debate, it is interesting to note that the Google App Engine Datastore is by default ACID. The BASE proponents are the Amazon Dynamo and Yahoo! PNUTS (Sherpa) system, as well as the Cassandra datastore that can be configured either in BASE or ACID mode on a per table basis.

The Google App Engine Datastore as well as most other NoSQL datastores support a very limited notion of transactions, and most systems do not support table joins used in relational databases. The Google Datastore stores its key values in a sorted fashion by the primary key, and thus range queries

are still efficient. Also Google itself notes that Datastore writes are expensive compared to reads that can be usually served from a cache. This is not surprising given the fact that the written data has to be replicated (using GFS) to ensure durability. Thus a common scalable application programming strategy Google proposes to be used is to batch datastore writes using a memory cache and only periodically flush the required writes to the datastore "backup". So the Google Datastore can be seen as quite database-like datastore system (without complex transactions or table joins) engineered for massive scalability.

Key/Value Datastores

The other interesting approach to datastores is that of pure key/value stores. These are systems built on top of massive distributed hash-tables, and the only thing one can do is lookup a value based on a key (md5 checksums of key material are often used). One of the ideas underlying these systems was presented in the peer-to-peer system Chord [48, 49]. It uses a technique called *consistent hashing* to allow for nodes to enter and leave a peer-to-peer content lookup network with minimal overhead. Basically if a network contains n nodes each containing a set of values, if one node leaves the system, only $\mathcal{O}(\frac{1}{n})$ data items have to be reallocated to the remaining nodes. In other words only the data that went missing by the node leaving needs to be reallocated while other nodes do not have to move the data they still store around to other nodes. Similar result also holds for new nodes entering the system. The Amazon Dynamo and its descendant the Cassandra system uses this idea to implement a scalable key/value store with replication, and it is used in internal Amazon infrastructure (to implement parts of Amazon S3, most likely file location lookup). Many of the other key/value stores use the consistent hashing primitive as well to do load balancing in a way that minimizes the effects of adding or removing servers to the servers remaining to serve the application load. For an overview on the performance of Cassandra, which copies heavily from Dynamo for its load balancing and configurable consistency levels, as well as from Bigtable for its on-disk storage design, see [35].

Also in-memory caching systems based on key-value lookup such as the open source `memcached` (`http://www.memcached.org/`) are heavily used in cloud computing offerings and usually used as very high performance caches to persistent databases. However, as `memcached` is just a least-recently-used cache (forgetting data is a feature, not a bug), we do not discuss it at length here. It should be noted that `memcached` seems to be one of the most widely deployed pieces of infrastructure, though. For example, the combination of

`memcached` with MySQL database is often used datastore solution for applications with small load, and for applications where the load is almost exclusively read-load and MySQL replication is used to improve the read capacity by having lots of read-only MySQL replicas of a single master MySQL database. However, once write load is significant, the replication overhead becomes problematic with a straightforward MySQL replication solution.

Scalable Consensus Datastores

One of the interesting pieces of technology Google uses in their system is the Chubby system [10, 11]. It is a highly fault tolerant database, where the contents of the database are replicated over a number of servers. If more than half of the servers are available, the database can serve requests. The system is used to mainly maintain configuration data for other Google services, for example BigTable uses Chubby to store the master node identity, servers that are up, and the location of the root table in GFS. Bigtable (as well as other Google services) has been designed to reboot quickly if needed, and it fetches the initial configuration data from Chubby. Google also uses Chubby internally to store the DNS records of internal services, which allows for atomic updates of DNS data. Chubby is thus a piece of the infrastructure that needs to be kept running 24/7. The design is based on the classic Paxos algorithm [37]. In Paxos writes are very expensive, but read performance can be made quite good by using extensive client caching and cache invalidation technologies [10, 11]. An open source implementation of the Paxos algorithm is Keyspace by Scalien (http://scalien.com/keyspace/). Also the Cassandra datastore can be configured to implements a majority algorithm on a table basis.

1.3.2 Scalable File Storage

Another key component in scalable cloud computing technologies is that of scalable file storage. The main reference here is the Amazon S3 system that implements a file storage that is geographically replicated for durability. Amazon says that the data should still be available even if two different datacenters are affected. In practice this means that all of the data must be available on at least three geographic location. In practice storing data in many different geographically disjoint locations means high latency for updating data. The Amazon S3 system uses standard HTTP methods to read and write data, and also data stored on S3 can be directly linked into on Web pages as standard URLs. Google has a new competing product with almost identical interface called Google storage. Scalable file storage

can use many methods in data storage inside the cloud: it can compress the customer data, it can deduplicate (detect identical data blocks and only store one copy of the data), and it can replicate the data not only for storage but also for caching purposes on geographically disjoint locations, trying to minimize latencies in accessing data stored on the file storage.

These scalable file storages can be used for example to: Store virtual machine images and disk images for servers, backups, serving static content to the Web, seeding content distribution networks. For example Amazon S3 can be used to seed BitTorrent feeds of data, thus feeding peer-to-peer distribution of content on the Internet.

For filesystem type interface to data, see the Google filesystem (GFS) [25], and the Hadoop HDFS filesystem (`http://hadoop.apache.org/common/docs/current/hdfs_design.html`) whose design is heavily influenced by GFS.

1.3.3 Scalable Batch Processing

In addition to scaling up datastores and storage horizontally to a large number of machines, also asynchronous batch processing of data needs to horizontally scale to a large number of computers. The main cloud reference here is the Google MapReduce system [18, 17] and its open source clone Apache Hadoop (`http://hadoop.apache.org/`) originally developed at Yahoo!.

The Google MapReduce is an implementation of MapReduce, a distributed batch programming paradigm based on functional programming techniques. It consists of a framework for automatically distributing batch jobs onto a large number of worker machines, and the framework takes care of the scheduling and synchronization between the jobs. An overview of the system is presented in Figure 1.5. The input data to MapReduce is given in a very large file, usually split into large (64MB is typical) chunks of data to be processed in parallel by n mapper tasks. Each one of the chunks contains number of records (for example lines of text), which are fed into a user provided *map* function record at a time. The map function processes each line at a time and decides for each record to produce some number of records consisting of a pair $(key, value)$. Next the MapReduce framework does what is called a *shuffle* operation, which groups all the data from all the parallel mappers using a (user provided) hash function to distribute them over m reducer tasks. This is a very network bandwidth heavy operation, as each one of the n mappers has to communicate the values it has for each of the m reducers in parallel, amounting to $\mathcal{O}(n \cdot m)$ pairs of file transfers. After the temporary files have been transfered by the shuffle, they are next sorted locally by the reducers in order to give the user provided *reduce* function,

which is presented with the list of values attached to each key, for example as $(key, (value_1, value_2, \ldots))$.

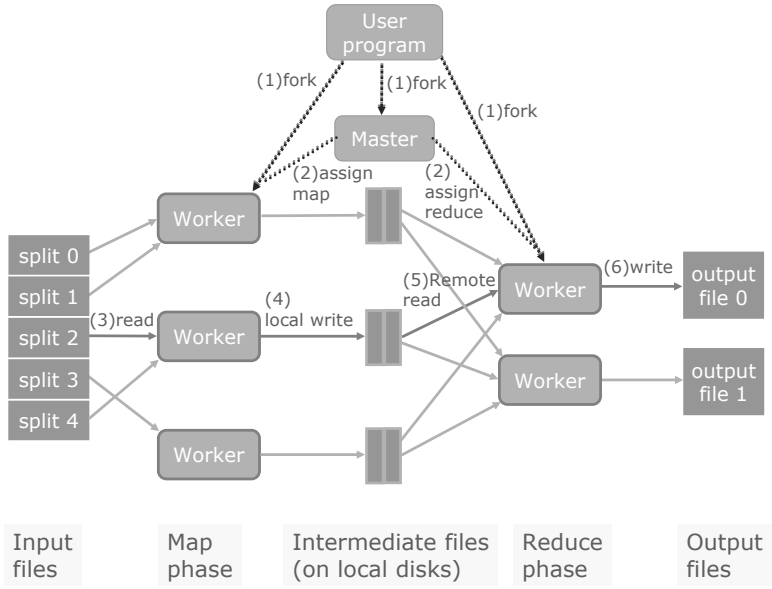

Figure 1.5: Overview of MapReduce

A really nice feature of the MapReduce framework is that it automatically parallelizes the work among the user provided number of computer nodes. Furthermore, the programs can be easily debugged, as the output of the program (following the rules of the framework) run in parallel is exactly the same as running the same program sequentially. This is the guarantee given by the functional programming framework. Also the functional programming paradigm gives very good fault tolerance: all the data produced by any number of mapper or reducer tasks can be lost and the framework can still continue making progress by rescheduling the re-execution of the lost jobs.

Google has been using the MapReduce framework to compute the production Web indexes Google uses to index the Web. For example, one of the heavy processing task is to compute the reverse Web link graph. That is, for each indexed site, collect all the sites linking to it. This perfectly matches the MapReduce framework. Also things like processing and doing statistics from log files matches the MapReduce framework nicely. Interestingly Google is not providing the full MapReduce feature to its customers. Also unfortu-

nately the MapReduce programming paradigm has been recently patented by Google.

The Apache Hadoop system is directly based on the design of the Google MapReduce and the Google Filesystem for its own distributed filesystem HDFS. The Hadoop system is written fully in Java and many cloud providers, for example Amazon, provides support for it in their service offering. Hadoop is used by many high volume production sites (`http://wiki.apache.org/hadoop/PoweredBy`), for example Facebook is running Hadoop clusters with 15 PetaBytes of storage using mainly the Hive (`http://hadoop.apache.org/hive/`) system implementing a SQL-style query language on top of the Hadoop MapReduce engine. Another Hadoop HDFS based database used mainly for batch processing is HBase (`http://hbase.apache.org/`), which is heavily inspired by Google Bigtable. The main applications for Hadoop seem to be log analysis, web indexing, and various data mining and customer analysis applications.

If a batch processing task fits the MapReduce framework, the framework gives good parallelization. In addition, MapReduce and Hadoop do not offer control of multiple parallel frameworks. In practice, there is a requirement to execute several different data processing frameworks, such as different versions of MapReduce and Hadoop, in parallel. The Nexus system presents a framework for running multiple frameworks in the same cluster [31]. The key idea of Nexus is to multiplex resources across frameworks and decouple job execution management from resource management. Figure 1.6 gives an overview of the Nexus framework.

1.3.4 Approaches to Fault Tolerance in Cloud Infrastructure

When providing scalable computing infrastructure on the cloud level the systems must be very fault tolerant and self-healing. When running datacenters with tens of thousands of computers there will always be some number of machines and hard disks that are broken. Most of the scalability solutions in the cloud space are built to have redundancy in a shared-nothing infrastructure. The aim of these systems is to provide an environment where one can power off any individual server in the datacenter, and the cloud infrastructure will recover automatically from the loss of the server without any manual intervention. The key techniques to achieve this is to have high redundancy in the datastore system for fault tolerance, and to never store any persistent data on the servers themselves: all data on the servers is just cached/preprocessed contents of the real application data that is stored in

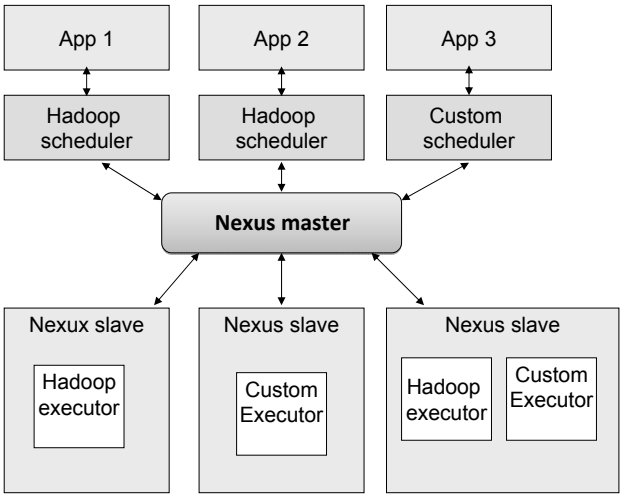

Figure 1.6: The Nexus system

the datastore. By doing so, all that is lost by powering down a server is the contents of a cache, that can be repopulated, and the server load can be redistributed among the remaining servers. Such a design also allows for powering off servers at off-peak hours, allowing the minimization of overall power usage of computing.

1.3.5 Latency and Clouds

On a much more philosophical level a key observation affecting the design decisions of future looking scalable cloud computing technologies is that computing capacity, memory capacity, storage capacity, and network bandwidth all improve at a much higher rate than latency improves. This is because ultimately latency is fixed by the speed of light, and especially when dealing with cloud computing systems where different computers of the cloud can be geographically very far from each other, latency certainly is an important issue. This has been observed by David Patterson in [42]. Maybe of much more practical interest are the three classical solutions to latency mentioned in the paper [42]: *Caching*, *Replication*, and *Prediction*.

If we look at the three above mentioned techniques in the scalable cloud computing technologies context, we observe all of the technologies are in use:

- *Caching* – Caching is one of the key components to cloud scalability, and is one of the reasons why cloud computing is heavily based on Web technologies that are themselves engineered with efficient caching in mind. In the cloud context especially memory based caches such as `memcached` have been introduced as additional write-through caches to lower back-end datastore read load.

- *Replication* – Replication is heavily used by all scalable cloud computing infrastructures. Many of the storage systems such as Amazon S3 and Google File System are using replication not only to provide improved data durability across geographically redundant replication, but also to improve the serving of "hot" files by replicating the highly requested files on a much higher number of file server nodes than the infrequently requested "cold" files. One of the implications of potentially very high replication ratios is that modifying data in-place becomes very expensive as basically all the (geographically distant) replicas have to be modified. The solution to this problem employed is to heavily emphasize write-once-read-many storage patterns with quite big block sizes in application design, see for example the Google BigTable design [13], as once written the files cannot be efficiently modified. Thus BigTable has been designed from the ground up to be run on a replicated file storage system to eliminate most of the random access write traffic to the replicated filesystem.

- *Prediction* – Quite a common architectural style these days is the precomputation of potential application database queries asynchronously beforehand and populating caches such as the main memory `memcached` beforehand with contents that the application might require in the future. This is once more done to hide latency from the user and to serve the common case queries from cache instead of a datastore. To implement this background processing one can employ asynchronous job queueing systems such as Amazon Simple Queue Service and scalable batch processing systems such as Hadoop to pre-populate the caches in advance.

As a practical consideration, if one develops applications that need to scale to large numbers of users on the cloud infrastructure, ACID datastore writes are going to become more and more expensive compared to the datastore reads. Thus applications need to be designed in a manner that tries to minimize the number of datastore writes per time unit to ensure application scalability, or to use a datastore that does not use ACID but BASE, and deal with the

potentially significant application complexity increase that is needed to deal with the inconsistent datastore.

1.4 Cloud from Application Programmer View

In terms of engineering, web application development is still in its infancy for obvious reasons – as long as the primary purpose of web development was the creation of web pages, there was no need to apply established software engineering principles to web development. However, cloud computing forces one to treat web development in the same fashion as software development in general. These include aspects like reusability, interoperability, and security, just to give some examples.

In the following, we will study different cloud computing approaches in terms of the type of cloud service they offer. The viewpoint is that of a programmer that might be interested in benefitting from the available cloud platform, not that of a potential cloud service developer that might wish to build a similar system.

1.4.1 Infrastructure as a Service

Infrastructure as a service (IaaS) is about providing a hosted computing infrastructure. A typical way to implement such a system is platform virtualization, where the user of the system can consider that the service corresponds to a piece of hardware and associated system software. One common approach seems to be that Linux type of a system is provided, where the developers can deploy their own software stack.

Examples of IaaS type systems include the following:

- Amazon EC2 (`http://aws.amazon.com/ec2/`)
- Eucalyptus project [41] (`http://open.eucalyptus.com/`)
- Ubuntu Enterprise Cloud (`http://www.ubuntu.com/cloud/`)

Next, we address these systems in more detail.

Amazon Elastic Computer Cloud

Amazon, which is better known from its Internet bookstore, was one of the first companies to enter into the cloud computing business. Amazon's Elastic Computer Cloud (Amazon EC2) was designed to make it easier for developers to use web-scale computing power [3]. Amazon promises 99.95% availability

for each EC2 region. Amazon EC2 offers a virtual computing environment where developer can launch multiple instances with variety of operating system or with custom application environment. Amazon offers a web service interface to launch instances.

One of key advantages in Amazon EC2 is that it allows a freedom for developers to choose their tools as they want. Amazon offers Amazon Machine Images (AMIs), which are preconfigured with several Linux distributions, Microsoft Windows Server, or OpenSolaris. Developers can customize AMI by choosing Amazon provided software (e.g. Java Application server, Database server). If offered operating systems or software do not meet developers' needs, they are always free build their own custom AMI. Since each developer can have the root access and can manage network access permissions, therefore the developer has total control over the software stack they use.

Amazon has three different kinds of pricing models: On Demand Instances, Reserved Instances, and Spot Instances. In On Demand Instances, the user pays for the capacity which is actually used. This approach is good when system needs to scale. In Reserved Instances, the user pays one time fee for each instance user wants reserve. With Spot Instances, Amazon is selling unused capacity of EC2. The price fluctuates based on supply and demand.

Amazon Elastic Block Store (EBS) offers persistent storage for Amazon EC2 instances [2]. EBS allows the user to create 1GB to 1TB volumes for use of EC2 instances. Instances see volumes as raw unformatted block devices and they can be used as any other block device (e.g. hard drive). EBS volumes are automatically replicated to prevent dataloss and they can be used as boot partitions.

Amazon divides locations into regions, e.g. US and Europe. These regions are divided into smaller areas - Availability Zones. EC2 instances can be placed into multiple locations in case of failure in an Availability Zone. Availability Zone are designed to be insulated from other AZs and they have inexpensive, low latency network connectivity to other Availability Zones in the same region.

Amazon EC2 uses Elastic IPs, static IPs that has been design for dynamic cloud computing. Elastic IP points to users account instead of EC2 instances. User controls the IP until the user chooses to release it. Elastic IP allows the user to remap IP to point instance, which the user has chosen. In case of a failure in Availability Zone, user can start new instance from other Availability Zone and keep running the service.

Amazon offers Virtual Private Cloud (VPC) technology for enterprises to extend their existing infrastructure [21]. VPC works as a secure isolated portion of the Amazon cloud. So far, Amazon VPC integrates EBS, EC2 and

Cloud Watch and rest of the Amazon cloud features are under development. Amazon provides enterprise a VPN connection and one cloud, where user can define up to 20 subnets.

Cloud Watch is the Amazon's cloud monitoring system. Cloud Watch enables user to monitor EC2 instances and Elastic Load Balancing (ELB) in real-time via Web Service interface. ELB distributes traffic automatically across multiple Amazon EC2 instances. From Cloud Watch, user can also enable Auto Scaling, which automatically scales capacity up or down depending upon the conditions that the user defines. This suits well in application that has high variability in usage.

Eucalyptus

Eucalyptus is an open source software used to implement cloud computing on compute clusters. Eucalyptus implements Infrastructure as a Service (IaaS) while giving the user the ability to run and control virtual machine instances deployed across a variety of physical resources [9]. Some aspects of underlying protocol and interface design for Eucalyptus are similar to Amazon Web Services (AWS). For example, the external interface to Eucalyptus is based on the API provided by Amazon [41]. The system is relatively easy to set-up, even on a local environment with limited resources. It is already a part of the Ubuntu Enterprise Cloud (UEC) installation.

Figure 1.7 presents an overview of the Eucalyptus system that follows a hierarchical design. The primary high-level components that comprise the Eucalyptus architecture are as follows:

- Node Controller (NC): An NC makes queries to discover the node's physical resources - the number of cores, the size of memory, the available disk space as well as to learn about the state of VM instances on the node (although an NC keeps track of the instances that it controls, instances may be started and stopped through mechanisms beyond NC's control) [41].

- Cluster Controller (CC): The CC schedules the distribution of virtual machines to the NC and collects resource capacity information [9]. Each CC may manage one or more NC(s). The Cluster Controller (CC) generally executes on a cluster front-end machine, or any machine that has network connectivity to both the nodes running NCs and to the machine running the Cloud Controller (CLC) [41].

- Storage Controller (Walrus): To put it simply, Walrus is a put/get storage service that implements Amazon's S3 interface, providing a

mechanism for storing and accessing virtual machine images and user data [41]. Walrus implements the REST (via HTTP), sometimes termed the "Query" interface, as well as the SOAP interfaces that are compatible with S3. Walrus provides two types of functionality. Firstly, users that have access to EUCALYPTUS can use Walrus to stream data into/out of the cloud as well as from instances that they have started on nodes. In addition, Walrus acts as a storage service for VM images. Root filesystem as well as kernel and ramdisk images used to instantiate VMs on nodes can be uploaded to Walrus and accessed from nodes [41].

- Cloud Controller (CLC): It is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes high level scheduling decisions, and implements them by making requests to cluster controllers [41].

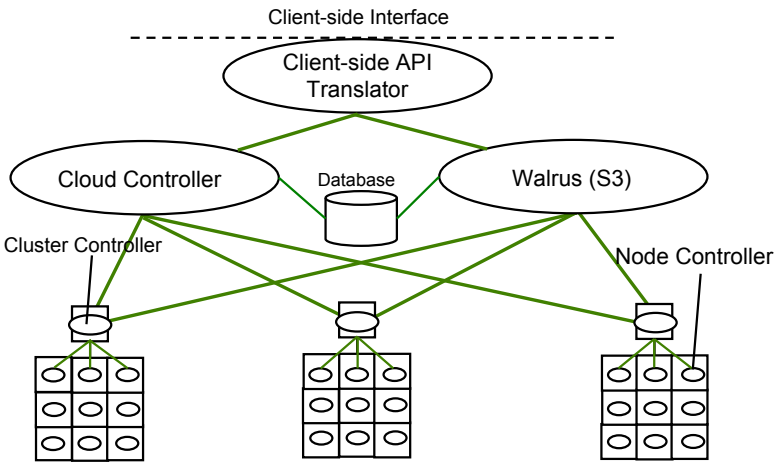

Figure 1.7: Overview of Eucalyptus

Each of these components runs as a web service on the respective machines; we have seen above that the machines distribution is separated for CLC/Walrus, CC(s) and NC(s). However it is also possible to set-up a test system with lesser resources. A minimal working "cloud" structure as mentioned in UEC set up guide is to have the CLC, Walrus and CC running on one machine and an NC service running on another.

Several performance measurements with Eucalyptus compared to the original AWS have been made. The result of these measurements is that a

private cloud can provide almost the same functionality and possibly better performance compared to the AWS. The performance can be easily improved with storage area networks (SAN) for storing the data in the S3/EBS directories (e.g. image files) in a better performing environment. This approach also helps a lot to reduce the time needed to start the virtual server instances. On the other hand, AWS offers unlimited scalability and thus it would be beneficial to combine both, private and public resources in a hybrid cloud [9].

Eucalyptus system has filled an important niche in the cloud-computing design space by providing a system that is easy to deploy atop existing resources, that lends itself to experimentation by being modular and open source, and that provides powerful features out-of-the-box through an interface compatible with Amazon EC2 [41]. Eucalyptus is an interesting product to build private cloud infrastructures for R&D. The performance of such an installation with commodity hardware is satisfying for most common scientific applications. Already now a large number of open source tools exist for cloud management. As also other cloud software providers start to implement Amazon AWS as a cloud computing interface AWS has the potential to become a de facto standard for cloud computing infrastructure services [9].

Ubuntu Enterprise Cloud

Company Canonical claims that Ubuntu is only Linux distribution to position itself as true Cloud OS. Ubuntu Enterprise Cloud(UEC) is one of the three components in Canonical's cloud strategy [56]. Other two components are Ubuntu Server edition and UbuntuOne. UEC and Ubuntu Server are aimed as IaaS and UbuntuOne is aimed as SaaS. Canonical's cloud project started with offering official AMI for Amazon EC2. Later, Canonical took Ubuntu Server Edition and integrated enchanted version of KVM [34] based Eucalyptus into the distribution. As a result Canonical got UEC – a user deployable cloud, which matches the API that AWS provides.

The architecture of UEC closely follows the architecture of Eucalyptus [41]. It has the same controllers (node controller, cluster controller and cloud controller) as Eucalyptus. One difference is that UEC includes also ESB Controller, which runs in the same machine as CC and is configured automatically when CC is installed. EBS Controller follows the same ideology as Amazon ESB [2].

1.4.2 Platform as a Service

Platform as a Service (PaaS) is about providing a computing platform that contains a complete solution stack, hosted as a service. Applications devel-

oped on top of PaaS can commonly be run as a part of the service. The biggest difference between PaaS and IaaS seems to be that PaaS usually assumes a certain kind of application model, together with associated libraries and system software, whereas in IaaS the developers have more freedom to select the systems they want to use. PaaS usually offers additional features such as load balancing, automatic provisioning and geographic replication.

Sample PaaS systems include the following:

- Google AppEngine (`http://code.google.com/appengine/`)
- Microsoft Azure (`http://www.microsoft.com/windowsazure/`)
- Heroku (`http://heroku.com/`)

Google AppEngine

Google App Engine is a service developed by Google to allow developers to run web applications on Google's infrastructure [27]. The reason for wanting to run web applications on Google's infrastructure is the access to their immense computing power and storage capabilities. The promise is that developers can start out small and then scale when the need arises.

One of the key advantages with App Engine is that there are no servers that need to be maintained by the developer, one can simply use part of the infrastructure Google already has in place. The need to have your own servers has traditionally been an issue for smaller projects, since they represent a considerable investment in hardware. By using App Engine the developer can simply use more processing power when the service usage grows.

App Engine allows developers to simply upload their code and have it deployed automatically, ready to be used by consumers. App Engine supports applications written in multiple languages. A Java runtime environment supports development with standard Java technologies, including JVM, Java servlets, and the Java programming language. It also supports any other language that uses a JVM-based interpreter or compiler such as JavaScript or Ruby. Python is also supported and App Engine offers a dedicated Python runtime environment, which includes a fast Python interpreter and the Python standard library. Both the Java and Python runtime environments are built to ensure that web applications run quickly, securely and without any interference from other applications running on App Engine.

App Engine provides its own storage solution called Datastore which utilizes BigTable as the method for storage. BigTable is not a relational and SQL compatible database and requires a different approach for storing and

retrieving data compared to a conventional SQL database. The advantages of BigTable is that it is fast and can scale to large tables and loads.

Google Datastore authors describe it as being "a sparse, distributed multi-dimensional sorted map", sharing characteristics of both row-oriented and column-oriented databases. It performs queries over data objects, known as entities. An entity has one or more properties, named values of one of several supported data types. A property can be a reference to another entity. The datastore supports executing multiple operations in a single transaction, and roll back the entire transaction if any of the operations fail. This type of feature is especially useful for distributed web applications.

Google currently supplies two standard Java interfaces for interacting with the Datastore, JDO (Java Data Objects), JPA (Java Persistence API), a low-level API is also available to allow developers direct access and the possibility to develop new interfaces. These interfaces allow developers to manage relational data in applications and include mechanisms that are meant to be used when trying to define classes for data objects and for performing queries.

JDO uses annotations on Java classes (POJO's) to describe how instances of the class are stored in the Datastore as entities, and how entities are recreated as instances when retrieved from the datastore. At the moment Datastore supports the use of JPA version 1.0. JPA also requires the use of annotations as in JDO but it also requires a `persistence.xml` file to be added which indicates to App Engine how to use the Datastore with this specific application. In order to make use of the App Engines persistence capabilities all objects used must be serializable.

All these features make App Engine a very attractive solution for deploying web applications that can scale without manual provisioning of computing resources.

Microsoft Azure

Azure is a Platform as a Service (PaaS) cloud offering from Microsoft [14]. It provides a platform for running mainly Windows applications and storing data in the cloud. These applications could be existing Windows applications that have been modified to run on cloud, or brand new ones written specifically for Microsoft Azure.

Developers can create applications for Microsoft Azure using familiar tools such as Visual Studio 2010. Azure applications are usually written using the .NET libraries, and are compiled to the Common Language Runtime (CLR). However, there is also support for the Java, Ruby and PHP languages.

Developers get a choice of language, but cannot control the underlying operating system or runtime. The platform provides a degree of automatic network configuration failover and scalability, but requires the developer to specify some application properties in order to do so.

The main components of the Windows Azure platform are:

- Windows Azure: Provides a Windows-based environment for running applications and storing data on servers in Microsoft data centers.
- SQL Azure: Provides data services in the cloud based on SQL Server.
- Windows Azure platform AppFabric: Provides cloud services for connecting applications running in the cloud.

Microsoft Azure may be a good solution to deploy existing applications for Windows and .NET platform. However, it raises the question on how well applications that have not been designed for the cloud can be scaled in the first place.

Heroku

Heroku [30] is a cloud application platform for the Ruby programming language. It was founded in 2007 by Orion Henry, James Lindenbaum, and Adam Wiggins. Heroku architecture consists of six components: HTTP reverse proxy, HTTP cache, routing mesh, dyno grid, SQL database with replication, and memory cache.

HTTP reverse proxy is the entry point for all requests coming into the platform. These front-end servers are used to manage DNS, load balancing, and fail-over. Heroku uses Nginx (`http://wiki.nginx.org/Main`) as its proxy.

All requests go through a HTTP cache (Varnish) [55]. If the requested content is available in the cache (a hit), the cache responds immediately and the request never reaches the application servers.

The routing mesh is a distributed pool of dynamic HTTP routers that balances requests across applications, dynos described below, tracks load, and intelligently routs traffic to available resources. The mesh easily handles frequent and instantaneous registration and de-registration of dynos. It is implemented using Erlang.

Actual application logic runs inside a dyno process. The number of dynos running for an application can be increased or decreased dynamically. According to the platform provider, it takes less than two seconds to start a

dyno for most applications. The dyno grid is spread across a large pool of server instances. The size of this pool varies with load.

Heroku provides a fully featured SQL database (PostgreSQL) for every application and an in-memory cache (Memcached). A dyno is a single process running Ruby code on a server in the dyno grid. In terms of computing power, four dynos are equivalent to one CPU-core on other systems. Each dyno is independent and includes the following layers:

- POSIX environment (Debian Linux).
- The Ruby VM, which then loads the application.
- The Thin application server.
- The Rack web server interface.
- Rails web framework. It is also possible to use other Rack-compliant frameworks.

We consider Heroku to be conceptually similar to the Google Application Engine (GAE) from a technical point of view. However there are several important differences worth mentioning:

- Heroku supports the Ruby programming language, while GAE supports Python and JVM languages.
- Heroku applications can use a SQL database.
- Heroku allocation of application threads is performed manually by the application provider. Thread allocation is automatic in GAE.

1.4.3 Software as a Service

Software as a Service (SaaS) can be considered as the most service oriented way to use cloud. So far, SaaS has proven useful as a business model among the cloud approaches. In SaaS, complete software systems that are ready-to-use is hosted in the software providers' servers and is offered to users to use over internet. The end-user willing to use this system pays the software provider a subscription fee for the service. This is completely different form the traditional way of software distribution and use, in which end-users need to purchase the license from the software provider and then install and run the software directly form on-site servers. Thus, SaaS allows some serious cost cutting for the companies (end-users) as they can avoid maintenance costs, licensing costs and the costs of the hardware required to run servers on-site.

Examples of such systems include the following:

- Salesforce (`http://salesforce.com`)
- Facebook (`http://www.facebook.com/`)
- Zynga farmwille game (`http://www.farmville.com/`)

Salesforce

Salesforce.com offers Customer Relationship Management (CRM) application services in the Software as a Service (SaaS) Industry [47, 57]. For end users, it offers services (applications) to industries and businesses of all sizes through online access, with minor implementation and no on-premise installation or maintenance of software or physical servers. These applications can be used to systematically record, store business data and to optimize different aspects of a company's business including sales, marketing, partnerships, and customer service. But due to the fact that CRM solutions should differ from one company to another, customization is obvious. That's why the cloud platform Force.com came into existence. Force.com [44] provides developers a platform to create data-oriented business applications which run against the salesforce.com database. This platform uses its own programming language called Apex. It has the following platforms [44]:

- Collaboration Platform: Chatter is a real time collaboration platform that brings together people, data, and content in a secure, private, trusted social framework. It facilitates creating customized profile across multiple business applications; provide real-time monitoring of business activities; secured content sharing; APIs to create new collaboration applications as add-ons; integration with other social networks like facebook and twitter.
- Development Platform: Development platform provides following services,
  - Database customization: Users can create and customize database relationships, formula fields, validation rules, reporting, tagging, auditing, and searches using the Web-based environment or the Eclipse-based IDE. It also generates user interface based on the data model defined which can be edited with page layout editor.
  - Programmable UI: User can build interfaces with customized behavior and look & feel. Force.com pages use a standard model-view-controller (MVC) design, HTML, and web technologies such

as CSS, AJAX, and Adobe Flash and Flex. It also has 60 predefined components that can be assembled with minimal coding in building-block fashion. Force.com also provides an IDE to implement cloud-based RIAs which can be deployed through the browser via the Adobe flash player, or directly to the desktop using the Adobe AIR runtime. It runs seamlessly online or offline while taking full advantage of the security, scalability, and reliability of Force.com.

- Programmable Cloud logic: Force.com code has similar syntax to Java and C#. Its Eclipse-based IDE can be used to create, modify, test and deploying applications.
- Visual process manager: Visual Process Manager, along with workflow and approvals, enables users to rapidly design and run any business process in the cloud without infrastructure, software, or code.
- Mobile deployment: Users can create complex mobile applications with point-and-click ease that work across BlackBerry, iPhone, Windows Mobile, and others with offline mobile access to key Salesforce CRM data.

- Cloud Infrastructure: Force.com is based on a multitenant architecture that makes it secure, reliable, and elastic. It has ISO 27001 security certification and used by nearly 60,000 companies including Cisco, Japan Post Network, and Symantec. It provides real time query optimization, upgradation, and scalability.

Facebook

Facebook is a social networking website that provides following services as SaaS to its users: Publisher (used to post information and messages which appear on the user's own Wall), Wall (space on each user's profile page that allows friends to see and post messages), Photo and video uploads, Gifts (virtual gift shop), Marketplace (allows users to post free classified ads in different catagories), Status updates, Events (to organize and notify the community with new events), Networks, groups and like pages, Chat, Pokes (to attract the attention of another user).

Facebook also provides platform that consists of a set of APIs and tools for developing social networking applications [38]. It is possible to develop Facebook applications using external server capacities from Cloud computing service providers like Amazon or Joyent [38].

In general, there are two types of applications on Facebook [22]:

- FBML Canvas applications: This kind of applications are rendered by Facebook using FBML (Facebook Markup Language). The application is hosted by the developer on their own server.

- IFrame Canvas applications and websites using Facebook: This kind of applications are usually rendered by the developer's server without using Facebook as an intermediary. IFrame Canvas applications are architecturally very similar to websites which incorporate data and widgets from Facebook.

Although the two types of applications do roughly the same things, they differ in the way user data is retrieved from Facebook, display static content and perform optimization.

The list of API's and SDK's currently provided by facebook platform [22] to develop applicaiton on Facebook are as follows:

- Core APIs:
  - Graph API: The Graph API can be used to read and write objects and connections in the Facebook social graph. Objects can be for example, album, photo, event, link, note, status message, video and so on. Whereas connections can be friend relationships, shared content, and photo tags. Every object in the social graph has a unique id and the data associated with the object can be fetched using that id.
  - Social plugins: Social plugins are the extensions of Facebook. These plugins are designed for not to share personal data with the sites on which they appear, but to show users with their activities on the facebook. These social plugins can be added to a site with a line of HTML code.

- Advanced APIs:
  - Facebook Query Language (FQL): FQL provides a SQL-style interface to query the data exposed by the Graph API. Batching multiple queries into a single call is possible. Query response format can be specified as either XML or JSON with the format query parameter.

  - Facebook Markup Language (FBML): FBML is used to build Facebook applications that can be hooked into several Facebook integration points, including the profile, profile actions, and canvas. FBML is HTML extension and is used in traditional FBML Canvas applications, and is rendered by Facebook directly.
  - XFBML is also a HTML extension provided by Facebook platform that can be used to incorporate FBML into an HTML page on a Facebook Connect (a set of API's to provide trusted connection between facebook and developer site) site or an iframe application.

- Facebook SDKs:

  - Android SDK (unofficial).
  - JavaScript SDK: JavaScript is used to access features of the Graph API. It also provides client-side functionality for authentication and sharing. Its recommended to load the SDK asynchronously in the site for better efficiency.
  - PHP SDK: It also supports access to Graph API.
  - Python SDK: This client library is designed to support the Facebook Graph API and the Facebook JavaScript SDK, which is the canonical way to implement Facebook authentication.
  - iPhone SDK: This mobile SDK is a Objective-C code that can be used to connect users' Facebook accounts with a mobile application. User authorization is required to fetch user profile data from the Graph API and to publish messages on user's wall. Facebook uses OAuth 2.0 protocol for authentication and authorization. There are also releases for PHP and Python SDKs for the Graph API to support mobile application development.

All the above Facebook SDKs are open source and are available on GitHub.

Zynga

Zynga is a social game developer, which develops games to play on social networks such as Facebook and MySpace, on mobile devices like the iPhone, on MSN games and my yahoo. Zynga makes some of the most popular social networking games that run on Facebook which includes Mafia Wars, Farmville, and Cafe World. The company's games attract 235 million users a month, which is more than half of Facebook's worldwide total of 400 million users [32].

1.4.4 Discussion

Since Cloud Computing has become a buzzword only relatively recently, there are no dominant technology designs yet that would be used predominantly. However, there are certain emerging trends, especially when approaching the development from programming perspective.

To begin with, although the paradigm shift from client-side programs to running the majority of applications in the cloud per se does not prescribe any new programming techniques, there have been some recent trends that deserve attention. However, in practice the trend seems to be that developers are increasingly using web technologies and scripting, as pointed out in [43]. In part, we believe that this is a consequence of the availability of increasing computing resources, but also the fact that modifiability, extendability and the access to ready-made web-enabled implementations simply are more important than the development of the most optimized implementation.

Finally, we fundamentally believe that it still remains a challenge to compose reliable software systems that are distributed and scalable in nature. Consequently, there is a need for improved understanding and better tools and methods for developing applications for any cloud programming platform.

1.5 Conclusions

This chapter has looked at cloud computing from both the application developers perspective as well as from looking at the technologies employed inside the cloud.

Virtualization is nowadays a mature technology that is heavily used both in clouds and in datacenters to remove the dependency of a server from the physical hardware it is running on, easing maintenance including hardware replacement and fail-over.

On the application development side it can be seen that the Web application development methodologies such as RESTful services and high level programming frameworks similar to the Google App Engine and Ruby on Rails are key building blocks for building Web applications hosted on the cloud.

Cloud computing can be used as a direct drop-in replacement for traditional applications using technologies such as Amazon EC2 and Amazon EBS. They basically provide virtual (Linux/Windows) machines and virtual (NAS) storage with traditional database products used as data storages. If the application does not have to scale to very large user numbers this is a viable alternative. It basically just uses clouds as virtualized hardware to

run traditional applications. If scalability to large user counts is needed, the other approach is to use scalable datastores such as, for example, Google App Engine Datastore, Cassandra, or Amazon SimpleDB to create scalable Web applications. In this framework the value add is that the cloud computing provider provides many of the pieces of the infrastructure to monitor and scale the applications to very large numbers of users but this requires some re-architecting the applications for scalability. The use of Web application development frameworks is a good start as it uses a similar division of work between the application itself and the datastore (database) used to store the data for the application.

To make cloud application deployment and administration cost-effective for large scale applications, scalable cloud based systems are generally architected in a shared-nothing architectural style where loosing any single server due to hardware failures will not effect the behavior of the cloud applications, as the cloud infrastructure will reconfigure the load of the failing server to the remaining servers. The key techniques to achieve this is to have high redundancy in the datastore system for fault tolerance, and to never store any persistent data on the servers themselves: all data on the servers is just cached or preprocessed contents of the real application data that is stored in the datastore.

When selecting if and how to employ the cloud technologies one should consider the scalability needs of the developed application as this gives requirements on the employed development and datastore methodologies. Many of the advanced cloud technologies are still in active development, and clear suggestions on which technologies to employ for which applications are not yet straightforward. Also the financial issues of cloud computing need to be considered, as common cloud pricing structure (pay-as-you-go) is at its best for handling computing loads that are bursty and hard to predict in advance and therefore carry investment risks in server and datacenter capacity planning.

References

[1] *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*. USENIX Association, 2006.

[2] *Amazon Elastic Block Store*. `http://aws.amazon.com/ebs/`.

[3] *Amazon Elastic Compute Cloud*. `http://aws.amazon.com/ec2/`.

[4] *AMD Virtualization (AMD-V^{TM}) Technology*. http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx.

[5] H. Amur et al. "Robust and Flexible Power-Proportional Storage". In: *ACM Symposium on Cloud Computing (ACM SOCC)*. 2010.

[6] M. Armbrust et al. "A view of cloud computing". In: *Commun. ACM* 53.4 (2010), pp. 50–58.

[7] M. Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report UCB/EECS-2009-28. Available from: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf. University of California at Berkeley, Electrical Engineering and Computer Sciences, 2009, p. 23.

[8] L. A. Barroso and U. Hölzle. "The Case for Energy-Proportional Computing". In: *IEEE Computer* 40.12 (2007), pp. 33–37.

[9] C. Baun and M. Kunze. "Building a private cloud with Eucalyptus". In: *E-Science Workshops, 2009 5th IEEE International Conference on*. 2009, pp. 33 –38. DOI: 10.1109/ESCIW.2009.5408006.

[10] M. Burrows. "The Chubby Lock Service for Loosely-Coupled Distributed Systems". In: *OSDI*. USENIX Association, 2006, pp. 335–350.

[11] T. D. Chandra, R. Griesemer, and J. Redstone. "Paxos made live: An engineering perspective". In: *PODC*. Ed. by I. Gupta and R. Wattenhofer. ACM, 2007, pp. 398–407. ISBN: 978-1-59593-616-5.

[12] F. Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *OSDI*. USENIX Association, 2006, pp. 205–218.

[13] F. Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (2008).

[14] D. Chappell. *Introducing The Windows Azure Platform*. Tech. rep. Microsoft Corporation, 2009.

[15] *Comparison of platform virtual machines*. http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines.

[16] B. F. Cooper et al. "PNUTS: Yahoo!'s hosted data serving platform". In: *PVLDB* 1.2 (2008), pp. 1277–1288.

[17] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI*. 2004, pp. 137–150.

[18] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters". In: *Commun. ACM* 51.1 (2008), pp. 107–113.

[19] G. DeCandia et al. "Dynamo: Amazon's highly available key-value store". In: *SOSP*. Ed. by T. C. Bressoud and M. F. Kaashoek. ACM, 2007, pp. 205–220. ISBN: 978-1-59593-591-5.

[20] H. Douglas and C. Gehrmann. "Secure Virtualization and Multi-core Platforms State-of-the-Art report". In: *SICS Technical Report* T2009:14A (2010), pp. 1–71. URL: `http://soda.swedish-ict.se/3800/`.

[21] *Extend Your Virtual IT Infrastructure With Amazon Virtual Private Cloud*. Tech. rep. Amazon Web Services, 2010.

[22] *Facebook Developers*. `http://developers.facebook.com/docs/`. 2010.

[23] R. T. Fielding and R. N. Taylor. "Principled design of the modern Web architecture". In: *ACM Trans. Internet Techn.* 2.2 (2002), pp. 115–150.

[24] A. Fox et al. "Cluster-Based Scalable Network Services". In: *SOSP*. 1997, pp. 78–91.

[25] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system". In: *SOSP*. Ed. by M. L. Scott and L. L. Peterson. ACM, 2003, pp. 29–43. ISBN: 1-58113-757-5.

[26] S. Gilbert and N. A. Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In: *SIGACT News* 33.2 (2002), pp. 51–59.

[27] *Google App Engine*. `http://code.google.com/appengine/`.

[28] N. Gude et al. "NOX: towards an operating system for networks". In: *SIGCOMM Comput. Commun. Rev.* 38.3 (2008), pp. 105–110. ISSN: 0146-4833. DOI: `http://doi.acm.org/10.1145/1384609.1384625`.

[29] T. Härder and A. Reuter. "Principles of Transaction-Oriented Database Recovery". In: *ACM Comput. Surv.* 15.4 (1983), pp. 287–317.

[30] *Heroku Homepage*. `http://heroku.com/`.

[31] B. Hindman et al. *Nexus: A Common Substrate for Cluster Computing*. Tech. rep. UCB/EECS-2009-158. EECS Department, University of California, Berkeley, 2009. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-158.html`.

[32] *Information Week*. http://www.informationweek.com/. 2010.

[33] *Intel® Virtualization Technology*. http://www.intel.com/technology/virtualization/technology.htm?iid=tech_vt+tech.

[34] *KVM Homepage*. http://www.linux-kvm.org/page/Main_Page.

[35] J. Laine. *Cloud Storage Systems in Telecom Services*. Master's Thesis, Aalto University, School of Science and Technology, Degree Programme in Computer Science and Engineering. 2010.

[36] A. Lakshman and P. Malik. "Cassandra - A Decentralized Structured Storage System". In: *LADIS 2009: The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware*. 2009.

[37] L. Lamport. "The Part-Time Parliament". In: *ACM Trans. Comput. Syst.* 16.2 (1998), pp. 133–169.

[38] A. Lenk et al. "What's Inside the Cloud? An Architectural Map of the Cloud Landscape". In: *ICSE '09: Proceedings of the Workshop on Software Engineering Challenges in Cloud Computing*. Available from: http://www.icse-cloud09.org/cloud-dashboard. 2009.

[39] N. McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (2008), pp. 69–74. ISSN: 0146-4833. DOI: http://doi.acm.org/10.1145/1355734.1355746.

[40] P. Mell and T. Grance. *The NIST Definition of Cloud Computing v15*. Version 15 available from: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

[41] D. Nurmi et al. "The Eucalyptus Open-Source Cloud-Computing System". In: *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. ISBN: 978-0-7695-3622-4. DOI: http://dx.doi.org/10.1109/CCGRID.2009.93.

[42] D. A. Patterson. "Latency lags bandwith". In: *Commun. ACM* 47.10 (2004), pp. 71–75.

[43] L. D. Paulson. "Developers Shift to Dynamic Programming Languages". In: *Computer* 40.2 (2007), pp. 12–15. ISSN: 0018-9162. DOI: http://dx.doi.org/10.1109/MC.2007.53.

[44] force.com platform. In: http://www.salesforce.com/platform/cloud-platform/, 2010.

[45] D. Pritchett. “BASE: An ACID Alternative”. In: *ACM Queue* 6.3 (2008), pp. 48–55.

[46] *QEMU Open Source Processor Emulator*. http://wiki.qemu.org/Index.html.

[47] “SalesForce products”. In: http://www.salesforce.com/crm/products.jsp, 2010.

[48] I. Stoica et al. “Chord: A scalable peer-to-peer lookup protocol for internet applications”. In: *IEEE/ACM Trans. Netw.* 11.1 (2003), pp. 17–32.

[49] I. Stoica et al. “Chord: A scalable peer-to-peer lookup service for internet applications”. In: *SIGCOMM*. 2001, pp. 149–160.

[50] M. Stonebraker. “SQL databases v. NoSQL databases”. In: *Commun. ACM* 53.4 (2010), pp. 10–11.

[51] M. Stonebraker et al. “MapReduce and parallel DBMSs: Friends or foes?” In: *Commun. ACM* 53.1 (2010), pp. 64–71.

[52] N. Tolia et al. “Delivering Energy Proportionality with Non Energy-Proportional Systems - Optimizing the Ensemble”. In: *HotPower*. Ed. by F. Zhao. USENIX Association, 2008.

[53] *TrustZone*. http://www.arm.com/products/processors/technologies/trustzone.php.

[54] L. M. Vaquero et al. “A break in the clouds: towards a cloud definition”. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (2009), pp. 50–55. ISSN: 0146-4833. DOI: http://doi.acm.org/10.1145/1496091.1496100.

[55] *Varnish Homepage*. http://varnish-cache.org/.

[56] S. Wardley, E. Goyer, and N. Barcet. *Ubuntu Enterprise Cloud Architecture*. Tech. rep. Canonical, 2009.

[57] “wikinvest”. In: http://www.wikinvest.com/wiki, 2010.

[58] *Wikipedia page for VMWare ESX*. http://en.wikipedia.org/wiki/VMware_ESX.