Cloud Computing *

Jens Lechtenbörger

IT Systems, Summer Term 2024

This presentation introduces cloud computing and serverless computing, starting from basic concepts of distributed systems.

1 Introduction

Let us look at essential questions and terminology of our topic.

1.1 Core Questions

- What is a distributed system?
- What do "cloud computing" and "serverless computing" mean?
  - How do they help to build distributed systems?

This presentation addresses the following questions:
What is a distributed system?
What are "cloud computing" and "serverless computing", and how do they help when building distributed systems?

1.2 Learning Objectives

- Explain distributed systems with their basic scalability techniques.
- Explain and contrast cloud computing and serverless computing based on definitions and examples.

Take some time to think about the learning objectives specified here.

1.3 Retrieval practice

- What is digital sovereignty?
  - Introduced as course goal
  - Revisited in OS part
    - With free software as precondition
      - Similarly for free firmware (and hardware)
- What is caching?

*This PDF document is an inferior version of an OER HTML page; free/libre Org mode source repository.

- What is server consolidation?
- What is Kubernetes?

Please take a brief break and write down answers to these questions, without using previous class material.

Agenda

The agenda for the remainder of this presentation is as follows.

We continue with basic notions and techniques of distributed systems, which form the foundation for cloud computing. Then, we look at characteristics of and models for cloud computing. Afterwards, we turn to serverless computing, which is a form of cloud computing.

2 Distributed Systems

Figure 1: "Internet of Things" by Wilgengebroed on Flickr under CC BY 2.0; from Wikimedia Commons

While the course so far looked at individual computers, we now turn to distributed systems, where multiple computers serve a common goal.

Such systems are everywhere.

2.1 Definitions

- A distributed system (DS) is ...
  - Leslie Lamport: "one in which the failure of a computer you didn't even know existed can render your own computer unusable"

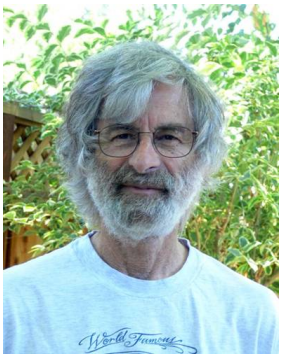

Figure 2: "Photo of Leslie Lamport" under CC0 1.0; from Wikimedia Commons

    - (Lamport is Turing Award winner, also with seminal contributions to DSs)
  - Tanenbaum and van Steen (Tanenbaum and Steen 2007): "a collection of independent computers that appears to its users as a single coherent system"
  - Coulouris et al. (Coulouris et al. 2011): "a system in which hardware and software components located at networked computers communicate and coordinate their actions only by passing messages"

    This slide shows several definitions for the term "distributed system". Please think about them for some seconds.
  - Comments, examples in notes

While the first definition by Lamport may sound like a joke, it captures the essence of DSs as expressed by the subsequent definitions, which are cited from standard textbooks on DSs: Such systems consist of multiple computers that communicate to achieve some common goal or functionality. It may neither be immediately visible to users which computers are involved, nor by whom they are controlled, nor what purposes they serve. (Note that any "smart" device is or embeds a Computer. Frequently, smart devices communicate with other machines, i.e., they are part of a DS.)

Typical examples for DSs include all "cloud" or web based systems, e.g., e-learning platforms where a browser on one computer communicates with processes on other machines.

For a counter-example, consider software on your own computer that also works when your computer is offline. E.g., if you disconnect your computer from the Internet and write a document that is stored on your local disk, you interact with a non-distributed system. If you connect your computer to the Internet, the situation is less clear: Depending on operating system and installed software, all kinds of interactions may happen "in the cloud": e.g., automatic backups; spell or virus checking; activity tracking.

Coming back to Lamport's definition, maybe consider this tweet (from December 2021) which highlights that some "smart" vacuum cleaners are part of a DS. Thus, one may be unable to use such devices if machines elsewhere on earth fail. As a side remark, I like to stay in control of my devices, which I believe to require free software. Besides, being "smart" usually means being distributed and, according to Hypponen's Law, being vulnerable.

Returning to topics of this presentation and web based examples of DSs, note that what we might think of as "web server" is frequently a complex DS itself, consisting of lots of machines, e.g., for load balancing, to be revisited with replication and in the context of IaaS later on. Briefly, a "web server" may be implemented by lots of physical machines that share the load created by millions of clients. In addition, each web server may access database servers or other machines in the background, say, to retrieve or validate payment information during client interactions or to track and trace user behavior under surveillance capitalism.

2.2 Internet vs Web

- Major concepts
  - Internet: **Network** of networks, an internetwork
    - Each network with hosts and links

      The Internet is an internetwork, i.e., a network of networks. Each network consists of two or more devices, also called hosts, that are connected by one or more links.

      Typical devices or hosts are computers in any shape and form as well as special purpose network devices such as routers. Routers are devices that connect different networks by forwarding messages between them. For that purpose, routers have links to at least two networks.

      Typical links are fibre or copper cables, e.g., for ethernet or DSL connections, or "nothing" in case of Wi-Fi.

      - E.g., our home networks, university networks, ISPs, etc.
        - **Connectivity** for heterogeneous devices in DSs, regardless of their home network

          The Internet connects all the networks to which we are used, e.g., end users' networks such as at home, in the university, or in companies, but also networks of internet service providers.
          In any case, the Internet offers world-wide connectivity for heterogeneous devices that are part of various DSs.
      - Connectivity enabled by various **protocols**

        To offer connectivity in view of heterogeneous devices, standardized protocols are necessary, which use abstractions to hide complexity, where multiple layers of protocols are used in practice. In general, protocols describe rules and conventions of communication, e.g., message formats, or sequencing of events.
        - IPv4 and IPv6 for **host-to-host** connectivity (IP = Internet Protocol)

          In case of the Internet, the Internet Protocol, IP for short, is the single, unifying protocol that provides host-to-host connectivity. Version 4 of that protocol was published in 1981 and is still widely used, with its successor, version 6, gaining adoption.
          Importantly, the IP specifies a message format, including an address format, and forwarding and routing functionality to transmit messages between different hosts.
          All IP messages includes source and destination IP addresses, and destination IP addresses are used by routers to forward messages in a hop-by-hop fashion from source to destination. E.g., a local Wi-Fi network at home might be connected over a DSL router to some provider's network, which in turn connects to other parts of the Internet. Then, local Wi-Fi devices send outgoing messages to the DSL router as first hop, which inspects the destination IP address. If the destination IP address is outside the local network, the DSL router forwards the message to a router in the provider's network as next hop. That router again inspects the destination IP address, and forwards the message to a suitable next hop etc.

            - DNS translates human-readable names to IP addresses, e.g., `www.uni-muenster.de` to `128.176.6.250` (IPv4) or `2001:4cf0:2:20::80b0:6fa` (IPv6)

              DNS is the domain name system of the Internet: While numeric IP addresses are great for identification and routing, they are hard to remember for human beings. Thus, we use hierarchical DNS names, whose components are separated by dots, and globally distributed DNS servers are consulted to translate such names to IP addresses as basis for message transmissions. E.g., we expect machines inside our university to have names ending in "uni-muenster.de", and university name servers know which IP addresses those machines have.

              Addresses of version 4 consist of 32 bits, written down with four 8-bit integer numbers separated by dots. In contrast, addresses of version 6 have a size of 128 bits, written down in a schema with hexadecimal numbers separated by colons.
            - TCP, UDP, QUIC for **process-to-process** connectivity (e.g., process of web browser talks with remote process of web server)

              Important protocols building on top of IP are TCP, UDP, and, more recently, QUIC. While properties of these protocols are not important for us, you should remember that they enable processes on different hosts to send messages to each other, and these protocols use so-called ports to identify processes on hosts, e.g., the well-known ports 80 for web servers and 53 for DNS servers.

              Recall that you configured port numbers when starting a web server inside a Docker container.

    - The web is an **application** using the Internet
        * Clients and servers talk HTTP (another protocol)
            - E.g., `GET` requests of HTTP ask for HTML pages (and more)
            - Web servers provide resources to web clients (browsers, apps)

        The web is one particular example of an Internet application, which is based on the protocol HTTP. That protocol specifies how web clients and servers communicate by exchanging messages. Such HTTP messages are then transmitted over the Internet with protocols such as TCP or QUIC and IP.

        As one example, HTTP specifies `GET` requests, with which clients can ask servers for resources, e.g., for HTML pages, images, JavaScript files.

        A short demo for this is (or was already) part of a class session.
    - Internet and web **contain** DSs

The Internet and the web contain numerous DSs: E.g., DNS is a distributed system on the Internet: Computers running nameserver software are queried by clients on other computers.

Learnweb and our web browsers form a DS as part of the web, with HTTP as underlying protocol.

2.3 Technical DS Challenges

- No shared memory, but message passing
- Concurrency
- Autonomy and heterogeneity
- Neither global clock nor global state
- Independent failures

- Hostile environment, safety vs security

Note that in non-distributed systems, within a process its threads share an address space. In addition, processes may share selected regions of memory, which allows them to share data structures as well as to coordinate and cooperate with little overhead. In distributed systems, such sharing and cooperation relies on message passing, adding additional complexity and latency.

Also note that even in non-distributed systems, concurrency may lead to race conditions, asking for mutual exclusion, and raising various MX related challenges. Clearly, such challenges also arise in DSs, but they are aggravated by several facts.

First, different parts of a system may be run by different autonomous organizations with different goals and different choices concerning hardware, software, and cooperation. Thus, heterogeneity is to be expected and needs to be overcome.

Second, it is difficult (or even impossible) to agree on such seemingly simple facts as the current time, which led to the development of logical time to avoid the need for globally synchronized time. Similarly, it should not come as a surprise that with multiple autonomous parts, no single party exists that could tell the current global state of a DS.

Moreover, different parts of a DS may fail at any point in time (e.g., due to power outage, hardware failure, or bugs), but they may also be attacked at any point in time, bringing all issues of single systems related to safety and security to the table.

2.4 DS Goals

- Make **resources** accessible
  - E.g., CPUs or GPUs, printers, files, communication and collaboration
- Openness
  - Accepted standards, interoperability
- Various distribution transparencies
- Scalability

(Source: (Tanenbaum and Steen 2007))

One major reason for the creation of distributed systems is to make resources accessible beyond a single machine. E.g., hardware such CPUs, GPUs, and printers are shared within or beyond organizations, files may be shared beyond organizations, communication and collaboration takes place on a global scale.

Successful DSs strive for openness, where accepted standards afford interoperability.

The definition of Tanenbaum and Steen above characterizes DSs as "a single coherent system", which hides the real complexity of multiple cooperating computers, which in turn is understood as transparency, to be explored on the next slide.

Finally, DSs aim for scalability, which means that they should be prepared for growth. This goal is revisited subsequently.

2.4.1 Distribution Transparencies

- Transparency = Invisibility (hide complexity)
- Sample selection of transparencies from ISO/ODP (Farooqui, Logrippo, and de Meer 1995)
  - **Location t.**: clients need not know physical server locations
  - **Migration t.**: clients need not know locations of objects, which can migrate between servers
  - **Replication t.**: clients need not know if/where objects are replicated

- **Failure t.**: (partial) failures are hidden from clients

Different forms of transparencies for DSs are listed here. Again, these transparencies are goals of DSs, and not every DS may address all of them.

E.g., clients may neither need to know locations of physical servers (which may change over time, with servers being added or removed) nor of objects (which can migrate between servers).

Moreover, clients may not need to know whether or where objects are replicated.

Finally, failures should be hidden from clients, which requires redundancy and automatic failover with switching from failed components to working ones.

2.4.2 Scalability

- Dimensions of scale
  - Numerical: Numbers of users, objects, services
  - Geographical: Distance over which system is scattered
  - Administrative: Number of organizations with control over system components

A system is scalable if it can handle growth "appropriately", where growth can be measured along several dimensions. E.g., for an IT system, we might expect that it tolerates arbitrary increases regarding the numbers of users, objects, or services. In addition, a single, coherent distributed system may be scattered over numerous system components on multiple continents, and different components may or may not be controlled by different organizations.

- Typical scalability techniques for IT systems
  - Replication, caching, partitioning

Numerical dimensions of scalability relate to the workload coming with increased numbers of users, objects, or services. To address such growth, techniques such as replication, caching, and partitioning are used, to be explained subsequently.

  - Scale up: Improve hardware; limited potential

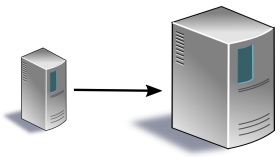

Figure 3: Figure under CC0 1.0

Scaling of compute resources comes in two variants, namely scale out and scale up.

Scale up (also called vertical scaling) means to upgrade given hardware (e.g., to add more RAM or more CPU cores). It should be obvious that the potential for scaling up is limited.

  - **Scale out** (**horizontal scaling**): Use partitioning and replication; (almost) unlimited potential

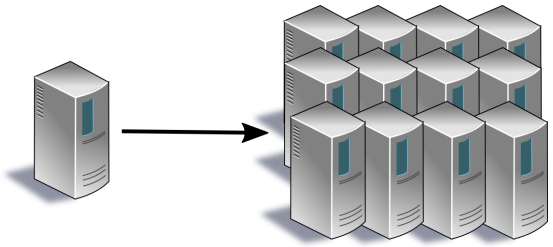

Figure 4: Figure under CC0 1.0

(Based upon: (Neuman 1994))
Scale out (also called horizontal scaling) means to add additional machines, often in the form of off-the-shelf PC hardware. Thanks to techniques such as partitioning and replication, the potential for scaling out is essentially unlimited by using more and more machine.

2.4.3 Replication

- To replicate = to **copy** to multiple machines/nodes
  - Copies (or nodes managing them) are called **replicas**
    - E.g., manually forked and cloned repositories with Git or automatically managed redundancy with HFDS or Kubernetes

Replication is a key technique in distributed systems to provide fault tolerance and to handle server load. To replicate simply means to provide redundant copies of "something", be it hardware, data, or services. E.g., in the context of the version control system Git, forking and cloning create copies of repositories. Thus, forks and clones are examples of manually managed replicas. Probably more interestingly, automatically managed replication exists as well, e.g., in distributed file systems such as HDFS in the Apache Hadoop ecosystem or Kubernetes as cluster management software for automating software deployment and scaling. In any case, replication comes with several positive effects and one major challenge as sketched next.

- **Positive effects**
  - Increased availability (usability in presence of faults)
    - System usable as long as "enough" replicas available
  - Reduced latency
    - Use local or nearby replica
  - Increased throughput
    - Distribute/balance load among replicas

Replication is a mechanism for fault tolerance, which improves the availability of systems under failure situations: Even if single replicas fail (e.g., due to power failures, bugs, or attacks), the overall system may still work as expected by serving requests from the remaining replicas.

In addition, replication usually reduces latency by serving requests from nearby replicas. As an example, this happens for major websites with the help of content delivery networks.

Moreover, replication enables load balancing, which increases the throughput of distributed systems: Each replica is powerful enough to serve a certain number of clients or customers, and by adding more replicas we scale horizontally and are able to serve more clients in total.

- Challenge: Keep replicas in sync (consistent)

  Replication does not only come with positive effects, though: As replicas are supposed to be copies of each other, their synchronization in response to updates presents a major challenge. Various consistency models with accompanying protocols exist in the context of distributed systems, which is beyond our scope.

2.4.4 Caching

- To cache = to **save** (intermediate) results close to client
  - Temporary form of replication, e.g.:
    * CPU caches keep data from RAM closer to CPU; in turn, RAM acts as cache for data from disk; in turn, disks act as caches for "cloud" data
    * Browser caches for web resources
    * SIEVE algorithm for caching

  Caching is a classical technique to speed up computations, even in non-distributed systems. As you know, the memory hierarchy embeds caching at multiple levels: CPUs are equipped with small and fast memory chips, called CPU caches, which provide fast access to currently used instructions and data, avoiding relatively slow accesses to main memory (RAM). Main memory in turn can cache data loaded from even slower disks, and disks can cache data from remote network locations.

  As an example in distributed systems, web browsers use the local file system to cache recently accessed web resources (e.g., images, HTML, or JavaScript files), making them locally available for upcoming visits of the same website (avoiding network latency and reducing load on web servers). Mechanisms such as conditional requests enable web browsers to detect whether their cached resources are still current.

  For caches of fixed size, replacement policies are necessary, e.g., SIEVE, which is a variant of the page replacement algorithm Clock.

- **Positive effects**
  - Reduced load on server/origin
  - Increased availability and throughput as well as reduced latency as with replication
- Challenge: Keep cache contents **up to date**

In essence, caching can be understood as temporary form of replication, leading to similar positive and negative effects.

2.4.5 Partitioning

- To partition = to **spread** data or services among multiple machines/nodes
  - Each node responsible for subset
  - (Sharding = partitioning in shared-nothing architecture)

Partitioning is a key technique in distributed systems to provide horizontal scalability. Here data or functionality (or both) are distributed (or partitioned) into disjoint subsets (or partitions), each of which is assigned to a different machine. (Sharding is a marketing term for this technique, where partitions are called shards.)

- **Effects**

  - Reduced availability: each node is additional point of failure
    * If node fails, its data/services are not available
    * (To improve availability, partitioning usually paired with replication)
  - Reduced latency and increased throughput
    * Each node operates on (small) subset
      · (Partial) results on subsets produced fast; combined into overall result
    * Nodes operate in parallel
      · (Think of search in large set of data)

On the negative side, partitioning reduces the availability of the overall system, as each machine is an additional point of failure, and the failure of a single machine makes at least part of the system unusable. To counteract this drawback, partitioning is usually combined with replication such that multiple replicas are responsible for the same partition.

On the positive side, each machine is now only responsible for a subset of the overall load, and all machines can operate in parallel, leading to reduced latency and increased throughput.

3 Cloud Computing

Cloud computing offers an infrastructure to build and use DSs.

3.1 Computing as Utility

- Computing as 5th utility (Buyya et al. 2009)
  - "Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered."

    For decades, there has been a vision of computing as utility, in the context of cloud computing with the quote shown here:

    "Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered."
  - Subscription-based pay-per-use of complex IT infrastructure, without heavy up-front investment/developments
    * Flexibility
  - Economies of scale for providers
    * Server consolidation with virtualization: Optimized use of hardware and space, energy, cooling

Thus, users should be able to subscribe to resources and services that require complex underlying IT infrastructure, where they only pay what they actually use. In particular, users do not have to set up the infrastructure themselves, but can flexibly use what they need without heavy up-front investments.

Providers benefit from economies of scale with optimized use of hardware and physical space, energy, or cooling, in particular based on virtualized compute resources with server consolidation.

3.2 NIST Definition

- (Mell and Grance 2011)

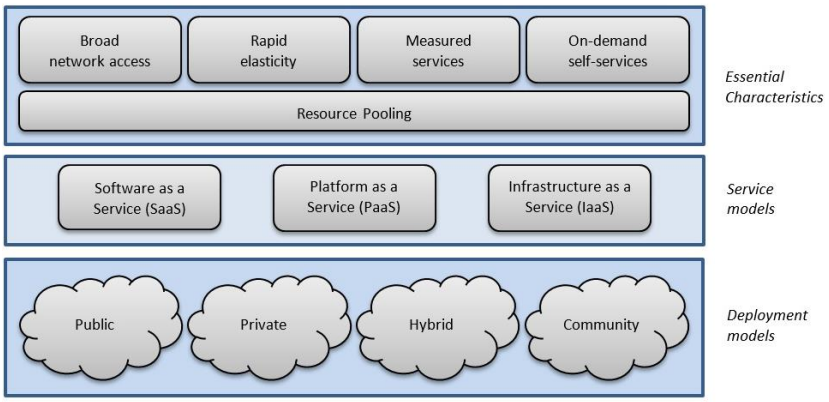

Figure 5: "NIST Visual model of cloud computing definition" by P Naveen, Wong Kiing Ing, Michael Kobina Danquah, Amandeep S Sidhu, and Ahmed Abu-Siada under CC BY 3.0; from Fig. 2 in P Naveen et al 2016 IOP Conf. Ser.: Mater. Sci. Eng. 121 012010

  - "**Cloud computing** is a **model** for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with **minimal management effort** or **service provider interaction**. This cloud model is composed of five **essential characteristics**, three **service models**, and four **deployment models**."
    * (Emphasis added)

(Figure source: (Naveen et al. 2016))

A frequently cited definition for cloud computing goes back to the National Institute of Standards and Technology in the US:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

Note how resource sharing mentioned in this definition aligns well with a major goal of DSs. Clearly, every IT system built using cloud technology is a distributed one.

We next look at the components of this definition in more detail.

3.2.1 NIST Definition: Characteristics

- **On-demand self-service**
  - Users provision servers with CPUs, memory, storage, networking
    * Without human interaction with service provider
- **Broad network access**

  - Capabilities accessible over Internet
- **Resource pooling**
  - Provider's computing resources assigned dynamically to multiple consumers
    * Multi-tenant
    * Customers without knowledge/control of exact locations
- **Rapid elasticity**
  - Capabilities quickly scalable up or down
  - Illusion of unlimited resources
- **Measured service**
  - Cloud systems control and optimize resource usage
    * For both providers and consumers

(Source: (Mell and Grance 2011))

The characteristics of cloud computing mentioned in the previous definition are as follows:

On-demand self-service refers to the ability of users to independently provision servers with specifications such as CPU, memory, storage, and networking. This is done without any need for human interaction with the service provider. Users can easily manage their own resources through a web-based interface or API.

Broad network access allows capabilities to be accessed through standard mechanisms over the internet, making it possible for remote users to utilize these services from anywhere at any time. It enables easy integration of cloud services into existing business processes and applications.

Resource pooling involves the dynamic assignment of computing resources by the provider to multiple consumers. These resources are multi-tenant, meaning they are shared among several customers who do not have explicit knowledge or control over the exact location of their data and workloads. Resource pooling ensures efficient utilization of resources while providing economies of scale.

Rapid elasticity signifies the quick scaling capability of cloud services based on demand. Resources can be scaled up during periods of high demand and then scaled down, or released, when no longer needed. This creates an illusion of having unlimited resources available whenever required. Rapid elasticity helps businesses maintain optimized performance levels even under fluctuating loads.

Measured service implies that cloud systems monitor and control resource usage automatically, ensuring fair allocation between different consumers. Both, providers and consumers, benefit from this feature since it leads to cost savings due to transparency regarding actual consumption patterns, enabling better budget management and forecasting.

3.2.2 NIST Definition: Service Models

- **Infrastructure as a Service** (IaaS)
  - Consumers deploy arbitrary software in VMs on provider's cloud infrastructure
    * "VMs in the cloud", e.g., major cloud providers and project seminar servers

IaaS provides consumers with the ability to deploy and run arbitrary software within virtual machines on a cloud provider's infrastructure. IaaS offers flexibility and control similar to traditional on-premises environments but eliminates the need for physical hardware maintenance. Examples include popular cloud platforms from Amazon, Google, or Microsoft, but also virtual servers provided inside the university, e.g., for project seminars.

- **Platform as a Service** (PaaS)
  - Consumers deploy applications using programming languages, libraries, and tools supported by provider
    - "Programming environment in the cloud", e.g., major cloud providers

PaaS enables consumers to develop, test, and deploy applications using programming languages, libraries, and tools supported by the provider. By abstracting away underlying infrastructure complexities, PaaS simplifies application development and deployment tasks. Users still have control over the deployed applications and possibly configuration settings for the application-hosting environment. Major cloud providers offer PaaS solutions as well.

- **Software as a Service (SaaS)**
  - Consumers use provider's applications on cloud infrastructure
    - "Applications in the cloud", e.g., office suite, CRM system, ERP system

SaaS delivers fully functional applications hosted on cloud infrastructure directly to end-users via the internet. SaaS offerings cover various domains, including office suites, customer relationship management, enterprise resource planning, and e-mail communication.

- (Anything as a Service (XaaS))
  - (X = Container, Function, Backend, Database, ...)

Beyond the previous definition, XaaS represents the trend towards delivering diverse IT components and capabilities as cloud services. With XaaS, there is virtually no limit to what could be offered "as a service", e.g., containers, functions, backends, databases, etc.

3.2.3 NIST Definition: Deployment Models

- **Public cloud**
  - Company manages cloud to be used by others
- **Private cloud**
  - Organization operates its own cloud
  - Exclusive use
- **Community cloud**
  - Community of consumers operates their own cloud
- **Hybrid cloud**
  - Two or more distinct private, community, or public cloud infrastructures
  - Standards for data and application portability

A public cloud refers to a type of cloud computing model where a third-party company manages and maintains the cloud infrastructure, which is made available for other organizations or individuals to use. Operating on a pay-per-use basis, they may allow rapid scaling of resources according to demand. They are suitable for a wide range of applications, including web hosting, big data analytics, and software development.

A private cloud denotes an organization operating its dedicated cloud infrastructure solely for internal use. Private clouds may reside either on-premises or off-site, managed by the organization itself or a trusted partner. Due to increased security and compliance features compared to public clouds, private clouds may be preferred when handling sensitive information or requiring strict governance policies.

A community cloud describes a collaborative effort involving multiple organizations sharing resources, expertise, and costs to build and operate a common cloud platform tailored to meet specific needs.

A hybrid cloud combines two or more distinct private, community, or public cloud infrastructures bound together by standards for data and application portability. Hybrid clouds aim to enable seamless migration of workloads between disparate environments depending on factors such as cost, availability, and performance considerations.

3.3 Cloud Caveats

- Rapid elasticity?

  Recall that the above definition of cloud computing includes the characteristic rapid elasticity and the service model IaaS. Note that IaaS is scalable in the sense that users can create more and more VMs or containers when demand increases. However, users need to manage such scaling themselves.

  E.g., consider a web server, which might run in a VM (or in a container). When the hosted website increases in popularity, scaling out can be applied to start the web server in multiple VMs. On their own, these additional servers are unreachable by clients, though. To make scale out work, one typically adds some kind of load balancer as contact point for clients. The load balancer knows about all VMs and their current load, so that it can distribute incoming requests appropriately, balancing the load among VMs.

  When popularity decreases, VMs (or containers) should be destroyed again to reduce operational costs. Again, such management operations are not included with IaaS. In essence, elastic scaling is not supported well with IaaS.

  To address elastic scaling, orchestration software such as Kubernetes with its support for autoscaling, or serverless computing offerings can be used.

- Digital sovereignty?
  - Provider reliability
  - Vendor lock-in

  Digital sovereignty encompasses the concept of maintaining control over one's digital assets, data, and decision-making processes. In a cloud computing context, key aspects related to digital sovereignty include provider reliability and vendor lock-in. Choosing a suitable cloud provider has immediate impact on uptime, support, and timely updates. Binding oneself to offerings of a single provider, in particular without established standards, may lead to vendor lock-in, limiting freedom of choice and potentially compromising negotiation power.

- Security and privacy concerns
  - Data and code on someone else's machines
  - Who has access when? Misconfigurations?
    - See (Buyya et al. 2018) for selected techniques, e.g., homomorphic encryption

  - Beware of marketing claims
    * E.g., see Google's 2024 analysis of "Microsoft's ongoing security struggles"

Security and privacy concerns arise because storing and processing data on external servers imply entrusting them to another party. Risks associated with placing critical assets on someone else's machines must be evaluated against potential benefits. Factors to consider include unauthorized access, notably surveillance by US authorities, misconfigurations leading to vulnerabilities, and ambiguous terms governing ownership, retention, and deletion of data.

The paper cited here points to security measures against some risks, e.g., homomorphic encryption, which enables computations over encrypted data.

Additionally, "security" may be misrepresented as marketing claim in cloud contexts. Maybe read what Google has to say about "Microsoft's ongoing security struggles", referring to several incidents in Microsoft's cloud infrastructure in 2024.

4 Serverless Computing

- (Kounev et al. 2023)
  - "Serverless computing is a cloud computing paradigm encompassing a class of cloud computing platforms that allow one to develop, deploy, and run applications (or components thereof) in the cloud without allocating and managing virtualized servers and resources or being concerned about other operational aspects."
    * Provider is responsible for operational aspects, e.g., fault tolerance, elastic scaling
    * Pay-per-use with fine granularity
    * Examples include AWS Lambda, Google Cloud Functions

One particular form of cloud computing is branded as "serverless computing". It can be defined as follows:

"Serverless computing is a cloud computing paradigm encompassing a class of cloud computing platforms that allow one to develop, deploy, and run applications (or components thereof) in the cloud without allocating and managing virtualized servers and resources or being concerned about other operational aspects."

Notably, with serverless computing, the provider is responsible for operational aspects, e.g., fault tolerance and elastic scaling. In addition, services must be offered under pay-per-use with a fine granularity.

Various companies offer serverless computing, including Amazon and Google.

Please take some time to think about differences and commonalities between cloud and serverless computing.

4.1 Self-Study

- How do the above service models of cloud computing relate to serverless computing?

Take a break to work on the self-study task here.

4.2 Sample Serverless Applications

Examples for serverless applications are as follows.

- Anomaly detection for industrial sensors

  - Cloud functions consume micro batches, e.g., thresholding or machine learning for anomaly detection

Anomaly detection for industrial sensors may be implemented with serverless cloud functions to analyze microbatches of sensor data using methods such as thresholding or machine learning algorithms. These approaches facilitate early identification of abnormal trends, allowing preventive actions before significant issues occur.

- Object storage
  - (Unlimited) storage of data with meta-data under unique identifiers
  - No details of servers necessary

Object storage is a classical cloud offering, which has always been close to "serverless". It provides nearly unlimited storage space for vast amounts of data accompanied by metadata under unique identifiers. Unlike traditional server-centric architectures, object storage does not require detailed knowledge of individual server configurations, thus streamlining administration tasks and enhancing overall simplicity.

- Serverless databases, SQL-as-a-Service
  - Capacity planning and autoscaling included

Serverless databases and SQL as a Service eliminate manual capacity planning; they autoscale resources according to demands. This approach reduces administrative overhead and aligns database functionality closely with application needs. Moreover, autoscaling ensures consistent performance levels regardless of varying load intensities.

- Serverless edge or fog computing
  - Computation and storage close to data sources
    * (In contrast to shipping of data to central data centers for computations)
    * E.g., real-time computer vision or analytics for mobile or (resource-constrained) IoT devices

Edge or fog computing shifts computation and storage closer to data sources, reducing latency and improving response times. Instead of transferring raw data to distant centralized data centers, preprocessing occurs near the source, resulting in faster results and lower bandwidth requirements. Also in this context, serverless approaches are emerging.

Applications include real-time computer vision or analytics for mobile devices and resource-constrained IoT devices.

(Source: (Kounev et al. 2023))

5 Conclusions

Let us conclude.

5.1 Summary

- Distributed systems are everywhere
  - Internet as core infrastructure
  - Networked machines coordinated with messages

- Cloud computing provides infrastructure for distributed systems
  - Different service models for different applications
  - Serverless computing as paradigm without operational concerns for users with pay-per-use

Distributed systems form the foundation of modern computing landscapes, with the Internet serving as a primary infrastructure facilitating interactions between interconnected devices worldwide. Through messaging, these machines coordinate activities and share resources efficiently.

Cloud computing extends the reach of distributed systems by offering service models upon which developers can construct innovative applications, from simple file storage to arbitrarily complex functionality. Among cloud computing offerings, serverless computing emerges as a prominent paradigm, empowering users to create and deploy applications, with pay-per-use pricing, without worrying about operational concerns.

Bibliography

Buyya, Rajkumar, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, et al. 2018. "A Manifesto for Future Generation Cloud Computing: Research Directions for the next Decade." *Acm Comput. Surv.* 51 (5): 1–38. https://doi.org/10.1145/3241737.

Buyya, Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2009. "Cloud Computing and Emerging It Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility." *Future Generation Computer Systems* 25 (6): 599–616. https://doi.org/10.1016/j.future.2008.12.001.

Coulouris, George, Jean Dollimore, Tim Kindberg, and Gordon Blair. 2011. *Distributed Systems: Concepts and Design*. 5th ed. USA: Addison-Wesley Publishing Company. http://www.cdk5.net/.

Farooqui, Kazi, Luigi Logrippo, and Jan de Meer. 1995. "The Iso Reference Model for Open Distributed Processing: An Introduction." *Computer Networks and Isdn Systems* 27 (8): 1215–29. http://www.sciencedirect.com/science/article/pii/016975529500087N.

Kounev, Samuel, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup, Ian Foster, Prashant Shenoy, Omer Rana, and Andrew A. Chien. 2023. "Serverless Computing: What It Is, and What It Is Not?" *Commun. Acm* 66 (9): 80–92. https://doi.org/10.1145/3587249.

Mell, Peter, and Timothy Grance. 2011. "The Nist Definition of Cloud Computing." *Nist Special Publication* 800-145. https://doi.org/10.6028/NIST.SP.800-145.

Naveen, P, Wong Kiing Ing, Michael Kobina Danquah, Amandeep S Sidhu, and Ahmed Abu-Siada. 2016. "Cloud Computing for Energy Management in Smart Grid - an Application Survey." *Iop Conference Series: Materials Science and Engineering* 121 (1): 012010. https://doi.org/10.1088/1757-899X/121/1/012010.

Neuman, B. Clifford. 1994. "Scale in Distributed Systems." In *Readings in Distributed Computing Systems*. IEEE Computer Society Press. http://clifford.neuman.name/publications/.

Tanenbaum, Andrew S., and Maarten van Steen. 2007. *Distributed Systems: Principles and Paradigms*. 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

The bibliography contains references used in this presentation.

License Information