Guest Lecture: Cloud Computing

CPEN400A - Building Modern Web Applications - Winter 2020-1

Julien Gascon-Samson, Assistant Professor

ÉTS Montréal / University of Quebec
Département of Software and IT Engineering
Montreal, Canada

Tuesday, Nov 17, 2020

Introduction

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

Guest Lecturer: Julien Gascon-Samson

- Assistant Professor at ÉTS Montréal / University of Quebec (Software & IT Engineering Dept.)
    - On the lookout for highly-motivated PhD / MSc students – funded positions available :-)
- Before: Post-Doctoral Fellow at UBC (ECE)
    - PhD from McGill University (Montreal, 2017)
    - Master's in Computer Engineering (École Polytechnique de Montréal, 2011)
    - Undergrad in Software Engineering (École Polytechnique de Montréal, 2009)
- Research
    - Internet Of Things (IoT) and Web of Things (WoT)
    - Cloud / Edge / Distributed Systems
    - Stream Processing
    - Publish/Subscribe
    - Networking for Multiplayer Games

ÉTS Montréal / University of Quebec

- Part of the *University of Quebec* network : system of 10 publicly run universities in Quebec
- Offers only **Engineering** programs (technical university)
  - Similar to University of Polytechnique (UMontreal) and some technical schools in Europe
- French language for courses
  - Undergrad: French-speaking students are admitted
  - Grad: French and English-speaking (courses can be done at McGill and Concordia through a partnership)

ÉTS Montréal / University of Quebec

- Applied research and teaching in engineering
- Industry partnerships & tech transfer
- Ranks 1st in the number of engineers trained in Quebec and 2nd in Canada

Definitions

1 Introduction

2 Definitions

3 Characteristics

4 Service Models

5 Deployment Models

6 Virtualization and Elasticity

7 Typical Cloud Services
- Data Storage in the Cloud
- Communications: Publish/Subscribe
- Batch Processing: Map/Reduce
- Serveless Computing / Function as a Service

8 Edge Computing

Distributed Computing

Distributed System: Definition (Wikipedia)

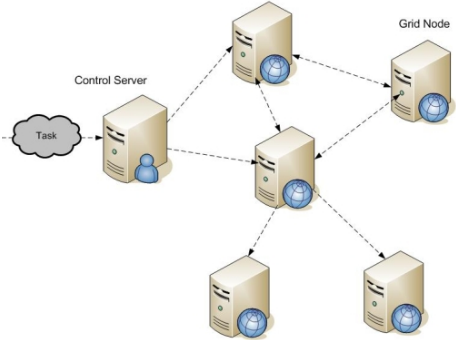
*A system whose **components** are located on **different networked computers**, which then **communicate** and **coordinate their actions** by **passing messages** to one another.*

- Very broad! Different models are possible:
  - Centralized
  - Peer-to-peer/"mesh"
  - Hybrid

Grid Computing

Grid Computing: Definition (Wikipedia)

*A **combination** of **computer resources** from multiple administrative domains applied to a **common task**.*



Utility Computing

Utility Computing: Definition (Wikipedia)

*The packaging of **computing resources** (computation, storage etc.) **as a metered service** similar to a traditional public utility.*

Cloud Computing?

- Grid Computing + Utility Computing?
- Very hard to define – can mean so many different things to different parties!
- Many definitions

Cloud Computing: Definition (NIST)

*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.*

Cloud Characteristics

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
    - Data Storage in the Cloud
    - Communications: Publish/Subscribe
    - Batch Processing: Map/Reduce
    - Serveless Computing / Function as a Service
8. Edge Computing

Cloud Characteristics (1)

1. On-demand Self Service

- Ability to provision computing capabilities without intervention
  - Computation (“aka machine”) time
  - Storage

Cloud Characteristics (1)

1. On-demand Self Service

- Ability to provision computing capabilities without intervention
  - Computation (“aka machine”) time
  - Storage

2. Broad network access

- Capabilities available over the network
- Accessible by *thin* and *thick* clients (e.g., desktop/laptops, mobile devices, etc.)

Cloud Characteristics (2)

3. Resource pooling

- Multi-tenancy: the same cloud infrastructure can serve multiple customers, host multiple VMs, applications
- Computing resources are *pooled* (to serve multiple users)
  - Storage
  - Processing
  - Memory
  - Network
- *Physical* and *logical* resources are dynamically assigned and reassigned according to consumer demand
- Location independence
  - Precise location of the resources
  - Only a general idea (e.g., Amazon EC2 US-east)

Cloud Characteristics (3)

4. Rapid elasticity

- *Elastic* provisioning – scaling up and down
- Can be done automatically
- To consumers: pool of resources might appear to be infinite

Cloud Characteristics (3)

4. Rapid elasticity

- *Elastic* provisioning – scaling up and down
- Can be done automatically
- To consumers: pool of resources might appear to be infinite

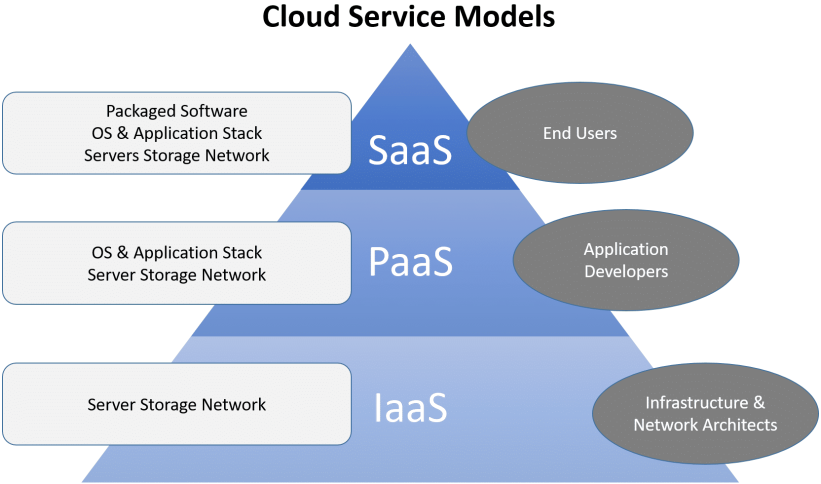
5. Measured service

- *Metering* of the different resources
  - CPU (e.g., \$/CPU time in ms)
  - Network bandwidth (e.g., \$/gb)
  - Processing (e.g., \$/X requests)
  - Storage (e.g., \$/gb)
- Monitoring, controlling, reporting
- Full transparency for cloud operator and consumer

Service Models

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

Hierarchy of Service Models (Source)


Cloud Service Models

Infrastructure as a Service (IaaS) (1)

- Consumer can provision virtualized computing resources (aka VMs)
    - Processing, storage, network, GPU
- Can include OS and applications, or be bare metal
    - Example: Amazon EC2, Azure
- Consumer doesn't manage the hardware (physical or virtualized)
    - But has control over the OS, storage, applications, and limited network settings
        - e.g., firewall, port redirection, VLans, etc

Infrastructure as a Service (IaaS) (2)

Virtualizing a machine implies that all of its components must be virtualized as well!

- Processor (CPU): virtualized from “real”, physical CPUs
  - Hardware acceleration is available on most recent processors
  - Support for multiple cores
  - “Standardized” metrics for modelling the performance of CPUs (e.g., Amazon vCPU)
  - Specific features (e.g., “bursting”)
- Memory
- Storage
- Networking: SDNs
  - Network configuration is defined by software and not purely by the hardware (routers, switches)
  - Bandwidth, firewalls, subnets, etc.
- GPUs and other devices

The components of a VM are not all necessarily located on the same physical machine!

Data storage in the cloud

- **File storage:**
  - Common/typical storage abstractions (file systems, folders, files, etc.)
  - Emulation of a "local" hard disk, but provided over the network
  - Protocols: NFS, etc.

Data storage in the cloud

- **File storage:**
  - Common/typical storage abstractions (file systems, folders, files, etc.)
  - Emulation of a "local" hard disk, but provided over the network
  - Protocols: NFS, etc.
- **Object storage:**
  - Storage of objects and metadata (BLOB)
  - ID for each "object"
  - Typically accessed thru standard access protocols (e.g., HTTP)
  - Version control systems (VCS) (e.g., Git, SVN) make use object storage
  - Different storage systems are available based on customer needs (costs, frequency of data reads/writes, throughput, latency, etc.)
  - Replication, versionning, encryption, availability in several "zones", etc.
  - e.g., Amazon S3, Google Cloud Storage, Amazon Glacier

Data storage in the cloud

- **File storage:**
  - Common/typical storage abstractions (file systems, folders, files, etc.)
  - Emulation of a “local” hard disk, but provided over the network
  - Protocols: NFS, etc.
- **Object storage:**
  - Storage of objects and metadata (BLOB)
  - ID for each “object”
  - Typically accessed thru standard access protocols (e.g., HTTP)
  - Version control systems (VCS) (e.g., Git, SVN) make use object storage
  - Different storage systems are available based on customer needs (costs, frequency of data reads/writes, throughput, latency, etc.)
  - Replication, versionning, encryption, availability in several “zones”, etc.
  - e.g., Amazon S3, Google Cloud Storage, Amazon Glacier
- **Block storage:**
  - Very low-level: emulates a fixed storage block
  - Can be mount as a networked hard disk, either over a filesystem or raw
  - Different storage technologies are available: SSD, magnetic, etc.
  - e.g., can be used for storing database data

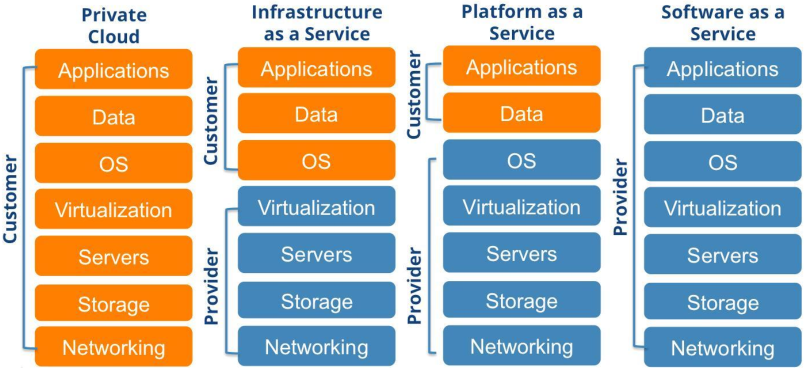
Platform as a Service (PaaS)

- Enables the deployment, management and execution of **consumer or acquired applications** onto cloud infrastructure (IaaS)
  - For customers: alleviates the need for managing the applications on their own infrastructure
  - Can be written into a variety of languages
  - Using a variety of libraries, services, tools supported by the provider
  - e.g., Web apps (Heroku, Google App Engine), APIs, microservices
- No control over underlying cloud infrastructure (IaaS)!
- Control over deployed applications
- Might have limited control over configuration settings of the hosting environment (e.g., config files)

Software as a Service (SaaS)

- Use the provider’s specific applications
  - Over the cloud provider’s infrastructure (hardware + software/PaaS)
- Accessible from various clients
  - Thin & thick clients, mobile, web (e.g., web-based email)
- Consumer does not manage the underlying cloud infrastructure (network, servers, OS, storage, applications)
- Exception: limited user-specific application configuration settings (e.g., GMail settings)

Provisioning in Service Models (Source)



Deployment Models

1 Introduction

2 Definitions

3 Characteristics

4 Service Models

5 Deployment Models

6 Virtualization and Elasticity

7 Typical Cloud Services
- Data Storage in the Cloud
- Communications: Publish/Subscribe
- Batch Processing: Map/Reduce
- Serveless Computing / Function as a Service

8 Edge Computing

Public cloud

- Open use by general public
- Owned by business, academic, government organization, or a combination
- Exists on the premises the of cloud provider
- Example: Amazon, Google, MS Azure

Private cloud

- Exclusive use of a single organization with multiple “internal” consumers
    - e.g. different business units within a given organization
- Owned, managed, operated by organization, or a third-party, or a combination
- May exist on or off premises
- Example: a large company (e.g., Amazon Internal Cloud)

Community cloud

- Exclusive use of a specific community of consumers from **organizations with shared concerns**
  - Mission, security requirements, policy, compliance considerations
- Owned, managed, operated by one or more organizations in the community, a third party, or a combination of them
- May exist on or off premises
- Examples: Amazon Government Cloud, clouds that comply with BC data policies (e.g., UBC Workspace)

Hybrid cloud

- A composition of two or more distinct cloud infrastructure

Virtualization

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

What is virtualization?

Decoupling the **physical resources (physical hardware)** into **virtual resources**

Why virtualize?

- Cloud provider might have heterogeneous hardware
- Offering a consistent configuration to the different customers of the cloud
  - CPU performance
  - Amount of memory
  - Storage
  - Network bandwidth
- Offering additional isolation (reliability)
- **Virtualization of resources happen at different levels based on the service model!**

Virtualization: IaaS (1)

Hardware-level virtualization (lowest level)

- 1 Physical machine ⇒ *n virtual* machines
- Hypervisor: VMWare, VirtualBox, MS HyperV, Xen, etc.
- Run over an OS or “bare-metal”
- Nowadays, virtualization is hardware-assisted: can run at near-native speeds

Virtualization (IaaS) (2)

Hardware-level virtualization (lowest level)

Virtualized Hardware

- CPU (modern CPUs support virtualization extensions - e.g., Intel VT, AMD-V)
- Memory: portions of the RAM of the host machine are reserved
- Storage: virtual hard drives and other I/O peripherals, data center storage
- Network: virtual network adapters, virtualized networks/subnets
- GPU: for specific applications
- Other devices / pass-thru (e.g., USB)

Virtualization: PaaS / SaaS

Virtualization of the **combined** resources of a pool of machines (VMs)

- Build over IaaS virtualization layer
- Processing power (CPU)
- Pools of memory
- Distributed data storage
- Virtualized networking and adressing

Elasticity

- Allocating a pool of resources from the provider in an “elastic” manner, according to current needs
- Resource allocation can be made directly according to user requirements (e.g., Amazon EC2 dashboard, or thru the command-line) - or -
- Can be triggered by the needs of higher-level apps & services deployed over higher-level layers (e.g., PaaS)

Elasticity: IaaS

Two approaches to scalability:

- Vertical: more powerful hardware (limited)
- **Horizontal: partitioning / sharding**

Elasticity: IaaS (Infrastructure as a Service)

- Allocating new VM instances
- Deallocating instances which aren't needed anymore
- Allocating storage, RAM, network (or a specific network configuration), etc. (can be properties of the VMs)

Elasticity: PaaS

Elasticity: PaaS (Platform as a Service):

- Automatic provisioning of VM/physical resources (IaaS layer) to execute the PaaS application
- The elasticity of the application itself might or might not be done automatically

Elasticity: PaaS

Elasticity: PaaS (Platform as a Service):

- Automatic provisioning of VM/physical resources (IaaS layer) to execute the PaaS application
- The elasticity of the application itself might or might not be done automatically

Example: for a request-based application, the PaaS "execution layer" could provision / allocate enough resources as necessary from the IaaS layer to satisfy the current volume of requests

- The unit of measure for control & billing purposes can then be different (higher-level) compared to the billing metrics for the IaaS layer
- For instance, the customer can be billed by the number of requests or for the execution time alloted for handling the requests (as opposed to billing for the "raw" CPU usage, memory, etc. of the VM)

Elasticity: SaaS

Elasticity: SaaS (Software as a Service):

- Fully managed provisioning of the PaaS layer
- e.g., Gmail will provision enough combined resources at the PaaS layer, which in turn will provision enough resources at the IaaS layer (type and # of VMs, network, etc.)

Data Storage in the Cloud

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

Data Storage in the Cloud

How data can be stored across different nodes?

- Distributed File Systems
  - Google FS, Hadoop
  - Provides file-system like abstractions in a distributd manner
- Block Storage
  - Amazon S3 (storage of objects, can be files)
- Databases:
  - SQL
  - NoSQL (e.g., Key-value Stores, MongoDB, etc.)

Data Storage in the Cloud: Properties

- Scalability
- High availability
- Low latency
- Durability
- Fault tolerant
- Predictable costs

Data Storage in the Cloud: Properties

- Scalability
- High availability
- Low latency
- Durability
- Fault tolerant
- Predictable costs

Tradeoff: the CAP Theorem

- Consistency
- Availability
- Partition tolerance

Pick only two :-)

- Cloud storage systems often opt for **eventual consistency**

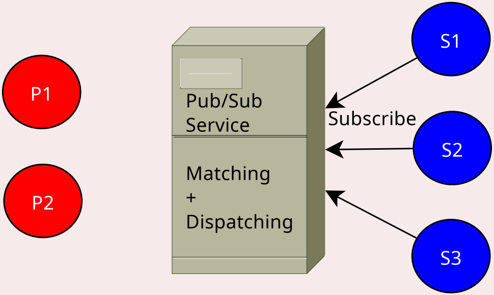
Communications: Publish/Subscribe

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

Publish/Subscribe Paradigm

- Provides an elegant way to decouple content producers (publishers) from content consumers (subscribers)
- Publications are matched against subscriptions
- Many *flavours* of publish/subscribe

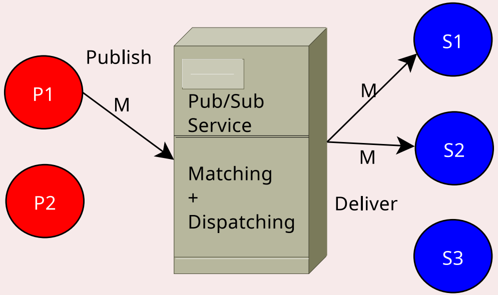
Publish/Subscribe - Example



Publish/Subscribe Paradigm

- Provides an elegant way to decouple content producers (publishers) from content consumers (subscribers)
- Publications are matched against subscriptions
- Many *flavours* of publish/subscribe

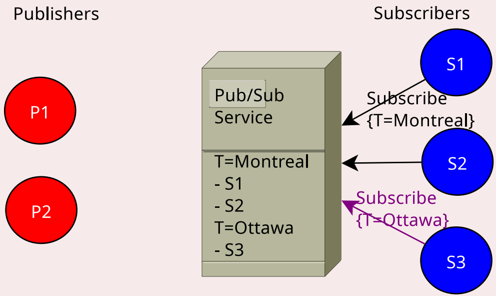
Publish/Subscribe - Example



Topic-Based Publish/Subscribe

- Very common flavour of pub/sub
- Subscription language: a key (topic name)
- Publications tagged with a topic T, sent to all subscribers of T

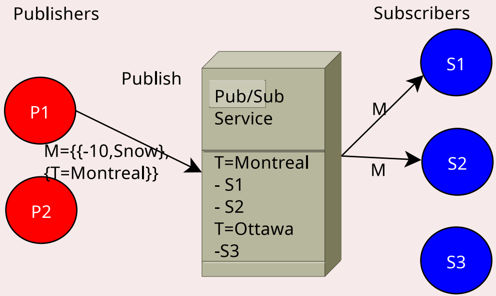
Example - Weather Reports



Topic-Based Publish/Subscribe

- Very common flavour of pub/sub
- Subscription language: a key (topic name)
- Publications tagged with a topic T, sent to all subscribers of T

Example - Weather Reports



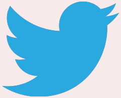
Applications of Topic-Based Pub/Sub

Traffic alert systems

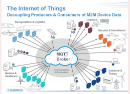
Weather alert systems

Mobile notif. frameworks

Social networks



IoT



Multiplayer Games

Desirable properties:

- Scalability & Elasticity
- Low Latency
- Reduced & Predictable Costs

Batch Processing: Map/Reduce

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

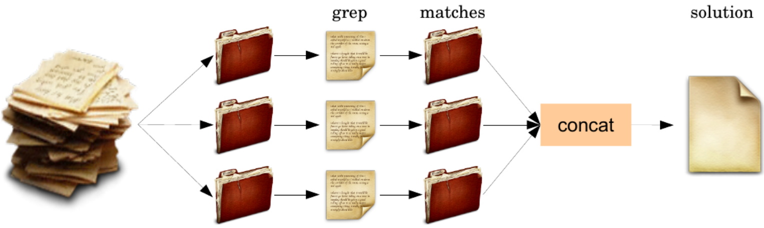
Map/Reduce (Source)

- Functional Decomposition:
  - Breaking a large problem broken into a set of small problems
- Each small problem:
  - can be solved by a functional transformation of input data
  - can be executed in complete isolation (parallel computing)

Examples (next slides) – what do these Linux programs do?

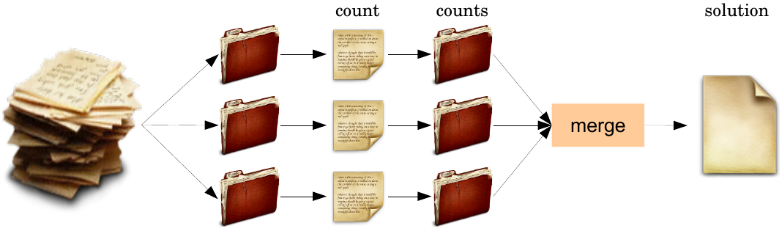
- **grep**
- **wc (word count)**

grep with MapReduce



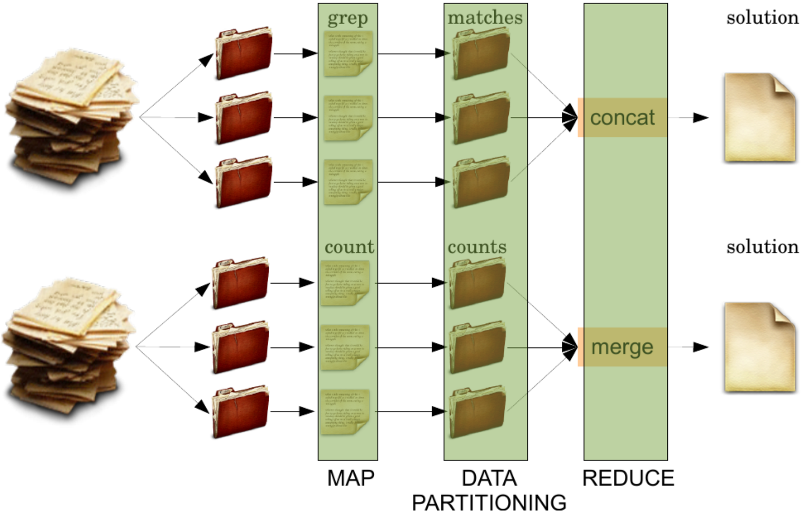
- Partitioning the files to be searched onto several nodes (map)
- Executing "grep" on each instance
- Partitioning the intermediate results to send them to a "reducer"
- Concatenation of all intermediate results

wc with MapReduce



- Partitioning the files on which the wordcount should be performed onto several nodes (map)
- Executing "count" on each instance to compute the number of occurences of each word
- Partitioning the intermediate "counts" to send them to a "reducer" (e.g., by hashing the words)
- Merging of all results (adding the partial counts for each word)

grep and wc with MapReduce



Serveless Computing / Function as a Service

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
   - Data Storage in the Cloud
   - Communications: Publish/Subscribe
   - Batch Processing: Map/Reduce
   - Serveless Computing / Function as a Service
8. Edge Computing

Function as a Service (FaaS)

- Application made of a set of functions
- Executed upon certain events being triggered
  - Web request
  - File upload
  - Change to DB
  - Timer
- Executed within containers (thin VMs)
  - Full isolation
  - FaaS functions are **stateless!**
    - Changes in state must be persisted to durable storage
- Example: Amazon Lambda, Google Cloud Functions, MS Azure Functions
- Some "functions" can be executed "at the edge": e.g., lambda@edge

Edge Computing

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
    - Data Storage in the Cloud
    - Communications: Publish/Subscribe
    - Batch Processing: Map/Reduce
    - Serveless Computing / Function as a Service
8. Edge Computing

Edge Computing

Edge Computing, from a systems point of view, aims at processing the data as close as possible to where the data is **produced** and **consumed**.

The definition by itself is very vague, and can refer to various models that are described in the literature.

- For instance, for cloud providers, the "edge" can refer to smaller/micro cloud deployoments that are more "localized"
- In other contexts, the "edge" can refer to processing the data onto the devices themselves that are becoming more and more powerful
    - For instance, Raspberry Pi devices run a full Linux OS

Edge Computing, why is it important?

The cloud is widespread and is typically “reachable” from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

Edge Computing, why is it important?

The cloud is widespread and is typically "reachable" from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

1. Limited connectivity
    - For devices that are deployed in an isolated environment, or if the wireless communication is intermittent

Edge Computing, why is it important?

The cloud is widespread and is typically “reachable” from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

1. Limited connectivity
   - For devices that are deployed in an isolated environment, or if the wireless communication is intermittent
2. Lower latency
   - Some critical apps must react very quickly to specific events – in that case, sending the data to the cloud (back and forth) can push the latency above a critical level
   - e.g., smart vehicles

Edge Computing, why is it important?

The cloud is widespread and is typically “reachable” from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

1. Limited connectivity
   - For devices that are deployed in an isolated environment, or if the wireless communication is intermittent
2. Lower latency
   - Some critical apps must react very quickly to specific events – in that case, sending the data to the cloud (back and forth) can push the latency above a critical level
   - e.g., smart vehicles
3. Alleviating the dependence to a third party
   - Some IoT devices are too strongly coupled to a specific cloud-based service, and can stop working if the cloud service ceases to exist, or if the operator changes its terms of use.

Edge Computing, why is it important?

The cloud is widespread and is typically “reachable” from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

1. Limited connectivity
   - For devices that are deployed in an isolated environment, or if the wireless communication is intermittent
2. Lower latency
   - Some critical apps must react very quickly to specific events – in that case, sending the data to the cloud (back and forth) can push the latency above a critical level
   - e.g., smart vehicles
3. Alleviating the dependence to a third party
   - Some IoT devices are too strongly coupled to a specific cloud-based service, and can stop working if the cloud service ceases to exist, or if the operator changes its terms of use.
4. Security requirements
   - It might be preferable to process the data locally to meet the needs of security requirements and policies.

Edge Computing, why is it important?

The cloud is widespread and is typically “reachable” from anywhere – however, in some cases, it might not be the best option (in the IoT landscape). **Why?**

1. Limited connectivity
   - For devices that are deployed in an isolated environment, or if the wireless communication is intermittent
2. Lower latency
   - Some critical apps must react very quickly to specific events – in that case, sending the data to the cloud (back and forth) can push the latency above a critical level
   - e.g., smart vehicles
3. Alleviating the dependence to a third party
   - Some IoT devices are too strongly coupled to a specific cloud-based service, and can stop working if the cloud service ceases to exist, or if the operator changes its terms of use.
4. Security requirements
   - It might be preferable to process the data locally to meet the needs of security requirements and policies.
5. Reducing the costs
   - In a context in which a lot of data is produced and consumed, processing some of the data locally can reduce the volume of data that is sent and processed in the cloud.

Table of Contents

1. Introduction
2. Definitions
3. Characteristics
4. Service Models
5. Deployment Models
6. Virtualization and Elasticity
7. Typical Cloud Services
    - Data Storage in the Cloud
    - Communications: Publish/Subscribe
    - Batch Processing: Map/Reduce
    - Serveless Computing / Function as a Service
8. Edge Computing