Data Integration and Large-scale Analysis (DIA)
08 Cloud Computing Fundamentals

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Dec 04, 2025

Announcements / Administrative Items

- **#1 Video Recording**
  - Hybrid lectures: in-person BH-N 243, zoom live streaming, video recording
  - https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09

- **#2 Exercises/Projects**
  - **Reminder:** exercise/project submissions by **Jan 30** (no extensions)
  - Make use of **virtual** / **in-person** (FR-766) office hours **Wed 4.30pm-6pm**

- **#3 Exam Registration**
  - Three exams now **open for registrations in Moses**
  - Written exams **Feb 05, 4pm** (BH-N 243), **Feb 12, 4pm** (BH-N 243), and **Mar 12, 4pm** (A 151)

Course Outline Part B:
Large-Scale Data Management and Analysis

12 Distributed Stream Processing

13 Distributed Machine Learning Systems

Compute/ Storage

- 11 Distributed Data-Parallel Computation
- 10 Distributed Data Storage

Infra

- 09 Cloud Resource Management and Scheduling
- 08 Cloud Computing Fundamentals

Agenda

- **Motivation and Terminology**
- **Cloud Computing Service Models**
- **Cloud, Fog, and Edge Computing**

Motivation and Terminology

Motivation Cloud Computing

- **Definition Cloud Computing**
  - **On-demand, remote storage and compute resources, or services**
  - **User:** computing as a utility (similar to energy, water, internet services)
  - **Cloud provider:** computation in data centers / multi-tenancy

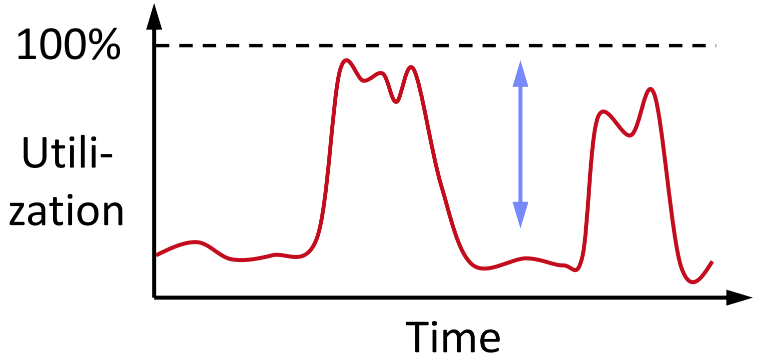
"Computing as a Utility"

- **Service Models**
  - **IaaS: Infrastructure as a service** (e.g., storage/compute nodes)
  - **PaaS: Platform as a service** (e.g., distributed systems/frameworks)
  - **SaaS: Software as a Service** (e.g., email, databases, office, github)

➔ **Transforming IT Industry/Landscape**
  - Since ~2010 increasing move from on-prem to cloud resources
  - System software licenses become increasingly irrelevant
  - Few cloud providers dominate IaaS/PaaS/SaaS markets (w/ 2018 revenue):
    Microsoft Azure Cloud ($ 32.2B), **Amazon AWS** ($ 25.7B), **Google Cloud** (N/A), **IBM Cloud** ($ 19.2B), **Oracle Cloud** ($ 5.3B), **Alibaba Cloud** ($ 2.1B)

Motivation Cloud Computing, cont.

- **Argument #1: Pay as you go**
  - No upfront cost for infrastructure
  - Variable utilization ➔ over-provisioning
  - **Pay per use or acquired resources**



- **Argument #2: Economies of Scale**
  - Purchasing and managing IT infrastructure at scale ➔ **lower cost** (applies to both HW resources and IT infrastructure/system experts)
  - Focus on **scale-out on commodity HW** over scale-up ➔ **lower cost**

100 days @ 1 node

≈

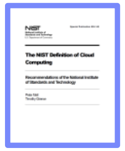
1 day @ 100 nodes

(but beware Amdahl's law: max speedup **sp = 1/s**)

- **Argument #3: Elasticity**
  - Assuming perfect scalability, work done in **constant time * resources**
  - Given virtually unlimited resources allows to reduce time as necessary

Characteristics and Deployment Models

[Peter Mell and Timothy Grance: The NIST Definition of Cloud Computing, **NIST 2011**]



- **Extended Definition**
  - ANSI recommended definitions for service types, characteristics, deployment models
- **Characteristics**
  - **On-demand self service:** unilateral resource provision
  - **Broad network access:** network accessibility
  - **Resource pooling:** resource virtualization / multi-tenancy
  - **Rapid elasticity:** scale out/in on demand
  - **Measured service:** utilization monitoring/reporting
- **Deployment Models**
  - **Public cloud:** general public, on premise of cloud provider
  - **Hybrid cloud:** combination of two or more of the above
  - **Community cloud:** single community (one or more orgs)
  - **Private cloud:** single org, on/off premises

MS Azure Private Cloud

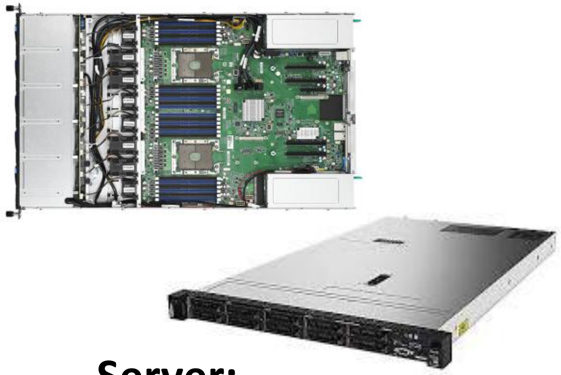
IBM Cloud Private

Cloud Computing Service Models

"Computing as a utility"

Anatomy of a Data Center

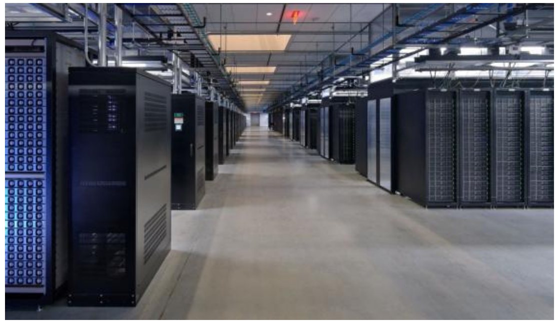
Commodity/Server CPUs:
Xeon E5-2440: 6/12 cores
Xeon Gold 6148: 20/40 cores
Xeon Gold 6430: 64/128 cores



Server:
Multiple sockets,
RAM, disks

Rack:
16-64 servers +
top-of-rack switch

Cluster:
Multiple racks + cluster switch



Data Center:
>100,000 servers

[Google
Data Center,
Eemshaven,
Netherlands]

Fault Tolerance

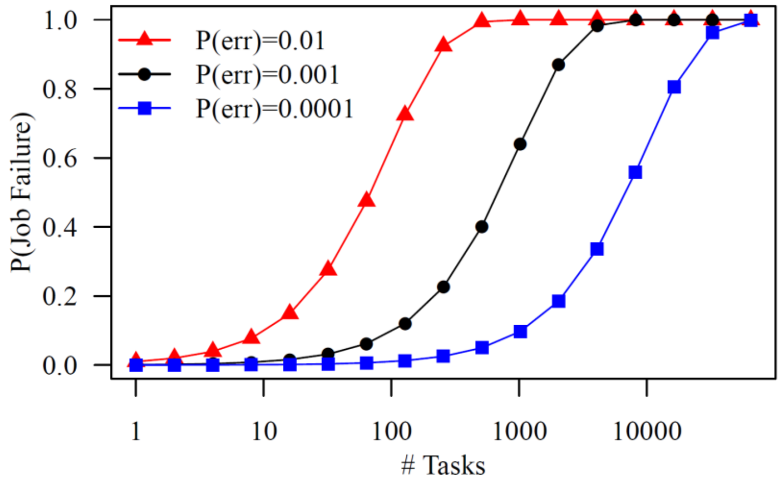
[Christos Kozyrakis and Matei Zaharia: CS349D: Cloud Computing Technology, lecture, **Stanford 2018**]

- **Yearly Data Center Failures**
  - ~0.5 overheating (power down most machines in <5 mins, ~1-2 days)
  - ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hrs)
  - ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hrs)
  - ~1 network rewiring (rolling ~5% of machines down over 2-day span)
  - ~20 rack failures (40-80 machines instantly disappear, 1-6 hrs)
  - ~5 racks go wonky (40-80 machines see 50% packet loss)
  - ~8 network maintenances (~30-minute random connectivity losses)
  - ~12 router reloads (takes out DNS and external vIPs for a couple minutes)
  - ~3 router failures (immediately pull traffic for an hour)
  - ~dozens of minor 30-second blips for dns
  - ~1000 individual machine failures (2-4% failure rate, at least twice)
  - ~thousands of hard drive failures (1-5% of all disks will die)

Fault Tolerance, cont.

- **Other Common Issues**
  - **Configuration issues**, partial SW updates, SW bugs
  - **Transient errors:** no space left on device, memory corruption, stragglers

- **Recap: Error Rates at Scale**
  - Cost-effective commodity hardware
  - Error rate increases with increasing scale
  - Fault Tolerance for distributed/cloud storage and data analysis

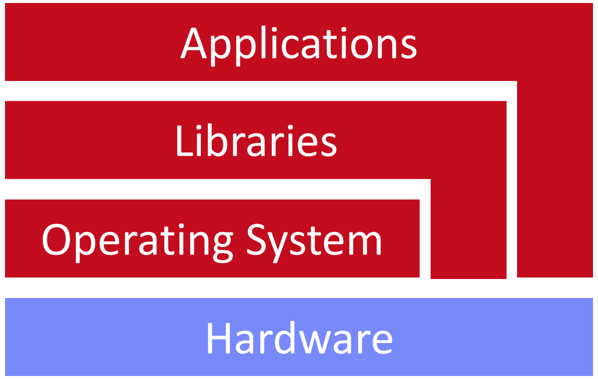


➔ Cost-effective Fault Tolerance

- **BASE** (basically **available**, soft state, **eventual consistency**)
- Effective techniques
  - ECC (error correction codes), CRC (cyclic redundancy check) for detection
  - **Resilient storage:** replication/erasure coding, checkpointing, and lineage
  - **Resilient compute:** task re-execution / speculative execution

Virtualization

- **#1 Native Virtualization**
  - Simulates most of the HW interface
  - Unmodified guest OS to run in isolation
  - **Examples:** VMWare, Parallels, AMI (HVM)
- **#2 Para Virtualization**
  - No HW interface simulation, but special API (hypercalls)
  - Requires modified quest OS to use hyper calls, trapped by hypervisor
  - **Examples:** Xen, KVM, Hyper-V, AMI (PV)
- **#3 OS-level Virtualization**
  - OS allows multiple secure virtual servers
  - Guest OS appears isolated but same as host OS
  - **Examples:** Solaris/Linux containers, Docker
- **#4 Application-level Virtualization**
  - **Examples:** Java VM (JVM), Ethereum VM (EVM), Python virtualenv



[Prashant Shenoy: Distributed and Operating Systems - Module 1: Virtualization, **UMass Amherst, 2019**]

Containerization

- **Docker Containers**
  - **Shipping container analogy**
    - Arbitrary, self-contained goods, standardized units
    - Containers reduced loading times ➔ efficient international trade
  - **#1 Self-contained package** of necessary SW and data (read-only image)
  - **#2 Lightweight virtualization** w/ shared OS and resource isolation via **cgroups**

- **Cluster Schedulers** (see **Lecture 09**)
  - Container orchestration: scheduling, deployment, and management
  - Resource negotiation with clients
  - Typical resource bundles (CPU, memory, device)
  - Examples: **Kubernetes**, **Mesos**, (**YARN**), **Amazon ECS**, **Microsoft ACS**, **Docker Swarm**

[Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, John Wilkes: Borg, Omega, and Kubernetes. **CACM 2016**]

➔ **from machine- to application-oriented scheduling**

Excursus: AWS Snowmobile (since 2016)

- **Snowmobile Service**
  - Data transfer on-premise → cloud via 100PB trucks

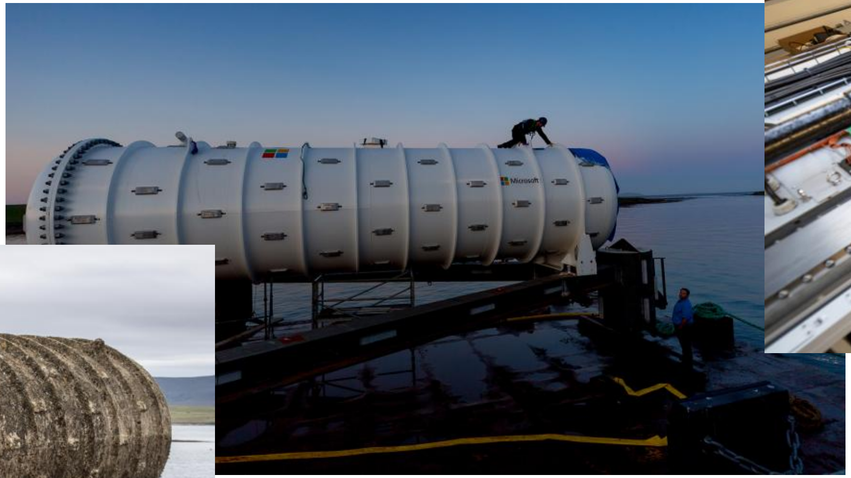
Real-World
"Containerization"
☺

100PB (1Gb Link)
~26 years → weeks

[https://aws.amazon.com/snowmobile/?nc1=h_ls]

Excursus: Microsoft Underwater Datacenter

- **Study for feasibility, and if logistically, environmentally, economically practical**


[https://news.microsoft.com/features/under-the-sea-microsoft-tests-a-datacenter-thats-quick-to-deploy-could-provide-internet-connectivity-for-years/, **06/2018**]

[https://news.microsoft.com/innovation-stories/project-natick-underwater-datacenter/, **09/2020**]

Infrastructure as a Service (IaaS)

- **Overview**
  - Resources for **compute**, **storage**, **networking** as a service
    ➔ Virtualization as key enabler (simplicity and auto-scaling)
  - **Target user:** sys admin / developer

- **Storage**
  - Amazon AWS Simple Storage Service (S3)
  - OpenStack Object Storage (Swift)
  - IBM Cloud Object Storage
  - Microsoft Azure Blob Storage

- **Compute**
  - Amazon AWS Elastic Compute Cloud (EC2)
  - Microsoft Azure Virtual Machines (VM)
  - IBM Cloud Compute

Infrastructure as a Service (IaaS), cont.

- **Example AWS Setup**
  - Create user and security credentials

```
> aws2 configure
  AWS Access Key ID [None]: XXX
  AWS Secret Access Key [None]: XXX
  Default region name [None]: eu-central-1
  Default output format [None]:
```

- **Example AWS S3 File Upload**
  - Setup and configure S3 bucket
  - WebUI or cmd for interactions

```
> aws2 s3 cp data s3://mboehm7datab/air --recursive
> aws2 s3 ls s3://mboehm7datab/air --recursive
  2019-12-05 15:26:45      20097 air/Airlines.csv
  2019-12-05 15:26:45     260784 air/Airports.csv
  2019-12-05 15:26:45       6355 air/Planes.csv
  2019-12-05 15:26:45    1001153 air/Routes.csv
```

- **Example AWS EC2 Instance Lifecycle**

```
> aws2 ec2 allocate-hosts --instance-type m4.large \
  --availability-zone eu-central-1a --quantity 2
```

Platform as a Service (PaaS)

- **Overview**
  - Provide environment setup (libraries, configuration), platforms, and services to specific applications ➔ additional charges
  - **Target user:** developer
- **Example AWS Elastic MapReduce (EMR)**
  - Environment for Apache Hadoop, MapReduce, and Spark over S3 data, incl entire eco system of tools and libraries

```
> clusterId=$(aws emr create-cluster --applications Name=Spark \
  --ec2-attributes ... --instance-type m4.large --instance-count 100 \
  --steps '[{"Args":["spark-submit","--master","yarn",'${sparkParams}'"--class", \
    "org.apache.sysds.api.DMLScript","./SystemDS.jar","-f","./test.dml"], ...]' \
  --scale-down-behavior TERMINATE_AT_INSTANCE_HOUR --region eu-central-1)
```

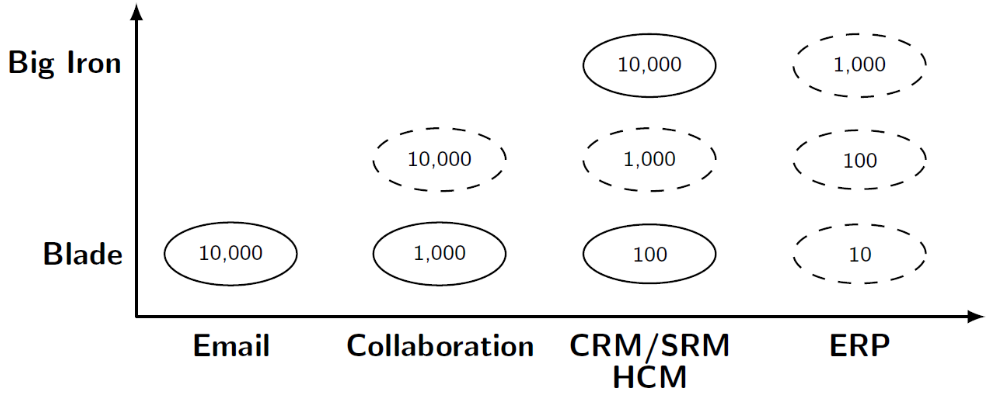
```
> aws emr wait cluster-running --cluster-id $clusterId
```

```
> aws emr wait cluster-terminated --cluster-id $clusterId
```

Software as a Service (SaaS)

- **Overview**
  - Provide application as a service, often via simple web interfaces
  - Challenges/opportunities: **multi-tenant systems** (privacy, scalability, learning)
  - **Target user:** end users
- **Examples**
  - **Email/chat services:** Google Mail (Gmail), Slack
  - **Writing and authoring services:** Microsoft Office 365, Overleaf
  - **Enterprise:** Salesforces, ERP as a service (SAP HANA Cloud)
  - **Database as a Service** (DBaaS)

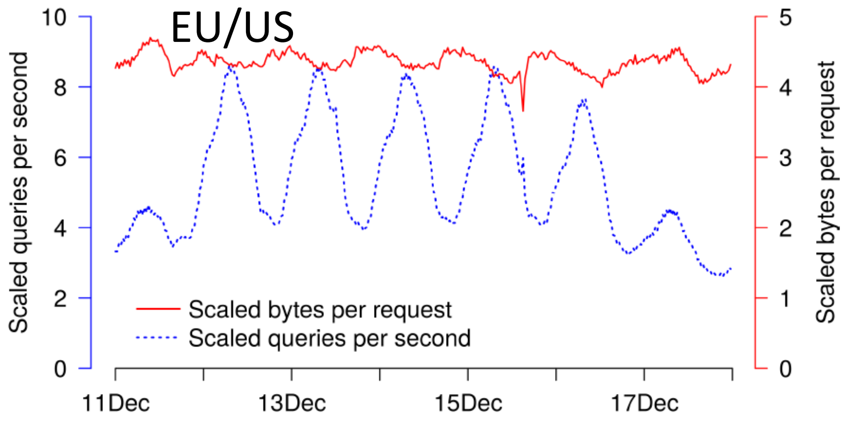
[Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, Jan Rittinger: Multi-tenant databases for software as a service: schema-mapping techniques. **SIGMOD 2008**]



Software as a Service (SaaS), cont.

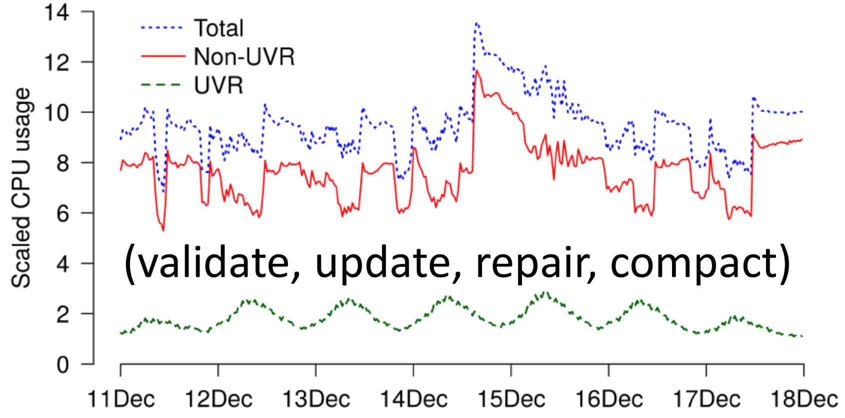
[Dan Ardelean, Amer Diwan, Chandra Erdman: Performance Analysis of Cloud Applications. **NSDI 2018**]

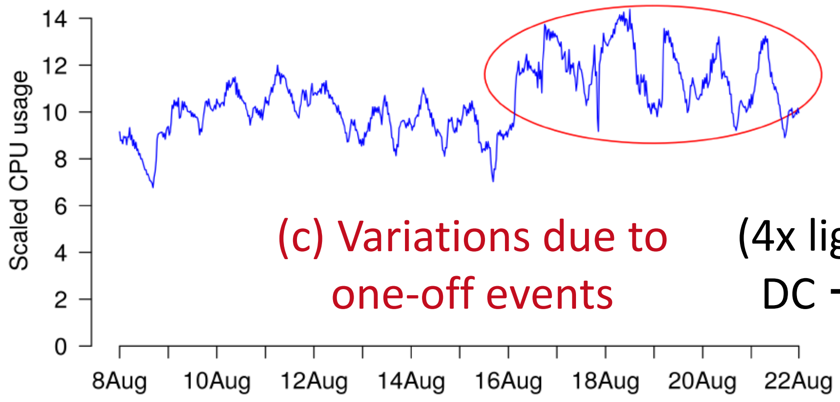
- **Performance Analysis on Gmail Data**
  - Coordinated bursty tracing via time
  - Vertical context injection into kernel logs



(a) Variations in rate and mix of user visible requests (UVR)

(b) Variations in rate and mix of essential non-UVR work





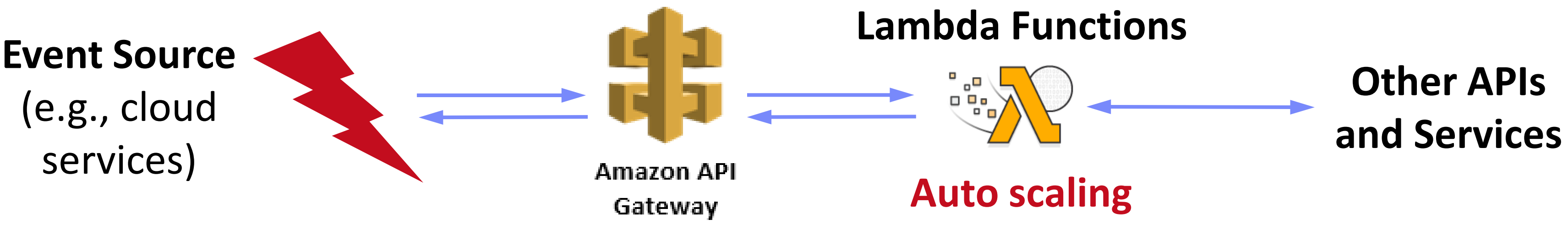
(c) Variations due to one-off events

(4x lightning Belgium DC → reconstruct)

Serverless Computing (FaaS)

[Joseph M. Hellerstein et al: Serverless Computing: One Step Forward, Two Steps Back. **CIDR 2019**]

- **Definition Serverless**
  - **FaaS:** functions-as-a-service (event-driven, stateless input-output mapping)
  - Infrastructure for deployment and auto-scaling of APIs/functions
  - Examples: **Amazon Lambda**, **Microsoft Azure Functions**, etc



Auto scaling
Pay-per-request
(1M x 100ms = 0.2$)

- **Example**

```java
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class MyHandler implements RequestHandler<Tuple, MyResponse> {
    @Override
    public MyResponse handleRequest(Tuple input, Context context) {
       return expensiveModelScoring(input); // with read-only model
    }
}
```

Serverless Computing (FaaS), cont.

[Joseph M. Hellerstein et al: Serverless Computing: One Step Forward, Two Steps Back. **CIDR 2019**]

- **Advantages** (One Step Forward)
  - **Auto-scaling** (the workload drives the allocation and deallocation of resources)
  - **Use cases: embarrassingly parallel functions**, **orchestration functions** (of proprietary auto scaling services), **function composition** (workflows)
- **Disadvantages** (Two Steps Backward)
  - **Lacks efficient data processing** (limited lifetime of state/caches, I/O bottlenecks due to lack of co-location)
  - **Hinders distributed systems development** (communication through slow storage, no specialized hardware)

→ "Taken together, these challenges seem both interesting and surmountable. [...] Whether we call the new results 'serverless computing' or something else, the future is fluid."

Func. Invoc. (1KB)	Lambda I/O (S3)	Lambda I/O (DynamoDB)	EC2 I/O (S3)	EC2 I/O (DynamoDB)	EC2 NW (0MQ)
303ms 1,045×	108ms 372×	11ms 37.9×	106ms 365×	11ms 37.9×	290μs 1×

Example AWS Pricing (current gen)

- **Amazon EC2 (Elastic Compute Cloud)**
  - IaaS offering of different node types and generations
  - **On-demand**, **reserved**, and **spot** instances

as of 12/2019

	vCores		Mem		
m4.large	2	6.5	8 GiB	EBS Only	$0.117 per Hour
m4.large	2	6.5	8 GiB	EBS Only	$0.12 per Hour
m4.xlarge	4	13	16 GiB	EBS Only	$0.24 per Hour
m4.2xlarge	8	26	32 GiB	EBS Only	$0.48 per Hour
m4.4xlarge	16	53.5	64 GiB	EBS Only	$0.96 per Hour
m4.10xlarge	40	124.5	160 GiB	EBS Only	$2.40 per Hour
m4.16xlarge	64	188	256 GiB	EBS Only	$3.84 per Hour

- **Amazon ECS (Elastic Container Service)**
  - PaaS offering for Docker containers
  - Automatic setup of Docker environment

Pricing according to EC2
(in EC2 launch mode)

- **Amazon EMR (Elastic Map Reduce)**
  - PaaS offering for Hadoop workloads
  - Automatic setup of YARN, HDFS, and frameworks like Spark
  - **Prices in addition to EC2 prices**

m4.large	$0.117 per Hour	$0.03 per Hour
m4.xlarge	$0.234 per Hour	$0.06 per Hour
m4.2xlarge	$0.468 per Hour	$0.12 per Hour
m4.4xlarge	$0.936 per Hour	$0.24 per Hour
m4.10xlarge	$2.34 per Hour	$0.27 per Hour
m4.16xlarge	$3.744 per Hour	$0.27 per Hour

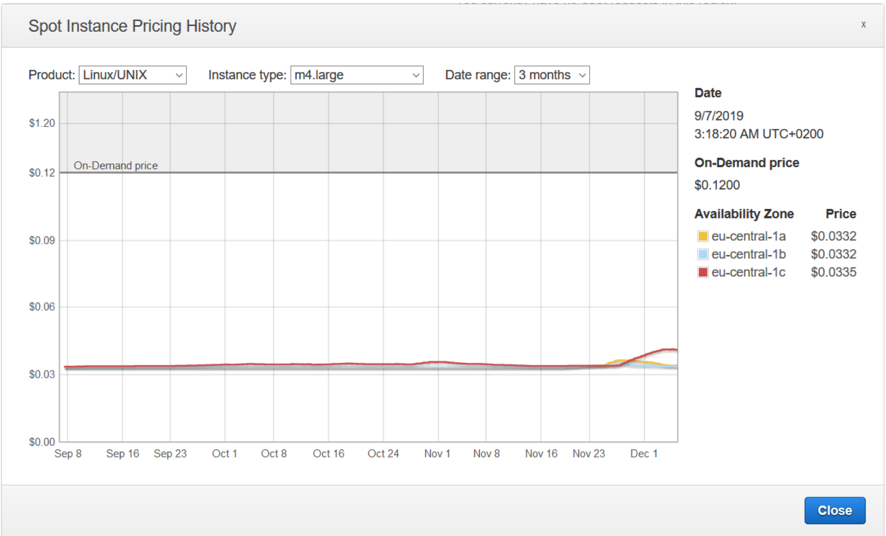
Example AWS Pricing (current gen), cont.

- **Spot Instances**
  - Unused cloud recourses for much lower prices → bidding market
  - Interruption behavior: hibernate, stop, terminate

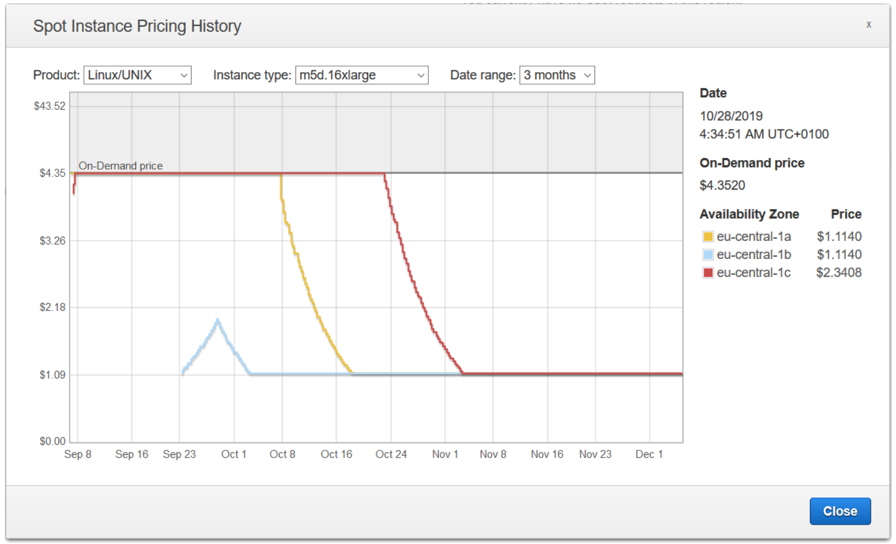
Self-regulating effect

- **Example Instance Types**

(**m4.large**, 2 vCPU, 8GB)



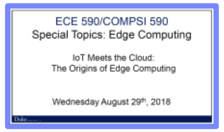
(**m5d.24xlarge**, 96 vCPU, 384GB)



[AWS EC2 Management Console, Spot Requests, **Dec 05 2019**]

Cloud, Fog, and Edge Computing

Cloud vs Fog vs Edge Overview

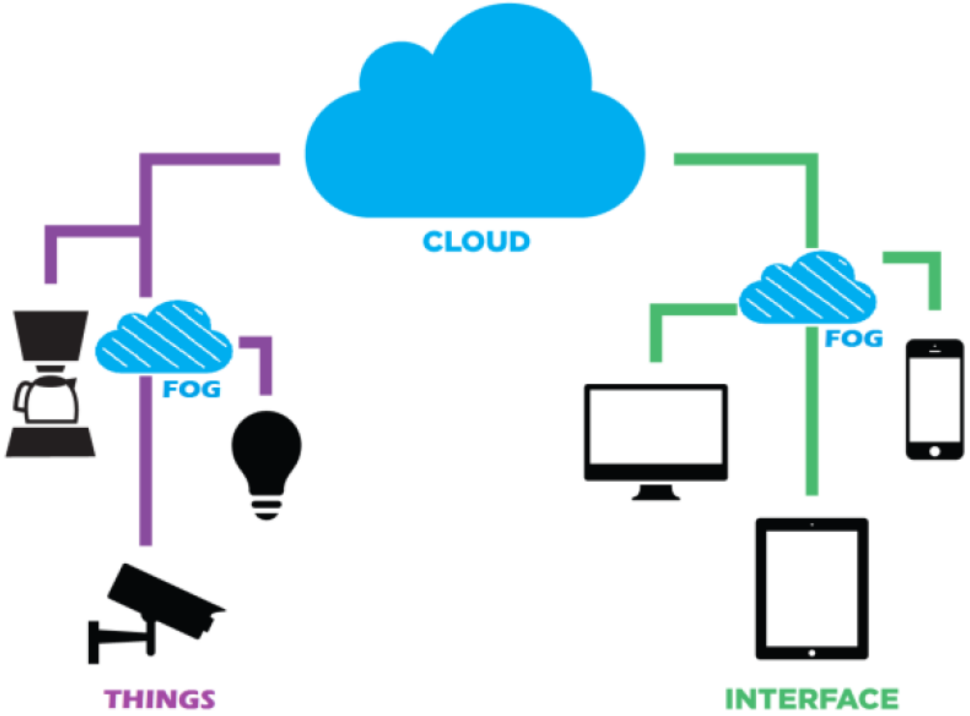


[Maria Gorlatova: Special Topics: Edge Computing; IoT Meets the Cloud – The Origins of Edge Computing, **Duke University 2018**]

- **Overview Edge Computing**
  - Huge number of mobile / IoT devices
  - Edge computing for **latency**, **bandwidth**, **privacy**

- **Fog & Edge Computing**
  - **Different degrees of application decentralization**
  - Reasons: **energy**, **performance**, **data**
  - Natural hierarchy, heterogeneity
  - Cloud as enabler for vibrant web ecosystem

  → **fog/edge for IoT** the same?



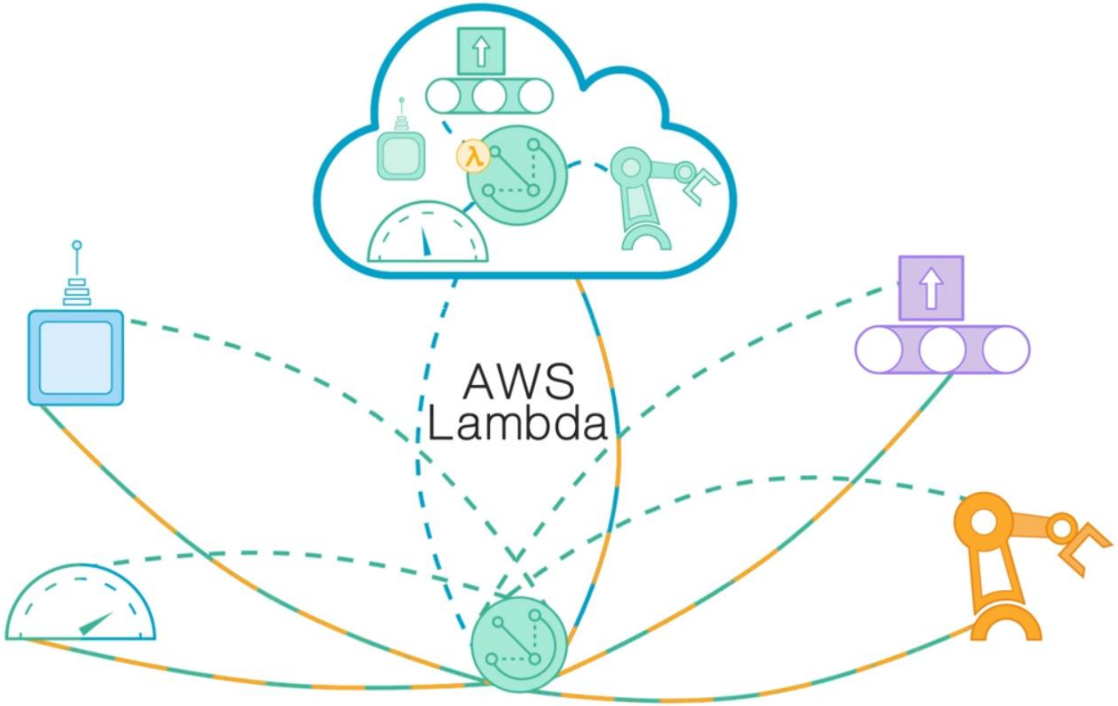
NebulaStream
(TU Berlin & BIFOLD FG DIMA)
[https://nebula.stream]

Example: AWS Greengrass

[**Credit:** https://aws.amazon.com/greengrass/?nc1=h_ls]

- **Overview AWS Greengrass**
  - Combine **cloud computing and groups of IoT devices**
  - Cloud configuration, group cores, connected devices to groups
  - Run lambda functions (FaaS) in cloud, fog, and edge – partial autonomy
- **System Architecture**
  - Central configuration and deployment
  - Decentralized operation

Customer Use cases:
"My data doesn't reach the cloud"



Excursus: Decentralized Infrastructure

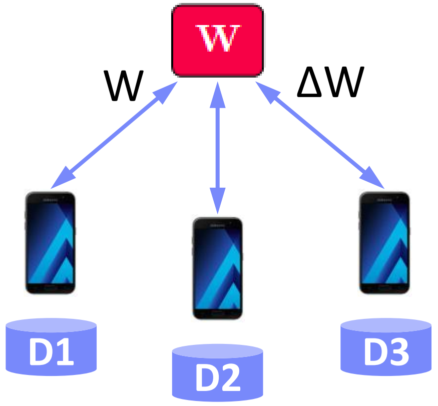
- **Public/Private Infrastructure Projects**
  - **Hierarchy of endpoints/data centers**
  - Analogy: **“City-Planning”**

[**Credit:** University of Maribor]

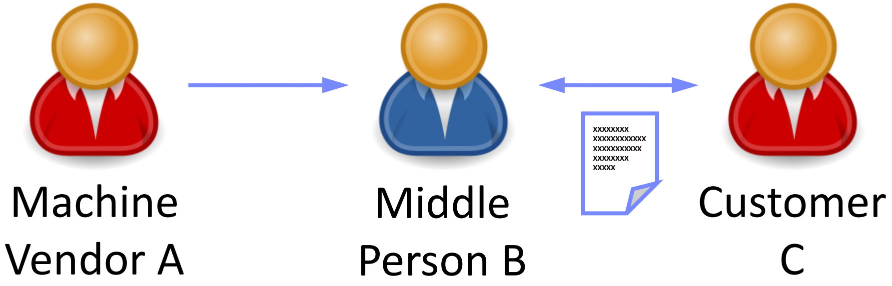
Federated Machine Learning

[Keith Bonawitz et al.: Towards Federated Learning at Scale: System Design. **SysML 2019**]

- **Overview Federated ML**
  - Learn model **w/o central data consolidation**
  - **Privacy** vs **personalization and sharing** (example: voice recognition)
  - Adaptation of parameter server architecture, w/ random client sampling and **distributed agg**
  - Training when phone idle, charging, **and on WiFi**



- **Example Data Ownership**
  - **Thought experiment:** B uses machine from A to test C's equipment.
  - Who owns the data?
    Negotiated in bilateral contracts



  - **Spectrum of Data Ownership:** Federated learning might create **new markets**

Spectrum of Data Sharing

- **Fine-grained Spectrum**
  - Spectrum of technologies with **performance/privacy/utility** tradeoffs
  - Different applications with different requirements → **Potential for new markets**

Increasing Privacy ← → **Increasing Utility/Perf**

Private Data (no sharing)	**Privacy-enhancing Technologies** (FHE, MPC, Differential Privacy)	**Aggregates** (Federated ML, Federated w/ secure comm.)	**Anonymized Data** (k-anonymity, pseudonyms)	**Surrogate Data** (char-preserving synthetic data, dataset distillation)	**Public Data** (full sharing, sharing w/ partners)

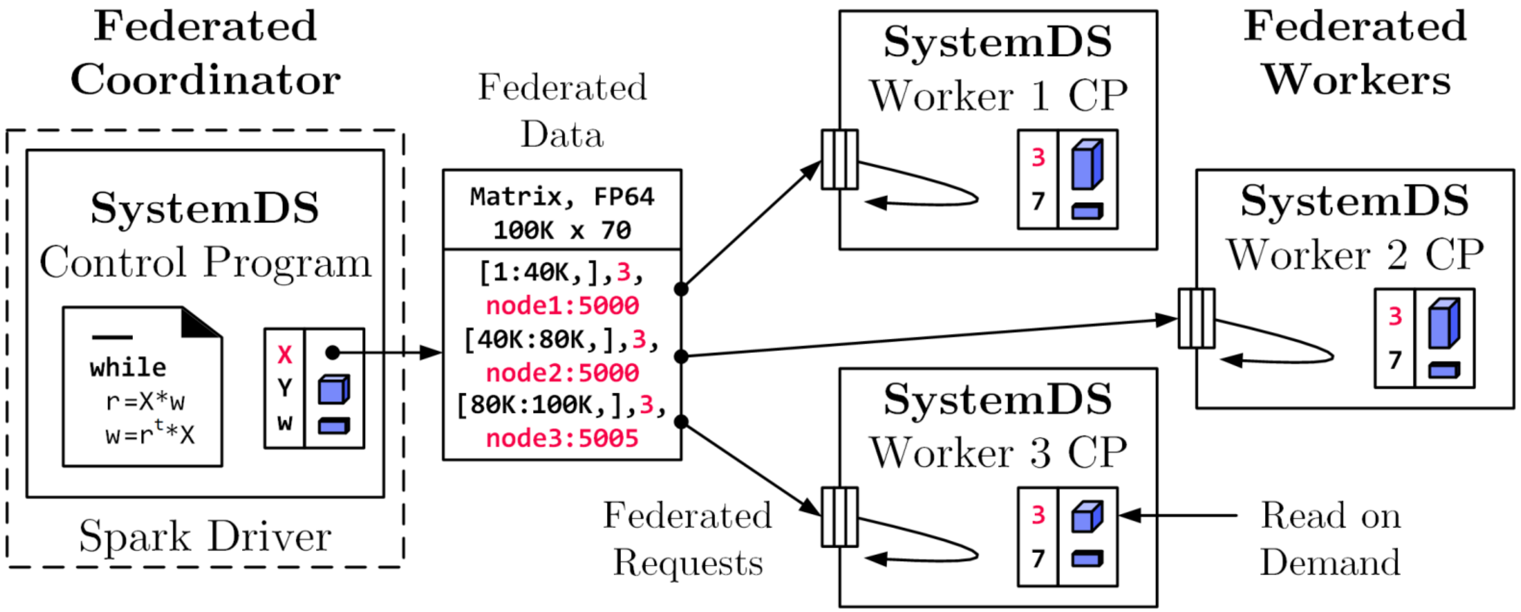
Key Property: no reconstruction of private raw data

Federated Learning in SystemDS

[SIGMOD'21, CIKM'22]

- **Federated Backend**
  - **Federated data** (matrices/frames) as meta data objects
  - **Federated linear algebra**, (and **federated parameter server**)

```
X = federated(addresses=list(node1, node2, node3),
  ranges=list(list(0,0), list(40K,70), ..., list(80K,0), list(100K,70)));
```



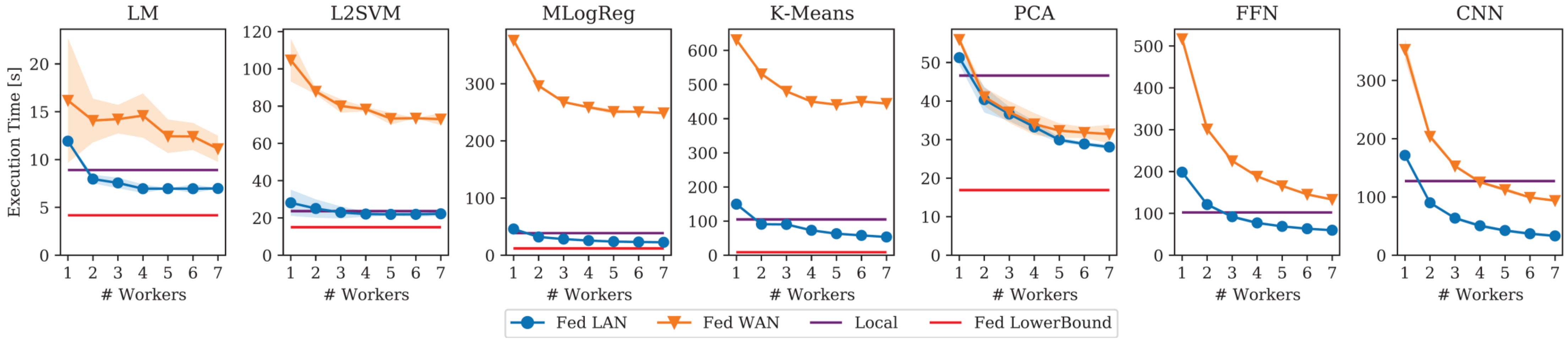
Federated Requests:
`READ, PUT, GET, EXEC_INST, EXEC_UDF, CLEAR`

➔ **Design Simplicity:**
(1) reuse instructions
(2) federation hierarchies

Federated Learning in SystemDS – Experiments

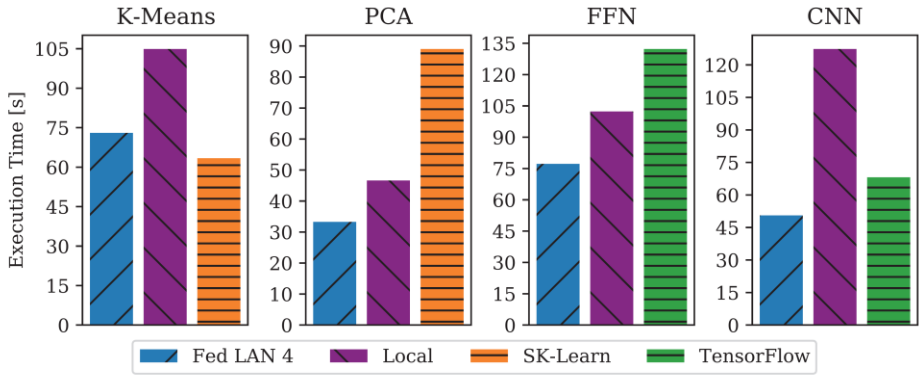
Reproducible Results

OPEN ACCESS



- **Workloads and Baselines**
  - **LM:** linear regression, lmCG
  - **L2SVM:** l2-regularized SVM
  - **MLogReg:** multinomial logreg
  - **K-Means:** Lloyd's alg. w/ K-Means++ init
  - **PCA:** principal component analysis
  - **FFN:** fully-connected feed-forward NN
  - **CNN:** convolutional NN

Comparisons w/ **Scikit-learn** and **TensorFlow**



Summary and Q&A

- **Cloud Computing Motivation and Terminology**
- **Cloud Computing Service Models**
- **Cloud, Fog, and Edge Computing**

- **Next Lectures (Large-scale Data Management and Analysis)**
  - **09 Cloud Resource Management and Scheduling** [Dec 11]
  - **10 Distributed Data Storage** [Dec 18]
  - **Holidays**
  - **11 Distributed, Data-Parallel Computation** [Jan 15]
  - **12 Distributed Stream Processing** [Jan 22]
  - **13 Distributed Machine Learning Systems** [Jan 29]