Advanced Internet Computing WS 2022

01 Introduction

Complexity, Interaction, Autonomy

- Heterogenous systems increasingly connected (Integration becomes more complex)
- Software- and Hardware-Architects cannot plan for all potential interactions upfront (increased interaction dynamics of systems and processes, people and software services)
- **Monitoring and Management of internet-scale** infrastructures becomes most important thing
- Autonomic & Services Computing includes
    - Self-Healing
    - Self-Configuring
    - Self-Optimizing
    - Self-Protecting
    - *Self- properties**

What it wrong with Internet Systems?

- Too difficult to build, deploy and maintain
- Need to address
    - Complexity
    - Interoperability
    - Trustworthiness
    - Robustness
    - Ease of use
    - Ease of maintenance
- In the future **infrastructure principles need to apply on higher levels of the software stack** (before: applied on systems level)

Software Evolution

- Requirements cannot be fully gathered upfront or be frozen
- Too many stake-holders started arising
- Requirements decentralized, complete control and pre-plan not possible
- When changed, impact whole product/process/service
- Evolution is intrinsic to software

Open-world assumption

- Services become key actors (SaaS, loosely coupled, accessed on demand)
- Services are owned by other people

Evolutions

- **Teamwork evolution**: from long-lived and stable to short-lived and flexible
- **Infrastructure/Software/Processes/Teamwork evolutions**

- Dependencies between parts of systems are no longer fixed and predetermined
    - Ability to deal with context changes and unanticipated events and people (self-* behaviors)
    - IoT: Support active objects providing service (Taggable objects, artifacts, sensor compositions/networks)

Ecosystems

A complex system with networked dependencies and intrinsic adaptive behavior has:

- Robustness & Resilience mechanisms: Achieving stability in the presence of disruption
- Measure of health: Diversity, population trends
- Built-in coherence
- Entropy-resistance (syntropy)

The co-evolution of science & technologies lead to people, things and systems being ecosystems

- Ecosystems = Architecture, structure + dynamics
- **Everything must be seen as a ecosystem**

New paradigms

- **Elasticity** (Resilience)
    - Acquire new resources, reduce quality at stress
    - Release resources, increase quality when stress is removed
    - Elasticity > Scalability (Resource elasticity, quality elasticity, costs & benefit elasticity)
- **Osmotic Computing**
    - Dynamic management of (micro)services across cloud and edge datacenters
- **Social compute units (SCUs)**
    - Integrate people, in the form of human-based computing, and software services into one composite system

Service-oriented computing (SoC)

Visions and principles

- Programmatic interaction between autonomous systems
- Web services: self-contained entities on the internet (loose coupling of systems)
- Virtualization of resources, utilization and consolidation of internet-based infrastructure
- Agile development through service composition
- Services can be consumed everywhere (servers, mobile, ...)

What is a service?

- Standardized interface
- Self-contained with no dependencies to other services
- Available
- Coarse-grained
- Context-independent
- Allows for service composition
- Quality of service (QoS) attributes which can be measured

Types of web services

- **Informational services**: Simple request/response sequences
- **Complex services**: Assembly and invocation of many pre-existing services (typically stateful)

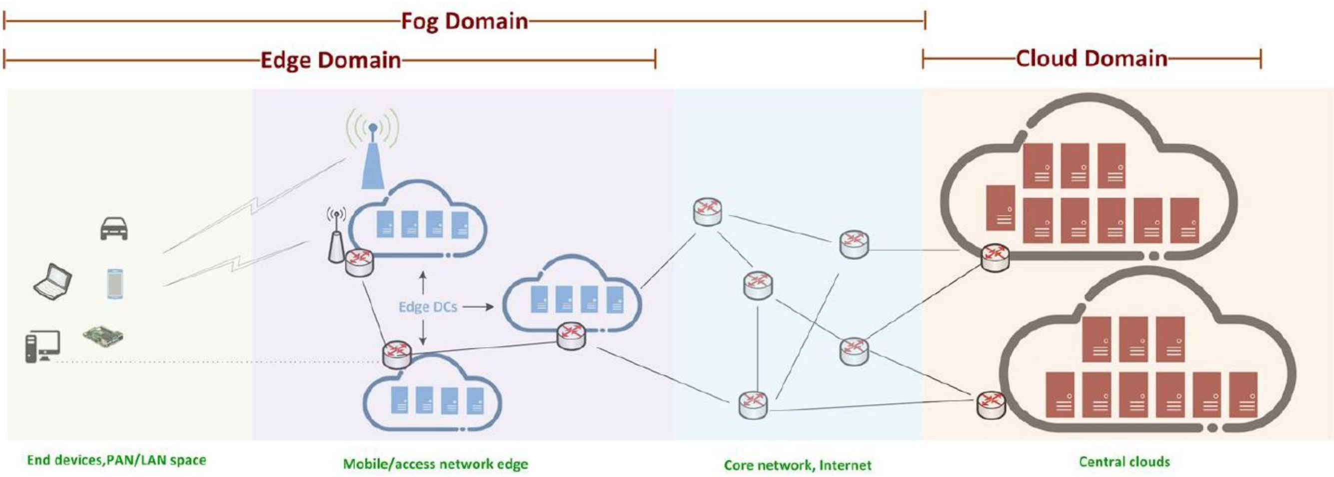
Services properties, states and principles

- **Functional properties**: Characteristics that define the overall behavior
- **Non-functional properties**: Quality attributes (cost, performance, ...)
- **Stateless services**: Can be invoked repeatedly without having to maintain context or state
- **Stateful services**: Context must be preserved from one invocation to the next
- **Loose coupling**: No need to know how partner application behaves or is implemented
- **Service granularity**
  - Simple services are atomic (request/response)
  - Complex services are coarse-grained (Interactions with other services and end users, multiple sessions, larger and richer data structures)
- **Synchronicity**
  - Synchronous / Remote procedure call: Request with a set of arguments, response with return value
  - Asynchronous / Message (document)-style: Send entire document
- **Well-definedness**: Service interaction must be well defined, rules for interfacing and interacting
  - Service interface: Defines functionality
  - Service implementation: Realizes specific service interface (e.g. by using a programming language of choice)

02/03 Edge Intelligence in the Distributed Compute Continuum

Current State

- Distributed systems are key to our society
- Base of our critical infrastructure and applications (smart cities, healthcare, autonomous vehicles)
- Interconnectedness of components (HW, SW, people) induces complexity



The distributed compute continuum is a ecosystem that needs to be programmed.

There are new families of applications that require:

- Real-time location-based access to data from the environment

- Appropriate compute and storage resources in close proximity

Edge intelligence is a **post-cloud computing** paradigm at the intersection of human augmentation, IoT and AI.

The cartesian blanket

- System control based Service Level Objectives (SLOs)
- SLOs are represented as thresholds on the cartesian space (resources, cost and quality)
- Minimum and maximum for every SLO
- The cartesian space is constraint to the infrastructure characteristics (represented as points)
- Possible configurations can be visualized as a stretched blanket over the points
- Helps to **determine if a service unit or service is in the "elasticity behavior"**

Fabric for Edge Intelligence

- Sensing (Sensor Data as a Service)
- Edge computer network with modular AI capabilities
- Intelligent orchestration mechanisms for decentralized and distributed infrastructure

AI for Edge

Topics:

- Topology (Edge orchestration)
- Content (Lightweight service frameworks)
- Service

Challenges:

- Model Establishment
- Algorithm Development
- Trade-off between optimality and efficiency

AI on Edge

Challenges:

- Data Availability (Bias on different devices)
- Model Selection (How can infrastructure itself change)
- Coordination Mechanisms

04 Cloud Computing

Motivation

Operating own IT on-premise

- High upfront investment
- High maintenance cost
- More or less fixed resources

- Stepwise (discrete) scaling

Using IT capacity from the Cloud

- Pay per use
- Lower maintenance cost
- Linear (continuous) scaling
- Fault-tolerant

Use cases for Cloud Computing

- Demand for a service varies with time (peak loads)
- Demand is unknown in advance (new startup)
- Batch workloads

Three Cloud Service Models

- Cloud Infrastructure as a Service (**IaaS**)
  - Delivers computer infrastructure (VMs, storage, ...)
  - Amazon EC2, Amazon EBS
- Cloud Platform as a Service (**PaaS**)
  - Deliver computing platform and solution stack (execution env/framework)
  - Google App Engine, Heroku
- Cloud Software as a Service (**SaaS**)
  - Google Docs

A cloud system may provide any or all of the service modes to the customers through external APIs.

IaaS vs. PaaS vs. SaaS

- **Speed vs. Customization**
  - SaaS is already there to use, but cannot always be customized
  - PaaS applications can be developed faster than IaaS (do not care about load balancing and networking), but IaaS offers highest customization possibilities
- **Cost**
  - PaaS can be cheaper to run, because optimized for efficiency
- **Vendor Lock-In**
  - SaaS and PaaS worse than IaaS

Cloud Development Models

- **Public cloud**: Open to general public, owned by organization selling cloud services
- **Private cloud**: Operated solely for one single organization
- **Community cloud**: Shared by several organizations
- **Hybrid cloud**: Composition of two or more cloud deployment models (private, community, public)

Advantages of public clouds:

- Lower costs (pay per use, no upfront costs)
- No maintenance

- High scalability
- High reliability

Advantages of private clouds:

- Self-reliance
- Flexibility (Custom environments)
- Security (Resources kept on-premise)
- Compliance (National/Local regulations)

Advantages of a hybrid clouds:

- Control: Private infrastructure for sensitive assets
- Flexibility: Take advantage of additional resources in the public cloud
- Cost-effectiveness: Pay only when extra power needed, save by using own infrastructure

Virtualization

- Fundamental technical and conceptual **enabler of cloud computing**
- Basic idea: **Abstract view on resources**
    - Platform wide
    - Memory
    - Storage
    - Network
- Higher degree of capacity utilization
    - Resources are shared between users
    - Backend parallelization
- Many different applications onto VMs in same data center (energy consumption and space usage lower)
- Fault tolerance: Save VM state (and take it somewhere else)
- **Hardware level virtualization** (VMs simulate machine)
- **Operating system virtualization** (OS kernel manages coexistence of multiple isolated user spaces)
    - Containers share OS and drivers
    - Small overhead, nearly native performance
    - Not as secure as VMs
    - More elastic than hypervisors
- OS-Level Virtualization: **Docker**
    - Leading containerization technology
    - Cross-platform portability for developers
    - Benefits: Deployment speed/agility, portability, reuse, manged via code

Cloud Quality of Service (QoS)

- **Measures** for (technical) quality of a service
    - Performance
    - Availability
    - Failure rate
    - Security
    - Trust

    - Compliance
    - Costs
- **Instance-level performance-related QoS metrics**
    - Round-trip time and response time (Time from issuing to receiving)
    - Network latency (Time that message spends in transport medium)
    - Processing time (Time needed to execute requested operation)
    - Wrapping time (Time that service or client need to pack/unpack messages)
    - Execution time (Processing time + wrapping time)
- **Aggregated QoS metrics**
    - Throughput (number of requests processed in a timeframe) == maximum processing rate
    - Availability (Probability that a given service is operative)
    - Redundancy as a rescue for bad availability

SLAs in the cloud

- Service Level Agreements (SLAs) are specific agreements between client and service on the QoS terms
- SLAs are typically mostly relevant for B2B (business-to-business) interactions
- SLAs contain
    - Service Level Objectives (SLOs)
    - Metrics and metric definitions
    - Concrete target values
    - Penalities for non-achievement
    - Validity period
    - Responsible monitoring entity

05 IoT-Cloud continuum

What is the Internet of Things?

- Sensing
    - What sensors are needed?
    - How to identify different users?
- Communication
    - What protocol to use?
    - How to communicate to cloud applications?
- Processing
    - How does decision making occur?
- Behavior
    - Events triggered, state changes
    - What happens when there are multiple devices, complex systems?
- Actuation
    - Privacy and ethics?
    - Medium and channel?

Related domains and terminology

- **Embedded systems**: Focus on system, not necessarily connected
- **Sensor networks**: Focus on sensor devices connected through wireless channels

- **Cyber-physical systems**: Focus on interaction between physical and cyber systems
- **Distributed systems**: Focus on time constraints
- **Pervasive computing**: Focus on anytime/anywhere computing

IoT use cases

- Industry 4.0, smart manufacturing
- Intelligent transportation systems, connected vehicles
- Smart homes and cities
- Logistics
- Environmental monitoring
- Healthcare

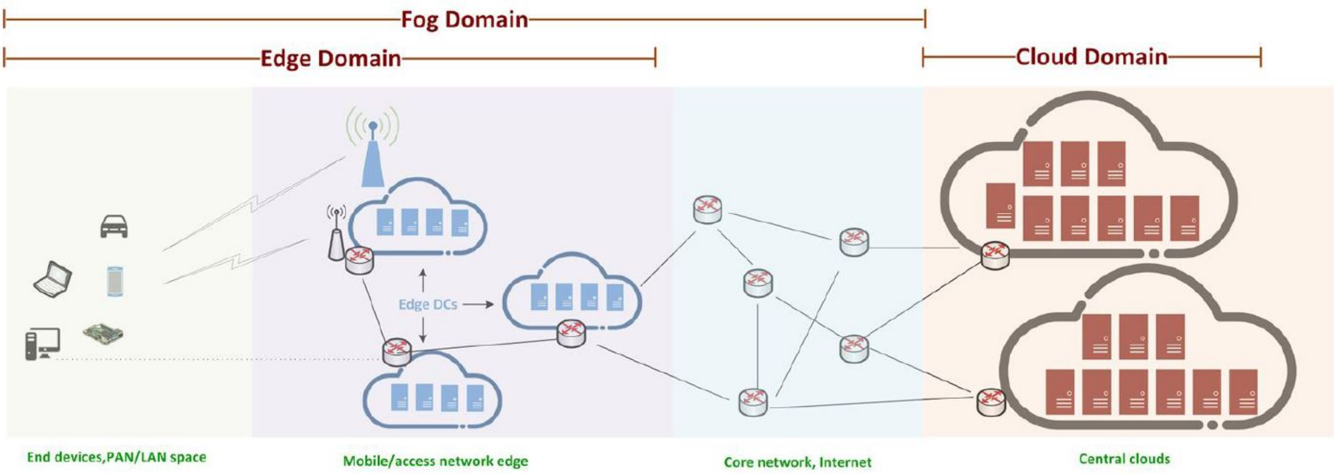
Device to Cloud Continuum

IoT-Cloud Architectural Spectrum

- Widely different applications within several domains
- Heterogeneous and highly complex systems
- Many standards, interoperability can be problematic

Architectural questions to keep in mind

- What do specific applications dictate?
- Where should reasoning or decision-making occur?
- What networking aspect is employed and what is its effect?
- How much does it cost, and how can it scale up?



- **End devices, PAN/LAN space** (part of fog and edge)
  - Low reliability
  - Volatility
  - Mobility
  - Mostly wireless connectivity
  - Small form factors
  - Battery constraints
  - Mobile, IoT, smart home, vehicles

    - **User/Service provider controlled**
- **Mobile/access network edge** (part of fog and edge)
    - Edge of the (mobile) network
    - Low latency to end device
    - Close to/collocated with 4G/5G base stations
    - General purpose compute infrastructure
    - Standards-based architecture & management/orchestration stacks
    - **Telecom operator controlled**
- **Core network, Internet** (part of fog)
    - Network Neutrality (NN) ?
    - "All traffic on the Internet must be treated equally."
- **Central clouds** (cloud domain)
    - "Unlimited" compute/storage resources
    - Full spectrum of cloud services
    - High availability
    - Lower cost
    - Higher latency vs. edge/fog
    - **Cloud provider controlled**

Networking technologies for the IoT

- **Wireless Local/Personal Area Networks**
    - Access to the internet over local gateway and wired connection
    - Wi-Fi, Bluetooth, ZigBee
    - High throughput, lower coverage, local gateway
- **Wide Area Networks**
    - Direct access to "remote" gateway (to the internet)
    - 4G LTE, 5G
    - Low Power Wide Area Networks: LoRa, SigFox, LTE-M, NB-IoT

Low Power Wide Area Networking

- Event driven or periodic data transmission
    - Typically infrequently, small packets
    - Low bandwidth requirements
- Very large number of devices (Networking hardware should be cheap)
- May need long-range wireless connectivity (Bluetooth, ZigBee not enough)
- Should operate at very low power (4G more power-hungry)

LoRaWAN or NB-IoT?

- **LoRaWAN** allows full end-to-end control of the network
    - Free spectrum, zero fees possible (managerial overhead)
    - Multiple service models available: full e2e solution, devices + gateways only, community gateways, community/commercial network server stack
- **NB-IoT** uses the mobile network operator's infrastructure
    - Network management overheads at IoT service provider
    - Like 4G/5G (SIM card)

    - Brings IP endpoints directly to the device (no gateway, direct to cloud server)
    - A bit more expensive device hardware

IoT software stacks

- **Device**: Interacts with the environment, generates data
- **Gateway**: First level of aggregation and data processing, device connectivity towards the Internet/Cloud
- **IoT cloud platform**: Central data processing/analysis, storage, end-to-end control, provision of services to other apps

Key design principles for IoT stacks

- **Loose coupling** (Each stack can be used independently of the other stacks)
- **Modularity** (Non dependence on hardware (if possible) and cloud provider)
- **Open standards** (Ensures interoperability)
- **Well-defined APIs** (eases integration)

Microservice-based design

- Fine-grained refinement of service-oriented architecture (SOA)
- Not a technology nor standard, but a set of application design principles, resulting in software architectures with **many independent components, consumed via services**

Problems with complex but monolithic web applications:

- No individual developer understands the entire application
- Difficult to scale (due to size or different resource requirements)
- Too big/complex for frequent re-deployments, DevOps challenging
- Tight coupling (low module isolation), errors propagate more easily
- Difficult to change implementation technology (language/framework)

Microservices

- Typically a large-scale microservice system can include hundreds to thousands of microservices (Netflix 500+ microservices, two billion API requests)
- Independently deployable small components (mostly as containers)
    - Each component communicates through well-defined APIs using lightweight mechanisms
- A microservice should perform a small, business-oriented functionality
- Most microservices are not externally exposed
- Developed by a small team
    - Arbitrary programming language, framework
    - Independent data backend (state is scoped to the microservice)
    - Independently versioned, deployed an scaled

Microservices: Advantages

- Small understandable functionality units focus on business needs
- Lightweight, quick to deploy and scale

- Good fault isolation
- Different technologies used in parallel, changed at will
- Possibly independent teams
- Easy continuous integration/continuous delivery (CI/CD) - DevOps

Microservices: Drawbacks

- Not the solution to every problem
- Highly distributed
  - A lot of Inter-service communication/coordination
  - Long invocation chains, increased latency
  - Integration testing becomes complex
  - Overload control, fault management challenging
- Individual data backends (duplication of data)
- Automating deployment requires tooling support
- Testing many components is complex and requires deploying many services
- **(Re-)Designing an application into a microservice architecture is not trivial!**

Taking microservices to the extreme

- Enter **Serverless Computing**
- Known as "Function as a Service" (FaaS) cloud model
  - AWS Lambda, Google Cloud Functions, Microsoft Azure Functions
- Trend: manage hardware -> manage VMs -> manage containers -> manage code/logic
- Serverless does not mean that servers/machines/containers are not deployed to execute your code, but the deployment is handled by the cloud provider transparently

How does serverless computing work?

- Provide the piece of **business logic** you want executed as a function and upload it to the cloud provider
- Define **mappings for inputs and outputs** (from/to other cloud services)
- Define **metadata**
  - Triggers (logical condition when the function should be executed)
  - Runtime parameters (timeouts, memory requirements)
  - Failure strategies (retry, fail)
- Functions can be simpler and cheaper if the use case is right
- When a trigger fires, the function gets executed
- Scaling is per-invocation, no wasted virtual resources for the user
- Pricing per use, can be cheaper or more expensive than VMs. Depends on:
  - Number of invocations
  - Amount of RAM the invocations use, and how long the memory is kept occupied
  - Additional charges from invocation of other cloud services (storage, DB, queue)

The cold start problem

- First time a function is called, a container needs to be launched (takes a while)
- Response time: ms - sec

- Large tail latencies
- Inappropriate for low latency systems
- More critical at the edge, with "slower" infrastructure

Solution strategies (active research topic):

- Keep functions warm (exploits predictions on what will be requested, costs resources)
- Make cold starting cheap (replacing docker with a lower-overhead runtime)

06 Fog & Edge Computing

- Typically cloud designs are centralized
- Collect data at servers in large, remote data centers for centralized processing
- Emergent IoT applications: Loads of data generated at the edge
- Increasing amount of computation resources "close" to the source of data

Edge & Fog

Situating **computation closer to end-devices**

- Computation includes data processing, compression, decision making
- Emerging applications range from autonomous vehicles, augmented reality to smart systems
- Low latency, decentralization, less signalling and communication overhead

Edge vs. Fog

- **Fog computing** has a wider scope
  - Deeply hierarchical, multi-layer architecture
  - Computation anywhere among collaborating entities all along the IoT cloud continuum
- **Edge computing** typically spans mostly up to the edge of the operator's network
  - Computation on-device or offloaded one hop away (other local device, edge server collocated with WiFi AP, gateway, LTE base station)
- **Multi-access edge computing (MEC)**: More narrow (but more clearly defined) case of edge computing
  - Architecture and interfaces standardized
  - Tailored to a network operator-controlled edge infrastructure
  - Consumer-oriented services (multimedia, gaming, AR/VR)
  - Operator and 3rd party-oriented services (radio network troubleshooting, video analytics, connected vehicles)
  - Critical enabler for 5G (enables ultra reliable and low latency communication (URLLC) and enhanced mobile BroadBand (eMBB) services)
  - Important difference with traditional clouds: end-user context available to edge apps

Typical workflow of a Multi-access edge computing application

- Application provider **prepares** the application (Needs to package it appropriately incl. standard-format "manifest")
- **Onboards** the application package to MEC system (Available over REST API of MEC orchestrator)
- Application provider or 3rd party **instantiates** the application

- The application instance **subscribes** to MEC services it wishes to consume
- If app has requested specific user traffic to be **offloaded** to it, MEP applies the appropriate traffic steering rules

Security and the IoT-Cloud Continuum

- **Confidentiality** (information should be kept secret)
- **Integrity** (information should not be modified in transit / at rest)
- **Non-repudiation** (information and actions cannot be denied, once performed e.g. payments)
- **Availability** (service or data are provided when needed, information originates from an identified source)
- **Security as a cross-cutting concern**

Security within the IoT-Cloud Continuum

- Heterogenous devices, loosely coupled software components, devices appear and disappear
- Software developed and maintained by different teams and communities which may originate in other engineering disciplines
    - Fragile full stack solutions
    - Wide attack surface
- Devices may belong in different administrative domains or legal jurisdictions (think privacy)
- Operation in unknown or untrusted environments - typically at the edge
- Malicious actors have advantage
    - Automated network attack tools, massive IoT numbers, out-of-date firmware/software
    - Exploits for most segments of IoT stack
    - Physical access and hardware tampering

Edge intelligence (intelligence at the edge)

- Massive amounts of data generated at the edge in IoT settings
- Valuable input for Machine Learning tasks
- Traditionally: centralized
- Challenges
    - Performance (vast amount of data to be communicated)
    - Privacy (sensitive information is generated)
    - New developments
        - Increased computational power and storage space on devices
        - Modern smartphones feature AI chips
        - Runtime environments on small form-factor devices
        - To take advantage of distributed resources for AI tasks
        - GDPR and "data minimization"

Decentralizing training and inference

Federated learning

- On-device training
- Sending outcome to server
- Server aggregates and updates global model

- Model available for devices

Current research challenges in federated learning

- **Device recruitment strategies**: Which subset of the devices to assign a learning task at any given round? (Consider processing, storage, battery, trustworthiness, data quality)
- **Volatility**: Devices can disappear or join at any time
- **Asynchrony**: Algorithms face challenges when end devices do not submit their data in a timely manner
- **Non independent and identically distributed data**: inaccuracies, personalization loss
- **Heterogeneity in the volume of training data per device**: A device that contributes a lot may lead to a biased model
- **Preventing privacy leaks**: Some private information may be inferred even if devices do not transmit the actual data
- **Incentives to misbehave**: Why waste battery when I can let the others do all the work?

Distributed deep neural networks

- Inference anywhere in the device-to-cloud continuum
- If a confident predication can take place already in the first layer, no need to propagate to the cloud
- Less communication, faster response
- New challenges: How to optimally place layers on host, how to provide the system support for dynamic management, resource allocation, migration of layers, model redeployment