Università di Pisa

Pesaresi Seminar Series

Beyond the Cloud

Exploring Serverless Computing and Cloud Continuum

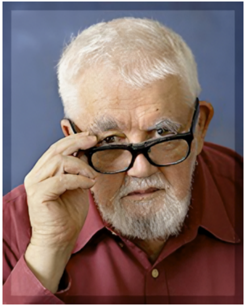
PART ONE

Valerio Besozzi

March 1, 2024

Introduction

A Little Historical Note on Cloud



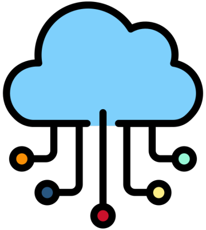
Computing may someday be organized as a public utility just as the telephone system is a public utility. Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system ... Certain subscribers might offer service to other subscribers.

(Professor John McCarthy, 1961)

A Little Historical Note on Cloud (cont'd)

From that speech, our story begins, arriving today at the modern concept of the cloud as we know it. Some of the most important moment in cloud evolution are:

- **1961** - Prof. J. McCarthy's speech for MIT's centennial celebration.
- **1968** - IBM launches CP-40/CMS, introducing *virtualization*.
- **1969** - ARPANET is launched.
- **1991** - World Wide Web opens to the public.
- **1997** - Ramnath K. Chellappa coins the term *"Cloud Computing"*.
- **1998** - Ian Foster et al. formalize the concept of *"Grid Computing"* [18].
- **1999** - VMWare introduces VMWare Workstation.
- **2006** - Amazon launches AWS.
- **2011** - NIST Cloud Computing Reference Architecture [32].



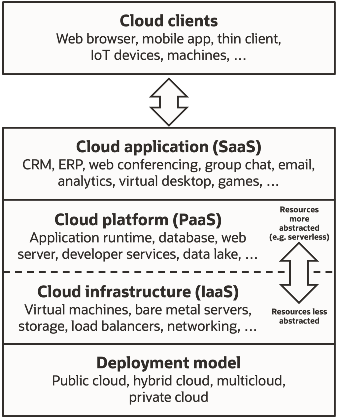
Introduction

Cloud Computing

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.

Three main categories of cloud computing service models [32] [8]:

- **Infrastructure as a Service (IaaS)**
- **Platform as a Service (PaaS)**
- **Software as a Service (SaaS)**



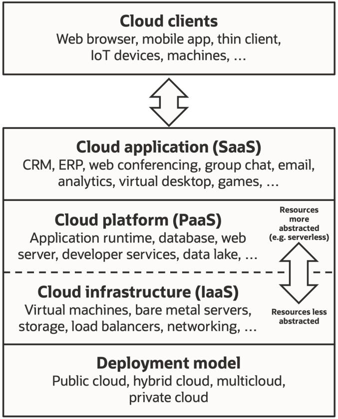
Introduction

Cloud Computing

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.

Three main categories of cloud computing service models [32] [8]:

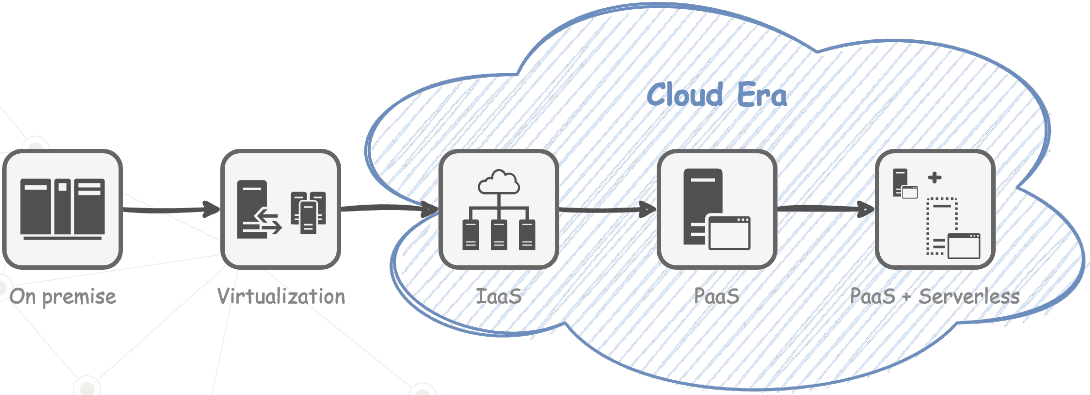
- **Infrastructure as a Service (IaaS)**
- **Platform as a Service (PaaS)**
- **Software as a Service (SaaS)**



Moving toward **Everything as a Service** (or **XaaS**).

Introduction: The Rise of Serverless

Serverless computing represents a significant evolution in cloud computing. It is a disruptive technology as it completely eliminates the need for managing infrastructure and back-end.

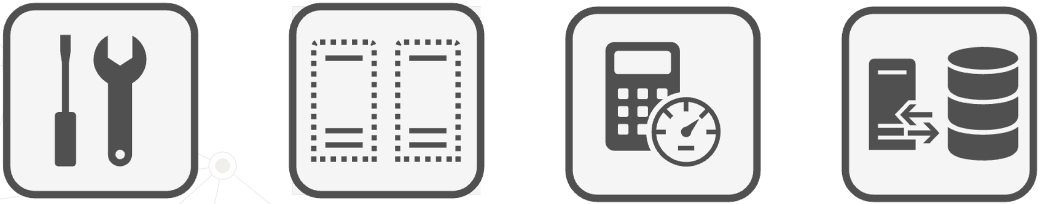


Serverless Computing

INTRODUCTION ON SERVERLESS

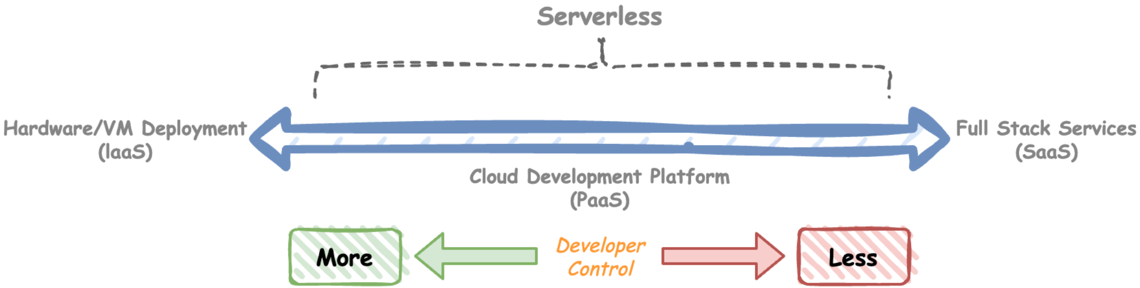
Serverless

Serverless computing is a form of cloud computing that allows users to run event-driven and granularly billed applications, without having to address the operational logic [16].



Introduction on Serverless (cont'd)

Depending on the different level of control offered to developers, a *serverless service* could fall into different levels of the NISC cloud reference model.



Serverless Service Models: Not Only FaaS...

Serverless Service Models

Function-as-a-Service (FaaS)

It is a serverless service model in which the cloud provider manages the resources, life-cycle, and event-driven execution of user-provided functions.

Backend-as-a-Service (BaaS)

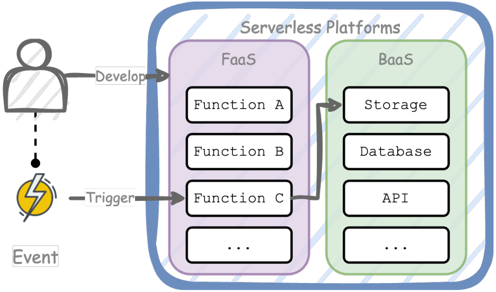
It is a serverless service model that focuses on providing specialized serverless frameworks to support specific application requirements (e.g., object storage, databases, or messaging services).

Container-as-a-Service (CaaS)

It is a cloud service model for deploying and managing containers.
It can be seen as a type of serverless computing, depending on the level of abstraction and automation it offers.

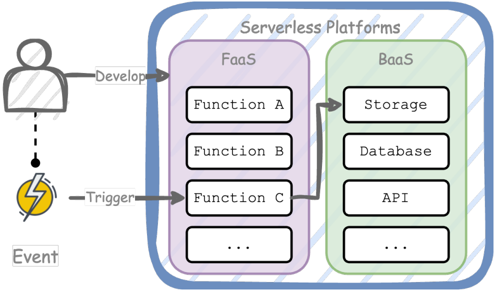
Putting All Together!

1. A pre-defined event triggers a serverless function that was bound to it earlier.
2. The serverless platform prepares the necessary execution environment for the triggered function to run (i.e., instance init., application transmission, and so on.).
3. After executions are completed, the serverless platform releases the resources previously acquired.



Putting All Together!

1. A pre-defined event triggers a serverless function that was bound to it earlier.
2. The serverless platform prepares the necessary execution environment for the triggered function to run (i.e., instance init., application transmission, and so on.).
3. After executions are completed, the serverless platform releases the resources previously acquired.



It's very simple, isn't it?

Current Serverless Platforms

Currently, there are several serverless options available, including both commercial and open source solutions.

- **Commercial:**
  - Amazon's AWS Lambda
  - Google's Cloud Functions
  - Microsoft Azure Functions
- **Open Source:**
  - OpenFaaS
  - Apache OpenWhisk
  - Knative

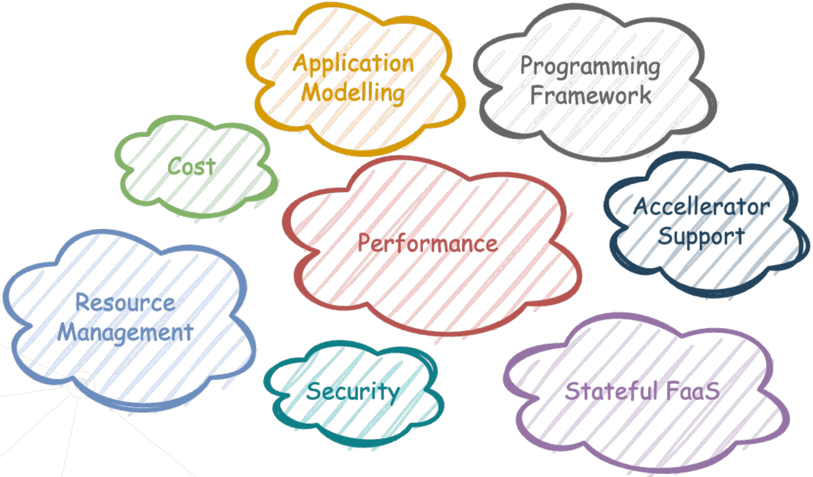
AWS Lambda · Cloud Functions · Azure Functions · OpenFaaS · OpenWhisk · Knative

Use Cases

- REST APIs
- Event-driven operations
- ML/AI Workloads
- Data Processing Pipelines
- Media Processing

Research Directions

Current Research Directions and Open Problems



Stateful FaaS

Serverless functions are developed in a stateless way.

- But there may be situations where it is necessary to transfer state.
- The situation is more complicated for serverless applications modeled as DAGs [11].

Solutions:

- Leverage on public cloud storage services (i.e., AWS S3) → **Overhead**!
- Use a distributed storage at the edge (i.e., Akka) [12] → **Depends on the case scenario**.
- Pass the state on each function call → **The client maintains the state**.
- Use an orchestrator to coordinate serverless functions and maintain state.
  - **AWS Step Functions**
  - **Azure's Durable Functions** [7]
  - **Apache OpenWhisk** through *Action Sequences*

STATEFUL FAAS

Serverless functions are developed in a stateless way.

- But there may be situations where it is necessary to transfer state.
- The situation is more complicated for serverless applications modeled as DAGs [11].

Solutions:

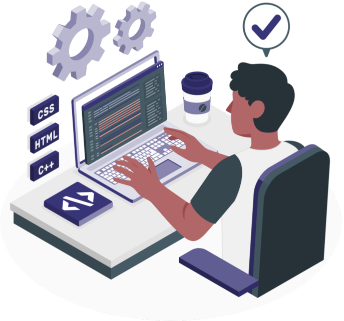
- Leverage on public cloud storage services (i.e., AWS S3) → **Overhead**!
- Use a distributed storage at the edge (i.e., Akka) [12] → **Depends on the case scenario**.
- Pass the state on each function call → **The client maintains the state**.
- Use an orchestrator to coordinate serverless functions and maintain state.
  - **AWS Step Functions**
  - **Azure's Durable Functions** [7]
  - **Apache OpenWhisk** through *Action Sequences*

But are there better solutions?

Programming Frameworks

Specific serverless frameworks are necessary to target different domains:

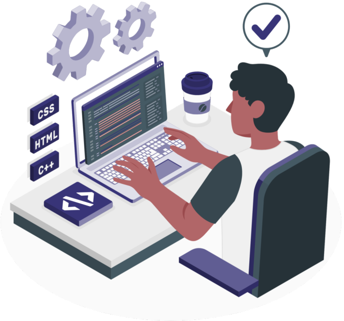
- **Numerical Computing**:
  - NumPyWren [41]
- **Video Processing**:
  - ExCamera [19], and Sprocket [3]
- **Internet of Things**:
  - AWS IoT Greengrass [29], and Azure IoT Edge [24]
  - tinyFaaS [37], AuctionWhisk [6], and PAPS [5]
- **Big Data Analytics**:
  - MapReduce for Serverless [21]
- **Machine Learning**:
  - BATCH [2], and Cirrus [10]



Programming Frameworks

Specific serverless frameworks are necessary to target different domains:

- **Numerical Computing**:
  - NumPyWren [41]
- **Video Processing**:
  - ExCamera [19], and Sprocket [3]
- **Internet of Things**:
  - AWS IoT Greengrass [29], and Azure IoT Edge [24]
  - tinyFaaS [37], AuctionWhisk [6], and PAPS [5]
- **Big Data Analytics**:
  - MapReduce for Serverless [21]
- **Machine Learning**:
  - BATCH [2], and Cirrus [10]



Mature tools are needed to facilitate the adoption of the serverless model.

Performance

Main Challenge → Minimize the startup time.

The startup time is influenced by three main phases [25]:

1. Scheduling and starting the resources needed to run the cloud function.
2. Setting up the environment where to run the cloud function.
3. Performing application-specific startup tasks.

Performance

Main Challenge → Minimize the startup time.

The startup time is influenced by three main phases [25]:

1. Scheduling and starting the resources needed to run the cloud function.
   - Use scheduling algorithms [45] [26].
   - Schedule functions in (existing) warm instances.
2. Setting up the environment where to run the cloud function.
3. Performing application-specific startup tasks.

Performance

Main Challenge → Minimize the startup time.

The startup time is influenced by three main phases [25]:

1. Scheduling and starting the resources needed to run the cloud function.
2. Setting up the environment where to run the cloud function.
3. Performing application-specific startup tasks.

In case of a **cold start**, the last two phases have a significant impact on the overall startup time.

Performance

Main Challenge → Minimize the startup time.

The startup time is influenced by three main phases [25]:

1. Scheduling and starting the resources needed to run the cloud function.
2. Setting up the environment where to run the cloud function.
3. Performing application-specific startup tasks.

In case of a **cold start**, the last two phases have a significant impact on the overall startup time.

And Now for Something Completely Different...

Notes on Virtualization

Depending on the level of abstraction, virtualization can be divided into three main categories [8]:

- **System-level Virtualization:**
  - Type I/Native Hypervisor (i.e., Xen).
  - Type II/Hosted Hypervisor (i.e., VMWare, KVM, VirtualBox).
  - i.e., Unikernels.
- **OS-level Virtualization:**
  - a.k.a. container.
  - i.e., Docker, FreeBSD Jails, and others.
- **Programming Language-level Virtualization:**
  - Execution through:
    - Interpretation
    - Just-In-Time compilation
  - i.e., JVM (Java), PVM (Python), and WebAssembly.

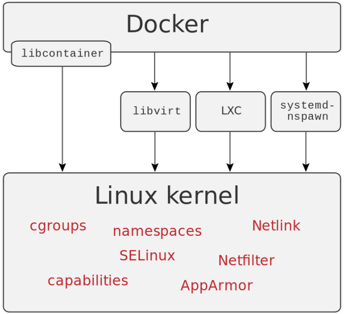
Notes on Virtualization

Depending on the level of abstraction, virtualization can be divided into three main categories [8]:

- **System-level Virtualization:**
  - Type I/Native Hypervisor (i.e., Xen).
  - Type II/Hosted Hypervisor (i.e., VMWare, KVM, VirtualBox).
  - i.e., **Unikernels**.
- **OS-level Virtualization:**
  - a.k.a. container.
  - i.e., **Docker**, FreeBSD Jails, and others.
- **Programming Language-level Virtualization:**
  - Execution through:
    - Interpretation
    - Just-In-Time compilation
  - i.e., JVM (Java), PVM (Python), and **WebAssembly**.

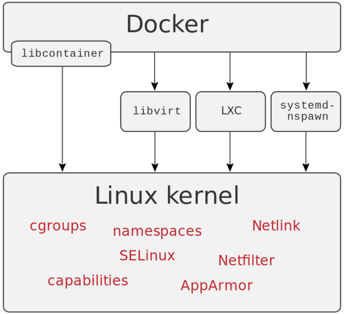
Docker

- OS-level Virtualization.
- Docker containers share the host OS kernel.
  - More lightweight than VMs.
  - But a kernel crash blocks all the containers on the same PM.
  - Also, shared kernel introduces potential security risks.
- Docker images make containers portable.
  - An image encapsulates an application and its dependencies.
  - It is possible to move containers between different systems.
- Each container has its own file system, libraries, and dependencies.
  - This can result in higher storage and memory usage.



Docker

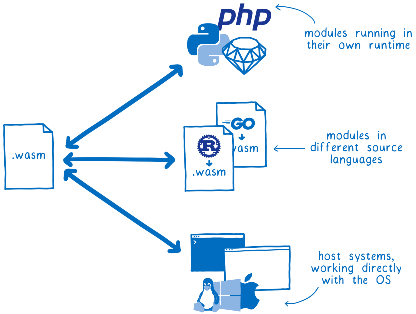
- OS-level Virtualization.
- Docker containers share the host OS kernel.
  - More lightweight than VMs.
  - But a kernel crash blocks all the containers on the same PM.
  - Also, shared kernel introduces potential security risks.
- Docker images make containers portable.
  - An image encapsulates an application and its dependencies.
  - It is possible to move containers between different systems.
- Each container has its own file system, libraries, and dependencies.
  - This can result in higher storage and memory usage.



Currently is the *de facto* standard virtualization solution used for serverless functions.

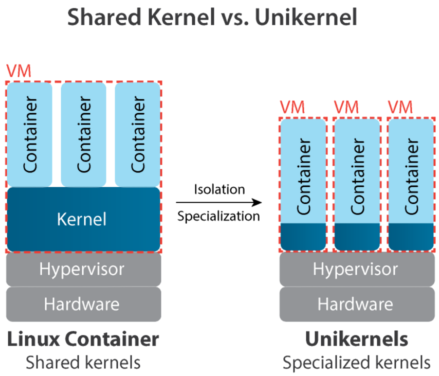
WebAssembly (*wasm*)

- Programming Language-level Virtualization.
- Presented in 2015, launched in 2017.
- It is a general-purpose virtual ISA [39].
  - It is designed to work with a stack-based virtual machine.
- It is both source language and target platform agnostic.
  - It supports compilation from C/C++, Rust, and any other LLVM-supported languages.
  - There are several Wasm runtime implementations available.
- The **WebAssembly System Interface (WASI)** provides an API for interacting with underlying resources [43].
  - It makes it possible to run WebAssembly outside of the browser.



Unikernel and Library OS

- System-level Virtualization.
- Based on the work done on Exokernel [15] and Nemesis [31].
- Exploit the concept of Library OS.
  - Reduce OS services to the minimum required for a specific application or service.
  - These are compiled, along with the application, into a single bootable VM image.
- Cloud (micro)service deployed within a Guest Library OS, running on a hypervisor.
- Popular examples: **MirageOS** [34], Hermit [30], Unikraft [28], and many others.



Unikernel and Library OS (cont'd)

Advantages

- Provide better performance than containers [22].
- Reduce the number of software layers deployed on a node in a cloud infrastructure.
- Reduce overheads caused by user-space/kernel-space transitions in the guest running in a VM.
- Reduce the attack surface.

Disadvantages

- Poor development tool support.
- Need for special compilation toolchain.
- Less flexible than containers.
- Lack of multi-thread support. Parallel applications split into multiple unikernels.
- As unikernels run in Ring 0 (a.k.a., supervisor mode), any vulnerabilities in the system could actually increase the attack surface [14].

Performance (cont'd)

There are several approaches for mitigating the cold start issue:

- **Data Cache-based optimizations**:
  - SOCK [36], alternative to Docker containers.
    - Targets Python workloads.
    - Exploits Python package caching and Zygote provisioning.
    - It is based on OpenLambda [23].
  - SAND [1], introduces a different approach to sandboxing.
    - Two levels of fault isolation:
      1. Isolation between different applications.
      2. Isolation between functions of the same application.
    - Application functions run in the same container, but as separate processes → Shared libraries loaded only once!
    - Interacting functions, located in the same host, communicate through a *local bus* → Reduce communication overhead.

Performance (cont'd)

- **Architecture Design optimizations**:
  - Use **Language-level Virtualization** → WebAssembly.
    - i.e., WasmEdge [33] and WoW [20].
    - Support for a large number of programming languages.
    - Currently, libraries and development tools are not mature yet…
    - …but something is moving (see containerd support for WASM).
  - Use **System-level Virtualization** → Unikernels.
    - i.e., USETL [17], framework for serverless ETL workloads.
    - Better performance w.r.t. standard containers (i.e., Docker).
    - Also here, lacks of development tools and libraries.
- **Instance Prewarm optimizations**:
  - Launch function instances in advance to serve the incoming request [44].
  - Different approaches can be used to predict future function invocations.

Performance (cont'd)

- **Snapshot-based optimizations**:
  - Use snapshots to reduce high latency due cold starts.
  - Capture the complete state of an initialized function. Saves it on storage.
  - When the same function is invoked again, restore its state through the saved snapshot.
  - i.e., SEUSS [9] → Based on Unikernels.

PERFORMANCE (CONT'D)

- **Snapshot-based optimizations**:
  - Use snapshots to reduce high latency due cold starts.
  - Capture the complete state of an initialized function. Saves it on storage.
  - When the same function is invoked again, restore its state through the saved snapshot.
  - i.e., SEUSS [9] → Based on Unikernels.

Currently, these proposals are still limited to the research domain. More work needs to be done before they can be implemented in real-life applications.

Accelerator Supports

Current serverless platforms are predominantly centered on CPU resources.

- However, some kinds of workloads could benefit from the use of hardware accelerators.
  - i.e., Video Processing, Machine Learning, Artificial Intelligence, and others.

How can the serverless model be adapted to support different types of heterogeneous accelerators?

Accelerator Supports

Current serverless platforms are predominantly centered on CPU resources.

- However, some kinds of workloads could benefit from the use of hardware accelerators.
  - i.e., Video Processing, Machine Learning, Artificial Intelligence, and others.

How can the serverless model be adapted to support different types of heterogeneous accelerators?

- BlastFunction [4] → **FPGA-as-a-Service**.
  - It is an FPGA sharing system for accelerating serverless applications.
  - It uses a *time-sharing* approach to maximize device utilization in a multi-tenant environment.

Accelerator Supports

Current serverless platforms are predominantly centered on CPU resources.

- However, some kinds of workloads could benefit from the use of hardware accelerators.
  - i.e., Video Processing, Machine Learning, Artificial Intelligence, and others.

How can the serverless model be adapted to support different types of heterogeneous accelerators?

- BlastFunction [4] → **FPGA-as-a-Service**.
  - It is an FPGA sharing system for accelerating serverless applications.
  - It uses a *time-sharing* approach to maximize device utilization in a multi-tenant environment.
- Molecule [13] → **Distributed Shim**.
  - It introduces *XPU-Shim*, which provides system call style interfaces to serverless runtime.
  - By using *XPUcalls*, serverless functions can be instantiated on different PUs (i.e., DPU), and communicate directly with each other.

Accelerator Supports

Current serverless platforms are predominantly centered on CPU resources.

- However, some kinds of workloads could benefit from the use of hardware accelerators.
  - i.e., Video Processing, Machine Learning, Artificial Intelligence, and others.

How can the serverless model be adapted to support different types of heterogeneous workflows?

Moving towards the **Kernel-as-a-Service** model.

Application Modelling

At present, the serverless model lacks application modeling techniques.

- Making the understanding of the system more difficult...
- ..and preventing rapid and frequent changes at high levels of abstraction.

There are various possible approaches:

- F(X)-MAN [38], extends the X-MAN component model.
  - It defines two types of services: atomic and composite.
  - Introduces connectors for hierarchical composition.
- Other approaches involve the usage of BPMN and TOSCA [46], or the usage of design patterns [35].

Application Modelling

At present, the serverless model lacks application modeling techniques.

- Making the understanding of the system more difficult...
- ..and preventing rapid and frequent changes at high levels of abstraction.

There are various possible approaches:

- F(X)-MAN [38], extends the X-MAN component model.
  - It defines two types of services: atomic and composite.
  - Introduces connectors for hierarchical composition.
- Other approaches involve the usage of BPMN and TOSCA [46], or the usage of design patterns [35].

However, further research on application modeling for serverless software development is necessary, since it has different requirements w.r.t. traditional software development.

Conclusions

Conclusions

What have we learned?

- Serverless computing represents a further evolution of the trend toward higher levels of abstraction in cloud computing models.
- It enables developers to write applications without dealing with the operational logic.
- As serverless applications are event-driven, computing resources are provisioned and instantiated by the cloud provider only when needed.
- However, this model is not yet mature, there are several open questions that need to be addressed.

CONCLUSIONS

What have we learned?

- Serverless computing represents a further evolution of the trend toward higher levels of abstraction in cloud computing models.
- It enables developers to write applications without dealing with the operational logic.
- As serverless applications are event-driven, computing resources are provisioned and instantiated by the cloud provider only when needed.
- However, this model is not yet mature, there are several open questions that need to be addressed.

Thank you for your attention!

Any Questions?

Bibliography I

[1] Istemi Ekin Akkus et al. “SAND: Towards High-Performance Serverless Computing”. In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 923–935. ISBN: 978-1-939133-01-4. URL: https://www.usenix.org/conference/atc18/presentation/akkus.

[2] Ahsan Ali et al. “BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching”. In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020, pp. 1–15. DOI: 10.1109/SC41405.2020.00073.

[3] Lixiang Ao et al. “Sprocket: A Serverless Video Processing Framework”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’18. Carlsbad, CA, USA: Association for Computing Machinery, 2018, pp. 263–274. ISBN: 9781450360111. DOI: 10.1145/3267809.3267815. URL: https://doi.org/10.1145/3267809.3267815.

Bibliography II

[4] Marco Bacis, Rolando Brondolin, and Marco D. Santambrogio. “BlastFunction: an FPGA-as-a-Service system for Accelerated Serverless Computing”. In: *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2020, pp. 852–857. DOI: 10.23919/DATE48585.2020.9116333.

[5] Luciano Baresi and Giovanni Quattrocchi. “PAPS: A Serverless Platform for Edge Computing Infrastructures”. In: *Frontiers in Sustainable Cities* 3 (2021). ISSN: 2624-9634. DOI: 10.3389/frsc.2021.690660. URL: https://www.frontiersin.org/articles/10.3389/frsc.2021.690660.

[6] David Bermbach et al. “AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms”. In: *Software: Practice and Experience* 52.5 (2022), pp. 1143–1169.

Bibliography III

[7] Sebastian Burckhardt et al. "Durable functions: semantics for stateful serverless". In: *Proc. ACM Program. Lang.* 5.OOPSLA (Oct. 2021). DOI: 10.1145/3485510. URL: https://doi.org/10.1145/3485510.

[8] Rajkumar Buyya, Christian Vecchiola, and S Thamarai Selvi. *Mastering cloud computing: foundations and applications programming*. Newnes, 2013.

[9] James Cadden et al. "SEUSS: skip redundant paths to make serverless fast". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. Heraklion, Greece: Association for Computing Machinery, 2020. ISBN: 9781450368827. DOI: 10.1145/3342195.3392698. URL: https://doi.org/10.1145/3342195.3392698.

Bibliography IV

[10] Joao Carreira et al. "Cirrus: a Serverless Framework for End-to-end ML Workflows". In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '19. Santa Cruz, CA, USA: Association for Computing Machinery, 2019, pp. 13–24. ISBN: 9781450369732. DOI: 10.1145/3357223.3362711. URL: https://doi.org/10.1145/3357223.3362711.

[11] Claudio Cicconetti, Marco Conti, and Andrea Passarella. "FaaS execution models for edge applications". In: *Pervasive and Mobile Computing* 86 (2022), p. 101689.

[12] Claudio Cicconetti, Marco Conti, and Andrea Passarella. "In-Network Computing With Function as a Service at the Edge". In: *Computer* 55.9 (2022), pp. 65–73. DOI: 10.1109/MC.2021.3130659.

Bibliography V

[13] Dong Du et al. “Serverless computing on heterogeneous computers”. In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’22. Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 797–813. ISBN: 9781450392051. DOI: 10.1145/3503222.3507732. URL: https://doi.org/10.1145/3503222.3507732.

[14] Nabil El Ioini et al. “Unikernels Motivations, Benefits and Issues: A Multivocal Literature Review”. In: *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*. ESAAM ’23. Ludwigsburg, Germany: Association for Computing Machinery, 2023, pp. 39–48. ISBN: 9798400708350. DOI: 10.1145/3624486.3624492. URL: https://doi.org/10.1145/3624486.3624492.

Bibliography VI

[15] D. R. Engler, M. F. Kaashoek, and J. O'Toole. "Exokernel: an operating system architecture for application-level resource management". In: *SIGOPS Oper. Syst. Rev.* 29.5 (Dec. 1995), pp. 251–266. ISSN: 0163-5980. DOI: 10.1145/224057.224076. URL: https://doi.org/10.1145/224057.224076.

[16] Erwin van Eyk et al. "The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures". In: *Proceedings of the 2nd International Workshop on Serverless Computing*. WoSC '17. Las Vegas, Nevada: Association for Computing Machinery, 2017, pp. 1–4. ISBN: 9781450354349. DOI: 10.1145/3154847.3154848. URL: https://doi.org/10.1145/3154847.3154848.

Bibliography VII

[17] Henrique Fingler, Amogh Akshintala, and Christopher J. Rossbach. "USETL: Unikernels for Serverless Extract Transform and Load Why should you settle for less?" In: *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*. APSys '19. Hangzhou, China: Association for Computing Machinery, 2019, pp. 23–30. ISBN: 9781450368933. DOI: 10.1145/3343737.3343750. URL: https://doi.org/10.1145/3343737.3343750.

[18] Ian Foster and Carl Kesselman, eds. *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. ISBN: 1558604758.

[19] Sadjad Fouladi et al. "Encoding, fast and slow:{Low-Latency} video processing using thousands of tiny threads". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 363–376.

Bibliography VIII

[20] Philipp Gackstatter, Pantelis A. Frangoudis, and Schahram Dustdar. "Pushing Serverless to the Edge with WebAssembly Runtimes". In: *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2022, pp. 140–149. DOI: 10.1109/CCGrid54584.2022.00023.

[21] Vicent Giménez-Alventosa, Germán Moltó, and Miguel Caballer. "A framework and a performance assessment for serverless MapReduce on AWS Lambda". In: *Future Generation Computer Systems* 97 (2019), pp. 259–274.

[22] Tom Goethals et al. "Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications". In: *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. 2018, pp. 1–8. DOI: 10.1109/SC2.2018.00008.

Bibliography IX

[23] Scott Hendrickson et al. “Serverless Computation with OpenLambda”. In: *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. Denver, CO: USENIX Association, June 2016. URL: https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/hendrickson.

[24] David Jensen. *Beginning Azure IoT Edge computing: extending the cloud to the intelligent edge*. Apress, 2019.

[25] Eric Jonas et al. “Cloud Programming Simplified: A Berkeley View on Serverless Computing”. In: (2019).

Bibliography X

[26] Dong Kyoung Kim and Hyun-Gul Roh. "Scheduling Containers Rather Than Functions for Function-as-a-Service". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2021, pp. 465–474. DOI: 10.1109/CCGrid51090.2021.00056.

[27] Samuel Kounev et al. "Serverless Computing: What It Is, and What It Is Not?" In: *Commun. ACM* 66.9 (Aug. 2023), pp. 80–92. ISSN: 0001-0782. DOI: 10.1145/3587249. URL: https://doi.org/10.1145/3587249.

Bibliography XI

[28] Simon Kuenzer et al. “Unikraft: fast, specialized unikernels the easy way”. In: *Proceedings of the Sixteenth European Conference on Computer Systems*. EuroSys ’21. Online Event, United Kingdom: Association for Computing Machinery, 2021, pp. 376–394. ISBN: 9781450383349. DOI: 10.1145/3447786.3456248. URL: https://doi.org/10.1145/3447786.3456248.

[29] Agus Kurniawan. *Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning*. Packt Publishing Ltd, 2018.

Bibliography XII

[30] Stefan Lankes, Jens Breitbart, and Simon Pickartz. "Exploring Rust for Unikernel Development". In: *Proceedings of the 10th Workshop on Programming Languages and Operating Systems*. PLOS '19. Huntsville, ON, Canada: Association for Computing Machinery, 2019, pp. 8–15. ISBN: 9781450370172. DOI: 10.1145/3365137.3365395. URL: https://doi.org/10.1145/3365137.3365395.

[31] I.M. Leslie et al. "The design and implementation of an operating system to support distributed multimedia applications". In: *IEEE Journal on Selected Areas in Communications* 14.7 (1996), pp. 1280–1297. DOI: 10.1109/49.536480.

[32] Fang Liu et al. *NIST Cloud Computing Reference Architecture*. en. 2011-09-08 00:09:00 2011. DOI: https://doi.org/10.6028/NIST.SP.500-292. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=909505.

Bibliography XIII

[33] Ju Long et al. "A Lightweight Design for Serverless Function as a Service". In: *IEEE Software* 38.1 (2021), pp. 75–80. DOI: 10.1109/MS.2020.3028991.

[34] Anil Madhavapeddy et al. "Unikernels: library operating systems for the cloud". In: *SIGARCH Comput. Archit. News* 41.1 (Mar. 2013), pp. 461–472. ISSN: 0163-5964. DOI: 10.1145/2490301.2451167. URL: https://doi.org/10.1145/2490301.2451167.

[35] Anil Mathew et al. "Pattern-based serverless data processing pipelines for Function-as-a-Service orchestration systems". In: *Future Generation Computer Systems* 154 (2024), pp. 87–100. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.12.026. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23004855.

Bibliography XIV

[36] Edward Oakes et al. "SOCK: Rapid Task Provisioning with Serverless-Optimized Containers". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 57–70. ISBN: 978-1-931971-44-7. URL: https://www.usenix.org/conference/atc18/presentation/oakes.

[37] Tobias Pfandzelter and David Bermbach. "tinyFaaS: A Lightweight FaaS Platform for Edge Environments". In: *2020 IEEE International Conference on Fog Computing (ICFC)*. 2020, pp. 17–24. DOI: 10.1109/ICFC49376.2020.00011.

Bibliography XV

[38] Chen Qian and Wenjing Zhu. "F(X)-MAN: An Algebraic and Hierarchical Composition Model for Function-as-a-Service". In: *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*. Ed. by Raúl García-Castro. KSI Research Inc., 2020, pp. 210–215. DOI: 10.18293/SEKE2020-022. URL: https://doi.org/10.18293/SEKE2020-022.

[39] Andreas Rossberg. *WebAssembly Core Specification*. W3C, Dec. 5, 2019. URL: https://www.w3.org/TR/wasm-core-1/.

[40] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. "Serverless Computing: A Survey of Opportunities, Challenges, and Applications". In: *ACM Comput. Surv.* 54.11s (Nov. 2022). ISSN: 0360-0300. DOI: 10.1145/3510611. URL: https://doi.org/10.1145/3510611.

Bibliography XVI

[41] Vaishaal Shankar et al. "Numpywren: Serverless linear algebra". In: *arXiv preprint arXiv:1810.09679* (2018).

[42] Blesson Varghese and Rajkumar Buyya. "Next generation cloud computing: New trends and research directions". In: *Future Generation Computer Systems* 79 (2018), pp. 849–861. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.020. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17302224.

[43] WebAssembly Community Group. *WebAssembly System Interface*. Feb. 29, 2020. URL: https://github.com/WebAssembly/WASI/blob/d8b286c697364d8bc4daf1820b25a9159de364a3/phases/snapshot/docs.md.

Bibliography XVII

[44] Jinfeng Wen et al. "Rise of the Planet of Serverless Computing: A Systematic Review". In: *ACM Trans. Softw. Eng. Methodol.* 32.5 (July 2023). ISSN: 1049-331X. DOI: 10.1145/3579643. URL: https://doi.org/10.1145/3579643.

[45] Song Wu et al. "Container lifecycle-aware scheduling for serverless computing". In: *Software: Practice and Experience* 52.2 (2022), pp. 337–352. DOI: https://doi.org/10.1002/spe.3016. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3016. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3016.

Bibliography XVIII

[46] Vladimir Yussupov et al. “Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA”. In: *Software: Practice and Experience* 52.6 (2022), pp. 1454–1495. DOI: https://doi.org/10.1002/spe.3073. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3073. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3073.

Appendix

Extended Introduction on Serverless

Serverless

Serverless computing is a form of cloud computing that allows users to run event-driven and granularly billed applications, without having to address the operational logic [16] .

Furthermore, a serverless service exhibits the following characteristics [40] [44]:

1. *NoOps*: The execution environment is hidden from the customer and completely managed by the cloud provider (no operations needed).
2. *Auto-scaling*: The cloud provider is responsible for providing and managing an auto-scaling service.
3. *Utilization-based billing*: The billing mechanism must only take into account the number of resources actually used by the customer (i.e. pay-as-you-go).
4. *Separation of computation and storage*: Generally, serverless computation should be stateless (This is not always true).

Serverless Service Models

Function as a Service

Function as a Service (FaaS) is a form of serverless computing in which the cloud provider manages the resources, life-cycle, and event-driven execution of user-provided functions [16].

Backend as a Service

Backend as a Service (BaaS) is a form of serverless computing focused on providing specialized serverless frameworks that cater to specific application requirements (i.e., object storage, databases, or messaging services) [25].

Container as a Service

Container as a Service (CaaS) is a cloud service model that allows users to deploy and manage containers in the cloud [42]. CaaS can be seen as a form of serverless computing, depending on the level of abstraction and automation it provides [27].

Serverless vs Traditional Software Development

Serverless software development differs from traditional non-cloud software development.

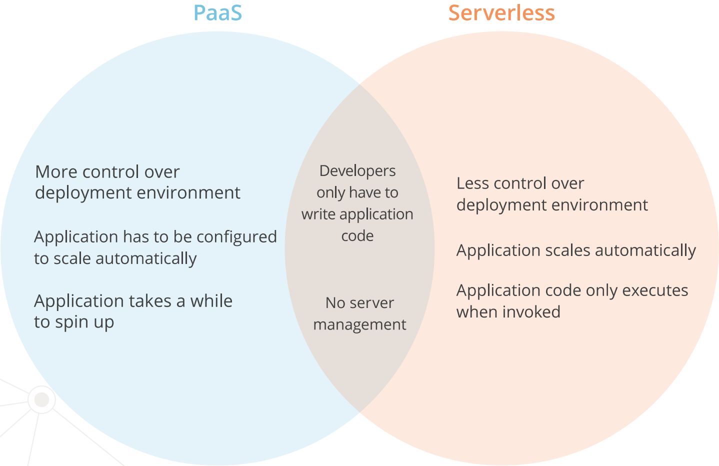
Features	Non-cloud SWD	Serverless SWD
Server management	Full management	No management
Functionality implementation	Implement everything from scratch	Implement only event-driven code
Invocation pattern	Client-side calls	Event triggers
Performance	Always activated	Activated only if triggered (**cold start**)
Cost	Pay for everything	Only pay for the resources you use

Traditional vs IaaS vs PaaS vs FaaS

On-premises	IaaS	PaaS	FaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Managed by User *Managed by CSP*

PaaS vs FaaS



Source:

https://www.cloudflare.com/en-gb/learning/serverless/glossary/serverless-vs-paas/

PESARESI SEMINAR SERIES

BEYOND THE CLOUD

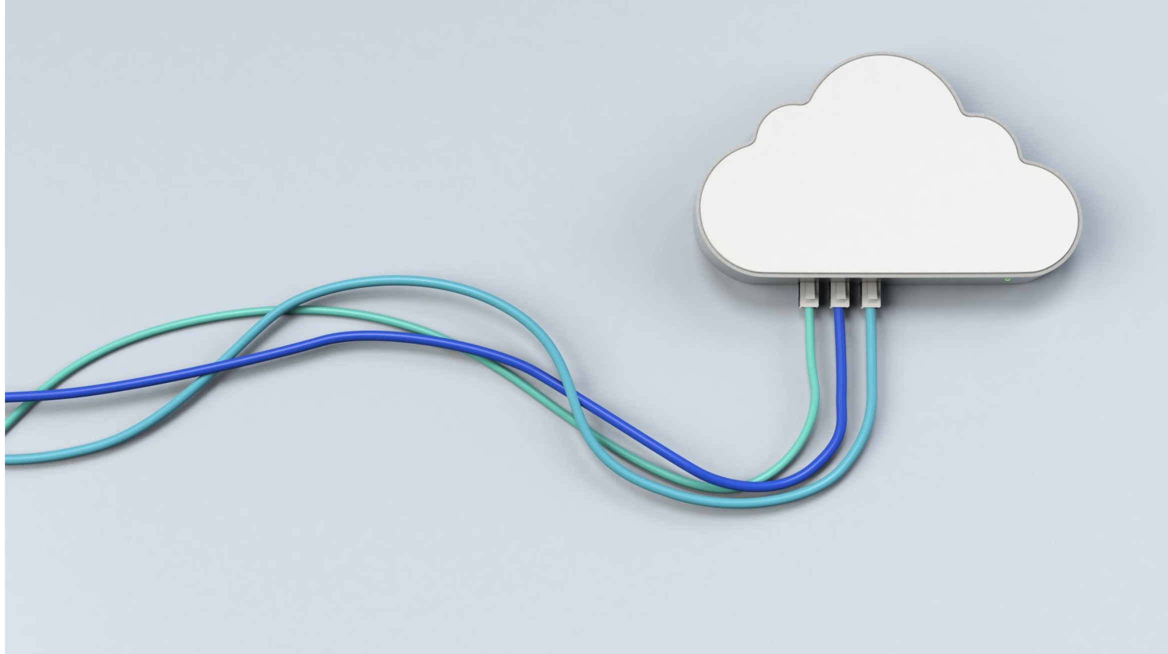
EXPLORING SERVERLESS COMPUTING AND CLOUD CONTINUUM

PART TWO

Lanpei Li

March 1st, 2024

Cloud / IoT Continuum



Modern computing paradigms

Cloud computing

Mobile cloud computing

Fog computing

Edge computing

“A model for enabling 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".

Advantages

Limitations

P. Mell, T. Grance, et al., The NIST Definition of Cloud Computing, Computer Security Division, National Information Technology Laboratory, 2011.

Modern computing paradigms

Cloud computing

Mobile cloud computing(MCC)

Fog computing

Edge computing

"A mobile device that can execute a resource-intensive application on a distant high-performance compute server or compute cluster and support thin client user interactions with the application over the Internet."

Advantages

Limitations

N.I.M. Enzai, M. Tang, A taxonomy of computation offloading in mobile cloud computing, in: 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2014, pp. 19–28.

Modern computing paradigms

Cloud computing

Mobile cloud computing

Fog computing

Edge computing

“ The process of extending Cloud Computing capabilities at the edge of the network. Fog incorporates computing, storage and network resources close to the IoT layer to facilitate the data processing”

Advantages

Limitations

F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in; Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

R. Cisco, M.Y. Upc, M. Nemirovsky, Fog computing, in: Proc. Cloud Assist. Serveys Eur. Conf. Bled, 2012, pp. 1–15.

Modern computing paradigms

Cloud computing

Mobile cloud computing

Fog computing

Edge computing

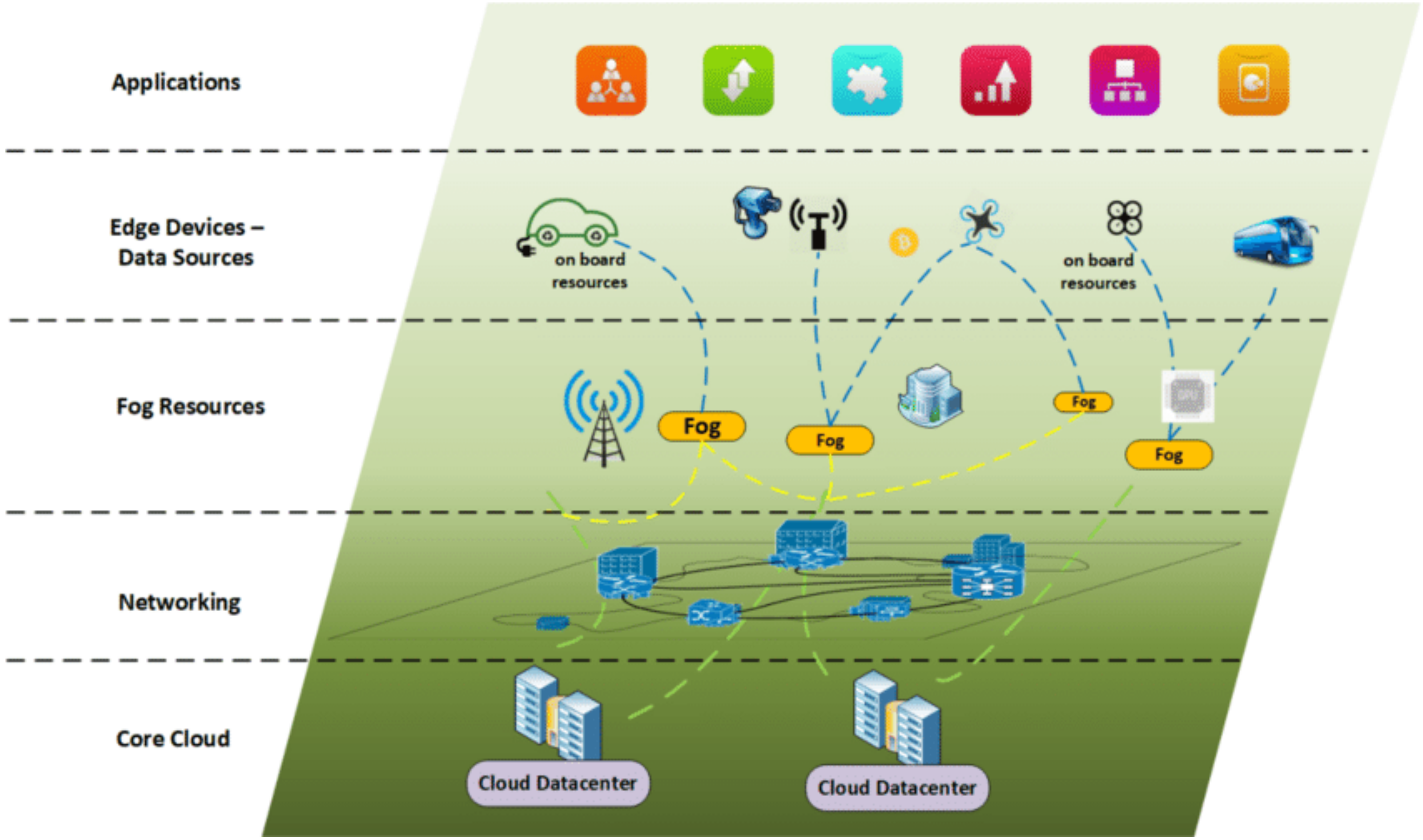
"A key technology to assist wireless networks with Cloud Computing-like capabilities to provide low-latency and context-aware services directly from the network Edge."

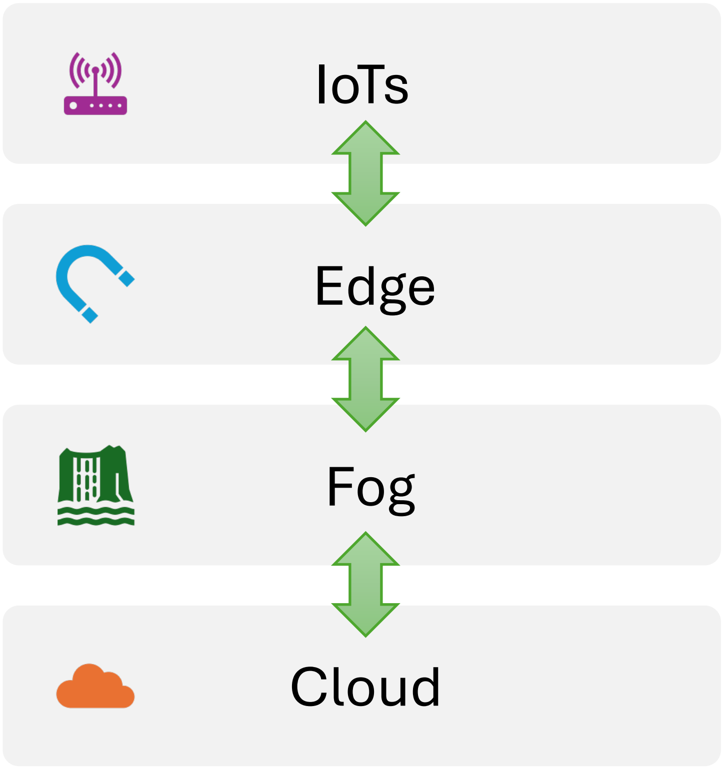
Advantages

Limitations

S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, et al., MEC In 5G networks, ETSI White Paper 28 (2018) 1–28.

Where does the cloud continue?





Bittencourt, Luiz Fernando et al. "The Internet of Things, Fog and Cloud Continuum: Integration and Challenges." ArXiv abs/1809.09972 (2018): n. pag.

Cloud Continuum

An extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities.

TANSTAAFL

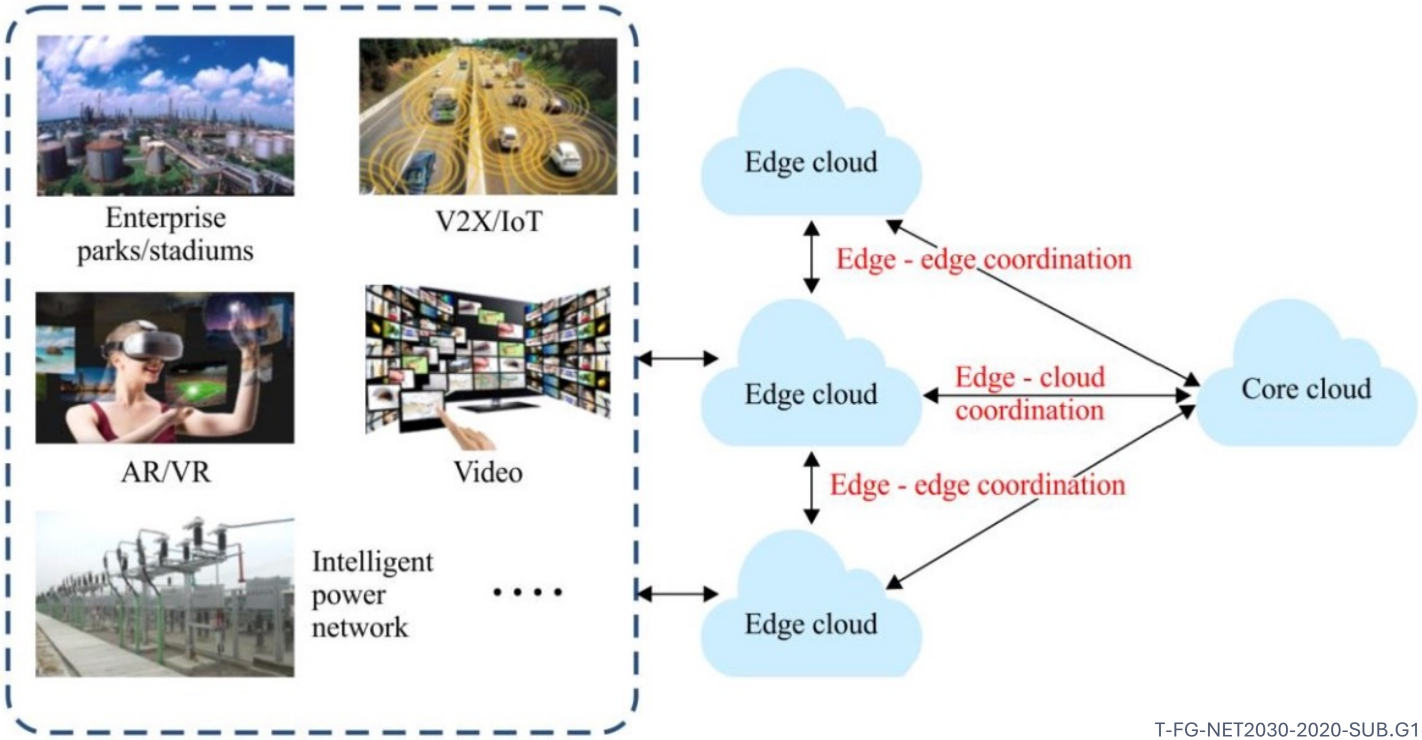
S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka and D. Taibi, "Cloud Continuum: The Definition," in IEEE Access, vol. 10, pp. 131876-131886, 2022, doi: 10.1109/ACCESS.2022.3229185.

Objective :

- Seamless and Integrated
- Efficient and Flexible
- Distributed and Scalable
- Energy consumption
- New Requirements

Use cases

- Immersive applications
- Autonomous vehicles
- Video streaming
- Space-Terrestrial
- Robotics
- IoT



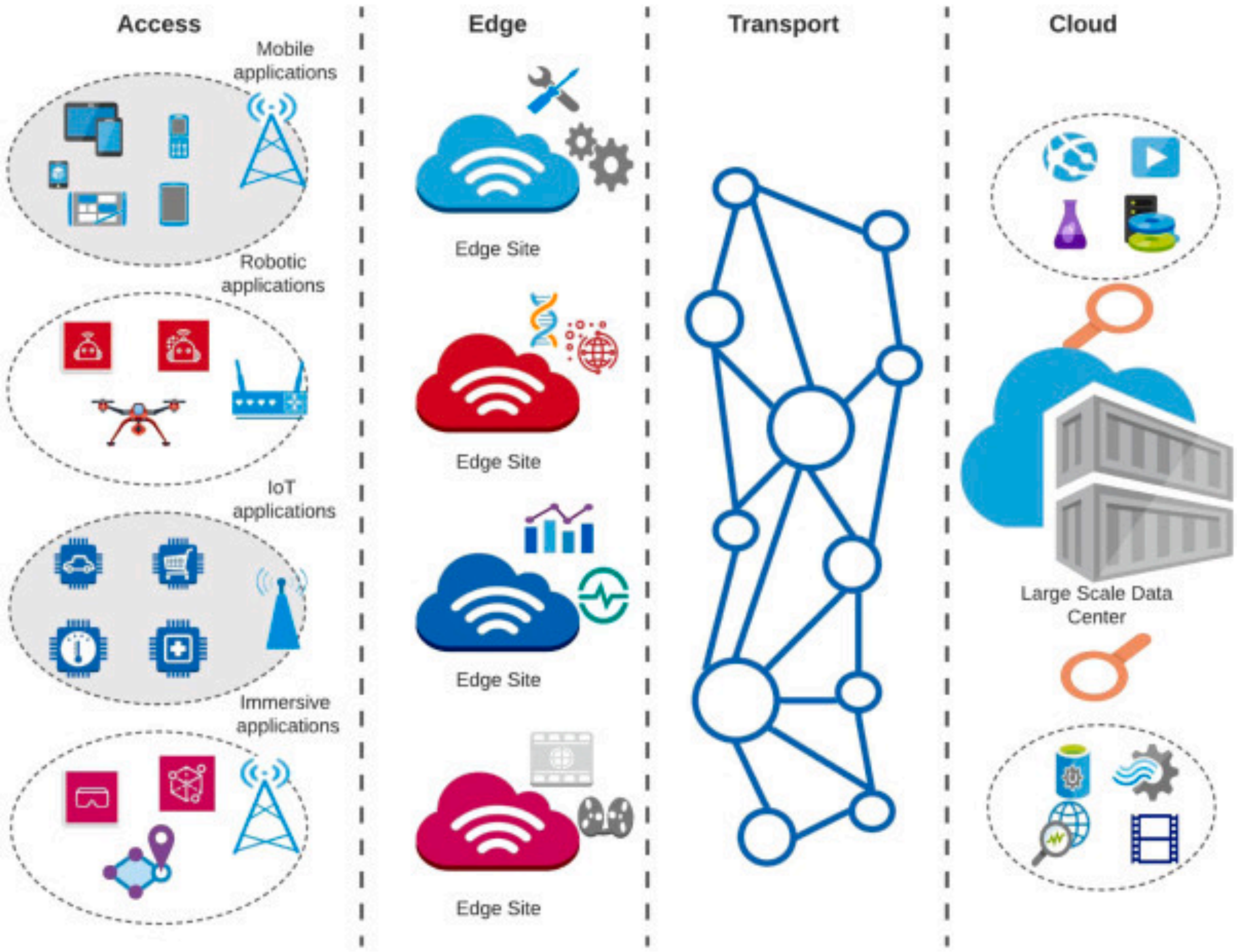
T-FG-NET2030-2020-SUB.G1-PDF-E

Characteristic

HETEROGENEOUS

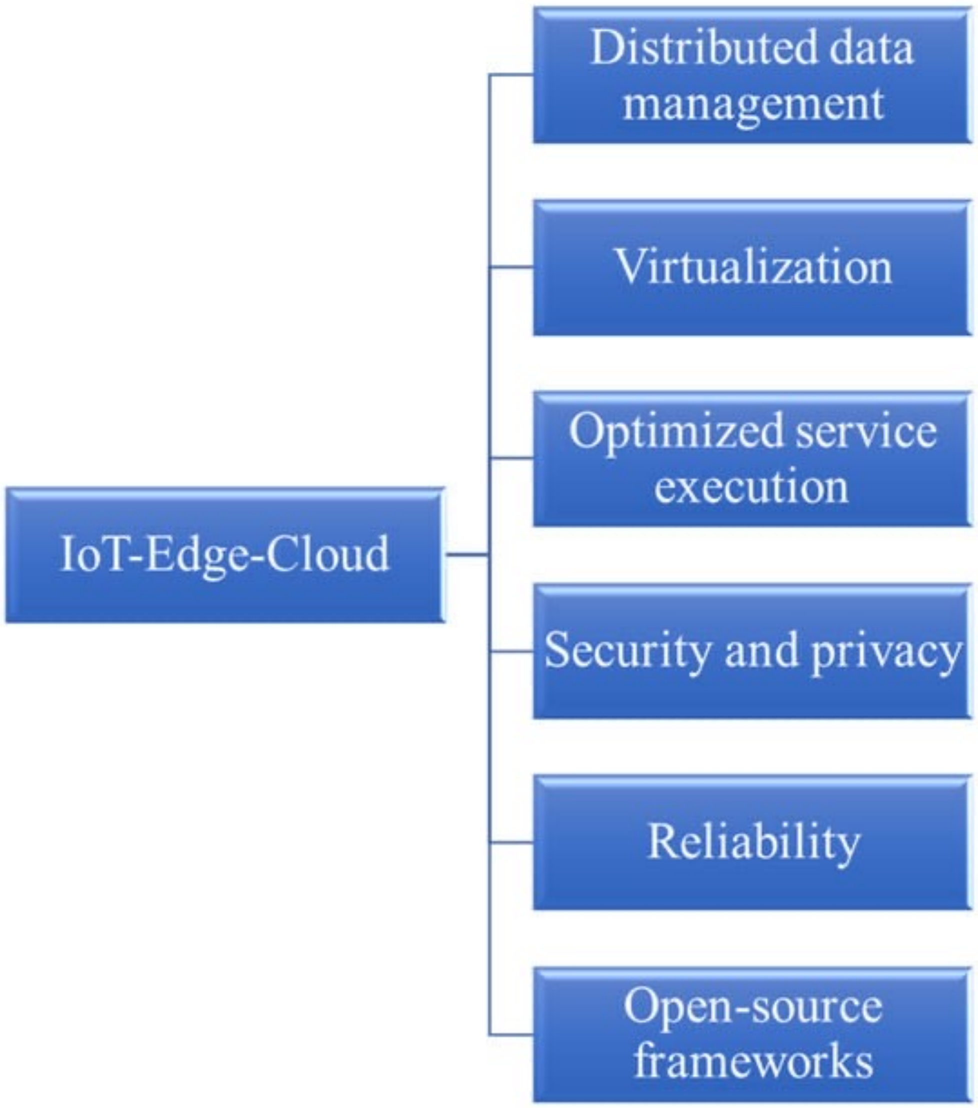
DISTRIBUTED

REQUIREMENTS



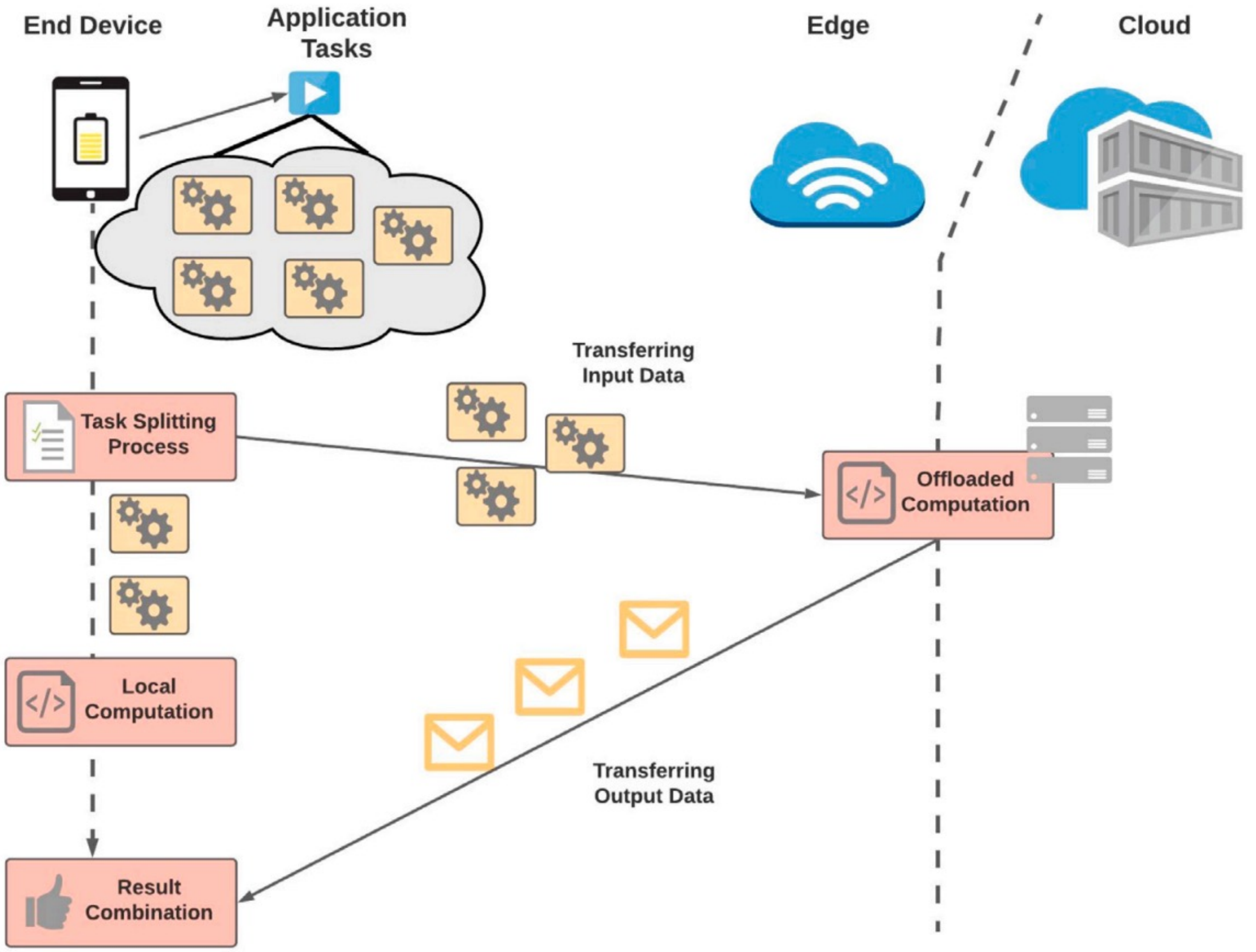
Challenges

- **Resource Orchestration**
  - Dynamic Allocation
  - Network Partitioning
  - Positioning
  - Localization
  - Job Scheduling
  - Task Offloading
- **Interoperability**
- **Performance**
  - Scalability
  - Mobility
  - Consistency
  - ...
- **Robustness**
- **Security**



Task Offloading

"The transfer of resource-intensive computational tasks to an external, resource-rich platform such as the ones used in Cloud, Edge or Fog Computing."



Saeik F., Avgeris M., Spatharakis D., Santi N., Dechouniotis D., Violos J., Leivadeas A., Athanasopoulos N., Mitton N., Papavassiliou S. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions

Challenges

- What to offload?
- Why to offload?
- When to offload (static or dynamic)?
- Where to offload?
- How to offload?

Objective

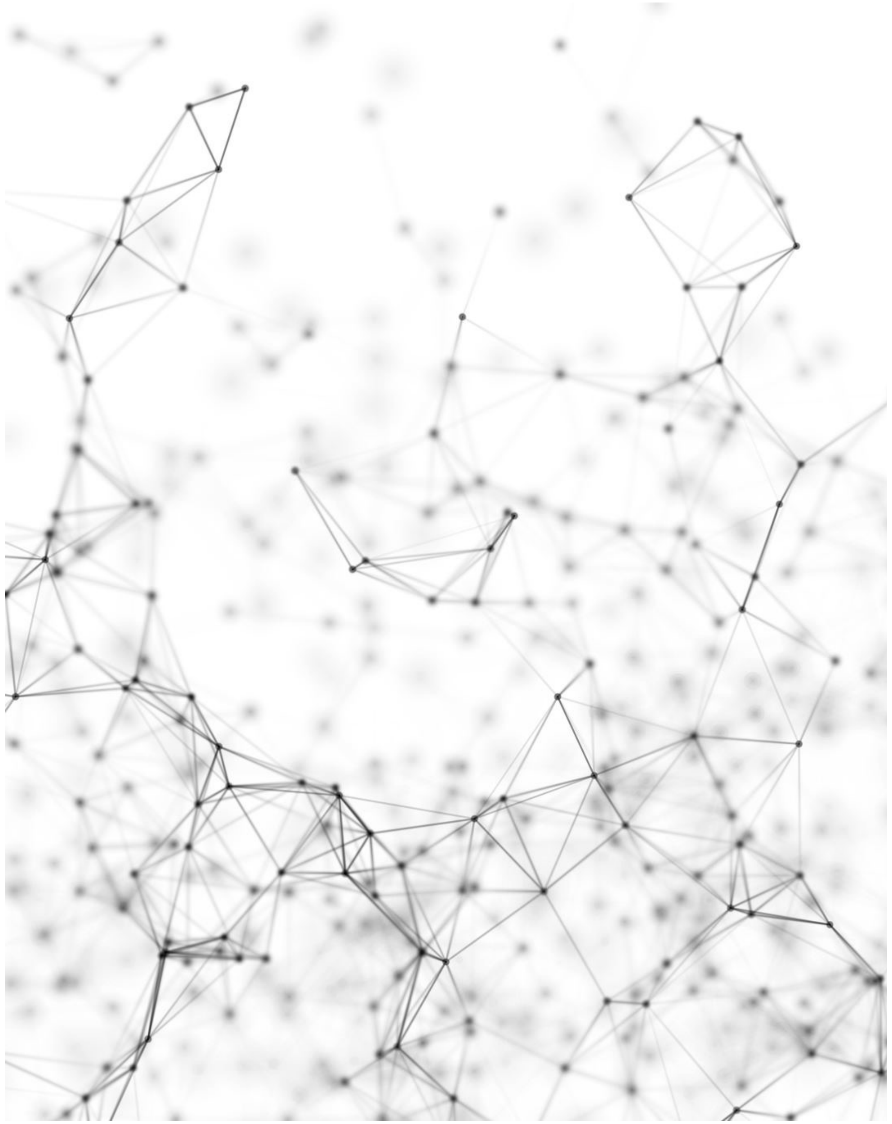
- Delay
- Energy
- Bandwidth
- Load balancing
- Deployment cost
- Model accuracy
- Multi-objective

Configuration views

- User/server-oriented edge architectures
- Offloading decision.
- Granularity-based offloading decision.
- Computation offloading sub-problems.
- Device-Edge-Cloud communication strategy.

Methodologies

- Mathematical optimization algorithms
- Control theory-based algorithms
- AI-based optimization algorithms



Problem formulation

- $Task_i \ : \ T_i =< d_i, c_i, t_i >$
- $Local\ execution\ time\ of\ task_i \ : \ \tau_i^l = \frac{c_i}{f_l}$
- $Offload\ time\ of\ task_i \ : \ \tau_i^u = \frac{d_i}{\mathcal{R}}$
- $Latency\ at\ edge\!: \tau_i^{mec} = \frac{c_i}{F_m^k}$
- $Latency\ at\ cloud\!: \tau_i^c = \frac{c_i}{F_c^k}$
- $Remote\ execution\ time\!: \tau_i^r = (1-\gamma_i)\ \tau_i^{mec} + \ \gamma_i\ \tau_i^c$ $where\ \ \gamma = \{0, 1\},\ edge\ or\ cloud?$

Choudhury, Alok, Manojit Ghose, and Akhirul Islam. "Machine learning-based computation offloading in multi-access edge computing: A survey." *Journal of Systems Architecture* (2024): 103090.

Latency minimization

$$\tau = \sum_{i=1}^{N} (1-\delta)\,\tau_i^l + \delta_i \tau_i^r \;\; where\; \delta = \{0,1\}$$

$$minimize\; \tau,\;\; such\; that:$$

$$\text{C1: } \tau_i \le t_d$$

$$\text{C2: } \sum_{i=1} c_i \le f_l$$

$$\text{C3: } \sum_{i=1} c_i \le F_m^k$$

Choudhury, Alok, Manojit Ghose, and Akhirul Islam. "Machine learning-based computation offloading in multi-access edge computing: A survey." *Journal of Systems Architecture* (2024): 103090.

RL-based Solution

- Task： $\left(W_k(\mathrm{t}), D_k(\mathrm{t}), T_k(\mathrm{t})\right)$
- Latency： $\tau_{\mathrm{k,s}} \approx \tau_{\mathrm{up}} + \tau_{proc} = \frac{D_k}{R_{k,s}} + \frac{W_k}{f_s}$
- Computation： $f(t) = \{\mathrm{f}_1(\mathrm{t}), \mathrm{f}_2(\mathrm{t}), \ldots f_S(\mathrm{t})\}\ with\ f_s(t) = \left(1 - \eta_F^s(t)\right)F_s$
- Storage： $q(t) = \{q_1(t), q_2(t), \ldots q_S(t)\}$ with $q_s(t) = \left(1 - \eta_Q^s(t)\right) Q_s$
- Networking： $g(t) = \left\{g_{i,j}(t) \middle| i, j \in S, i \neq j\right\}$ with $g_{i,j}(t) = h_{i,j}(t) d_{i,j}^{-\zeta}$

RL-based Solution

- **Action**： $a(t) = \{\alpha_{k,s}(t), f_s(t)\} \in \mathcal{A}$
- **Observation**： $s(t) = \{Env_t,\ Task_t\} = \{f(t), q(t), g(t), W_k(t), D_k(t), T_k(t)\} \in \mathcal{S}$
- **Reward**： $\mathcal{R}(t) = \mathcal{F}(t) - \mathcal{P}(t) = max\frac{1}{S}\sum_{k\in K}\sum_{s\in S}\alpha_{k,s}(t)\tau_{k,s}(t) - C_0\left[\sum_{s\in S}\left(1 - \alpha_{k,s}(t)\right)\right]$

Wireless Signal Processing and Networking Laboratory (WSPN), Beijing University of Posts and Telecommunications (BUPT), “A joint computing offloading and resource allocation method for 5G multi-access edge computing”, 2022.9.22.

Evaluation Metrics

- LATENCY
- ENERGY CONSUMPTION
- BANDWIDTH UTILIZATION
- RESPONSE TIME
- SYSTEM COST
- ALGORITHM EFFICIENCY
- MULTI OBJECTIVE FUNCTION

Challenges in ML-based offloading

ENERGY CONSUMPTION

PROBLEM FORMULATION TECHNIQUE

PARTITIONING GRANULARITY

RESOURCE UTILIZATION

RESOURCE SCHEDULING

MOBILITY

Conclusion

- Cloud / IoT Continuum
- Task Offloading
- ML-based solution

Thank you and Questions