The promise and the reality of serverless

Gustavo Alonso

Systems Group

Department of Computer Science

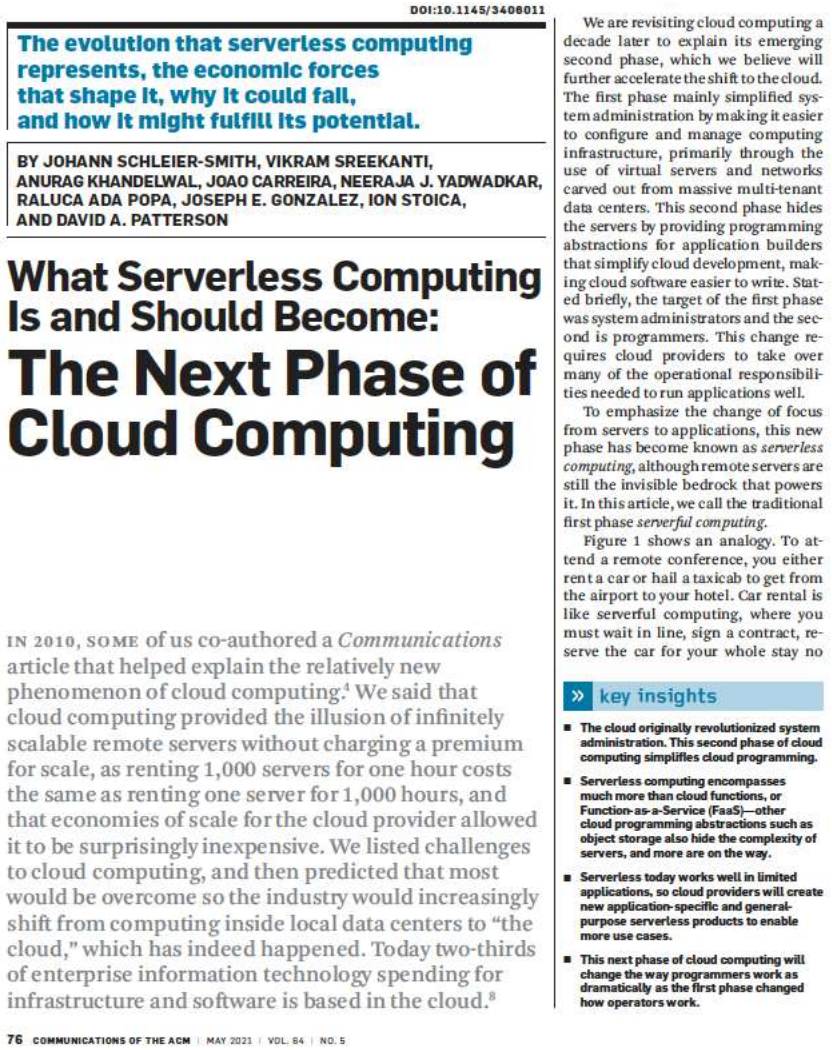
ETH Zurich, Switzerland

Some background

The promise

What Serverless Computing Is and Should Become
CACM'21

https://cacm.acm.org/magazines/2021/5/252179-what-serverless-computing-is-and-should-become/fulltext



DOI:10.1145/3406011

The evolution that serverless computing represents, the economic forces that shape it, why it could fail, and how it might fulfill its potential.

BY JOHANN SCHLEIER-SMITH, VIKRAM SREEKANTI, ANURAG KHANDELWAL, JOAO CARREIRA, NEERAJA J. YADWADKAR, RALUCA ADA POPA, JOSEPH E. GONZALEZ, ION STOICA, AND DAVID A. PATTERSON

What Serverless Computing Is and Should Become:

The Next Phase of Cloud Computing

IN 2010, SOME of us co-authored a *Communications* article that helped explain the relatively new phenomenon of cloud computing.[4] We said that cloud computing provided the illusion of infinitely scalable remote servers without charging a premium for scale, as renting 1,000 servers for one hour costs the same as renting one server for 1,000 hours, and that economies of scale for the cloud provider allowed it to be surprisingly inexpensive. We listed challenges to cloud computing, and then predicted that most would be overcome so the industry would increasingly shift from computing inside local data centers to "the cloud," which has indeed happened. Today two-thirds of enterprise information technology spending for infrastructure and software is based in the cloud.[8]

We are revisiting cloud computing a decade later to explain its emerging second phase, which we believe will further accelerate the shift to the cloud. The first phase mainly simplified system administration by making it easier to configure and manage computing infrastructure, primarily through the use of virtual servers and networks carved out from massive multi-tenant data centers. This second phase hides the servers by providing programming abstractions for application builders that simplify cloud development, making cloud software easier to write. Stated briefly, the target of the first phase was system administrators and the second is programmers. This change requires cloud providers to take over many of the operational responsibilities needed to run applications well.

To emphasize the change of focus from servers to applications, this new phase has become known as *serverless computing*, although remote servers are still the invisible bedrock that powers it. In this article, we call the traditional first phase *serverful computing*.

Figure 1 shows an analogy. To attend a remote conference, you either rent a car or hail a taxicab to get from the airport to your hotel. Car rental is like serverful computing, where you must wait in line, sign a contract, reserve the car for your whole stay no

» key insights

- **The cloud originally revolutionized system administration. This second phase of cloud computing simplifies cloud programming.**
- **Serverless computing encompasses much more than cloud functions, or Function-as-a-Service (FaaS)—other cloud programming abstractions such as object storage also hide the complexity of servers, and more are on the way.**
- **Serverless today works well in limited applications, so cloud providers will create new application-specific and general-purpose serverless products to enable more use cases.**
- **This next phase of cloud computing will change the way programmers work as dramatically as the first phase changed how operators work.**

Slide courtesy of Ana Klimovic (ETH Zurich)

The reality

Serverless Computing: One Step Forward, Two Steps Back

Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti,
Alexey Tumanov and Chenggang Wu
UC Berkeley
{hellerstein,jmfaleiro,jegonzal,jssmith,vikrams,atumanov,cgwu}@berkeley.edu

ABSTRACT

Serverless computing offers the potential to program the cloud in an autoscaling, pay-as-you go manner. In this paper we address critical gaps in first-generation serverless computing, which place its autoscaling potential at odds with dominant trends in modern computing: notably data-centric and distributed computing, but also open source and custom hardware. Put together, these gaps make current serverless offerings a bad fit for cloud innovation and particularly bad for data systems innovation. In addition to pinpointing some of the main shortfalls of current serverless architectures, we raise a set of challenges we believe must be met to unlock the radical potential that the cloud—with its exabytes of storage and millions of cores—should offer to innovative developers.

offers the attractive notion of a platform in the cloud where developers simply upload their code, and the platform executes it on their behalf as needed at any scale. Developers need not concern themselves with provisioning or operating servers, and they pay only for the compute resources used when their code is invoked.

The notion of serverless computing is vague enough to allow optimists to project any number of possible broad interpretations on what it might mean. Our goal here is not to quibble about the terminology. Concretely, each of the cloud vendors has already launched serverless computing infrastructure and is spending a significant marketing budget promoting it. In this paper, we assess the field based on the serverless computing services that vendors are actually offering today and see why they are a disappointment when viewed in light of the cloud's potential.

The business case

Starling: A Scalable Query Engine on Cloud Function Services

Matthew Perron
MIT CSAIL
mperron@csail.mit.edu

Raul Castro Fernandez
MIT CSAIL

David DeWitt
MIT CSAIL
david.dewitt@outlook.com

ABSTRACT

Much like on-premises systems, the natural choice for running database analytics workloads in the cloud is to provision a cluster of nodes to run a database instance. However, an-

that allow
cally, this s
on a per qu
require dat

Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure

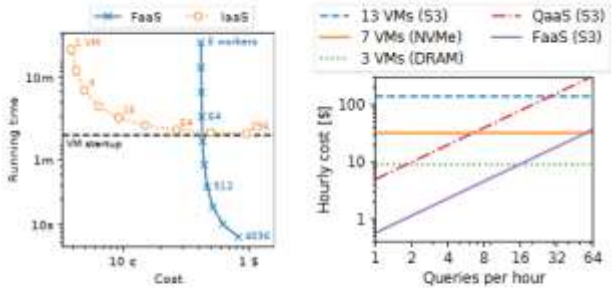
Ingo Müller
ingo.mueller@inf.ethz.ch
ETH Zurich

Renato Marroquín
marenato@inf.ethz.ch
ETH Zurich

Gustavo Alonso
alonso@inf.ethz.ch
ETH Zurich

ABSTRACT

Serverless computing has recently attracted a lot of attention from research and industry due to its promise of ultimate elasticity and operational simplicity. However, there is no consensus yet on whether or not the approach is suitable for data processing. In this paper, we present Lambada, a serverless distributed data processing framework designed to explore how to perform data analytics on serverless computing. In our analysis, supported with extensive experiments, we show in which scenarios serverless makes sense from an economic and performance perspective. We address several important technical questions that need to be solved to support data analytics and present examples from several domains where serverless offers a cost and performance advantage over existing solutions.



(a) Job-scoped resources. (b) Always-on resources.

Figure 1: Comparison of cloud architectures.

1 INTRODUCTION

Understanding serverless

Some data

Serverless in the Wild
USENIX ATC'20

https://www.usenix.org/system/files/atc20-shahrad.pdf

Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini *

Microsoft Azure and Microsoft Research

Abstract

Function as a Service (FaaS) has been gaining popularity as a way to deploy computations to serverless backends in the cloud. This paradigm shifts the complexity of allocating and provisioning resources to the cloud provider, which has to provide the illusion of always-available resources (*i.e.*, fast function invocations without cold starts) at the lowest possible resource cost. Doing so requires the provider to deeply understand the characteristics of the FaaS workload. Unfortunately, there has been little to no public information on these characteristics. Thus, in this paper, we first characterize the entire production FaaS workload of Azure Functions. We show for example that most functions are invoked very infrequently, but there is an 8-order-of-magnitude range of invocation frequencies. Using observations from our characterization, we then propose a practical resource management policy that significantly reduces the number of function cold starts, while spending fewer resources than state-of-the-practice policies.

1 Introduction

Function as a Service (FaaS) is a software paradigm that is becoming increasingly popular. Multiple cloud providers offer FaaS [5, 17, 21, 28] as the interface to usage-driven, stateless (serverless) backend services. FaaS offers an intuitive, event-based interface for developing cloud-based applications. In contrast with the traditional cloud interface, in FaaS, users do not explicitly provision or configure virtual machines (VMs) or containers. FaaS users do not pay for resources they do not use either. Instead, users simply upload the code of their functions to the cloud; functions get executed when "triggered" or "invoked" by events, such as the receipt of a message (*e.g.*, an HTTP request) or a timer going off. The provider is then responsible for provisioning the needed resources (*e.g.*, a container in which to execute each function), providing high function performance, and billing users just for their actual function executions (*e.g.*, in increments of 100 milliseconds).

Obviously, providers seek to achieve high function performance at the lowest possible resource cost. There are three main aspects to how fast functions can execute and the resources they consume. First, function execution requires having the needed code (*e.g.*, user code, language runtime libraries) in memory. A function can be started quickly when the code is already in memory (warm start) and does not have to be brought in from persistent storage (cold start). Second, keeping the resources required by all functions in memory at all times may be prohibitively expensive for the provider, especially if function executions are short and infrequent. Ideally, the provider wants to give the illusion that all functions are always warm, while spending resources as if they were always cold. Third, functions may have widely varying resource needs and invocation frequencies from multiple triggers. These characteristics severely complicate any attempts to predict invocations for reducing resource usage. For example, the wide range of invocation frequencies suggests that keeping resources in memory may work well for some functions but not others. With respect to triggers, HTTP triggers may produce invocations at irregular intervals that are difficult to predict, whereas timers are regular.

These observations make it clear that providing high function performance at low cost requires a deep understanding of the characteristics of the FaaS workload. Unfortunately, there has been no public information on the characteristics of production workloads. Prior work [3, 15, 24, 25, 27, 44] has focused on either (1) running benchmark functions to assess performance and/or reverse-engineer how providers manage resources; or (2) implementing prototype systems to run benchmark functions. In contrast, what is needed is a comprehensive characterization of the users' *real* FaaS workloads on a *production* platform from the provider's perspective.

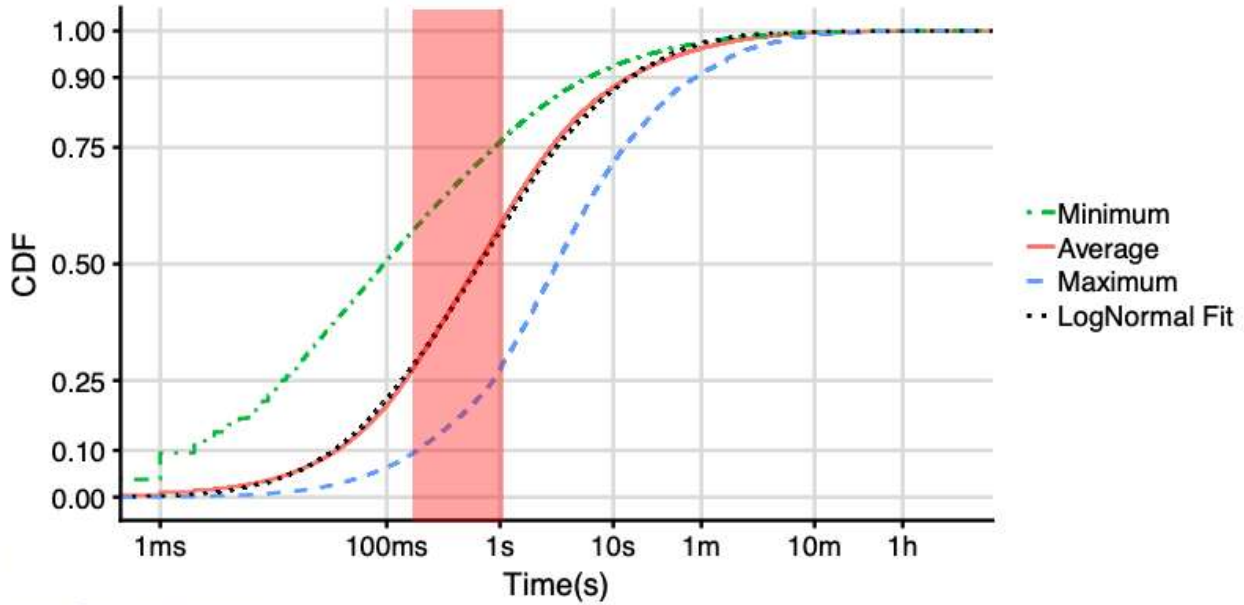
Characterizing production workloads. To fill this gap, in this paper, we first characterize the entire production FaaS workload of Azure Functions [28]. We characterize the real functions and their trigger types, invocation frequencies and patterns, and resource needs. The characterization produces many interesting observations. For example, it shows that most functions are invoked very infrequently, but the most popular functions are invoked 8 orders of magnitude more frequently than the least popular ones. It also shows that functions exhibit a variety of triggers, producing invocation patterns that are often difficult to predict. In terms of resource needs, the characterization shows a 4x range of function memory usage and that 50% of functions run in less than 1 second.

Researchers can use the distributions of the workload characteristics we study to create realistic traces for their work.

*Shahrad is affiliated with Princeton University, but was at MSR during this work. Laureano and Tresness are now with Facebook and D. E. Shaw.

FaaS function characteristics

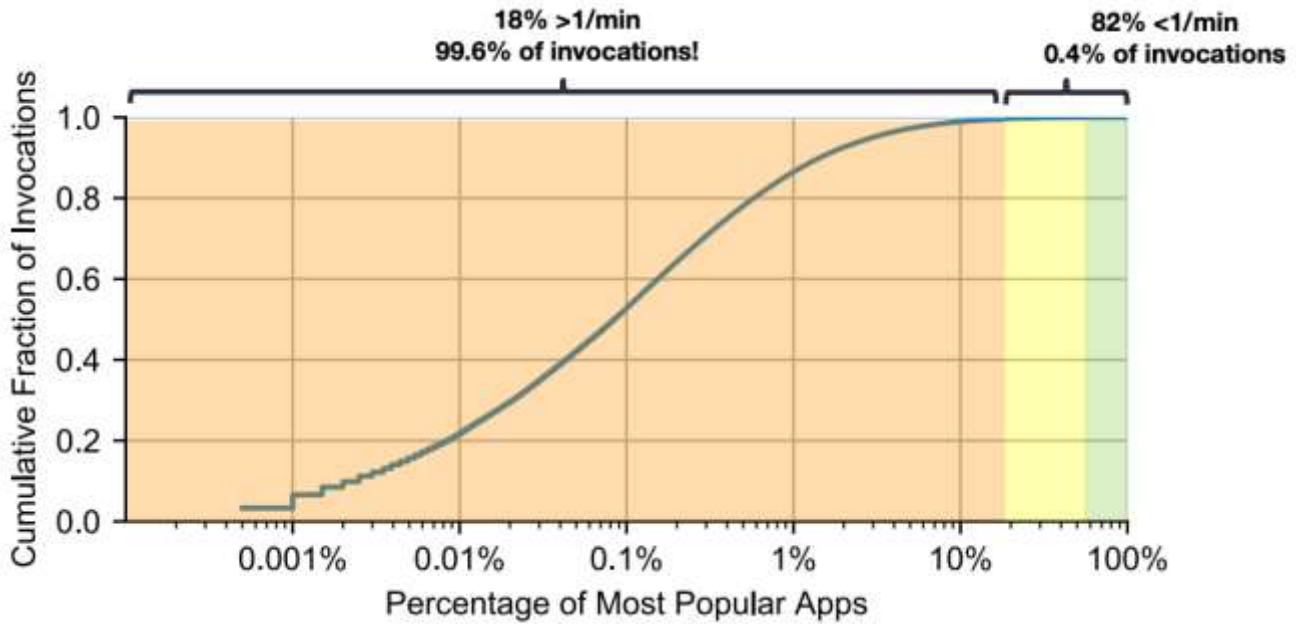
- Short-lived (up to ~15 minutes)
  - Example from Azure Functions:



- Executions are short
  - 50% of apps on average run for <= 0.67s
  - 75% of apps on run for <= 10s max

FaaS function characteristics

- Skewed invocation frequency
  - Example from Azure functions



FaaS function characteristics

- Short-lived (up to ~15 minutes)
- Small resource footprint (e.g., up to 3GB RAM)
- Skewed invocation frequency
- Stateless

Provider side

- Scheduling at this granularity is a potential challenge: many functions with a short lifetime (15 minutes) that need to be allocated
- Start up time becomes a problem because the functions typically run for a short time (in milliseconds): keep functions warm, reserve containers, etc.
- Not compete with other deployment forms (compute time in serverless much more expensive than in VMs, creates market for short lived computations but makes no sense for services running continuously)

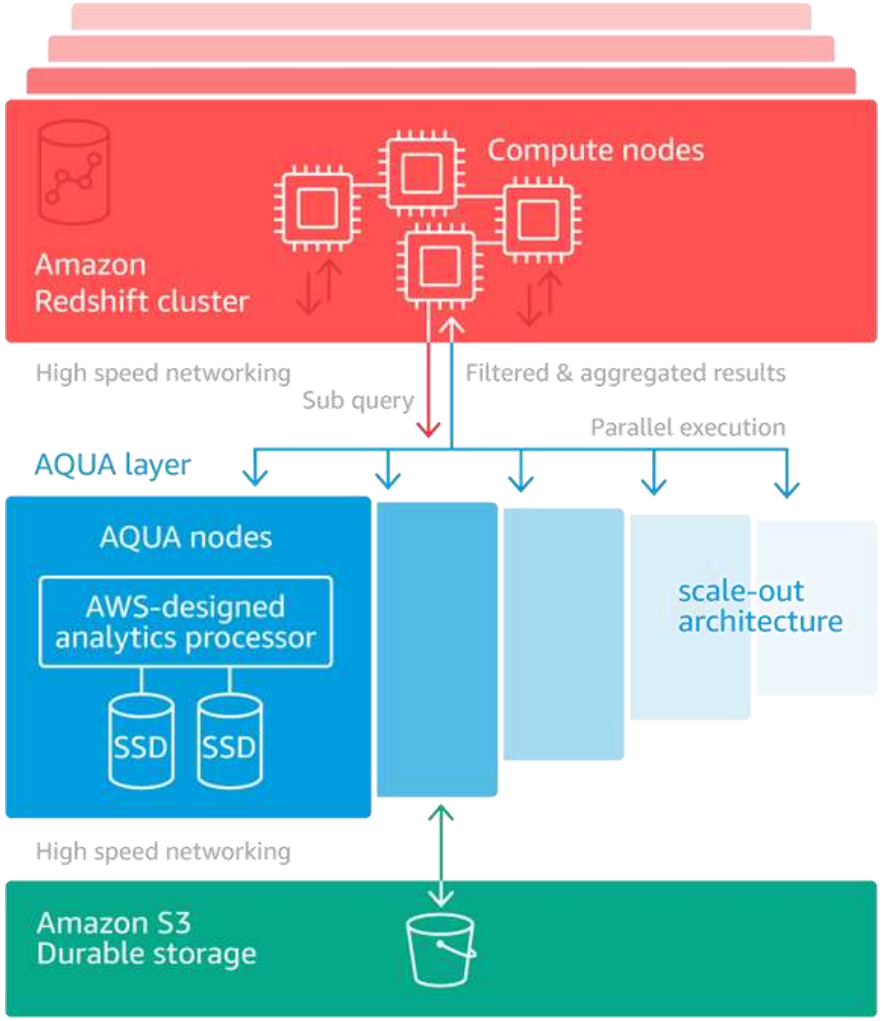
The question that needs to be asked

Does it even make sense to run data analytics on current serverless infrastructure?

Data Analytics

- Heavy queries
- Not necessarily repetitive (use case dependent)
- Lots of data involved
- Potentially heavy algorithms (e.g., out of core sorting)
- Scatter-gather patterns very common
- …

Speeding data analytics in the cloud



Useful things that can be done …

- Functions can be used to extend an engine

Amazon Redshift announces support for Lambda UDFs and enables tokenization

Posted On: Oct 26, 2020

Amazon Redshift, a fully-managed cloud data warehouse, now supports Lambda user-defined functions (UDFs) enabling you to use an AWS Lambda function as a UDF in Amazon Redshift. This functionality enables you to write custom extensions for your SQL query to achieve tighter integration with other services or third-party products. For example, you can write Lambda UDFs to enable external tokenization of data by integrating with vendors like Protegrity, or access other services such as Amazon DynamoDB or Amazon SageMaker in your Redshift query.

Using Amazon Redshift Lambda UDFs, you can register an AWS Lambda function as UDF in Amazon Redshift cluster, and invoke the UDF from Redshift SQL queries. You can include more powerful operations in the AWS Lambda function such as accessing storage or network resources, which will allow you to integrate with external services. Lambda UDFs can be written in any of the programming languages supported by AWS Lambda, such as Java, Go, PowerShell, Node.js, C#, Python, Ruby, or a custom runtime. You can use Lambda UDFs in any SQL statement such as SELECT, UPDATE, INSERT, or DELETE, and in any clause of the SQL statements where scalar functions are allowed.

Useful things that can be done

Using Cloud Functions as Accelerator for Elastic Data Analytics

HAOQIONG BIAN, EPFL, Switzerland
TIANNAN SHA, EPFL, Switzerland
ANASTASIA AILAMAKI, EPFL, Switzerland

Cloud function (CF) services, such as AWS Lambda, have been applied as the new computing infrastructure in implementing analytical query engines. For bursty and sparse workloads, CF-based query engine is more elastic than the traditional query engines running in servers, i.e., virtual machines (VMs), and might provide a higher performance/price ratio. However, it is still controversial whether CF services are good suites for general analytical workloads, in respect of the limitations of CFs in storage, network, and lifetime, as well as the much higher resource unit prices than VMs.

In this paper, we first present micro-benchmark evaluations of the features of CF and VM. We reveal that for query processing, though CF is more elastic than VM, it is less scalable and is more expensive for continuous workloads. Then, to get the best of both worlds, we propose Pixels-Turbo - a hybrid query engine that processes queries in a scalable VM cluster by default and invokes CFs to accelerate the processing of unpredictable workload spikes. In the query engine, we propose several optimizations to improve the performance and scalability of the CF-based operators and a cost-based optimizer to select the appropriate algorithm and parallelism for the physical query plan. Evaluations on TPC-H and real-world workload show that our query engine has a 1-2 orders of magnitude higher performance/price ratio than state-of-the-art serverless query engines for sustained workloads while not compromising the elasticity for workload spikes.

Advancing serverless

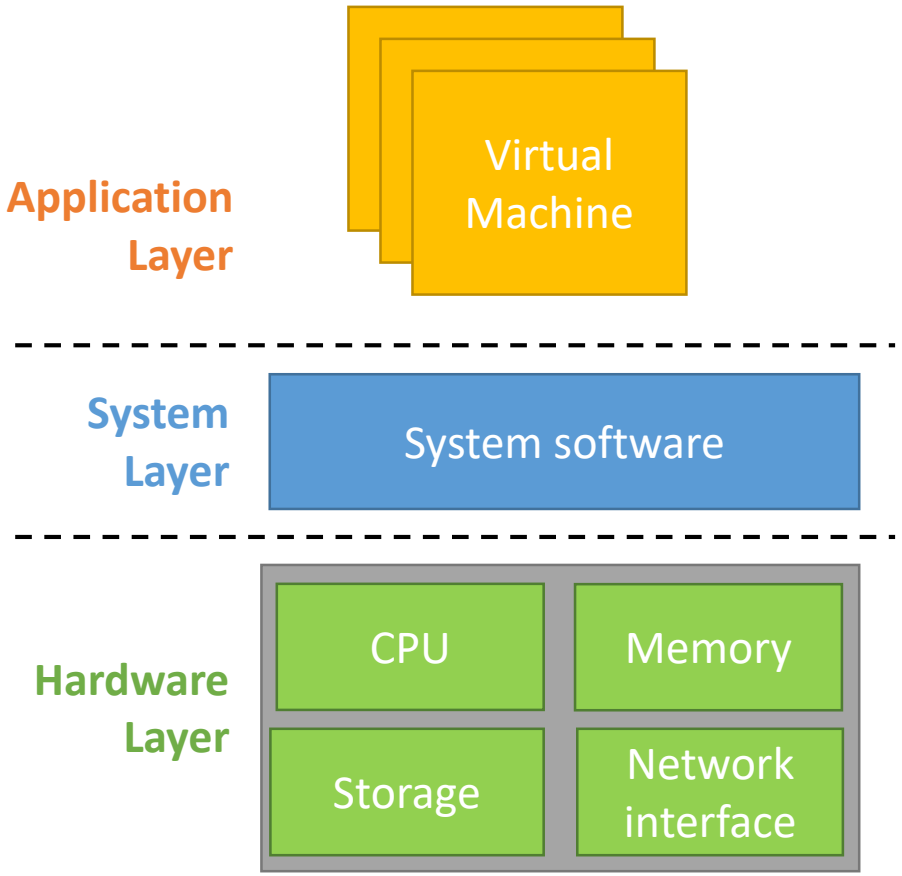
Serverless can be done better

- Current offerings are bound to legacy systems
- Many spurious constraints
  - Technical constrains => stateless
  - Business case constraints => pricing
  - Resource constraints => memory allocation
  - Cloud provider vs cloud user constraints => what problem are we solving?
- From a research perspective, we should not ignore reality, but we should not let spurious constraints dominate the agenda

Two ideas from our own research

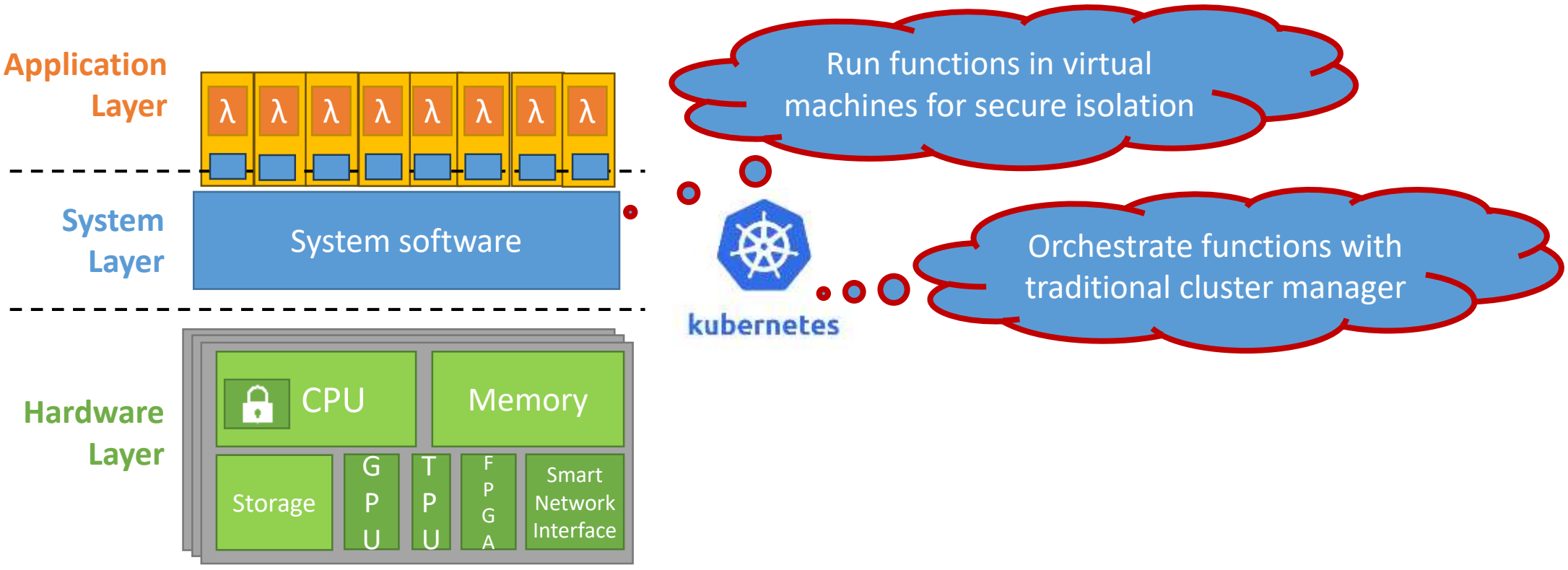
- Redesign the stack => Dandelion
- Enable networking => Boxer (and see presentation later)

The cloud stack today



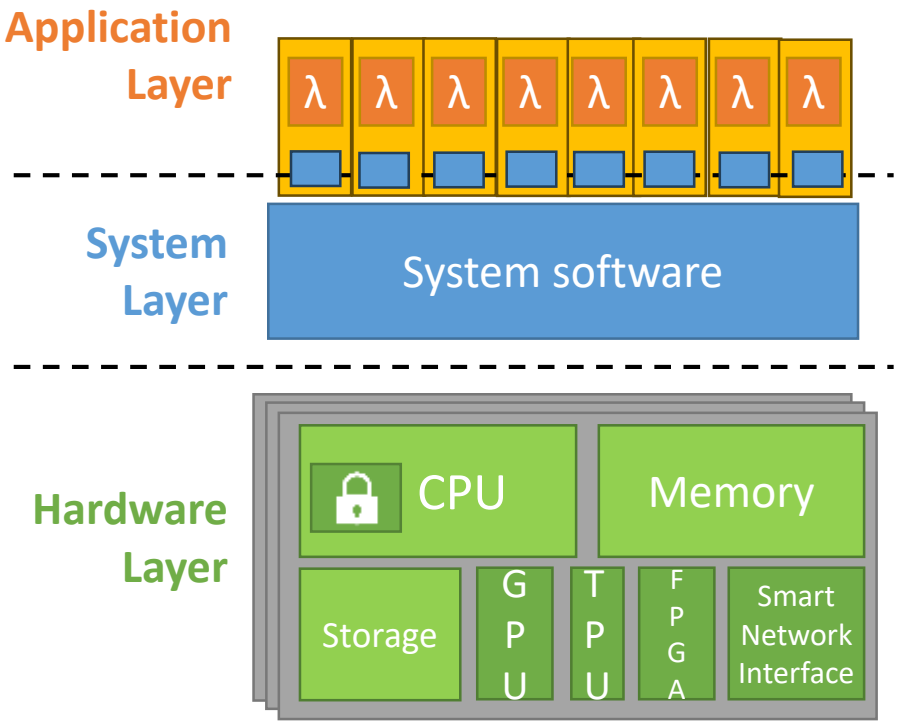
Slide courtesy of Ana Klimovic (ETH Zurich)

Current approach: re-use existing system software



Slide courtesy of Ana Klimovic (ETH Zurich)

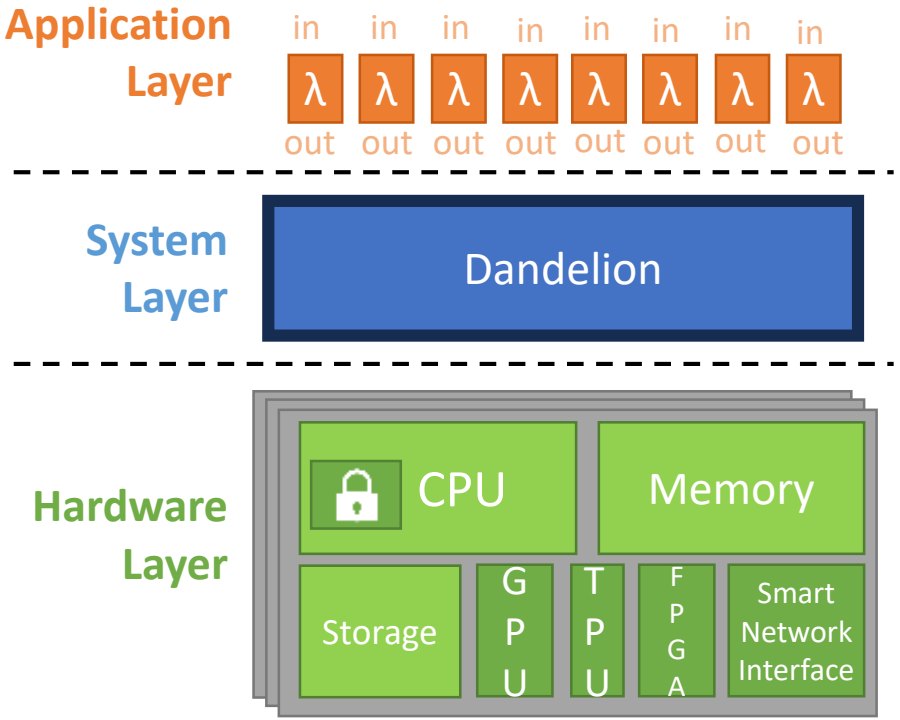
Current approach has several key problems



- ✗ high latency to boot functions
- ✗ high memory overhead
- ✗ cluster manager becomes bottleneck at high load
- ✗ limited scheduling optimizations as function dependencies are not known to the platform
- ✗ no support for heterogeneous hardware

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform

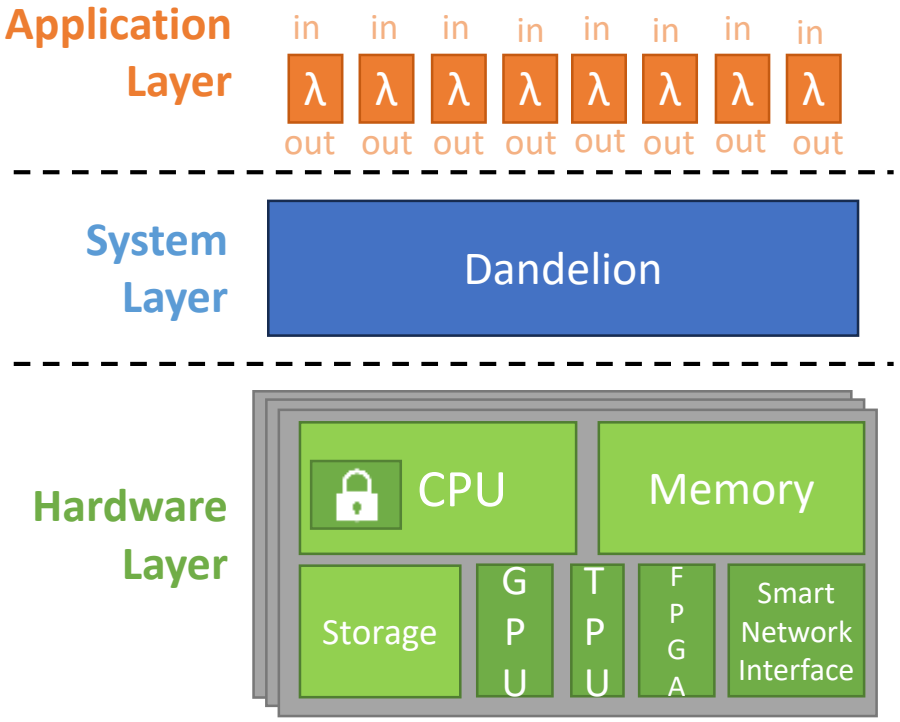


KEY IDEA: *treat functions as functions!*

Function = snippet of **code** that computes on *declared **inputs*** and produces *declared **outputs***

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

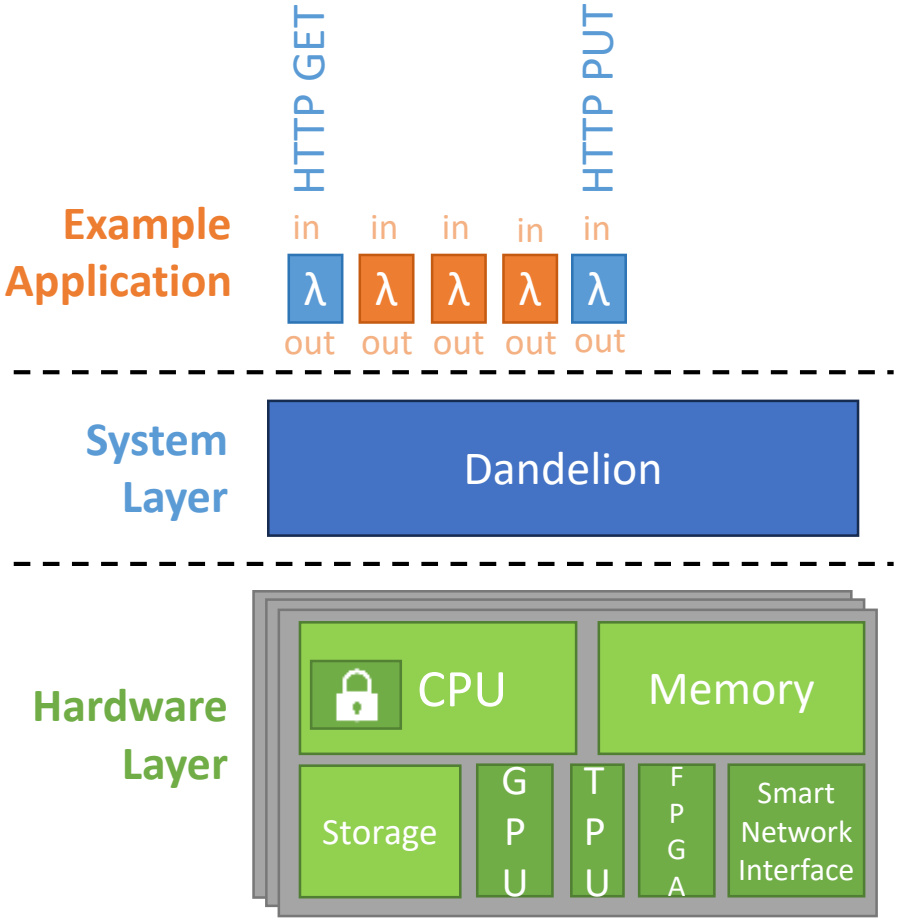
Function = snippet of **code** that computes on *declared **inputs*** and produces *declared **outputs***

Application = composition (DAG) of compute functions (*untrusted* user code) & I/O functions (*trusted* platform code)

↳ enable interaction with external storage services and between user compute functions

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

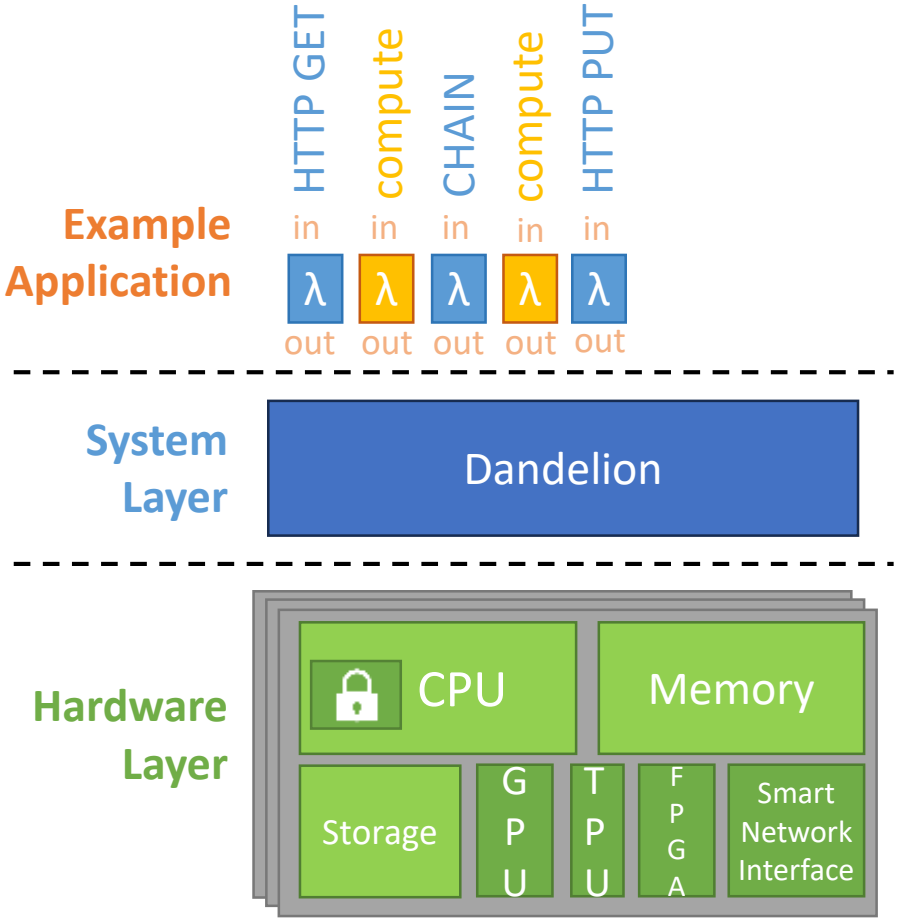
Function = snippet of **code** that computes on *declared **inputs*** and produces *declared **outputs***

Application = composition (DAG) of compute functions (*untrusted* user code) & I/O functions (*trusted* platform code)

↳ enable interaction with external storage services and between user compute functions

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

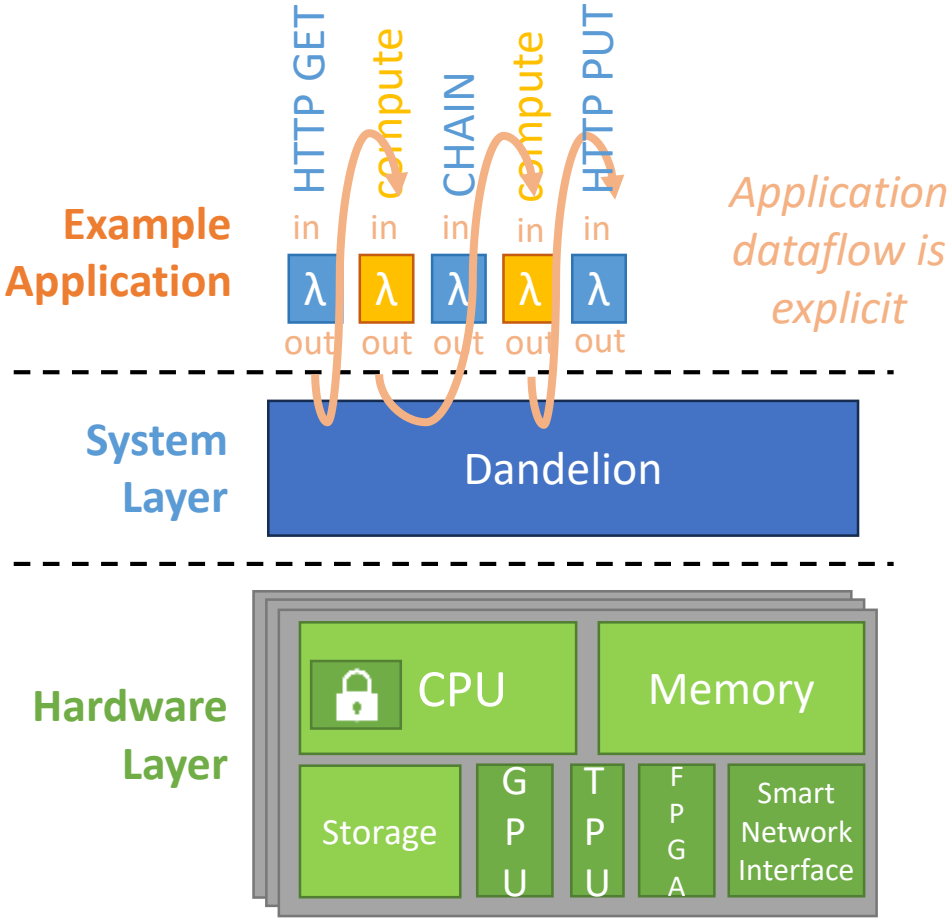
Function = snippet of **code** that computes on *declared **inputs*** and produces *declared **outputs***

Application = composition (DAG) of compute functions (*untrusted* user code) & I/O functions (*trusted* platform code)

↳ enable interaction with external storage services and between user compute functions

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

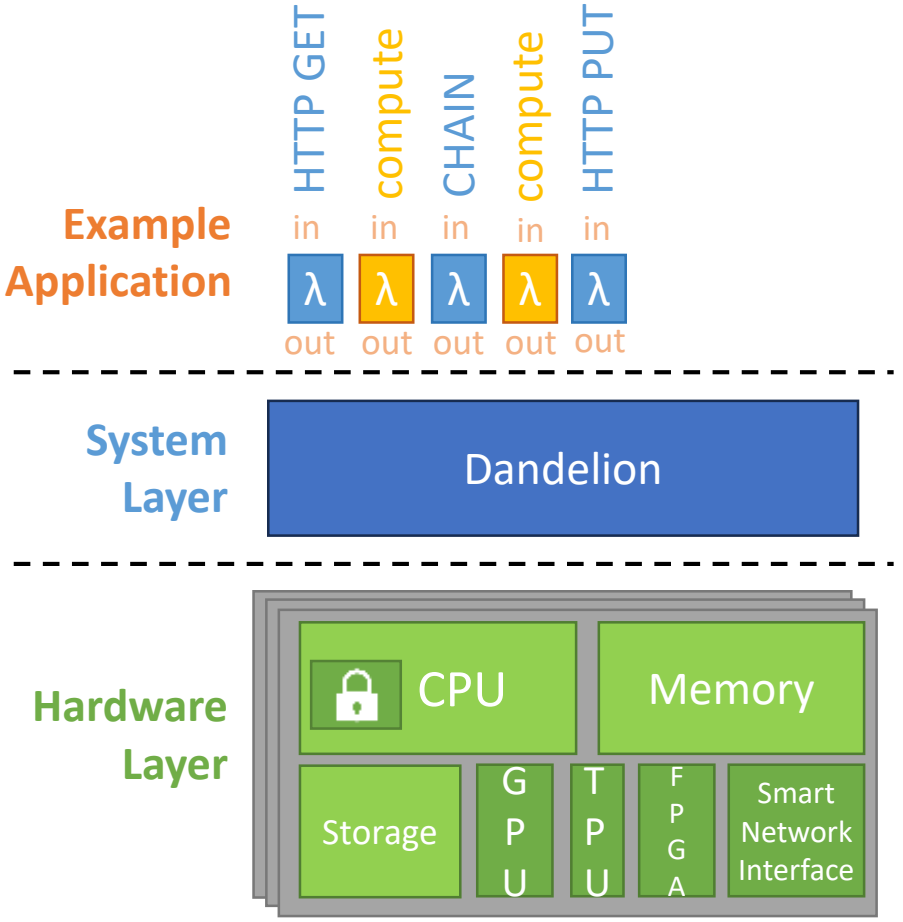
Function = snippet of **code** that computes on *declared **inputs*** and produces *declared **outputs***

Application = composition (DAG) of compute functions (*untrusted* user code) & I/O functions (*trusted* platform code)

enable interaction with external storage services and between user compute functions

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

Benefits:

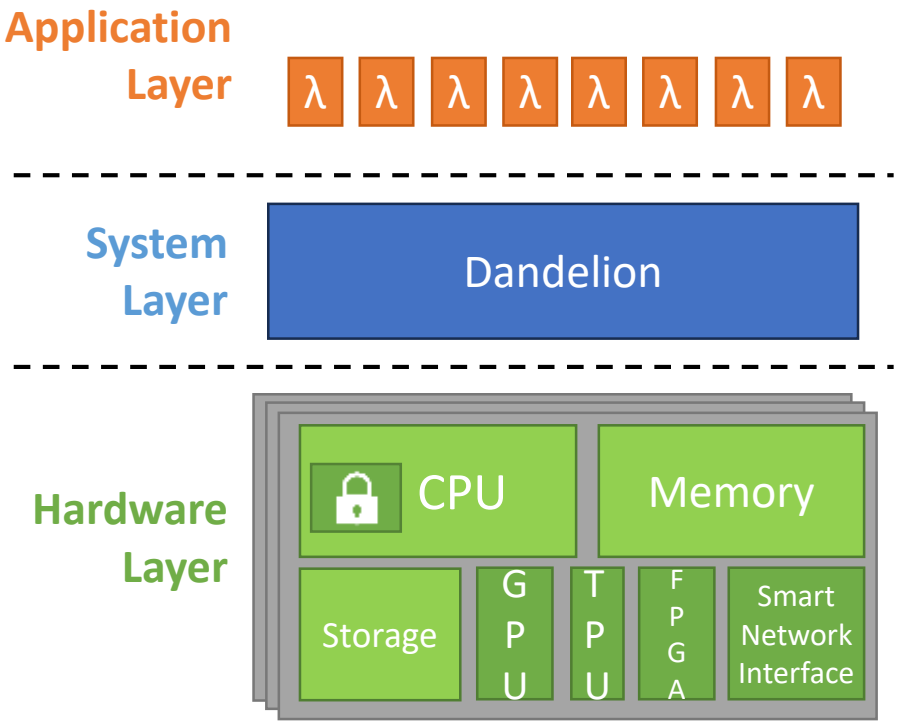
✓ Securely isolate functions without VMs (reduce attack surface by eliminating syscalls in user code)

✓ Optimize function scheduling with dataflow info

✓ Offload pure compute and pure I/O tasks to heterogeneous hardware

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion: a new FaaS platform



KEY IDEA: *treat functions as functions!*

ETH Team:

Tom Kuchler

Ana Klimovic

Slide courtesy of Ana Klimovic (ETH Zurich)

Dandelion = Truly serverless

- The key element of serverless is that it hides a complex infrastructure, making it easier to use
  - But the way it is done is by treating functions as black boxes
- Declarative Serverless
  - Functions declare their needs: I/O, communication, connectivity, resources needed, etc.
  - The serverless platform uses that information to optimize the deployment while providing a far better support and incurring less overhead in enforcing important features: isolation, performance, resource efficiency, etc.

Data Analytics on Serverless

Option 1 (redesign the engine)

- Build platforms that can run queries on current serverless
  - Hack around limitations
  - Accept constraints
  - Significant development cost
  - Unstable environment
  - …

Option 2 (replace the stack)

- Build a better serverless platform
  - Tailor it to wider use cases
  - Not supported by vendors
  - Higher costs
  - Often not truly serverless
  - …

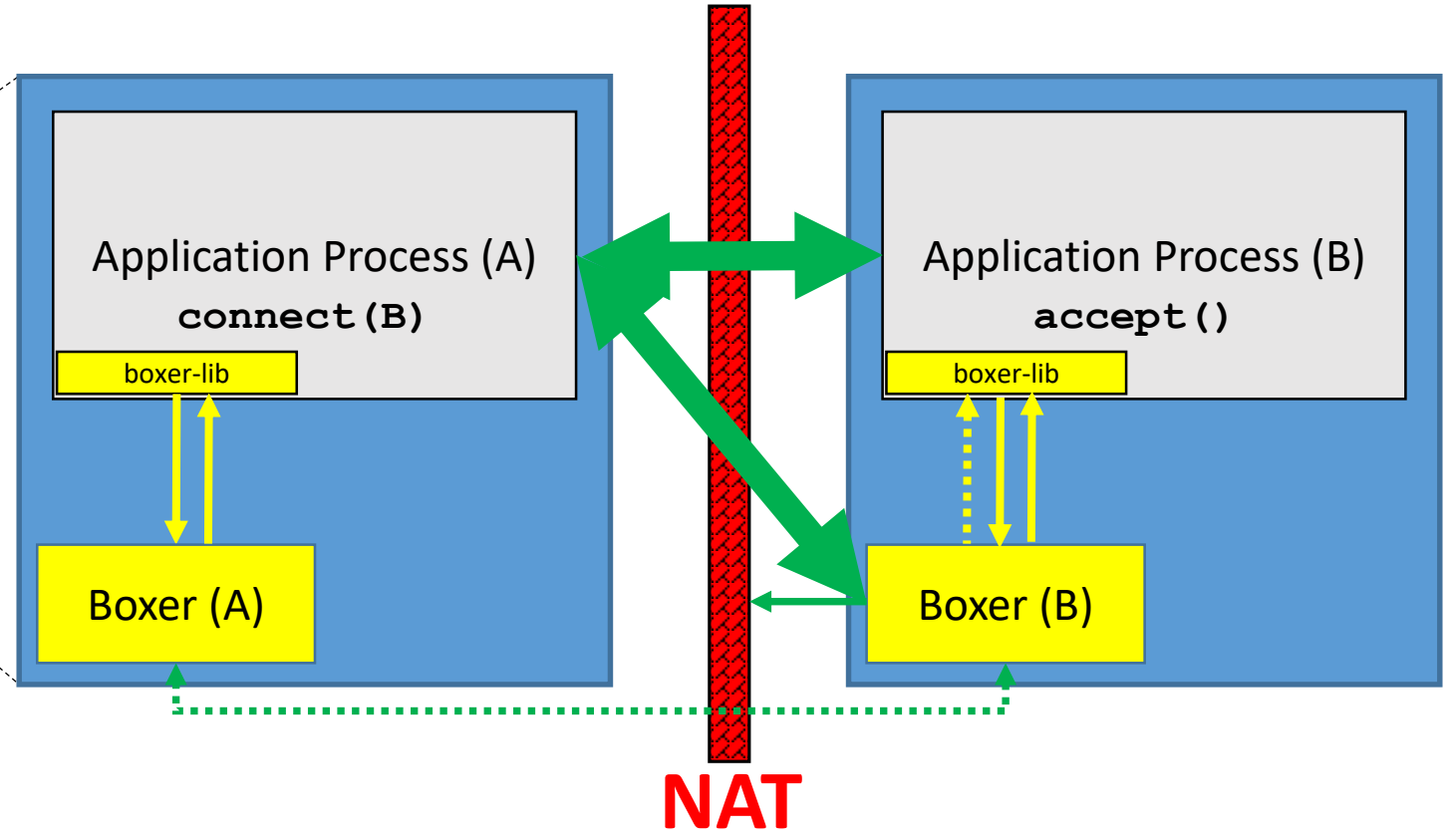
Data Analytics on Serverless

Option 3 (use what is there)

- There are very many data analytic engines in the cloud
- Do we want to redo all that work?
- What if we can just run existing analytic engines of current serverless?

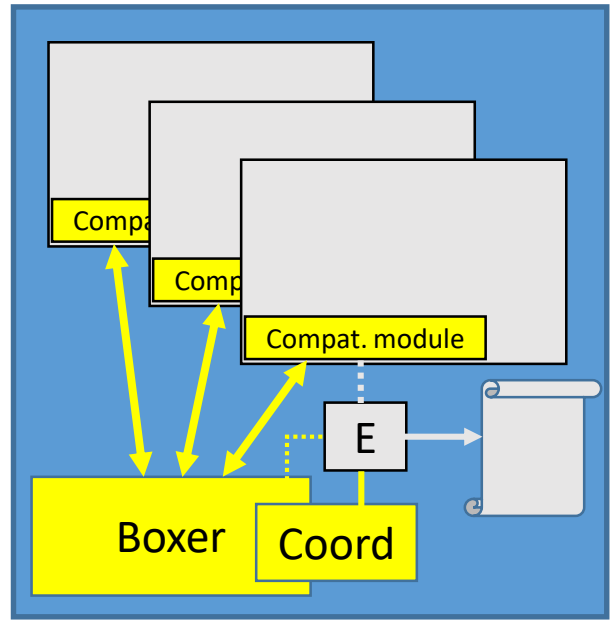
Boxer: Achieving TCP in Serverless

- Boxer NAT hole-punching service
  - For local connection request
  - For remote hole punch request
- Transparent compatibility module
  - Intercepts relevant libc calls SOCKET, BIND, CONNECT, LISTEN, ACCEPT, CLOSE
  - Transparent to function process
- **Boxer not involved after TCP connection established** (no SEND, RECV, EPOLL etc.)

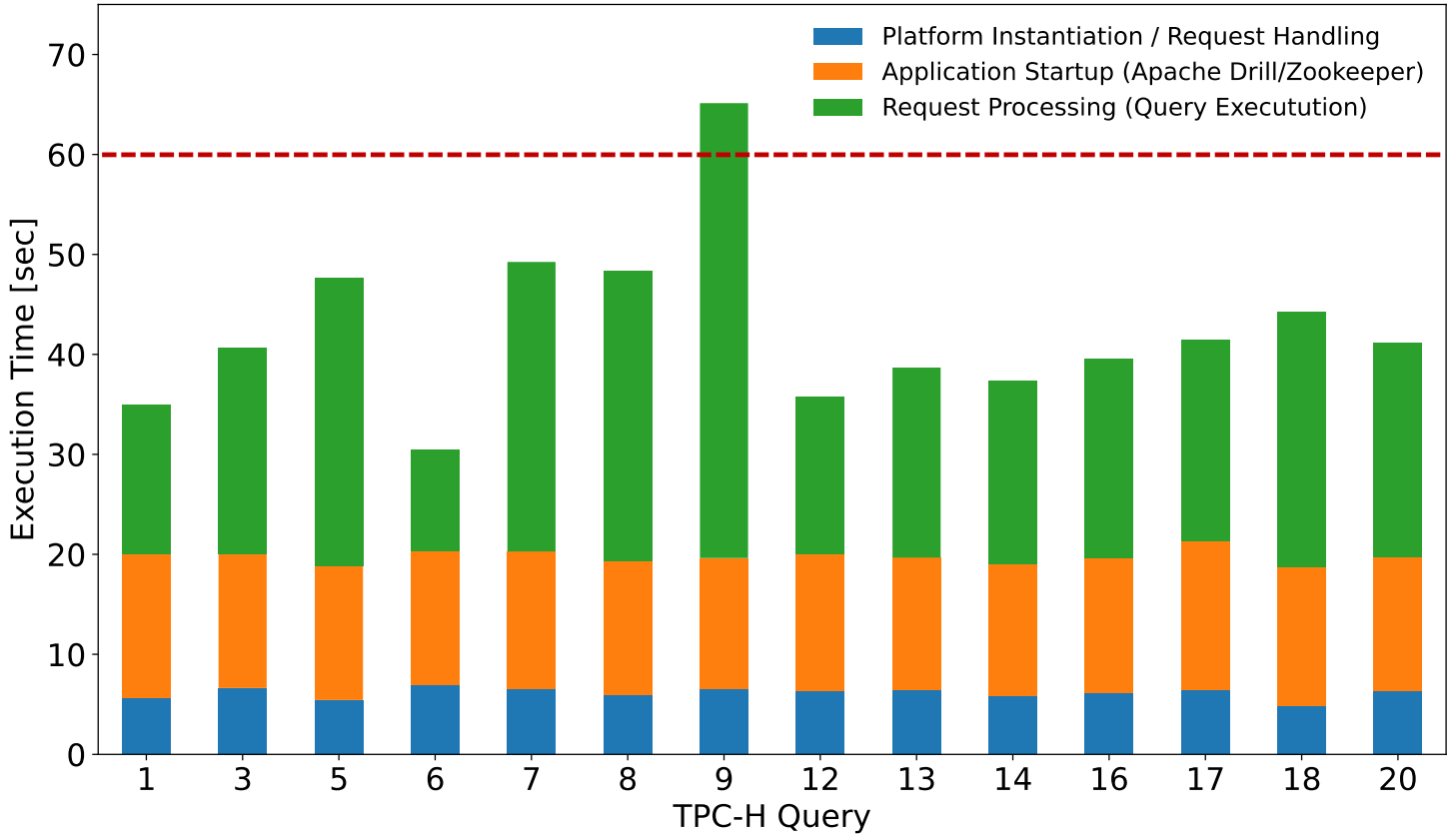


… more than just networking

- Hostname resolution,
- File system redirection, …
- Coordination service
  - Distributed barriers,
  - Node membership,
  - Process id and role assignments, …



Per-request Datacenter



- TPC-H benchmark
  sf.10 - 12GBytes,
  largest relation of almost 60 million tuples.
- Data stored in S3
- 8-node Apache Drill + Zookeeper
- Off-the-self configuration, not optimized

Data Analytics on Serverless

- Boxer implements an overlay that makes serverless look like a “regular” VM
- Analytic engines run unmodified and can be used to run any query in the same way they run on VMs
- It is not perfect:
  - The engines are not optimized for serverless (e.g., fast startup)
  - There are many things that reduce efficiency (e.g., resource waste)
- But this already gives us analytics on serverless
- Or does it?

Conclusions

Serverless as the new autonomous DB

- Serverless is here to stay but it will evolve in significant ways
- Currently, not suitable for data analytics
  - But still worthwhile to explore the space
- We are exploring two approaches:
  - A radical redesign of the serverless stack (from the provider perspective)
  - An incremental extension of the serverless stack to facilitate the transition
- Boxer allows to run existing engines on serverless to explore the space and get important empirical data on what happens when running analytics on serverless