C10 Vendor lock-in

Migrating a serverless application from one cloud provider to another cloud provider is very time-intensive and often requires partial rearchitecting of the application. Serverless offerings are mostly built on top of proprietary software instead of open-source solutions. This means that there is little to no compatibility between, e.g., the blob storage offerings of two cloud providers, which leads to the commonly reported vendor lock-in for serverless applications.

Closing Statement

The group discussed the changes to the traditional software engineering lifecycle from the perspective of software engineers that are responsible for the development and operation of software systems running on serverless cloud platforms. Based on these changes, the group identified a number of challenges for the planning, design, implementation and operation of serverless applications. While the discussion focussed mostly on these challenges, the group is confident that they can be overcome by a combined effort from industry and academia.

4.3 Serverless Applications and Requirements (Topic 3)

Josef Spillner (ZHAW – Winterthur, CH), Bartosz Balis (AGH University of Science & Technology – Krakow, PL), Jesse Donkervliet (VU University Amsterdam, NL), Nicola Ferrier (Argonne National Laboratory, US), Ian T. Foster (Argonne National Laboratory – Lemont, US), Maciej Malawski (AGH University of Science & Technology – Krakow, PL), Panos Patros (University of Waikato, NZ), Omer F. Rana (Cardiff University, GB), and Florian Wamser (Universität Würzburg, DE)

Why/when should applications be serverless?

As it stands, Serverless Computing expands on state-of-the-art cloud computing by further abstracting away software operations (ops) and larger parts of the hardware/software stack. One could consider functions, the execution unit of serverless computing, as "lightweight" containers, invoked with a set of inputs and expected to produce a set of outputs, when triggered.

From a user perspective, Serverless reduces system operation effort, simplifies development, supports highly variable and unpredictable workload patterns, enables the complete removal from dynamic memory of applications not in use – referred to as Scale to zero – and, under the right circumstances, can reduce software operation cost. From an operator perspective, it reduces costs by increasing resource efficiency and crucially, it incentivizes innovation for sustainability because the operator bears the cost of idleness.

Considering both the current state-of-the-art of serverless computing as well as its expected evolution over the year, this report aims to identify the types of applications that are currently well-supported by today's serverless platforms, and then, move on to discuss novel and upcoming applications with challenging characteristics, which would require serverless to evolve in order to satisfy them.

What are the unique characteristics?

We identified the following four unique characteristics (UC) for current serverless:

- UC-1 Stateless/Idempotency, which describes the pure-function behavior of invocations;
- UC-2 Fixed Memory, which limits the amount of resident memory an invocation is allowed to have;
- UC-3 Short Running, enforcing a short time limit on the execution of invocations;
- UC-4 Little Control, referring to the abstraction of ops, such as scheduling and autoscaling, from the user.

The transparency trade-off

However, are software engineers and problem owners ready to relinquish all this control of their applications? We identified a tradeoff between resource abstraction and resource control, essentially a tug-of-war between ease of programming vs. efficiency and cost.

This could be mitigated by exposing tuning knobs, such as resource management, to use by FaaS developers, a concept inspired by "open implementation analysis and design" [Maeda, Murphy, Kizales, 1997]. For a more technical example, consider the developer passing hints from the application through an interface exposed by the backend stack. This could be incarnated by pragmas, event interface, rate-limiting contracts, etc.

All in all, is it worthwhile to exchange ease of programming, deployment, maintenance and operation, to enable fine-grained control for developer customization? From a platform design perspective, such a requirement endangers efficiency in application and backend. However, it could help FaaS providers with resource allocation and scheduling decisions, while saving cost. Thus, the question persists if serverless should be even enabling any type of ops.

Scoping applications on their path towards serverless

The suitability of serverless computing concepts to deliver application functionality opens a maturity-chronology spectrum associated with application enablement. This spectrum can be roughly divided into three serverless phases: "Serverless 1.0" starting around 2014, "Serverless 1.5" representing the state of commercially available technology in 2021, "Serverless 2.0", bringing finer granularity and control in the coming years. Potentially more phases (3.0, 4.0, ...) will follow that are currently unclear but may nevertheless still not be sufficient for certain types of applications.

For some early adopter applications in the "serverless 1.0" phase, the initial serverless concepts around FaaS (λ, OW/ICF, GCF, AF) in the mid-2010s were already suitable. Further applications have been enabled recently by an expanded set of serverless computing offerings including FaaS-alike flavours of CaaS [GCR, Fargate, IBM CodeEngine, ACS/Dapper that permit stateful tasks/inter-instance communication/multiple CPUs, academic approaches like funcX], relaxed limits in FaaS invocations, and low-latency BaaS that characterise "serverless 1.5".

In the near future, based on recent scientific progress, "serverless 2.0" will make it easier to build applications that currently require unaffordable effort [e.g. ExCamera, deep learning, NumPyWren], and will furthermore allow for new classes that are currently unreachable [e.g.

online gaming, federated learning, agritech on the edge]. This trend will be driven primarily by four factors:

1. Hardware advances such as disaggregation and continuums; including better heterogeneity, specialised hardware (GPUs, TPUs), storage, networking (mobile radio heads)
2. Changes in BaaS, primarily the ubiquitous ability to run functions next to data (fusing the concepts of FaaS, stored procedures in DBs, UDFs in big data tools)
3. Autonomic middleware assisting the decomposition and placement of application code into managed services
4. Improvements in the design of serverless platforms, including
   a. selected control knobs for applications (possibly with some declarative language),
   b. including "only" a maximum runtime instead of a fixed short runtime (see "elastic" execution time for applications),
   c. differentiated and guaranteed quality (QoS or QoE guarantees) including real-time constraints (see also [RTserverless]), as well as
   d. aligned mature toolsets to convey the platform benefits directly to software engineers including testing, tracing and debugging, covering the entire DevOps cycle.

Application domains

The main application domains we discussed are:

1. Scientific Computing
2. Machine Learning and Artificial Intelligence
3. Online Gaming
4. Mobile Serverless and Telecommunication
5. Big Data Analysis
6. IoT, Agriculture and Cyber Physical Systems
7. Web Services

Below we focus on the selected four domains as representing the key challenges for current and future serverless platforms.

Domain: Scientific computing

Why use serverless?

Modern science relies on large scale experiments, simulations and data-driven analysis methods. Scientists analyze time series of global archives, often on the order of hundreds of Terabytes up to Petabytes, and hundreds of thousands of data images in order to generate a single layer of global, sometimes geospatial, information with added value. For this processing, extreme performance and often local processing is required. Computational requirements cover the full spectrum – from functions providing "support" for HPC environments to initial (approximate) analysis closer to the point of data capture. Many tasks are available as functions, can be reused and are available in R, R-Shiny Apps, Python, while we can think of larger HPC-jobs as "fat" functions that can be also considered serverless.

What can serverless provide today?

The serverless model is appealing for scientists because of the ease of programming, whereby scientists can focus on implementing scientific procedures, easily collaborate on function development, and reuse existing functions. Many scientific applications include fine-grained tasks, e.g. high-throughput scientific workflows, machine learning tasks, or interactive analytics.

Scheduling of tasks

Resource allocation associated with serverless platforms is highly dynamic and elastic, so the scientists can gain quick on-demand access to computing resources suitable for exploratory interactive data analysis, processing of streaming data from instruments etc. It is noteworthy that in the context of scientific computing the cold-start problem or high start-up times typical, e.g., for serverless containers, are less significant in comparison to job queue wait times in HPC systems. Accelerated time-to-science is thus another potential advantage of serverless computing applied to scientific use cases. Scientific applications are diverse in terms of software, complex dependencies on libraries, and packages, often requiring legacy software, so current approach to containerisation, deployable to serverless CaaS, is a perfect solution to these problems.

Challenging application requirements

Dynamic provisioning (on-demand access) is radically different from the typical batch-queue model used in scientific computing. Moreover, scientific computing often involves long-running tasks with high memory usage, while cloud functions currently are not suited to run as long as batch jobs and have fixed memory limits. Scientific applications often rely on specialized hardware, including all types of accelerators (GPU, potentially TPU for tensor tasks) and fast I/O (burst buffers, nvram) which are available in state-of-the-art HPC systems but not in cloud datacenters. For large-scale tightly-coupled parallel simulations fast interconnects and communication substrates are required (MPI), for which workarounds like using cloud storage or other means are now developed (NumPyWren), but need better solutions in the future. Scientific pipelines (workflows) benefit from data locality, difficult to achieve with stateless functions.

Ultimate vision

We envision that with Serverless 2.0 some of these requirements will be soon fulfilled, but the ultimate goal of "Serverless Supercomputer" will be possible not earlier than with the advent of "Serverless 4.0" era, where intra-datacenter latency will match the current leadership HPC interconnects and the distinction between a datacenter and supercomputer will disappear.

Domain: Machine learning and artificial intelligence

What are the characteristics of this domain?

Machine Learning is an emerging area in function-based processing, combining both learning on edge devices combined with inference-based models (MobileNetV1, MobileNetV2 and Faster R-CNN – i.e. pretrained models on cloud systems) that can be deployed on such

devices.[7] These computationally reduced versions of machine learning algorithms provide great opportunities for deploying function-based processing. Conversely, a number of hardware vendors (e.g. NVidia, Huawei, Intel, etc) are increasingly developing hardware accelerators aimed at improving the performance of machine learning algorithms, these range in complexity from support for specialist data structures (e.g. matrices and matrix/vector manipulation), to inclusion of specialist dedicated hardware that can be used to improve processing of data associated with machine learning algorithms (e.g. video analysis). Understanding how serverless approaches can be used to deploy (sustainably – combining both energy and economic efficiency) ML functions can be used to support a variety of different types of applications. To provide an example: the size of the models, number of parameters and computational complexity of these two MobileNet models include: MobileNetV1 (570M Multiply-Accumulate (MACs) and 4M parameters (which can include weights connecting layers and other model parameters such as learning rate)); MobileNetV2 (300M MACS, 3M parameters). Understanding benchmarks that can be used to characterise performance (and accuracy) of ML algorithms, realised as functions on edge devices is also being undertaken within the MLCommons Consortium (bringing together academia and industry). The benchmarks being proposed in this work could directly be used to undertake capacity planning for serverless implementations of ML functions.

ML functions can also vary in their computational time requirements – from algorithms that need to execute over long time frames (e.g. multiple days) to process different input data, to others that can be used to pre-process data prior to processing (<1min). Additionally, other ML pre-processing functions can be triggered by events observed in the environment (e.g. availability of sensor data, movement of people etc). Understanding where the serverless paradigm aligns with ML function implementation is an important consideration – as not all of these functions may be suitable to be realised using the serverless paradigm (especially when considering the economics of deployment). The following diagram demonstrates the possible mechanisms for distributing ML functions using serverless approaches: (i) we can partition the data (sharding of a data stream); (ii) partitioning a model (e.g. with the use of federated learning, where multiple models are independently constructed, and then integrated at a central site); (iii) aggregating the outcome of multiple functions and combining this with additional parameter optimisation using a cloud-based backend server.

What could serverless provide today?

Today's serverless can or, with modest system tweaks, could support ML and open up a number of opportunities in providing:

- Function-based implementation of ML algorithms at different levels of complexity, from ML that can be deployed within a data centre to functions at the edge;
- Programming support for implementing ML functions and developing software libraries that can be used to realise functions. A variety of libraries already exist – such as TF-Lite, use of "distillation" and quantization approaches to reduce the complexity of learned models to deploy over resource constrained environments. Another similar approach is the ability to migrate functions between edge and cloud resources – e.g. use of Osmotic computing approaches that enable migration of functions as lightweight containers;
- Deployment mechanisms that can be used to place ML functions across the IOT-edge-cloud continuum. Function placement driven by performance, cost and energy constraints can provide a useful basis for making more effective use of these within other application areas;

[7] https://dl.acm.org/doi/10.1145/3398020

- Serverless utilising increasingly available hardware accelerators – support for "hardware aware" function optimisation
- Specialist compilers that are able to create serverless functions that can be adapted to hardware characteristics

Challenging application requirements

Some of the characteristics and limitations of available serverless systems remain important open challenges, such as:

- Need to support often long-running functions that may have high memory and I/O requirements. Understanding whether a serverless approach would be most relevant in his context, and where alternative approaches may be more suitable for such deployments. Another challenge in this context would be understanding how to partition machine learning algorithms or general workloads across the iot-edge-cloud continuum.
- Need to support observability and manageability of functions, especially if these ML functions are part of other applications, for instance using a learning algorithm that is used as a component within a larger workflow. In this context, understanding the level of "control" a user has on configuring and deploying these ML functions remains an important overall consideration
- A deployment environment, e.g. as used in funcX/Parsl to dynamically deploy ML functions based on user demand, and aligned with the characteristics of the hardware platform. Matchmaking between function characteristics and hardware device properties also remains an important research challenge to increase adoption.
- Secure and privacy-aware ML functions, especially when dealing with sensitive data that may have GDPR/data privacy constraints, is also an important requirement for serverless deployment. Using encrypted data (e.g. using fully or partially homomorphic encryption) or utilising functions that carry particular security credentials also remains an important requirement for some ML applications. The research challenge here lies in identifying mechanisms for certifying ML functions based on "certificate servers" prior to their use.

Domain: Online Gaming

Gaming is a massive industry, generating a revenue of $180 billion in 2020.[8] But despite its size, developing and operating games and their surrounding ecosystems is challenging. We envision today's and future serverless technology addressing these challenges. In this section, we argue why online gaming can benefit from serverless, how today's serverless technology can help, and in which direction serverless technology needs to develop to better support the online gaming domain.

Benefits

The characteristics of serverless is promising for the online gaming domain. Without being comprehensive, we discuss here three areas where serverless can help. First, online games typically have large workload variance over time.[9] The popularity of games is difficult to

[8] https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990
[9] https://atlarge-research.com/pdfs/2011-nae-dynamic.pdf

predict, but can be the difference between attracting tens or tens of millions of players. Importantly, the number of players in an online game typically goes down over time, and many games fail to attract a significant number of players. To manage this risk, game developers need the ability to scale to zero. At the same time, successful games that attract large numbers of users have significant daily, weekly, and yearly workload variation patterns. Games and their ecosystem services require strong scalability to support this. Second, Successful online games require continuous operational support to provide a service to players and meet QoS and other NFR constraints. Because such support is labor intensive, it requires risky and costly investments from development companies and individual developers. Third, online games operate as parts of a large ecosystem, and as such require good integration (e.g., high availability, scalability, fault tolerance) with other services. Using serverless applications can simplify development effort required to meet these goals.

Technology assessment

Today's serverless platforms are a promising technology for several areas of the online gaming ecosystem. Using the house-like metaphor from Iosup et al.,[10] we envision serverless technology to automatic content generation (e.g., generating worlds on demand), game analytics (e.g., analyzing player behavior and detecting toxicity), the social meta game (e.g., web apps where users share player-created content), and the virtual world (e.g., player authentication, matchmaking).

Future directions

While these applications are promising, we identify several challenges that require serverless to develop beyond its current capabilities. We briefly describe three such challenges here. First, games can have stringent QoS requirements such as low jitter and latency in the range of tens of milliseconds, which requires low (cold) start times and guarantees on tail latencies, and good performance isolation to prevent performance variability in areas that will result in reduced player experience. Second, using an architecture with large numbers of small components can make keeping consistent state between players will become more difficult. Third, the gaming ecosystem contains parts (such as game server instances) that do not use a programming model that fits easily with the request/reply model used by today's FaaS platforms.

Domain: Mobile and telco serverless

Overview

The mobile networking area is currently seeing a significant increase in connected devices, including IoT devices and smart mobile devices. Due to the operators' business models and the potential for additional revenue models for network and telcos, operators are forced to act efficiently and in line with the demands. In particular, this means being efficient in the direction of scaling and elasticity. More precisely, mobile networks are typically geographically distributed and have to deal with a highly variable amount of device messages at the edge and on central entities. This requires a massive scaling in both directions, spatially and in terms of

[10] https://arxiv.org/pdf/1802.05465.pdf

resources – all things that Serverless entails. Contrary to requirements, Mobile Serverless also has some inherent functionality that conforms to the serverless paradigm: many network core functions are sold today as software to avoid large, rigid hardware boxes. Cellular functions are already available today as separate functions (virtual network functions, especially with the use of Software-Defined Networking (SDN) environments). These functions are usually already short-running and often even already stateless.

Benefits

The most important points that can be envisioned for Mobile Serverless are: it leverages the efficient and scalable architecture, which provides benefits from reduced operational costs and function isolation. Mobile core network functions are expected to be fully integrated into or partially merged with user functions. Besides these specific points, Mobile Serverless can generally benefit from better maintainability and updatability as mobile functions are encapsulated in atomic tasks or functions, including better resilience with the replication of functions for fallback purposes and chaos monkey functions.

Challenges

For the next generation of Serverless Computing the challenging application requirements include runtime and latency guarantees for network and signaling processing functions, the cold start problem (especially with distributed deployment), and locality, since some functions must be performed in specific locations. Ultimately, more and more security and privacy requirements play a role in the discussion with Serverless, as mobile networks typically offer larger attack surfaces and have many common elements where sensitive information about users is stored.

Anti-Hypothesis: What is "not" serverless, and why/when should applications sometimes not be serverless?

Having the above mentioned application domains in mind, significant assumptions exist in current vendor-based serverless models regarding function execution time, memory constraints, "cold start" overhead, and execution costs (e.g. per unit time execution costs $\gg$ VM/container execution costs).

The question arises if these constraints are inherent to the serverless model or just technical limitations which will be relaxed in the future?

- Are they really constrained or just driven by economic models?
- What is not serverless, i.e. when should we just not use serverless?
- What is the anti-pattern equivalent for serverless? (Definition: "An anti-pattern is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive.")

From a general point of view:

1. The first obvious thing that strikes against the use of serverless for some applications is the fact that partitioning an application into functions may take too long or can even be counterproductive.

2. Sometimes applications depend on a strong requirement of I/O or other hardware components like shared or fast memory, which is hardly possible to achieve in serverless at the moment.
3. In addition to the last point, heterogeneity given by functions across multiple hardware and infrastructure can also lead to severe challenges in resilience and coordination.
4. There is furthermore limited reproducibility in serverless computing, which also applies to the performance of these functions.
5. It is also important to note that applications can require particular QoS and QoE guarantees (e.g. gaming) that have to be supported across multiple executions of these functions.
6. Finally, there are also restrictions from the architecture and platform side in direction to elasticity for applications – consider for example that the "unlimited resource" assumption does not completely hold for edge devices in case your platform spans over heterogeneous computing devices.

From a technical point of view, typical restrictions and the ones above arise from the fact that Serverless Computing is based on a number of paradigms, including the fact that simple direct communication between functions is not normally possible. Often there is also only a limited amount of synchronization possible and serverless commonly only allows few modifications and control of the workflow and scheduling of functions, which is required by some applications. One severe problem, regarding the performance point of view, is also that shared memory capabilities and heavy memory optimization is not possible. Ideas like OpenMP will not be possible since such approaches require strict locality of the memory.

Next steps – Takeaway for the wider community

We ask interested research communities (cloud and systems, software engineering, performance) to reflect on better application enablement. This encompasses concrete actions such as:

1. Helping to complete the transition to "Serverless 2.0" by measuring and optimising the recently introduced prototypes, both from industry and from academia, to overcome current limits in massive scalability and startup latency.
2. Understanding the cost and economics of using Serverless functions in applications, and developing a "cost calculator" that is able to make effective assessment of potential costs for an application user (along similar lines to AWS Cost Calculator).
3. Performing more empirical research with companies to also learn from failures and hesitation in addition to success cases. This will uncover current system limitations and turn that into common knowledge, not confined to the serverless platform product owners.
4. Support for serverless functions that can co-exist and meet different types of application requirements – such as security, performance, usability etc. Security remains an important challenge and will become increasingly important as new platforms and applications communities make use of Serverless.
5. Answering fundamental questions such as the "greenness" and cost efficiency of serverless computing in a way that practical advice for application engineers can be derived.
6. Co-designing forward-looking serverless architectures that abstract from the underlying isolation layers (container, µ-VM, WASM) and are prepared to work in heterogeneous hardware environments. The co-design should be conducted in conjunction with "borderline applications" that are not just yet enabled but might be with the new design.

This way, progress into the next stages of serverless computing can be documented with timestamped examples. Examples include smart edge-based systems with dynamic resource autodiscovery, uniform management interfaces, and awareness about end-to-end characteristics such as networked invocation duration to account for latency variations, interrupted connections and other QoS concerns, for instance in connected vehicles.

4.4 Evaluation of Serverless Systems (Topic 4)

Cristina Abad (ESPOL – Guayaquil, EC), Kyle Chard (University of Chicago, US), Pooyan Jamshidi (University of South Carolina – Columbia, US), Alessandro Vittorio Papadopoulos (Mälardalen University – Västerås, SE), Robert P. Ricci (University of Utah – Salt Lake City, US), Joel Scheuner (Chalmers and University of Gothenburg, SE), Mohammad Shahrad (University of British Columbia – Vancouver, CA), and Alexandru Uta (Leiden University, NL)

Opening Statement

The group discussed the topic of serverless computing from the perspective of performance evaluation and benchmarking of the serverless platforms and applications that can be built on those platforms.

Reproducibility in serverless

One of the overarching challenges in computer science is reproducibility [1, 2, 3]. Previous studies focusing on cloud computing [4] have already shown that results, especially performance data [5] are difficult to reproduce across studies. We expect this behavior to be exacerbated in serverless scenarios: On the one hand, from the client perspective, the underlying system is opaque. On the other hand, cloud providers have a clear view of the system design and implementation, but the client workloads are opaque to them. We believe this is an opportunity for the two parties to work together toward achieving better experimental reproducibility.

Directions

The performance evaluation of serverless systems can be classified into six types of evaluations, according to their goal. We describe each, next.

Evaluation of existing platforms and reverse-engineering:

Performed when we want to know well a serverless platform performs; for example, as in [6]. This type of evaluation primarily employs micro-benchmarks to measure a very specific resource such as CPU speed for floating-point operations. The evaluation results can be used to choose a suitable service, optimize configurations, guide design decisions of serverless applications, or parametrize a theoretical performance model.