Imaginary Machines: A Serverless Model for Cloud Applications

Michael Wawrzoniak[1], Rodrigo Bruno[2], Ana Klimovic[1], Gustavo Alonso[1]
[1]Systems Group, Dept. of Computer Science, ETH Zurich
[2]INESC-ID/Técnico, U. Lisboa

ABSTRACT

Serverless Function-as-a-Service (FaaS) platforms provide applications with resources that are highly elastic, quick to instantiate, accounted at fine granularity, and without the need for explicit run-time resource orchestration. This combination of the core properties underpins the success and popularity of the serverless FaaS paradigm. However, these benefits are not available to most cloud applications because they are designed for networked virtual machines/containers environments. Since such cloud applications cannot take advantage of the highly elastic resources of serverless and require run-time orchestration systems to operate, they suffer from lower resource utilization, additional management complexity, and costs relative to their FaaS serverless counterparts.

We propose *Imaginary Machines*, a new serverless model for cloud applications. This model (1.) exposes the highly elastic resources of serverless platforms as the traditional network-of-hosts model that cloud applications expect, and (2.) it eliminates the need for explicit run-time orchestration by transparently managing application resources based on signals generated during cloud application executions. With the Imaginary Machines model, *unmodified cloud applications become serverless applications*. While still based on the network-of-host model, they benefit from the highly elastic resources and do not require runtime orchestration, just like their specialized serverless FaaS counterparts, promising increased resource utilization while reducing management costs.

ACM Reference Format:
Michael Wawrzoniak[1], Rodrigo Bruno[2], Ana Klimovic[1], Gustavo Alonso[1], [1]Systems Group, Dept. of Computer Science, ETH Zurich, [2]INESC-ID/Técnico, U. Lisboa . 2024. Imaginary Machines: A Serverless Model for Cloud Applications. In *Proceedings of The 2nd Workshop on Serverless Systems, Applications and Methodologies (SESAME'24).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cloud users today can choose between two main options for developing and managing their applications: renting VMs on-demand or using providers' serverless compute and storage services. VMs come with a familiar programming/execution environment of network-of-hosts, but provide a lower level of resource elasticity, and users need to explicitly manage the infrastructure at run-time (e.g., using cluster managers, turn machines on/off based on the application load.) Serverless platforms automate infrastructure management and provide highly elastic resources but use a restrictive FaaS programming model with limitations that reduce its broader applicability [10]. To overcome the limitations, researchers have been exploring new serverless platform designs [13], additional infrastructure [11], and constructing new FaaS-specialized applications [12].

This paper explores how to get the best of both worlds – applying the serverless paradigm to cloud applications designed for networked VMs with the familiar network-of-hosts programming model. We explore how to automate infrastructure orchestration à la serverless for cloud applications. We propose a serverless model for cloud applications, which we call *Imaginary Machines*, that preserves the familiar, well-loved network-of-hosts programming model of VM-based applications with the orchestration automation of serverless platforms. We also discuss approaches to realize this model and propose an overlay approach based on an evolution of Boxer [16] to realize the model on publicly available FaaS resources.

2 SERVERLESS CLOUD APPLICATIONS

We believe that cloud applications based on the network-of-hosts model can also be serverless, taking advantage of highly elastic resources and automated infrastructure orchestration. Our perspective is based on the following observations and insights.

Serverless resources can suit cloud applications: Although commonly viewed as lightweight and restricted, resources made available by FaaS can match many cloud application requirements. FaaS platforms continue to reduce limitations and increase resource limits, but even today's public platforms, such as AWS Lambda [4], can match many cloud application requirements of memory, compute capacity, state persistence, and reliability for some use cases.

Resources available in today's FaaS platforms have been increasing in size to the point where functions available today are comparable to the largest VMs available 15 years ago (memory and cpu), the environment in which many of the popular cloud applications have been originally designed for [6, 8]. AWS Lambda functions can be configured up to 10GB of memory and 6vCPU cores [5], matching largest AWS EC2 CPU instances introduced 12 years earlier [2].

Second, the wide adoption of disaggregated storage resulted in cloud applications that are structured into multiple tiers of functionality, where many services do not depend on the local persistent state, which makes these layers a match for the ephemeral state provided by publicly available FaaS functions. For example, a completely unmodified application logic tier of a popular DeathStarBench [9] microservice benchmark runs in AWS Lambda [14].

Third, FaaS functions are becoming more reliable and are no longer short-lived. The maximum execution time of FaaS functions

SESAME'24, 22-25 April 2024, Athens, Greece,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Michael Wawrzoniak[1], Rodrigo Bruno[2], Ana Klimovic[1], Gustavo Alonso[1]
[1]Systems Group, Dept. of Computer Science, ETH Zurich
[2]INESC-ID/Técnico, U. Lisboa

has been increasing, in AWS Lambda, it is now 15 minutes [3].

Unbundling serverless resources from programming model: Serverless resources do not have to be bundled with the FaaS programming model, even in mainstream publicly available serverless platforms such as AWS Lambda. An evolution of Boxer [16], a platform providing the network-of-hosts programming model on top of AWS Lambda FaaS, has been demonstrated to run unmodified off-the-shelf distributed data processing systems such as Apache Spark and Drill [15], unmodified Apache Zookeeper, and the logic layer of the DeathStarBench microservice benchmark [14, 17]. These unmodified cloud applications run on AWS Lambda functions just as if they were running on VM-based networked hosts, each function instance having a routable network address, a resolvable hostname, and function-to-function network transport.

These projects demonstrated the feasibility of using the network-of-hosts model with serverless resources. However, they also took a step back by reintroducing the need for a serverful orchestration system (docker-compose [7] was used in [15]) to manage executions of individual functions.

Serverless orchestration for cloud applications: We believe that automated infrastructure orchestration, an essential property of serverless, can also be realized for network-of-hosts model cloud applications. We first describe a perspective on automated resource orchestration of existing FaaS and then propose how to achieve an analogous mechanism for the network-of-hosts model.

The FaaS programming model enables automated infrastructure orchestration by embedding resource allocation signals in application control flow. The two signals generated during application execution are (1.) function invocation resulting in resource allocation of a function instance (and invocation of the function event handler), and (2.) function handler exit, when resources are released. As a ubiquitous and essential optimization, the resources are normally not released immediately after function handlers exit, instead, to provide low-latency warm-starts, function instances are first suspended for a limited time before they are terminated. FaaS applications are factored into collections of functions (possibly structured as dataflows [1]) that compose by invoking each other (directly or by producing events) based on internal control flow. Each such function invocation is a resource allocation signal used by FaaS platforms to dynamically rescale resources and create the illusion of avoiding resource orchestration.

Cloud applications based on the network-of-hosts model execute processes on hosts that 'compose' by communicating with each other via a network. The analogous resource allocation signals to those embedded in FaaS functions (above) are also present in cloud applications. Generally, any signal that indicates that any application process on a host function could be runnable is potentially a resource allocation signal for that host function (analogous to a function invocation). Similarly, any signal that indicates that all application processes of a host are idle is a resource release signal for that host function (analogous to a function event handler exiting). To achieve the automated infrastructure orchestration for the network-of-hosts model, the platform must observe such signals and perform the necessary on-demand resource allocation actions of allocating and releasing host functions, just like in FaaS.

Although not as explicit as in FaaS, cloud applications provide such signals; these signals are already used by local operating systems of running hosts to perform local application process scheduling (e.g., in response to a signal indicating that data is available for a waiting blocked process, the scheduler may choose to make the process runnable). However, in contrast to VM-based infrastructure, to achieve serverless fine-grain scaling, host functions of cloud applications must be allocated on demand only when needed and released (suspended or terminated) when not used. This means that, instead of the host's local operating system, which may be suspended or not even started, the appropriate resource allocation signals must be observed and acted on *externally* to the host function. Fortunately, assuming that cloud applications run entirely on the serverless platform, the allocation signals for all non-running host functions must be generated by (and observable at) the running host functions or external network requests, all observable by the serverless platform. As a concrete example, when application processes initiate network communication to a valid destination host that is not running (e.g., during initialization or scaling up), the platform must use this signal to allocate the new destination host function automatically and in a transparent way to the already running application processes. Conversely, as application processes on a running host become idle, the platform suspends and eventually possibly terminates the host, automatically releasing unused application resources. More signaling scenarios must be handled, which we do not describe here, but based on this approach, the resources used by network-of-hosts applications are allocated and released on-demand and transparently to the application, providing a form of automated infrastructure orchestration.

In order to provide application transparency, the resource allocation latencies (time to start or resume a host) must be sufficiently short (and infrequent) to be within tolerable network latencies of the applications. Fortunately, as evidenced by publicly available systems such as AWS Lambda, latency to instantiate new host functions (cold ~200ms) can be in the range of wide-area network latencies, suggesting they are within the tolerable range of connection timeouts that many cloud applications can handle.

3 IMAGINARY MACHINES MODEL

We refer to the serverless execution model of cloud applications as the Imaginary Machines (IM) model.

- *From the application perspective*, the Imaginary Machines model presents a network-of-hosts programming model, where all possible (configured) hosts are already instantiated. Cloud application processes run *imagining* as if all network destinations were already running and available.
  Significantly, the model *does not* include any explicit runtime infrastructure orchestration just like in the FaaS model).
- *From the platform perspective*, the system must support the programming model that the applications expect (above) while performing on-demand, automatic, transparent, and fine-grained resource orchestration for the application.

We are in the process of realizing a version of the IM model by extending Boxer [14] serverless overlay system that already provides network-of-hosts model on top of AWS Lambda FaaS by also including the necessary function allocation mechanism.

4 CONCLUSION

We motivated and described an alternative serverless model for cloud applications called Imaginary Machines. Given that the dominant programming model in the cloud is based on the network-of-hosts model, providing applications based on that model with access to serverless resource elasticity and reduced operational complexity has the potential for a great impact.

REFERENCES

[1] [n. d.]. AWS Step Functions. https://aws.amazon.com/step-functions/

[2] 2008. Amazon EC2 now provides High-CPU instance types. Retrieved 2023-12-03 from https://aws.amazon.com/articles/feature-guide-amazon-ec2-high-cpu-instance-types/

[3] 2018. AWS Lambda enables functions that can run up to 15 minutes. Retrieved 2024-01-27 from https://aws.amazon.com/about-aws/whats-new/2018/10/aws-lambda-supports-functions-that-can-run-up-to-15-minutes/

[4] 2020. AWS Lambda. Retrieved 2020-08-17 from https://aws.amazon.com/lambda

[5] 2020. AWS Lambda now supports up to 10 GB of memory and 6 vCPU cores for Lambda Functions. Retrieved 2024-01-27 from https://aws.amazon.com/about-aws/whats-new/2020/12/aws-lambda-supports-10gb-memory-6-vcpu-cores-lambda-functions/

[6] 2023. Apache Spark history. Retrieved 2023-12-03 from https://spark.apache.org/history.html

[7] 2023. Docker Compose. Retrieved 2023-07-27 from https://docs.docker.com/compose/

[8] 2023. Initial ZooKeeper code contribution from Yahoo! Retrieved 2023-12-03 from https://issues.apache.org/jira/browse/ZOOKEEPER-1

[9] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *ASPLOS*.

[10] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *CIDR*.

[11] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *OSDI*. 427–444.

[12] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *SIGMOD*.

[13] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. 2020. Cloudburst: Stateful Functions-as-a-Service. *PVLDB* (2020).

[14] Michael Wawrzoniak, Rodrigo Bruno, Ana Klimovic, and Gustavo Alonso. 2024. Boxer: FaaSt Ephemeral Elasticity for Off-the-Shelf Cloud Applications. arXiv:2204.5701258 [cs.DC]

[15] Michael Wawrzoniak, Gianluca Moro, Rodrigo Bruno, and Gustavo Alonso. 2024. Off-the-shelf Data Analytics on Serverless. In *CIDR*.

[16] Michael Wawrzoniak, Ingo Müller, Rodrigo Bruno, and Gustavo Alonso. 2021. Boxer: Data Analytics on Network-enabled Serverless Platforms. In *CIDR*.

[17] Michael Wawrzoniak, Ingo Müller, Rodrigo Bruno, Ana Klimovic, and Gustavo Alonso. 2022. Short-lived Datacenters. arXiv:2202.06646 [cs.DC]