COLUMN: View from the Cloud

Serverless Is More: From PaaS to Present Cloud Computing

Erwin van Eyk
Delft University of Technology

Lucian Toader
Vrije Universiteit Amsterdam

Sacheendra Talluri
Delft University of Technology

Laurens Versluis
Vrije Universiteit Amsterdam

Alexandru Uță
Vrije Universiteit Amsterdam

Alexandru Iosup
Delft University of Technology and Vrije Universiteit Amsterdam

Editor:
George Pallis
gpallis@cs.ucy.ac.cy

In the late 1950s, leasing time on an IBM 704 cost hundreds of dollars per minute. Today, cloud computing—using IT as a service, on demand with pay-per-use—is a widely used computing paradigm that offers large economies of scale. Born from a need to make platform as a service (PaaS) more accessible, fine-grained, and affordable, serverless computing has garnered interest from both industry and academia. This article aims to give an understanding of these early days of serverless computing: what it is, where it comes from, what the current status of serverless technology is, and what its main obstacles and opportunities are.

The 1950s saw the emergence of two technologies that are currently shaping the world: containerization in shipping and time sharing in computing. By allowing shipping to become standardized and automated, containerization gave rise to manufacturing and retail ecosystems, and ultimately to the economic phenomenon of globalization.[1] By enabling multiple clients to share the same physical infrastructure, time sharing gave rise to cloud computing and the modern digital ecosystems, which are key drivers for growth in knowledge-based societies.[2]

Whereas few companies or people could afford the cost of time-sharing services and paid dearly for simple computer simulations in the late 1950s, today more than 80% of companies, along with many private individuals, use the hundreds of services accessible through cloud computing.[3,4] Following with remarkable regularity the evolution observed in the history of containerization, cloud services have adapted to offer better-fitting containers that require less time to load (boot) and to provide increased automation in handling (orchestrating) containers on behalf of the client.

1089-7801/18/$33.00 ©2018 IEEE

Serverless computing promises more: to achieve full automation in managing fine-grained containers. Already, IT spending on serverless computing is expected to exceed $8 billion per year, by 2021.[5] To understand what serverless is and what it can deliver, we trace the evolution of computing technology that has given rise to serverless computing, analyze the current status of serverless technology, and identify the main obstacles and opportunities we see in delivering on its promise.

What is serverless computing? We have proposed the following definition:

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.[6]

This definition places serverless as a computing abstraction, partially overlapping with platform as a service (PaaS). With serverless, developers focus on high-level abstractions (e.g., functions, queries, and events) and build applications that infrastructure operators map to concrete resources and supporting services. This effectively separates concerns, with developers focusing on the business logic and on ways to interconnect elements of business logic into complex workflows. Meanwhile, service providers ensure that the serverless applications are orchestrated—that is, containerized, deployed, provisioned, and available on demand—while billing the user for only the resources used. These separated roles have also emerged for physical containers, with manufacturers and retailers in the role of developers, and shipping lines in the role of service providers.

Clients of serverless computing could use the function as a service (FaaS) model, which we define this way:

Function as a service (FaaS) is a form of serverless computing in which the cloud provider manages the resources, lifecycle, and event-driven execution of user-provided functions.[6]

With FaaS, users provide small, stateless functions to the cloud provider, which manages all the operational aspects to run these functions. For example, consider the ExCamera application, which uses cloud functions and workflows to edit, transform, and encode videos with low latency and cost.[7] A majority of the tasks in these operations can be executed concurrently, allowing the application to improve its performance through parallelizing these tasks.

To deploy ExCamera using the traditional infrastructure as a service (IaaS) model, a user would need to spin up virtual machines (VMs), provision them, orchestrate the workflows, manage resources as needed, and manage the variety of dynamic issues (e.g., faults and inconsistencies). This would require considerable expertise and continuous effort in orchestrating ExCamera, and yet result in significant amounts of underutilized but paid-for resources. Instead, by leveraging serverless computing, ExCamera defers the operational complexity to the cloud provider, using FaaS to manage the operational lifecycle of the individual video tasks.

However promising, serverless computing is still an emerging technology. Understanding how applications such as ExCamera can leverage it requires finding answers to questions such as these:

- What are the computer technologies underlying serverless?
- What is the status of the current serverless technology?
- What can we expect from the field of the serverless computing in the foreseeable future?

We address these questions with a threefold contribution. First, we identify the concepts leading to serverless computing, with deep historical roots in concrete and abstract innovations in computer science. Second, we analyze the current state of the technology and the complex technological ecosystems it consists of. Finally, we identify and analyze important obstacles and opportunities in this emerging field that we—as a community—need to address to make the serverless promise a reality. We conclude with a forewarning question: can we reproduce the successes and avoid the downsides of physical containerization?

THE LONG ROAD TO SERVERLESS

In this section, we analyze the evolution of computer technology that led to serverless computing—going back to the 1960s. All the breakthroughs indicate that serverless computing would not have been possible a decade ago, when it would have missed enabling technologies such as the distinction between IaaS and PaaS (standardized by NIST), fine-grained containerization (e.g., Docker), and even a diverse set of applications.[3] In Figure 1, we distinguish six main dimensions of these critical breakthroughs that together led to the emergence of serverless.

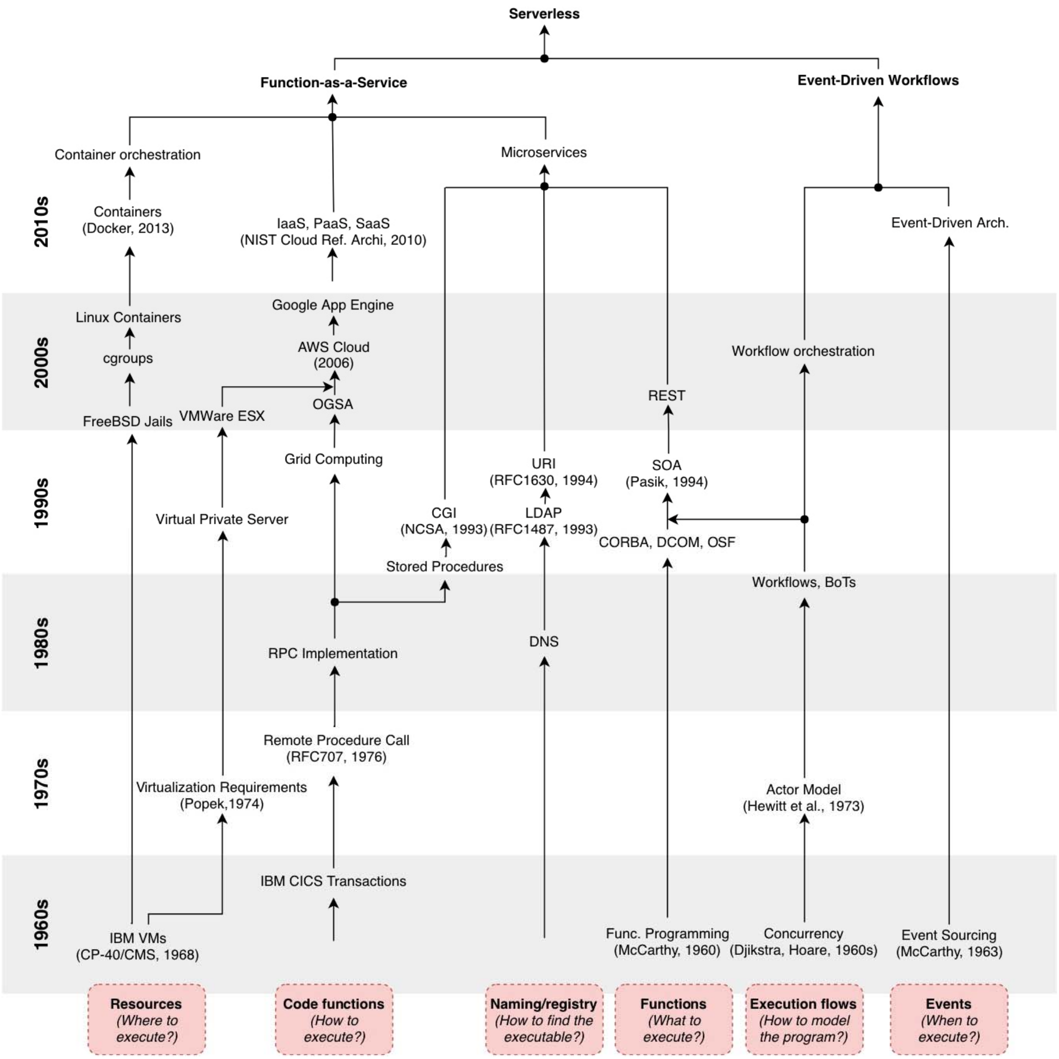

Figure 1. A history of computer science concepts leading to serverless computing.

CONTAINERIZED RESOURCES

Complementary to time sharing, virtualization abstracts away the physical machine to reduce the operational effort and to allow the same physical resources to be shared across multiple applications and users (multiplexing). Although associated in recent memory with VMware's ESX technology (2001), virtualization was invented much earlier; it was used in production in late-1960s

IBM mainframes. Virtualization was finally conceptualized nearly a decade later.[8] With the emergence of the Internet in the 1990s, these early concepts of virtualization were introduced online to enable shared hosting through virtual private servers. Soon after the release of ESX, cloud computing emerged, making virtual resources available over the Internet.

Like their physical counterparts, digital containers protect their content from external abuse. They do this by adding a layer of abstraction over the resources provided by the system. FreeBSD added jails—independent partitions of the system with their own directory subtree, hostname, IP address, and set of users. Linux followed with cgroups (2006), a mechanism to group processes and to assign each group separate resources. The Linux Containers project (LXC, 2008) bundled cgroups and kernel namespaces, along with better tooling. Built on LXC, Docker (2013) offered convenient container orchestration, fostering an entire ecosystem based on digital containers.

Serverless computing is the latest result of this long-term process of defining virtualization abstractions, to eliminate concerns related to server provisioning and management. Although it exposes abstract resources (e.g., functions) to the user, these are mapped to concrete resources (e.g., containers), continuing the transition from “bare metal” to “bare code.”

Code as Functions

The ability to execute arbitrary “cloud functions” is essential to serverless computing.

Technology has emerged and reemerged often for running domain- and context-specific remote functions. We can trace this concept to as early as 1968, when, with IBM’s Customer Information Control System (CICS), users were able to associate user-provided programs with transactions. RPC (specification in 1976, implementation in 1984) enabled the invocation of arbitrary procedures located in remote systems, over a communication network. Derived from RPC, stored procedures for databases (1980s) and Common Gateway Interface (CGI) scripts for webservers (1990s) aimed to bring support for executing functions to specific domains. Google App Engine—with other PaaS platforms following its example—started allowing users to asynchronously execute arbitrary tasks in the background.

In contrast to these context-specific implementations, serverless computing aims to provide a full abstraction for arbitrary, event-driven execution of generic functions.

Naming and Discovery

Managing and invoking services, including functions, depends on being able to name and discover services. Derived from a long line of technological innovations, current approaches follow the pathbreaking concepts of Lightweight Directory Access Protocol (LDAP, 1993) and URI (1994). LDAP uses naming and properties and enables distributed directory services over TCP/IP. URI provides unique identifiers for resources, encoded as character strings.

Serverless extends naming and discovery with function versioning and aliases—e.g., offered by Amazon Web Services (AWS). With versioning, it is possible to work with different immutable versions of a function simultaneously. Aliases are mutable pointers to a version and can be used to transition a version from one stage to another (e.g., from development to production) without changing the deployed application.

Functions as Computation

Serverless computing relies on the concept of function as computation, which stems from a long tradition of ever-higher-level abstractions and specialization in computer science.

Functional programming[9] departed from procedural programs, to allow the developer to manage abstract data types and control flows, instead of the concrete details of memory and processors. The application of object-oriented principles to distributed systems lead to the creation of

DCOM (Distributed Component Object Model), CORBA (common object request broker architecture), and OSF (the Open Software Foundation) in the 1990s. In the 2000s, we climbed up the specialization ladder by contextualizing and interconnecting services—e.g., through service-oriented architecture[10] (SOA) and architectures based on REST (Representational State Transfer).

These previous developments gradually led to microservices: self-contained applications providing specific functions over well-defined protocols.[11] Continuing this trend, serverless development is effectively a hyperspecialization of services.

Execution Flows

Serverless computing depends on the ability to coordinate execution flows.

Concurrency[12] has been an early and vital model for the evolution of computing, allowing multiple processes to make progress at the same time, while remaining under the developer's control. This model has many applications, and many other models are rooted in concurrency, including generalized processes, threads, and actors.[13]

Over the past two decades, we have moved toward a declarative form of expressing concurrency. Workflows declare the structure of applications, leaving the concrete execution and synchronization of workflow tasks to the runtime system. This model has a multitude of applications[14] and underlies our view of serverless computing.

Events to Trigger Functions

The first computer programs were synchronous, carefully crafted to follow a particular code path. This model made programs difficult to create and modify and less robust to changing conditions. Soon, event sourcing addressed the need to record, order, and respond to requests for state changes.

With the proliferation of high-level languages and advanced operating systems, the concept of linking disparate computation together with special communication constructs took hold. Device drivers were early examples of this event-driven programming. With the rise of the Internet, event-driven distributed systems became widely used—with events mapped intuitively to the asynchrony of real-world networks.

Indeed, in modern systems, event-based protocols allow systems in an ecosystem to communicate without excessive dependence on the implementation details of each individual system. Owing to its highly networked nature, serverless computing is apt to leverage this idea, through well-defined event protocols and ways to manage events—for example, by using message queues.

SERVERLESS NOW

In this section, we discuss how ExCamera could use serverless computing. We explain the current and emerging technological ecosystem for serverless, and detail the expected benefits from using serverless: better resource management, scaling, and more insight and control.

The State of Serverless Technology

To execute parts of the workloads using small serverless functions, ExCamera parallelizes video transcoding using AWS Lambda, one of the many FaaS platforms.[15] These various platforms differ in focus, target domain, assumed model, and architectural decisions. Next to the closed-source FaaS platforms, addressing the lack of insight and vendor lock-in, several open-source platforms have emerged, including Apache OpenWhisk, Fission, and OpenLambda.[16]

To address the complexity of working with many different FaaS platforms and their incompatible APIs, the community has focused on informal standardization. Serverless frameworks—e.g., Apex or the Serverless Framework—provide a common programming model that enables easier, platform-agnostic development, along with better interoperability of functions.

Much serverless tooling already exists to allow developers to defer nonessential tasks, sparking the emergence of a diverse ecosystem of (serverless) services, from serverless databases to monitoring to security. For example, workflow engines, such as Azure Logic Apps, Fission Workflows, and PyWren,[17] abstract away the complexity of networking in the composition of higher-order functions and services.

Benefits of Current Serverless Technology

Serverless computing promises more value than other cloud operations: equal or better performance while reducing the operational costs of applications. This has led industry—rather than academia—to drive the initial development and adoption of this paradigm. Figure 2 illustrates the main case for serverless computing.

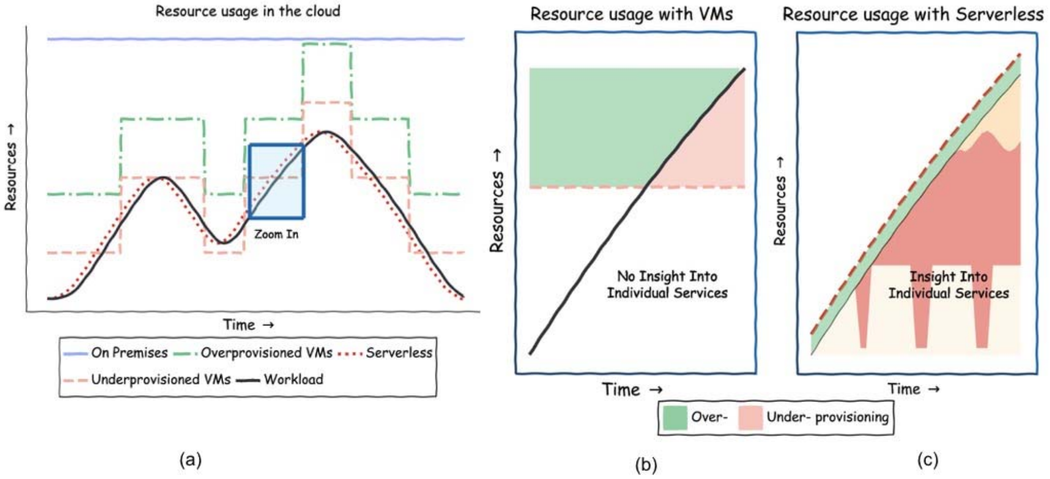

Figure 2. A case for serverless computing: higher resource utilization, finer granularity, and more detailed control than with container-based or self-hosted computing. (a) Resource utilization in the cloud. (b) For the zoom-in, resource utilization with virtual machines (VMs). (c) For the zoom-in, resource utilization with serverless computing.

Benefit 1: Improved Resource Management

In the traditional cloud model, the user is responsible for selecting and deploying the concrete resources. To avoid overburdening the user with options, the range of options is generally limited to large, multifunctional resource types (e.g., VMs or containers). Applications rarely fit these resources, and, to mitigate the overhead incurred by the large, general-purpose resources, applications are coarse-grained.

As illustrated in Figure 2a, coarse-grained applications lead to inaccurate autoscaling decisions, causing severe under- or over-provisioning. In contrast, serverless computing means applications are fine-grained, which means the cloud provider can more closely match abstract resource demand to actual system resources.

Benefit 2: More Insight and Control

In the traditional model, the user is responsible for deploying, monitoring, and other operational tasks related to the lifecycle of coarse-grained applications (see Figure 2b). However, many cloud users do not have the necessary expertise. Moreover, the operators lack context, so they have to make autoscaling decisions without accurate profiling or insights from the deployed applications.

With serverless, the increased responsibility for the operator gives more insight and control. Operators select the resources; deploy and provision resources; implement and control the monitoring of resource usage, workload intensity, and application behavior; and can autoscale or migrate the application. They can profile and model the granular services constituting the serverless application (see Figure 2c), offer this information to users, and improve the decisions made with these insights.

Benefit 3: Granular Scaling

In the traditional model, applications consist of large, multifunctional VMs with multiminute provisioning times. These VMs act as black boxes and thus are difficult to model and predict for operators. Although applications are typically bottlenecked by only one of the resources in one part of the application, the operators can only scale the entire application to resolve the bottleneck. Eliminating some of these issues is possible but requires the user to rearchitect the application, typically as microservices. The effectiveness depends highly on the user's expertise[11]—most users cannot benefit.

With serverless, the operator can better scale the individual, granular services or functions, using deep insights. The contrast between Figure 2b and Figure 2c illustrates this situation.

Other Benefits

Other benefits motivate the adoption of serverless computing. The shift from capital expenses to operational expenses more accurately aligns the costs to the actual business processes. The independent services allow teams to choose the right tools and dependencies for a use case without impacting other parts of the system and organization. The high-level abstraction allows software developers to iterate on these distributed systems more quickly, while limiting the need for extensive expertise in distributed systems. Etc.

PERSPECTIVES ON SERVERLESS

Although serverless computing already offers many benefits, many obstacles could inhibit its further adoption. In joint work with the Standard Performance Evaluation Corporation RG Cloud Group (https://research.spec.org/working-groups/rg-cloud.html), we have identified more than 20 detailed challenges and opportunities for serverless computing.[5,18] Here, we identify the top five obstacles and opportunities arising from them (see Table 1).

Table 1. Obstacles and opportunities for serverless computing.

Obstacle	Opportunity
Fine granularity and cost	Nontrivial resource management, workflows of functions, orchestration, fine-grained "pay-per-use" pricing, optimizing cost–performance tradeoffs
Data privacy	Fine-grained access control and function-level auditing and provenance, full GDPR (General Data Protection Regulation) compliance
Performance	Fine-grained scheduling and resource management, new performance models and fairness mechanisms that help reduce resource contention and performance variability

Data-intensive applications	Fine-grained data-centric programming models
API jungle	Service discovery and brokering, fine-grained software lifecycle management, standardized multicloud APIs, interoperability, portability, multimodal services

First, the fine granularity for expressing computation adds significant overhead to the resource management and scheduling layers. To overcome this, we envision significant research efforts invested in coscheduling and orchestration for workflows of functions. Moreover, from a user perspective, we need new tools for navigating the cost–performance tradeoffs to explore the complexity of fine-grained pricing models.

Second, data privacy is important for the clients and nontrivial for the providers to offer—for example, ensuring full GDPR (General Data Protection Regulation; https://www.eugdpr.org) compliance. With its fine-grained nature, serverless computing allows for more enhanced access control, function-level auditing, and provenance for seamless, efficient GDPR compliance.

Third, in modern clouds, performance suffers from significant variability due to resource contention, virtualization, and congestion overhead. These issues are amplified in serverless computing, because of its granular nature. However, the increased insight and control over the operational lifecycle provides cloud providers with opportunities to minimize these performance issues by being able to more accurately monitor, profile, and schedule these fine-grained services.

Fourth, data-intensive applications are not naturally expressed in the—stateless—FaaS paradigm. We envision the design and implementation of fine-grained, data-centric, serverless programming models. One promising research direction is investigating distributed promises in serverless environments.

Finally, the API jungle generated by the fast-evolving serverless APIs, frameworks, and libraries represents an important obstacle for software lifecycle management and for service discovery and brokering. To overcome this, significant effort must be invested in multicloud API standardization, interoperability, and portability to avoid lock-in and to enable seamless service discovery.

CONCLUSION

Serverless computing is a promising technology, with a burgeoning market already formed around it. By analyzing the computer technology leading to it, we conclude that this model could not have appeared even a decade ago. Instead, it is the result of many incremental advances, spanning diverse domains: from the increasingly granular resource abstractions, to the emergence of abundant amounts of resources available nearly instantly, to the reduction of costs and complexity of distributed applications.

The current serverless technology offers its customers fine billing granularity, detailed insight and control, and the affordable ability to run arbitrary functions on demand. However, this technology has not been demonstrated beyond selected, convenient applications. We have identified several obstacles and opportunities and have argued that industry and academia must work together. Can we make serverless computing available for many, without the drawbacks of the technology and processes underlying physical containerization?

ACKNOWLEDGMENTS

This work is supported by the Dutch project Vidi MagnaData, by the Dutch COMMIT/ program and the COMMIT/ project commissioner, and by generous donations from Oracle Labs.

REFERENCES

1. M. Levinson, *The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger*, Second Edition, Princeton University Press, 2016.
2. G. Pendse, “Cloud Computing: Industry Report and Investment Case,” Nasdaq, 22 June 2017; http://business.nasdaq.com/marketinsite/2017/Cloud-Computing-Industry-Report-and-Investment-Case.html.
3. *Uptake of Cloud in Europe. Digital Agenda for Europe report.*, European Commission, Publications Office of the European Union, Luxembourg, September 2014.
4. *State of the Cloud 2018*, Cloudability, 2018.
5. “Function-as-a-Service Market by User Type (Developer-Centric and Operator-Centric), Application (Web & Mobile Based, Research & Academic), Service Type, Deployment Model, Organization Size, Industry Vertical, and Region - Global Forecast to 2021,” Markets and Markets, February 2017; https://www.marketsandmarkets.com/Market-Reports/function-as-a-service-market-127202409.html.
6. E. van Eyk et al., “The SPEC cloud group’s research vision on FaaS and serverless architectures,” *Proceedings of the 2nd International Workshop on Serverless Computing* (WoSC 17), 2017, pp. 1–4.
7. S. Fouladi et al., “Encoding, fast and slow: Low-latency video processing using thousands of tiny threads,” *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (NSDI 17), 2017, pp. 363–376.
8. G.J. Popek and R.P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Communications of the ACM*, vol. 17, no. 7, July 1974, pp. 412–421.
9. J. McCarthy, “Recursive functions of symbolic expressions and their computation by machine,” *Communications of the ACM*, vol. 3, no. 4, April 1960, pp. 184–195.
10. N. Josuttis, *SOA in Practice: The Art of Distributed System Design*, O’Reilly Media, 2007.
11. R. Heinrich et al., “Performance Engineering for Microservices: Research Challenges and Directions,” *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion* (ICPE 17 Companion), 2017, pp. 223–226.
12. E. Dijkstra, “Cooperating Sequential Processes,” dissertation, Department of Mathematics, Eindhoven Technological University, 1965.
13. C. Hewitt, B. Bishop, and R. Steiger, “A universal modular ACTOR formalism for artificial intelligence,” *Proceedings of the 3rd international joint conference on Artificial intelligence*, vol. IJCAI 73, 2013, pp. 235–245.
14. N. Russell, W.M.P. van der Aalst, and A.H.M. ter Hofstede, *Workflow Patterns: The Definitive Guide*, MIT Press, 2016; https://mitpress.mit.edu/books/workflow-patterns.
15. Survey of the CNCF serverless WG, GitHub, 2018; https://github.com/cncf/wg-serverless.
16. S. Hendrickson et al., “Serverless computation with open-lambda,” *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing* (HotCloud 16), 2016, pp. 33–39.
17. E. Jonas et al., “Occupy the cloud: Distributed computing for the 99%,” *2017 Symposium on Cloud Computing*, 2017, pp. 445–451.
18. E. van Eyk et al., “A SPEC RG cloud group’s vision on the performance challenges of FaaS cloud architectures.,” *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (ICPE 18), 2018, pp. 21–24.

ABOUT THE AUTHORS

Erwin van Eyk is an MSc student in computer science at the Delft University of Technology (TU Delft), where he works on serverless function and workflow scheduling. Van Eyk received a BSc in computer science from TU Delft. He leads the Serverless working group of the Standard Performance Evaluation Corporation RG Cloud Group. Contact him at e.vaneyk@atlarge-research.com; https://erwinvaneyk.nl.

Lucian Toader is an MSc student at Vrije Universiteit Amsterdam, where he studies modern distributed systems. His work on massivizing computer systems led him to serverless. Toader received a BSc in electronics and telecommunication from Politehnica University of Bucharest. Contact him at l.toader@atlarge-research.com.

Sacheendra Talluri is an MSc student in computer science at the Delft University of Technology. In the spring of 2018, he was a research intern at Databricks, working on resource management and scheduling across the memory-storage stack. Talluri received a BTech in information and communication technology from the Dhirubhai Ambani Institute of Information and Communication Technology. Contact him at s.talluri@atlarge-research.com.

Laurens Versluis is a PhD student at Vrije Universiteit Amsterdam, where he studies modern distributed systems. His work on massivizing computer systems focuses on resource management and scheduling, with applications in cloud computing. Versluis received an MSc in computer science from the Delft University of Technology. Contact him at l.f.d.versluis@vu.nl.

Alexandru Uță is a postdoctoral researcher at Vrije Universiteit Amsterdam, where he studies modern distributed systems. His work on massivizing computer systems focuses on resource management and scheduling, with applications in cloud computing and big data. Uță received a PhD in computer science from Vrije Universiteit Amsterdam. Contact him at a.uta@vu.nl.

Alexandru Iosup is a full professor and the University Research Chair at Vrije Universiteit Amsterdam, where he leads the Massivizing Computer Systems group. He's also an associate professor in the Distributed Systems group at the Delft University of Technology. Iosup received a PhD in computer science from the Delft University of Technology. He has received the Netherlands ICT Researcher of the Year award, Netherlands Teacher of the Year award, and several Standard Performance Evaluation Corporation SPECtacular community awards. He's a member of the Young Academy of the Royal Academy of Arts and Sciences of the Netherlands. He's the elected chair of the SPEC Research Cloud Group. Contact him at a.iosup@vu.nl.

Contact department editor George Pallis at gpallis@cs.ucy.ac.cy.