

# Serverless Computing

Edited by

Cristina Abad<sup>1</sup>, Ian T. Foster<sup>2</sup>, Nikolas Herbst<sup>3</sup>, and Alexandru Iosup<sup>4</sup>

1 ESPOL – Guayaquil, EC, [cristina.abad@gmail.com](mailto:cristina.abad@gmail.com)

2 Argonne National Laboratory – Lemont, US, [foster@anl.gov](mailto:foster@anl.gov)

3 Universität Würzburg, DE, [nikolas.herbst@uni-wuerzburg.de](mailto:nikolas.herbst@uni-wuerzburg.de)

4 VU University Amsterdam, NL, [alexandru.iosup@gmail.com](mailto:alexandru.iosup@gmail.com)

---

## Abstract

In the backbone of our digital society, cloud computing enables an efficient, utility-like ecosystem of developing, composing, and providing software services. Responding to a trend to make cloud computing services more accessible, fine-grained, and affordable, *serverless computing* has gained rapid adoption in practice, and garnered much interest from both industry and academia.

However successful, serverless computing manifests today the opportunities and challenges of emerging technology: a rapidly growing field but scattered vision, plenty of new technologies but no coherent approach to design solutions from them, many simple applications but no impressive advanced solution, the emergence of a cloud continuum (resources from datacenters to the edge) but no clear path to leverage it efficiently, and overall much need but also much technical complexity.

Several related but disjoint fields, notably software and systems engineering, parallel and distributed systems, and system and performance analysis and modeling, aim to address these opportunities and challenges. Excellent collaboration between these fields in the next decade will be critical in establishing serverless computing as a viable technology.

We organized this Dagstuhl seminar to bring together researchers, developers, and practitioners across disciplines in serverless computing, to develop a vision and detailed answers to the timely and relevant, open challenges related to the following topics:

- Topic 1: design decisions for serverless systems, platforms, and ecosystems,
  - Topic 2: software engineering of serverless applications, but also systems, platforms, and ecosystems
  - Topic 3: applications and domain requirements for serverless computing,
  - Topic 4: evaluation of serverless solutions,
- and beyond (privacy, cyber-physical systems, etc.).

In this document, we report on the outcomes of Dagstuhl Seminar 21201 “Serverless Computing” by integrating diverse views and synthesizing a shared vision for the next decade of serverless computing.

**Seminar** May 16–21, 2021 – <http://www.dagstuhl.de/21201>

**2012 ACM Subject Classification** General and reference → Performance; Computer systems organization → Cloud computing; Computer systems organization → Grid computing; Software and its engineering → Distributed systems organizing principles; Software and its engineering → Software organization and properties

**Keywords and phrases** Cloud computing, Cloud continuum, data-driven, design patterns, DevOps, experimentation, model-driven, serverless computing, simulation, software architecture, systems management, vision

**Digital Object Identifier** 10.4230/DagRep.11.4.34



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 4.0 International license

Serverless Computing, *Dagstuhl Reports*, Vol. 11, Issue 04, pp. 34–93

Editors: Cristina Abad, Ian T. Foster, Nikolas Herbst, and Alexandru Iosup



DAGSTUHL  
REPORTS

Dagstuhl Reports  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Executive Summary

*Cristina Abad (ESPOL – Guayaquil, EC)*

*Ian T. Foster (Argonne National Laboratory – Lemont, US)*

*Nikolas Herbst (Universität Würzburg, DE)*

*Alexandru Iosup (VU University Amsterdam, NL)*

License © Creative Commons BY 4.0 International license  
© Cristina Abad, Ian T. Foster, Nikolas Herbst, and Alexandru Iosup

Serverless computing holds a significant promise for the modern, digital society. For the past seven decades, our society has increasingly required ever-cheaper, ever-more convenient, and ever-faster computing technology. In the late-1950s, leasing time on an IBM 701 cost \$15,000 per month (\$135,000 in 2020 dollars). Today, we can lease many times this computing power for mere pennies but need to be careful about the actual cost of doing so. Cloud computing, that is, the utility providing IT as a service, on-demand and pay-per-use, is a widely used computing paradigm that offers large economies of scale and promises extreme environmental efficiency. Born from a need to make cloud computing services more accessible, fine-grained, and affordable, *serverless computing* has garnered interest from both industry and academia. In our vision, serverless computing can meet this need, but to do this it will have to overcome its current status of emergent technology or risk its demise.

Cloud computing is already an established technology. Today, more than three-quarters of the US and European companies, and many private individuals, use cloud computing services<sup>1</sup>. The serverless market is blooming<sup>2</sup> and has already exceeded \$200 billion in 2020<sup>3</sup>. The cost of one hour on a cloud computer leased on-demand can be lower than a cent<sup>4</sup> and all the major cloud providers offer inexpensive access to diverse and state-of-the-art hardware. However cheap, cloud computing still poses daunting operational challenges to software professionals, in particular, how to manage the selection, operation, and other aspects of using cloud infrastructure (in short, *servers*). Correspondingly, it poses significant challenges to systems designers and administrators, related to keeping the cloud infrastructure efficient and sustainable.

An emerging class of cloud-based software architectures, *serverless computing*, focuses on providing software professionals the ability to execute arbitrary functions with low or even no overhead in server management. Serverless computing leverages recent developments in the miniaturization of software parts through *microservice-based architectures*, in the operationalization of small self-contained execution units through *containers*, and in their integration in service models such as *Function-as-a-Service (FaaS)*. Truly, serverless is more [12]. Early research successes [6, 15, 17, 18, 22] complement numerous industrial applications [9], from business-critical to scientific computing, from DevOps to side-tasks. Already, IT spending on serverless computing should exceed \$8 billion per year, by 2021.<sup>5</sup>

However promising, serverless computing has yet to mature and presents many hard, open challenges. There are numerous signs and reports [11, 14] that serverless computing poses critical challenges in software engineering, parallel and distributed systems operation,

<sup>1</sup> European Commission, Uptake of Cloud in Europe, Digital Agenda for Europe report by the Publications Office of the European Union, Luxembourg, Sep 2014. and Flexera, State of the Cloud Report, 2020.

<sup>2</sup> Gartner Inc. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17% in 2020. Press Release.

<sup>3</sup> Frank Gens. Worldwide and Regional Public IT Cloud Services 2019–2023 Forecast. Tech. Rep. by IDC, Doc. #US44202119, Aug 2019.

<sup>4</sup> Amazon AWS, Microsoft Azure, and Google Compute Engine offer VMs in this price range.

<sup>5</sup> “Function-as-a-Service Market - Global Forecast to 2021,” marketsandmarkets.com, Feb 2017.

and performance engineering [10]. For example, software engineering could help overcome challenges in the developer experience [23], including testing, tooling, functionality, and training and education. The systems side requires, among others, new approaches for deployment, monitoring, and general operation, and also specific advances in security, cost predictability, and life-cycle management for cloud functions. Performance engineering raises many hard aspects, such as performance optimization, engineering for cost-efficiency, and various forms of fast online scheduling. These combined challenges are distinctive from the general challenges of cloud computing, for example, because the fine-grained, often event-driven nature of serverless computing typically requires approaches that are lightweight and able to respond without delay.

The goal of the seminar is to *combine the views of a diverse and high-quality group of researchers spanning three disciplines*: software engineering, parallel and distributed systems, and performance engineering. The Dagstuhl Seminar will be a catalyst. Attendees discussed the open challenges and opportunities of serverless computing for the next decade, with a focus on at least the following crucial aspects and questions:

- Envision serverless systems and applications in the next decade. How to leverage the freedom from operational concerns? How to overcome the challenge and enjoy the benefits of fine granularity?
- How to properly engineer serverless software and systems? What are the emerging architectural patterns for serverless systems and applications? How to test and debug serverless systems and applications?
- How to characterize, model, and analyze serverless systems and applications? How to understand the diverse serverless workloads?
- How to manage the resources used in serverless operations? How to schedule and orchestrate in this environment? How to manage specific application classes, such as computer vision, enterprise workflows, HPC, DevOps?
- How to deploy and manage the full lifecycle of serverless applications? How to add ML-capabilities to feedback loops? How to break through the operational silos?
- How to support privacy, security, dependability, and other desirable operational properties for serverless applications and systems?
- Beyond computer systems, how to consider serverless systems and applications from a holistic, cyberphysical perspective?

## Core topics

The seminar focussed on the following key topics related to serverless computing:

**Topic 1. Design decisions for serverless systems, platforms, and ecosystems.** As the serverless model is increasingly being adopted in industry [9], the challenges of properly designing these systems and the platforms on which they run are becoming more apparent. These challenges include important problems [10], such as: how to reduce the serverless overhead added by the platform to the (commonly lightweight) functions representing the business logic of the application (e.g., see [20]), how to ensure proper performance isolation while making efficient use of the shared infrastructure (e.g., see [1]), how to partition the functions [5, 6], and how to properly schedule functions and route requests to these functions (e.g., see [2]), in such a way that the service level objectives (SLO's) are adequately met, among other important challenges. There is also the question of running serverless workloads alongside conventional applications, e.g., HPC, big data, machine learning. The experiences

of the attendees to the seminar, some of which have already started working in these domain and others with established experience in prior technologies from which we may learn and transfer knowledge (e.g., grid computing), will enable us to focus on determining which of these decisions the community should be focusing on, and how to establish adequately prioritized research agendas.

**Topic 2. Software engineering of serverless applications, but also systems, platforms, and ecosystems.** To increase the domain of application for serverless computing, the functionality it can express needs to become increasingly more complex, which contrasts with the perceived simplicity of the model [23]. What is the trade-off between simplicity and expressiveness? Which composition models can ensure that serverless workflows can be maintained and developed (and updated) long term? Serverless functions should become increasingly interoperable, and applications should become able to leverage the services of any serverless platform [6]. How to make serverless functions vendor-agnostic and how to run serverless applications across cloud federations? Which architectural patterns are useful for serverless applications? How to consider and support the legacy part of serverless applications? The development processes, from the macro-view of how teams coordinate to deliver applications that could operate in an external ecosystem, to the micro-view of how to develop and test a serverless function, will have to consider the new aspects raised by serverless computing. What are effective development processes? What tools and IDE features are needed? What versioning and testing, and what CI/CD protocols should be used? How to evolve legacy software toward serverless-native applications? How to ensure open-source software becomes FAIR software [13]?

**Topic 3. Applications and domain requirements for serverless computing.** Preliminary studies of serverless applications at large [9] have shown that there is a wide variety of scenarios for which industry and academia are adopting serverless approaches. From business-critical workloads, to automating DevOps, scientific computing, and beyond, the diversity of the applications and domains for which serverless is being applied poses significant challenges when attempting to optimally manage the resources and infrastructure on which these applications depend. It is important to properly understand the variety of these applications and domain requirements, engaging both academia and industry in the discussion.

These requirements should relate to various aspects in software engineering, parallel and distributed systems, and performance engineering. For example, a domain-based approach could help increase scalability [3]; considering the structure of packages in composing a deployable serverless application could improve scheduling performance [2]; and serverless functions and architectures should be considered during performance tests [8, 28].

**Topic 4. Evaluation of serverless computing systems, platforms, and ecosystems.** The performance trade-offs of serverless systems are not yet well understood [28], thus highlighting the importance of proper evaluation and benchmarking of these systems. However, the high level of abstraction and the opaqueness of the operational-side make evaluating these platforms particularly challenging. As recent efforts are starting to focus on this topic [24, 28], it is important to engage the community on an early discussion on the best approaches to tackle this problem. How to understand and engineer the performance of serverless systems? How to translate the findings, when serverless systems are opened to external developers (as platforms) or take part in much larger systems of systems (and even ecosystems)? How to account for parts of the ecosystem being closed-source and even acting as black-boxes? How to identify and even explain the performance bottlenecks such systems, platforms, and

ecosystems experience? How to use evaluation results with other performance engineering techniques to control and improve the performance of serverless systems, platforms, and ecosystems?

An important focus of inquiry has recently become prominent in computer systems research: the reproducibility of evaluation results and of experiments in general [19, 21]. Not doing so can result in misleading results [26], and in results that cannot be obtained again [25] sometimes even under identical circumstances and by their original authors [7]. This leads to a possible loss of faith in the entire field [4, 27]. “How to benchmark serverless solutions reproducibly?” is an important question to address with diverse expertise and fresh ideas.

### Synopsis and Planned Actions

We would like to thank the Dagstuhl staff and sponsors for this unique seminar opportunity even under the constraints of the pandemic. During the seminar, we had almost 45h of online meetings (not counting sub-meetings): some 9-10h of online meetings each seminar day. Three 3h sessions per day were spread around the clock to allow participation from various timezones. Even under these constraints, we experienced enormous participation and active discussion involvement. In brief, the seminar week was structured as follows:

After each participant presented her/himself to the plenary, we formed four working groups according to the topics above. The discussions were kick-started by four distinguished keynotes, in plenary, with the respective talk abstracts included in this report:

- “Serverless Predictions: 2021-2030” given jointly by Pedro García López (Universitat Rovira i Virgili – Tarragona, ES) and Bernard Metzler (IBM Research-Zurich, CH)
- “Developer Experience for Serverless: Challenges and Opportunities” given by Robert Chatley (Imperial College London, GB)
- “Federated Function as a Service” given jointly by Kyle Chard (University of Chicago, US) and Ian T. Foster (Argonne National Laboratory – Lemont, US)
- “Characterizing Serverless Systems” given by Mohammad Shahrad (University of British Columbia – Vancouver, CA)

Each of the four working groups held five 3h sessions with their teams, including three 1h one-on-one meetings with the other groups. The four working groups report individually on their outcomes and list identified research challenges. In a consolidation phase, we identified and planned nine focused topics for future joint research among the participants.

Complemented by a Slack workspace for the seminar participants, a focused continuation of discussions beyond the seminar week was enabled: Among others, a discussion initiated and led by Samuel Kounev on the notion of serverless computing, started during the seminar, continued well beyond. We include the outcome of this “panel discussion” in Section 5.1 of this report.

The organizers and participants decided to jointly work toward at least one high-profile magazine article reporting on the seminar outcome and research agenda.

Furthermore, during the seminar the motion was raised to establish a conference series on serverless computing. We see good potential for a new conference on “Serverless Software and Systems” as a cross-community event embracing, at least, the disciplines of software engineering, system engineering, and performance engineering. Working potentially in concert with an existing workshop series in the field, we plan to initiate this step in the coming months. We hope that one day in the future, we can proudly look back and say that this Dagstuhl seminar 21201 was an important trigger event.

**References**

- 1 Z. Al-Ali, S. Goodarzy, E. Hunter, S. Ha, R. Han, E. Keller, and E. Rozner. Making serverless computing more serverless. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.
- 2 G. Aumala, E. Boza, L. Ortiz-Avilés, G. Totoy, and C. Abad. Beyond load balancing: Package-aware scheduling for serverless platforms. In 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2019.
- 3 Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, André van Hoorn, Henning Schulz, Daniel S. Menasché, and Vilc Queupe Rufino. Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests. *J. Syst. Softw.*, 165:110564, 2020.
- 4 Monya Baker. Is there a reproducibility crisis? *Nature*, 533(7604):452–454, 2016.
- 5 Edwin F. Boza, Xavier Andrade, Jorge Cedeno, Jorge R. Murillo, Harold Aragon, Cristina L. Abad, and Andres G. Abad. On implementing autonomic systems with a serverless computing approach: The case of self-partitioning cloud caches. *Comput.*, 9(1):14, 2020.
- 6 Ryan Chard, Yadu N. Babuji, Zhuozhao Li, Tyler J. Skluzacek, Anna Woodard, Ben Blaiszik, Ian T. Foster, and Kyle Chard. funcx: A federated function serving fabric for science. In *HPDC '20: The 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 65–76. ACM, 2020.
- 7 Christian S. Collberg and Todd A. Proebsting. Repeatability in computer systems research. *Commun. ACM*, 59(3):62–69, 2016.
- 8 Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, Dusan Okanovic, and André van Hoorn. Microservices: A performance tester’s dream or nightmare? In *ICPE '20: ACM/SPEC International Conference on Performance Engineering*, 2020, pages 138–149. ACM, 2020.
- 9 Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, and Alexandru Iosup. A review of serverless use cases and their characteristics. Technical Report SPEC-RG-2020-5, SPEC RG Cloud Working Group, May 2020.
- 10 Erwin Van Eyk, Alexandru Iosup, Cristina L. Abad, Johannes Grohmann, and Simon Eismann. A SPEC RG cloud group’s vision on the performance challenges of FaaS cloud architectures. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018*, pages 21–24, 2018.
- 11 Erwin Van Eyk, Alexandru Iosup, Simon Seif, and Markus Thömmes. The SPEC cloud group’s research vision on FaaS and serverless architectures. In *Proceedings of the 2nd International Workshop on Serverless Computing, WOSC@Middleware 2017, Las Vegas, NV, USA, December 12, 2017*, pages 1–4, 2017.
- 12 Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uta, and Alexandru Iosup. Serverless is more: From PaaS to present cloud computing. *IEEE Internet Comput.*, 22(5):8–17, 2018.
- 13 Wilhelm Hasselbring, Leslie Carr, Simon Hettrick, Heather S. Packer, and Thanassis Tiropanis. From FAIR research data toward FAIR and open research software. *Inf. Technol.*, 62(1):39–47, 2020.
- 14 Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. [www.cidrdb.org](http://www.cidrdb.org), 2019.
- 15 Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2016, Denver, CO, USA, June 20-21, 2016.*, 2016.

- 16 Alexandru Iosup, Catia Trubiani, Anne Kozirolek, José Nelson Amaral, Andre B. Bondi, and Andreas Brunnert. Flexibility is key in organizing a global professional conference online: The ICPE 2020 experience in the COVID-19 era. *CoRR*, abs/2005.09085, 2020.
- 17 Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A berkeley view on serverless computing. *CoRR*, abs/1902.03383, 2019.
- 18 Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 427–444. USENIX Association, 2018.
- 19 Ravi Madduri, Kyle Chard, Mike D’Arcy, Segun C. Jung, Alexis Rodriguez, Dinanath Sulakhe, Eric Deutsch, Cory Funk, Ben Heavner, Matthew Richards, Paul Shannon, Gustavo Glusman, Nathan Price, Carl Kesselman, and Ian Foster. Reproducible big data science: A case study in continuous fairness. *PLOS ONE*, 14(4):1–22, 04 2019.
- 20 Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid task provisioning with serverless-optimized containers. In *USENIX Annual Technical Conference (USENIX ATC 18)*, July 2018.
- 21 A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. Tuma, and A. Iosup. Methodological principles for reproducible performance evaluation in cloud computing. *IEEE Trans. Software Eng.*, pages 1–1, 2019.
- 22 Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In Jay R. Lorch and Minlan Yu, editors, *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, pages 193–206. USENIX Association, 2019.
- 23 Mike Roberts. Serverless architectures. <https://martinfowler.com/articles/serverless.html>, 2016. Continuous development of the material.
- 24 Joel Scheuner and Philipp Leitner. Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software*, 2020.
- 25 Dag I. K. Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Trans. Software Eng.*, 31(9):733–753, 2005.
- 26 Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan S. Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. Is big data performance reproducible in modern cloud networks? In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 513–527, 2020.
- 27 Erik van der Kouwe, Gernot Heiser, Dennis Andriess, Herbert Bos, and Cristiano Giuffrida. Benchmarking flaws undermine security research. *IEEE Secur. Priv.*, 18(3):48–57, 2020.
- 28 Erwin van Eyk, Joel Scheuner, Simon Eismann, Cristina L. Abad, and Alexandru Iosup. Beyond microbenchmarks: The SPEC-RG vision for a comprehensive serverless benchmark. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, page 26–31, 2020.

## 2 Table of Contents

### Executive Summary

*Cristina Abad, Ian T. Foster, Nikolas Herbst, and Alexandru Iosup* . . . . . 35

### Overview of Talks

Serverless Predictions: 2021-2030 (Keynote Abstract – Topic 1)  
*Pedro García López and Bernard Metzler* . . . . . 44

Developer Experience for Serverless: Challenges and Opportunities (Keynote Abstract – Topic 2)  
*Robert Chatley* . . . . . 45

Federated Function as a Service (Keynote Abstract – Topic 3)  
*Kyle Chard and Ian T. Foster* . . . . . 45

Characterizing Serverless Systems (Keynote Abstract – Topic 4)  
*Mohammad Shahrad* . . . . . 48

Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms  
*Cristina Abad* . . . . . 48

Accelerating Reads with In-Network Consistency-Aware Load Balancing  
*Samer Al-Kiswany* . . . . . 49

A tool set for serverless  
*Ahmed Ali-Eldin Hassan* . . . . . 49

Serverless execution of scientific workflows  
*Bartosz Balis* . . . . . 50

Using Serverless Computing for Streamlining the Data Analytic Process  
*André Bauer* . . . . . 50

Challenges for Serverless Databases  
*A. Jesse Jiryu Davis* . . . . . 50

Using Serverless to Improve Online Gaming  
*Jesse Donkeroliet and Alexandru Iosup* . . . . . 51

Understanding and optimizing serverless applications  
*Simon Eismann* . . . . . 51

Autonomous resource allocation methods for serverless systems  
*Erik Elmroth* . . . . . 52

Is Serverless an Opportunity for Edge Applications?  
*Nicola Ferrier* . . . . . 52

HyScale into Serverless: Vision and Challenges  
*Hans-Arno Jacobsen* . . . . . 53

Serverless Workflows for Sustainable High-Performance Data Analytics  
*Nikolas Herbst* . . . . . 54

Massivizing Computer Systems: Science, Design, and Engineering for Serverless Computing  
*Alexandru Iosup and Erwin van Eyk* . . . . . 54



Machine Learning to enable Autonomous Serverless Systems <i>Pooyan Jamshidi</i> . . . . .	56
Self-Aware Platform Operations and Resource Management <i>Samuel Kounev</i> . . . . .	57
From design to migration and management: FaaS platforms for application porting to optimized serverless implementation and execution <i>Georgios Kousiouris</i> . . . . .	57
Software Development Using Serverless Systems <i>Philipp Leitner</i> . . . . .	58
Running and Scheduling Scientific Workflows on Serverless Clouds: From Functions to Containers <i>Maciej Malawski</i> . . . . .	58
The case for a hybrid cloud model for serverless computing <i>Vinod Muthusamy</i> . . . . .	59
Performance Evaluation in Serverless Computing <i>Alessandro Vittorio Papadopoulos</i> . . . . .	60
Federated AI on Serverless Edge Clusters Powered by Renewable Energy <i>Panos Patros</i> . . . . .	60
Is serverless computing the holy grail of fog computing application design paradigms? <i>Guillaume Pierre</i> . . . . .	61
Performance Evaluation of Serverless Applications <i>Joel Scheuner</i> . . . . .	62
FaaS orchestration <i>Mina Sedaghat</i> . . . . .	62
LaSS: Running Latency Sensitive Serverless Computations at the Edge <i>Prashant Shenoy and Ahmed Ali-Eldin Hassan</i> . . . . .	63
Fitting Serverless Abstractions and System Designs to Next-Generation Application Needs <i>Josef Spillner</i> . . . . .	64
Architectural Patterns for Serverless-Based applications <i>Davide Taibi</i> . . . . .	64
Continuous testing of serverless applications <i>André van Hoorn</i> . . . . .	65
Serverless Compute Primitives as a Compilation Target <i>Soam Vasani</i> . . . . .	66
Network Challenges in Serverless Computing <i>Florian Wamser</i> . . . . .	66
Decision Support for Modeling and Deployment Automation of Serverless Applica- tions <i>Vladimir Yussupov</i> . . . . .	67

**Working groups**

Design of Serverless Systems, Platforms, and Ecosystems (Topic 1) <i>Samer Al-Kiswany, Ahmed Ali-Eldin Hassan, André Bauer, André B. Bondi, Ryan L. Chard, Andrew A. Chien, A. Jesse Jiryu Davis, Erik Elmroth, Alexandru Iosup, Hans-Arno Jacobsen, Samuel Kounev, Vinod Muthusamy, Guillaume Pierre, Mina Sedaghat, Prashant Shenoy, Davide Taibi, Douglas Thain, Erwin van Eyk, and Soam Vasani</i> . . . . .	68
Software Engineering of Serverless Applications, but also Systems, Platforms, and Ecosystems (Topic 2) <i>Simon Eismann, Robert Chatley, Nikolas Herbst, Georgios Kousiouris, Philipp Leitner, Pedro García López, Bernard Metzler, Davide Taibi, Vincent van Beek, André van Hoorn, Guido Wirtz, and Vladimir Yussupov</i> . . . . .	73
Serverless Applications and Requirements (Topic 3) <i>Josef Spillner, Bartosz Balis, Jesse Donkerliet, Nicola Ferrier, Ian T. Foster, Maciej Malawski, Panos Patros, Omer F. Rana, and Florian Wamser</i> . . . . .	77
Evaluation of Serverless Systems (Topic 4) <i>Cristina Abad, Kyle Chard, Pooyan Jamshidi, Alessandro Vittorio Papadopoulos, Robert P. Ricci, Joel Scheuner, Mohammad Shahrads, and Alexandru Uta</i> . . . . .	86

**Panel discussions**

Toward a Definition for Serverless Computing <i>Samuel Kounev, Cristina Abad, Ian T. Foster, Nikolas Herbst, Alexandru Iosup, Samer Al-Kiswany, Ahmed Ali-Eldin Hassan, Bartosz Balis, André Bauer, André B. Bondi, Kyle Chard, Ryan L. Chard, Robert Chatley, Andrew A. Chien, A. Jesse Jiryu Davis, Jesse Donkerliet, Simon Eismann, Erik Elmroth, Nicola Ferrier, Hans-Arno Jacobsen, Pooyan Jamshidi, Georgios Kousiouris, Philipp Leitner, Pedro García López, Martina Maggio, Maciej Malawski, Bernard Metzler, Vinod Muthusamy, Alessandro Vittorio Papadopoulos, Panos Patros, Guillaume Pierre, Omer F. Rana, Robert P. Ricci, Joel Scheuner, Mina Sedaghat, Mohammad Shahrads, Prashant Shenoy, Josef Spillner, Davide Taibi, Douglas Thain, Animesh Trivedi, Alexandru Uta, Vincent van Beek, Erwin van Eyk, André van Hoorn, Soam Vasani, Florian Wamser, Guido Wirtz, and Vladimir Yussupov</i> . . . . .	89
---	----

<b>Participants</b> . . . . .	92
-------------------------------	----

<b>Remote Participants</b> . . . . .	92
--------------------------------------	----

### 3 Overview of Talks

#### 3.1 Serverless Predictions: 2021-2030 (Keynote Abstract – Topic 1)

*Pedro García López (Universitat Rovira i Virgili – Tarragona, ES) and Bernard Metzler (IBM Research-Zurich, CH)*

**License** © Creative Commons BY 4.0 International license

© Pedro García López and Bernard Metzler

**Joint work of** Pedro García López, Aleksander Slominski, Michael Behrendt, Bernard Metzler

**Main reference** Pedro García López, Aleksander Slominski, Michael Behrendt, Bernard Metzler: “Serverless Predictions: 2021-2030”, CoRR, Vol. abs/2104.03075, 2021.

**URL** <https://arxiv.org/abs/2104.03075>

Within the next 10 years, advances on resource disaggregation will enable full transparency for most Cloud applications: to run unmodified single-machine applications over effectively unlimited remote computing resources. In this article, we present five serverless predictions for the next decade that will realize this vision of transparency – equivalent to Tim Wagner’s Serverless SuperComputer or AnyScale’s Infinite Laptop proposals.

The major hypothesis is that transparency will be achieved in the next ten years thanks to novel advances in networking, disaggregation, and middleware services. The huge consequence is the unification of local and remote paradigms, which will democratize distributed programming for a majority of users. This will realize the old and ultimate goal of hiding the complexity of distributed systems. The projected developments to reach the ultimate goal (Serverless End Game) include the following:

- Prediction 1: Serverless Clusters (Multi-tenant Kubernetes) will overcome the current limitations of direct communication among functions, hardware acceleration, and time limits.
- Prediction 2: Serverless Granular computing will offer 1-10  $\mu$ s microsecond latencies for remote functions thanks to lightweight virtualization and fast RPCs.
- Prediction 3: Serverless memory disaggregation will offer shared mutable state and coordination at 2-10  $\mu$ s microsecond latencies over persistent memory.
- Prediction 4: Serverless Edge Computing platforms leveraging 6G’s ms latencies and AI optimizations will facilitate a Cloud Continuum for remote applications.
- Prediction 5: Transparency will become the dominant software paradigm for most applications, when computing resources become standardized utilities.

We discuss the basis of these predictions as well as technical challenges and risks. The predictions are mapped to phases to reach the final goal.

In conclusion, we argue that full transparency will be possible soon thanks to low latency and resource disaggregation. The Serverless End Game will unify local and remote programming paradigms, changing completely the way we currently create distributed applications. This is the ultimate goal of distributed systems, to become invisible using transparent middleware, and to simplify how users access remote resources.

### 3.2 Developer Experience for Serverless: Challenges and Opportunities (Keynote Abstract – Topic 2)

Robert Chatley (*Imperial College London, GB*)

**License** © Creative Commons BY 4.0 International license  
© Robert Chatley

**Joint work of** Robert Chatley, Goko, Adzic, Thomas Allerton, Hang Li Li

**Main reference** Robert Chatley, Thomas Allerton: “Nimbus: improving the developer experience for serverless applications”, in Proc. of the ICSE ’20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June – 19 July, 2020, pp. 85–88, ACM, 2020.

**URL** <https://doi.org/10.1145/3377812.3382135>

In this keynote talk we present a number of industrial case studies of building serverless systems, from the developer point of view. We examine how economic aspects – for example billing models – affect architectural design decisions for serverless applications, and also how available tooling inhibits or enhances developer experience when building, running and evolving serverless systems. We look at some current challenges, and propose some possible future directions aiming to address these.

After completing his PhD, Robert spent many years working in industry as a senior software engineer and a consultant before returning to university life. His work now bridges industry and academia, focussing on developing skills and knowledge in software engineers to build technical competence and improve developer productivity. His role at Imperial combines a strong focus on education with industry-focussed research. Robert’s main interests are in developer experience – trying to support and improve developer productivity through advances in tools, technologies and processes.

### 3.3 Federated Function as a Service (Keynote Abstract – Topic 3)

Kyle Chard (*University of Chicago, US*) and Ian T. Foster (*Argonne National Laboratory – Lemont, US*)

**License** © Creative Commons BY 4.0 International license  
© Kyle Chard and Ian T. Foster

**Main reference** Ryan Chard, Yadu N. Babuji, Zhuozhao Li, Tyler J. Skluzacek, Anna Woodard, Ben Blaiszik, Ian T. Foster, Kyle Chard: “funcX: A Federated Function Serving Fabric for Science”, in Proc. of the HPDC ’20: The 29th International Symposium on High-Performance Parallel and Distributed Computing, Stockholm, Sweden, June 23-26, 2020, pp. 65–76, ACM, 2020.

**URL** <https://doi.org/10.1145/3369583.3392683>

## Introduction

The serverless paradigm has revolutionized programming by allowing programmers to develop, run, and manage scalable *applications* without needing to build and operate the *infrastructure* that would normally be required to host those applications [5]. In particular, the popular function as a service (FaaS) model reduces application development to two tasks: defining and registering (or discovering) high-level programming language functions, and invoking those functions. The underlying FaaS platform, traditionally operated by a cloud provider, then deals with the complexities of provisioning and managing the servers, virtual machines, containers, and programming environments needed to run those functions.

Most current FaaS offerings adopt the powerful simplifying assumption that functions run on a single, centralized, and homogeneous platform, whether a commercial (public) cloud like AWS, Azure, or Google, or a dedicated (private) cluster as in the case of OpenWhisk. In such environments, FaaS systems provide simple and intuitive APIs that democratize access to seemingly unlimited remote elastic computing capacity. But modern computing environments are increasingly distributed and heterogeneous.

For example, quasi-ubiquitous machine learning methods require access to specialized hardware (e.g., AI accelerators) and introduce new workload patterns (e.g., large-memory training and short-duration inference) and interactive and event-based computing models (e.g., from automated laboratories and robots) that require instantaneous access to specialized computing capabilities. Sensor network applications often require that data be processed near to data sources to reduce bandwidth needs and/or enable rapid response.

To address these concerns, we propose a new *federated FaaS model* in which function executions can be dispatched to arbitrary computing resources, chosen for example on the basis of latency, cost, data locality, security, or other concerns. This new model preserves and leverages powerful features of conventional FaaS (e.g., simple function registration and invocation APIs, abstraction of infrastructure) while also allowing programmers to operate effectively in a distributed computational continuum [2]. In effect, federated FaaS aims to allow computation to flow to wherever makes sense for a particular purpose.

### **funcX: early experiences with federated FaaS**

funcX [4] is a federated FaaS platform designed to address some of the challenges outlined above. funcX adapts the traditional cloud-hosted FaaS model by enabling users to route function invocations to a distributed set of user-deployed funcX endpoints. Thus, users can add their own computing system (e.g., cluster, cloud, laptop) to the funcX ecosystem by deploying an endpoint and they may then use those endpoints to execute functions. From a user's perspective, funcX looks like any other FaaS system: users register functions with the cloud-hosted funcX service, they may then invoke that function by specifying input arguments and the target endpoint. funcX manages the complexity of execution, authenticating with the remote endpoint, reliably executing the function (optionally inside a container), and caching results (or exceptions) until retrieved by the user.

Over the past year we have applied funcX to a range of research applications and as the basis for building other services (e.g., DLHub [3]). We have found that funcX can effectively abstract the complexity of using diverse computing resources, simplify authentication and authorization, reduce the difficulties associated with scaling resources to support workloads, remove the challenge of porting applications between different systems and data centers, and enable new application modes such as event-based and interactive computing.

We have also identified limitations of the federated FaaS model as realized in our work to date. For example, many applications cannot easily be mapped to the FaaS paradigm; funcX's centralized data and state management restrict the application patterns that can be implemented, and require that the ratio of data size to compute must be reasonable to keep transfer overheads manageable; containers fail to solve portability problems in HPC environments; and the coarse-grained allocation models of HPC systems do not lend themselves well to function execution. These are all topics that we are addressing in current work.

### **Open challenges**

The federated FaaS model introduces fascinating research challenges, including the following.

**Data and State.** Traditionally, FaaS functions are stateless. However, many applications require that data be passed to, from, and between functions. (Indeed, data locality is one of the main reasons to apply a federated model.) Conventional cloud-hosted FaaS platforms

meet these needs via universal object storage; however, such storage is not generally accessible in federated settings. There is a need to explore FaaS application communication patterns, data-centric programming models for FaaS, transparent wide-area data staging, and shared data substrates for low-latency data sharing.

**Environment management.** FaaS systems leverage containerized environments that enable dependencies to be met while sandboxing execution in multi-tenant environments. Cloud FaaS systems have developed new software container technologies with rapid startup time and low cold start overheads [1]. The heterogeneous environments found in federated FaaS create more challenges, such as diverse container technologies and slow resource provisioning [7].

**Scheduling.** Increasingly heterogeneous computing environments create a continuum of computing capacity: from edge computing devices through to supercomputers. Federated FaaS makes it easy to route functions to execute anywhere, and thus exposes a fabric on which broad scheduling policies can be explored. Such scheduling policies may consider data locations, transfer costs, resource provisioning time, resource costs (monetary and/or availability), hardware performance, and function needs [6].

**Security, policies, regulations.** Federated FaaS is distinguished from conventional FaaS by a quite different security model. In a federated environment, each computing endpoint may be located in a distinct administrative domain with unique authentication and authorization systems, policies, and regulations. Centralized FaaS systems typically operate within a single administrative domain. Federated FaaS requires methods for bridging domains and for ensuring that policies and regulations are enforced.

## Summary


Federated FaaS provides a potential solution to long-standing remote computing challenges. In so doing, it enables a range of new application scenarios and moves us closer to a truly integrated treatment of local and remote computing. It also exposes fascinating new research challenges that will only grow in importance as both application demands and technologies continue to develop.

## References

- 1 A Agache et al., Firecracker: Lightweight virtualization for serverless applications, 17th USENIX Symposium on Networked Systems Design and Implementation, 2020, pp. 419–434.
- 2 P Beckman et al., Harnessing the computing continuum for programming our world, *Fog Computing: Theory and Practice* (2020), 215–230.
- 3 R Chard et al., Dithub: Model and data serving for science, 35rd IEEE International Parallel and Distributed Processing Symposium, 2019, pp. 283–292.
- 4 R Chard et al., FuncX: A federated function serving fabric for science, 29th International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2020, p. 65–76.
- 5 S Eismann et al., Serverless applications: Why, when, and how?, *IEEE Software* 38(2020), no. 1, 32–39.
- 6 R Kumar et al., Coding the computing continuum: Fluid function execution in heterogeneous computing environments, *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2021, pp. 66–75.
- 7 T Shaffer et al., Lightweight function monitors for fine-grained management in large scale Python applications, *IEEE International Parallel and Distributed Processing Symposium*, 2021, pp. 786–796

### 3.4 Characterizing Serverless Systems (Keynote Abstract – Topic 4)

Mohammad Shahrad (*University of British Columbia – Vancouver, CA*)

License  Creative Commons BY 4.0 International license  
© Mohammad Shahrad

This keynote is dedicated to understanding the importance of characterizing serverless systems from different perspectives. To make the case, two characterization studies will be presented: 1) a cluster-wide characterization of the entire serverless workload at Azure Functions [1], and 2) a detailed micro-architectural study of Apache OpenWhisk [2]. The insights gained by the first study lead to designing an adaptive scheduling policy reducing cold starts and resource wastage, and the observations in the second study reveal inefficiencies in cloud-grade processors in serving serverless workloads. The talk also emphasizes the importance of reproducibility through open-sourcing traces or tools.

#### References

- 1 Mohammad Shahrad, Rodrigo Fonseca, and Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. *Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider*. USENIX Annual Technical Conference (USENIX ATC), 2020
- 2 Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. *Architectural implications of function-as-a-service computing*. 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2019.

### 3.5 Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms

Cristina Abad (*ESPOL – Guayaquil, EC*)

License  Creative Commons BY 4.0 International license  
© Cristina Abad

**Joint work of** Gabriel Aumala, Edwin F. Boza, Luis Ortiz-Avilés, Gustavo Totoy, Cristina L. Abad  
**Main reference** Gabriel Aumala, Edwin F. Boza, Luis Ortiz-Avilés, Gustavo Totoy, Cristina L. Abad: “Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms”, in Proc. of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019, pp. 282-291, IEEE, 2019.  
**URL** <https://doi.org/10.1109/CCGRID.2019.00042>

Fast deployment and execution of cloud functions in Function-as-a-Service (FaaS) platforms is critical, for example, for user-facing services in microservices architectures. However, functions that require large packages or libraries are bloated and start slowly. An optimization is to cache packages at the worker nodes instead of bundling them with the functions. However, existing FaaS schedulers are vanilla load balancers, agnostic to packages cached in response to prior function executions, and cannot properly reap the benefits of package caching. We study the case of package-aware scheduling and propose PASch, a novel Package-Aware Scheduling algorithm that seeks package affinity during scheduling so that worker nodes can re-use execution environments with preloaded packages. PASch leverages consistent hashing and the power of two choices, while actively avoiding worker overload. We implement PASch in a new scheduler for the OpenLambda framework and evaluate it using simulations and real experiments. We evaluated PASch with varying cluster sizes and skewness of package popularity distribution, and found that it outperforms a regular balancer by as much as 318x (median speedup). Furthermore, for the workloads studied in this paper, PASch can outperform consistent hashing with bounded loads – a state-of-the-art load balancing algorithm – by 1.3x (mean speedup), and a speedup of 1.5x at the 80th percentile.

### 3.6 Accelerating Reads with In-Network Consistency-Aware Load Balancing

*Samer Al-Kiswany (University of Waterloo, CA)*

**License** © Creative Commons BY 4.0 International license  
© Samer Al-Kiswany

**Joint work of** Hatem Tahruri, Ibrahim Kettaneh, Ahmed Alquraan, Samer Al-Kiswany

**Main reference** Hatem Tahruri, Ibrahim Kettaneh, Ahmed Alquraan, Samer Al-Kiswany: “FLAIR: Accelerating Reads with Consistency-Aware Network Routing”, in Proc. of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), pp. 723–737, USENIX Association, 2020.

**URL** <https://www.usenix.org/conference/nsdi20/presentation/tahruri>

Replication is the main reliability technique for many modern cloud services that process billions of requests each day. Unfortunately, modern strongly-consistent replication protocols – such as multi-Paxos, Raft, Zab, and Viewstamped replication (VR) – deliver poor read performance. This is because these protocols are leader-based: a single leader replica (or leader, for short) processes every read and write request, while follower replicas (followers for short) are used for reliability only.

I present FLAIR, a novel approach for accelerating read operations in leader-based consensus protocols. FLAIR leverages the capabilities of the new generation of programmable switches to serve reads from follower replicas without compromising consistency. The core of the new approach is a packet-processing pipeline that can track client requests and system replies, identify consistent replicas, and at line speed, forward read requests to replicas that can serve the read without sacrificing linearizability. An additional benefit of FLAIR is that it facilitates devising novel consistency-aware load balancing techniques. Following the new approach, the research team designed FlairKV, a key-value store atop Raft. FlairKV implements the processing pipeline using the P4 programming language. We evaluate the benefits of the proposed approach and compare it to previous approaches using a cluster with a Barefoot Tofino switch. The evaluation indicates that, compared to state-of-the-art alternatives, the proposed approach can bring significant performance gains: up to 42% higher throughput and 35-97% lower latency for most workloads.

### 3.7 A tool set for serverless

*Ahmed Ali-Eldin Hassan (Chalmers University of Technology – Göteborg, SE)*

**License** © Creative Commons BY 4.0 International license  
© Ahmed Ali-Eldin Hassan

**Joint work of** Ahmed Ali-Eldin Hassan, Prashant Shenoy

Designing full stack serverless edge applications is a challenge. System dynamics of most edge applications poses challenges to what application can be developed using the serverless model to run on the edge. Specific challenges include function startup times, and state management. In this line of research, our aim is to develop models, tools, and frameworks that can enable programmers and system owners to harness the power of serverless computing for edge systems. We will initially focus on the problem of startup times and state management. Our aim is to eventually build an entire tool-base that enables for optimizes compiling applications into serverless functions, optimizes the deployment of serverless based applications, and optimizes the runtime on the edge.



### 3.8 Serverless execution of scientific workflows

*Bartosz Balis (AGH University of Science & Technology – Krakow, PL)*

**License** © Creative Commons BY 4.0 International license  
© Bartosz Balis

**Joint work of** Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, Kamil Figiela  
**Main reference** Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, Kamil Figiela: “Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions”, *Future Gener. Comput. Syst.*, Vol. 110, pp. 502–514, 2020.  
**URL** <https://doi.org/10.1016/j.future.2017.10.029>

Scientific workflows, consisting of a large number of tasks structured as a graph, are an important paradigm for automation in scientific computing. We discuss the applicability of serverless infrastructures to compute- and data-intensive workflows, and options for designing serverless workflow execution architecture. We also present cost analysis and implications with regard to resource management for scientific applications in the serverless paradigm. The approach is experimentally evaluated using the HyperFlow workflow management system and real workflow applications. Our findings indicate that the simple mode of operation makes the serverless approach attractive and easy to use, although for larger workflows traditional IaaS infrastructure is more cost-efficient. We conclude that a hybrid approach combining VMs with cloud functions for small tasks could be a good execution model for scientific workflows.

### 3.9 Using Serverless Computing for Streamlining the Data Analytic Process

*André Bauer (Universität Würzburg, DE)*

**License** © Creative Commons BY 4.0 International license  
© André Bauer

The discipline of data analytics has grown significantly in recent years as a means to make sense of the vast amount of data available. It has permeated every aspect of computer science and engineering and is heavily involved in business decision-making. However, data analytics projects are often done manually. To accelerate and improve such projects, there are, for example, Federated Learning and the best practices of DataOps. Since such approaches need a high degree of flexibility and should generate as little overhead as possible, I am interested in how far Serverless Computing can be used to guarantee these conditions.

### 3.10 Challenges for Serverless Databases

*A. Jesse Jiryu Davis (MongoDB – New York, US)*

**License** © Creative Commons BY 4.0 International license  
© A. Jesse Jiryu Davis

**Joint work of** Judah Schvimer, A. Jesse Jiryu Davis, Max Hirschhorn  
**Main reference** Judah Schvimer, A. Jesse Jiryu Davis, Max Hirschhorn: “eXtreme Modelling in Practice”, *Proc. VLDB Endow.*, Vol. 13(9), pp. 1346–1358, 2020.  
**URL** <https://doi.org/10.14778/3397230.3397233>

Academic research into serverless platforms has focused primarily on FaaS, not on the backend services such as databases that support FaaS applications. Furthermore, the literature mostly discusses how to use serverless platforms from the application developer’s perspective, rather

than how to implement them from the provider’s perspective. My research goal is to review the state of the art for implementing serverless platforms, particularly serverless databases. I am interested in methods for scaling and balancing tenants in multi-tenant serverless databases, and moving tenants between servers efficiently and without disruption. I am also interested in testing and validation methods for distributed systems algorithms, including formal methods such as TLA+.

### 3.11 Using Serverless to Improve Online Gaming

*Jesse Donkervliet (VU University Amsterdam, NL) and Alexandru Iosup (VU University Amsterdam, NL)*

**License** © Creative Commons BY 4.0 International license  
© Jesse Donkervliet and Alexandru Iosup

**Main reference** Jesse Donkervliet, Animesh Trivedi, Alexandru Iosup: “Towards Supporting Millions of Users in Modifiable Virtual Environments by Redesigning Minecraft-Like Games as Serverless Systems”, in Proc. of the 12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020, USENIX Association, 2020.

**URL** <https://www.usenix.org/conference/hotcloud20/presentation/donkervliet>

Serverless computing offers potential for the simpler and more cost-efficient deployment of large-scale systems. Online games are a billion dollar industry supported by large-scale distributed systems. How can these systems benefit from serverless computing? How to design real-time online games for serverless platforms? How to meet the QoS requirements of these systems? How to guarantee sufficient consistency between users? How to schedule its components cost- and energy-efficiently? I am interested in learning more about these questions and exploring their answers.

### 3.12 Understanding and optimizing serverless applications

*Simon Eismann (Universität Würzburg, DE)*

**License** © Creative Commons BY 4.0 International license  
© Simon Eismann

**Joint work of** Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup

**Main reference** Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup: “Serverless Applications: Why, When, and How?”, IEEE Softw., Vol. 38(1), pp. 32–39, 2021.

**URL** <https://doi.org/10.1109/MS.2020.3023302>

Serverless application are a novel computing paradigm with rapidly growing industry adoption. However, there are still many open questions about the characteristics of serverless applications, such as how many serverless functions does a typical serverless microservice consist of. Additionally, there are still a number of manual configurations that developers need to fine-tune in order to optimize their applications.

We collected 89 serverless applications from white literature, grey literature, open-source projects, and scientific computing and analyze their characteristics to provide insight into the current state of serverless applications. Further, we present approaches to model the performance of serverless workflows and serverless functions with different sizes to enable the automated optimization of serverless functions and workflows.

### 3.13 Autonomous resource allocation methods for serverless systems

*Erik Elmroth (University of Umeå, SE)*

**License**  Creative Commons BY 4.0 International license  
© Erik Elmroth


With the overall research direction being on how to build autonomous or semi-autonomous resource management systems for IT systems, we have done a lot of work on resource allocation topics such as scaling, orchestration, scheduling, service differentiation and all kind of techniques trying to control performance, efficiency, reliability, etc. And we have been doing this for systems spanning from servers and clusters to rack-scale systems, datacenters, edge environments, and so on. Serverless systems are obviously within scope for this.

In the past few years we have also spent more and more efforts on handling the situations that cannot be controlled, by focusing on anomaly detection, primarily trying to identify performance issues and their root causes but also considered functional and security anomalies, which are not always easy to distinguish.

When looking more specifically into serverless systems, we have recently initiated a project where we on one hand try to determine in advance what resources are needed and how they should be allocated to meet particular performance requirements. As these systems are increasingly building on machine learning models, we are also digging deeper into the questions of when to retrain the models, what data to use for retraining, and ultimately what data to save for future retraining of the machine learning models used in the management systems.

### 3.14 Is Serverless an Opportunity for Edge Applications?

*Nicola Ferrier (Argonne National Laboratory, US)*

**License**  Creative Commons BY 4.0 International license  
© Nicola Ferrier  
**URL** [www.sagecontinuum.org](http://www.sagecontinuum.org)

Deploying AI at the edge creates an opportunity to develop software defined sensors, using cameras and microphones, along with appropriate software to enable scientists to obtain measurements specific to their application. Some processing methods may require resources that exceed available edge resources. In addition, having multiple scientists seeking to use the same edge device might require off-loading some computations. Serverless architecture for these applications could support a seamless method to have methods run on the edge, cloud, or high-performance computing centers.

### 3.15 HyScale into Serverless: Vision and Challenges

*Hans-Arno Jacobsen (University of Toronto, CA)*

**License** © Creative Commons BY 4.0 International license  
© Hans-Arno Jacobsen

**Joint work of** Yuqiu Zhang, Hans-Arno Jacobsen

**Main reference** Anthony Kwan, Jonathon Wong, Hans-Arno Jacobsen, Vinod Muthusamy: “HyScale: Hybrid and Network Scaling of Dockerized Microservices in Cloud Data Centres”, in Proc. of the 39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019, pp. 80–90, IEEE, 2019.

**URL** <https://doi.org/10.1109/ICDCS.2019.00017>

Microservices, in contrast to traditional monolithic architectures, consist of several smaller dedicated processes working together to provide services to users and are widely adopted in industry due to better flexibility and reliability. Container technologies, such as Docker, provide a lightweight environment for deploying the microservices in computing clusters. Containers are independent units that package software and its dependencies together. Similar to virtual machines (VMs), containers are a virtualization technology that allows a single computing resource to be shared among multiple microservices. However, different from VMs which virtualize resources at the hardware level, containers are virtualized at the operating system level. Containers provide weaker isolation, but are much smaller in size, take much less time to start/stop, and bear lower overhead. This results in improving resource utilization in terms of the number of machines required to host a given workload on the host.

The large-scale adoption of containers for hosting microservices requires the use of container orchestration middleware, such as Kubernetes, to efficiently manage and deploy them. Therefore, an important issue arises, which is to schedule and place containerized applications on available hosts. When submitting an application for deployment, the container orchestration middleware must place it on one of the available resources, considering the limitations of the application and aiming to maximize the use of computing resources. From a cost-efficient perspective, the container orchestration middleware should consider factors such as the capacity of available machines, application performance, quality of service, energy consumption, and operation costs. Typically, a container orchestration middleware provides a unified control interface that is responsible for the whole cluster. The control interface, which we call an autoscaler, runs the autoscaling algorithm to automatically scale up or down the number of allocated resources of containers based on system usage, user requirements, and costs. Our HyScale project is dedicated to solve this issue by building a cost-efficient, SLO-aware autoscaler for container orchestration systems (service level objective-aware).

Going into the serverless era, we envision HyScale to have even more impact and practical use. First, since serverless services are widely adopting containers as the underlying infrastructure, HyScale should be able to seamlessly work with any container-based serverless frameworks to provide elasticity and scalability improvements. Second, the principle of ‘scaling from zero to infinity’ intrinsic to serverless computing and the fact that serverless function executions are mostly short-lived and small in size, finer-grained and faster-reacting autoscaling policies are required to meet the specific needs for this new paradigm. This also puts challenges upon HyScale design to account for the faster autoscaling decision making needs. Moreover, the problem of cold start becomes even more inevitable in the serverless context, where higher requirements of the cooldown period are expected. This needs more meticulous thinking in autoscaler design as it is usually difficult to find a balance between cost efficiency and cooldown period reduction. All in all, autoscaling in serverless is definitely an interesting and promising research area where HyScale can be devoted to in the near future.

### 3.16 Serverless Workflows for Sustainable High-Performance Data Analytics

*Nikolas Herbst (Universität Würzburg, DE)*

**License** © Creative Commons BY 4.0 International license  
© Nikolas Herbst

**Main reference** Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup: “Serverless Applications: Why, When, and How?”, *IEEE Softw.*, Vol. 38(1), pp. 32–39, 2021.

**URL** <https://doi.org/10.1109/MS.2020.3023302>

With the current serverless technologies like FaaS as early enabling technology, we see huge potential for the next generation of serverless computing. Current technical limitations can be overcome: among the current major limitations, we see in accordance with [1] [2] (1) missing ways for direct low-latency communication of functions, (2) efficient state transfer of intermediate results via dis-aggregated memory, (3) in-transparency in terms of real resource consumption and a priori cost estimates, and (4) missing intelligence in placement and scheduling of distributed serverless workflows in the cloud to edge continuum.

We envision a resource-efficient serverless computing platform enabling the specification, management, and automated execution of high-performance data analytic workflows for experts as well as non-experts. A low entry-barrier (NoOps) and flexibility (fine-granular PayPerUse including scale-to-zero) of the envisioned platform could foster interdisciplinary research across research domains based. Besides a sustainable serverless compute infrastructure, we envision a data analytic workflow engine that can leverage serverless technology for ease of assembly, configuration, and efficient operation with a high degree of reusability for distributed data sources. It should support end-to-end data analysis including steps like initial data-quality assessment for a result confidence rating, feature selection, model federation, tuning, method chaining, model-(re-)training, and more.

#### References

- 1 Pedro García López, Aleksander Slominski, Michael Behrendt, Bernard Metzler: Serverless Predictions: 2021-2030. CoRR abs/2104.03075 (2021)
- 2 Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, Chenggang Wu: Serverless Computing: One Step Forward, Two Steps Back. CIDR 2019

### 3.17 Massivizing Computer Systems: Science, Design, and Engineering for Serverless Computing

*Alexandru Iosup (VU University Amsterdam, NL) and Erwin van Eyk (VU University Amsterdam, NL)*

**License** © Creative Commons BY 4.0 International license  
© Alexandru Iosup and Erwin van Eyk

**Main reference** Alexandru Iosup, Alexandru Uta, Laurens Versluis, Georgios Andreadis, Erwin Van Eyk, Tim Hegeman, Satcheendra Talluri, Vincent van Beek, Lucian Toader: “Massivizing Computer Systems: A Vision to Understand, Design, and Engineer Computer Ecosystems Through and Beyond Modern Distributed Systems”, in *Proc. of the 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pp. 1224–1237, IEEE Computer Society, 2018.

**URL** <https://doi.org/10.1109/ICDCS.2018.00122>

The idea of enabling businesses, governments, scientific labs, and society at-large to use IT infrastructure for fine-grained, daily operations, without detailed management of operational logic, emerged in the 1950s and remains a grand challenge in computer science. After a

hiatus between roughly the 1970s through the 2000s, in the 2010s the cloud has picked up the challenge. In the 2020s instance of this challenge, serverless computing, the cloud provider manages the resources, lifecycle, and execution of user-provided functions, all packaged together into fine-grained applications (and fine-grained monitoring, accounting, and billing). But, beyond the low-hanging fruits of this models, and the early emergence of Function-as-a-Service (FaaS), the challenge remains largely untouched.

In this talk, we posit that the principles, challenges, and approach of **massivizing computer systems** [1] could help. Massivizing computer systems is a multi-disciplinary, mixed-methods approach, spanning at least distributed computer systems, performance engineering, and software engineering. We highlight in this talk several points:

1. We do not have to start from scratch to understand why serverless is more (than PaaS cloud) [2]: Using a historiographical approach focusing on technology, we can *explain the origins of serverless computing, and predict (envision) its evolution* focusing on the most important aspects and avoiding the common pitfalls of the past.
2. Building the serverless systems and applications of the future depends – much like the containerization of transport did in its early decades – on *modeling the architecture of serverless operations*. The SPEC RG reference architecture for FaaS [3] is an example of this.
3. Designing new parts and composites is essential for serverless computing, because the current technology raises many technical issues, and the interplay between non-functional properties such as performance, elastic scalability, dependability, security, and sustainability (in particular, energy-efficiency) is complex. The task is daunting and will require many different designers to be able to share and work together, so we need to also design processes, in other words, to *design the design of serverless systems and applications* [4]. A similar argument can be made about *optimization and tuning*.
4. Understanding and analyzing serverless ecosystems is necessary – paraphrasing every scientist and engineer ever, we cannot hope to use what we do not understand, lest it collapses when we least expect it. *Real-world experiments and benchmarking* are important activities here [5]. But real-world experimentation is too costly and time-consuming for large-scale, long-term operations. Instead, *simulation*-based approaches, e.g., based on simulators such as OpenDC [6], are important. Much like a single model cannot capture entirely complex real-life situations, the community should provide multiple simulators (models), and consider *predictions based on ensembles of models*. *Reproducibility* is another important aspect of this line of work [7].
5. Last, but not least, we need a forum to discuss serverless-related topics, especially focusing on the interplay between non-functionals. *The SPEC RG Cloud Group* provides such a forum and is inclusive. Developing its flagship workshop, HotCloudPerf, and merging it with others to form a serverless conference, could provide an annual selective event. Sharing data and software artifacts, FAIRly, would benefit all and be greatly facilitated by such a community/conference. We also envision here a *Memex-like approach to preserve diverse operational traces representative of serverless computing*.

## References

- 1 Alexandru Iosup, Alexandru Uta, Laurens Versluis, Georgios Andreadis, Erwin Van Eyk, Tim Hegeman, Sacheendra Talluri, Vincent van Beek, Lucian Toader: *Massivizing Computer Systems: A Vision to Understand, Design, and Engineer Computer Ecosystems Through and Beyond Modern Distributed Systems*. ICDCS 2018: 1224-1237.

- 2 Erwin Van Eyk, Lucian Toader, Satchendra Talluri, Laurens Versluis, Alexandru Uta, Alexandru Iosup: *Serverless is More: From PaaS to Present Cloud Computing*. IEEE Internet Comput. 22(5): 8-17 (2018)
- 3 Erwin Van Eyk, Alexandru Iosup, Johannes Grohmann, Simon Eismann, André Bauer, Laurens Versluis, Lucian Toader, Norbert Schmitt, Nikolas Herbst, Cristina L. Abad: *The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms*. IEEE Internet Comput. 23(6): 7-18 (2019)
- 4 Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent van Beek, Giulia Frascaria, Ahmed Musaafer, Satchendra Talluri: *The AtLarge Vision on the Design of Distributed Systems and Ecosystems*. ICDCS 2019: 1765-1776
- 5 Erwin Van Eyk, Joel Scheuner, Simon Eismann, Cristina L. Abad, Alexandru Iosup: *Beyond Microbenchmarks: The SPEC-RG Vision for a Comprehensive Serverless Benchmark*. ICPE Companion 2020: 26-31
- 6 Fabian S. Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin van Eyk, Laurens Versluis, Vincent van Beek, Alexandru Iosup (2021) *OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters*. CCGrid 2021
- 7 Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petr Tuma, Alexandru Iosup: *Methodological Principles for Reproducible Performance Evaluation in Cloud Computing*. IEEE TSE.

### 3.18 Machine Learning to enable Autonomous Serverless Systems

*Pooyan Jamshidi (University of South Carolina – Columbia, US)*

License © Creative Commons BY 4.0 International license  
© Pooyan Jamshidi

**Joint work of** Nabor Chagas Mendonça, Pooyan Jamshidi, David Garlan, Claus Pahl  
**Main reference** Nabor Chagas Mendonça, Pooyan Jamshidi, David Garlan, Claus Pahl: “Developing Self-Adaptive Microservice Systems: Challenges and Directions”, IEEE Softw., Vol. 38(2), pp. 70–79, 2021.  
**URL** <https://doi.org/10.1109/MS.2019.2955937>

Serverless computing offers cloud functions, a new type of cloud service that offers fine granularity and lower latency. However, building systems with this new computing platform comes with its challenges: (1) functions are stateless and may need to download large amounts of code/data when they boot up, (2) functions have very limited runtime before they are killed, (3) Storage is limited, but much faster comparing with outside services, (4) the number of available cloud workers depends on the overall load of service providers and the load can only be predicted, (5) node failures occur when running at a large scale, (6) the dependencies differ in functions comparing with an on-premise machine, and (7) latency to the cloud makes roundtrips costly. (8) the cost of acquiring and running a function may vary over time and across providers.

Although researchers have addressed some of these challenges, I am, in particular, interested in developing a vendor-agnostic framework that application developers can build their serverless systems with functionalities such as load balancing between cloud providers and reconfiguring the serverless pipeline to optimize the performance and reliability of the system. The framework can also dynamically map functions to compute nodes based on performance, reliability, and cost trade-off. In addition, automated fault detection and repair will enable resilient and robust serverless application development. We rely on our recent advancement in machine learning, particularly Causal AI (Causal Structure Learning, Causal Inference, Counterfactual Reasoning, Causal Transfer Learning), to enable the proposed capabilities in the serverless framework.

### 3.19 Self-Aware Platform Operations and Resource Management

*Samuel Kounev (Universität Würzburg, DE)*

**License** © Creative Commons BY 4.0 International license  
© Samuel Kounev

**Joint work of** Samuel Kounev, Simon Eismann, Johannes Grohmann, Erwin Van Eyk, Nikolas Herbst

**Main reference** Simon Eismann, Johannes Grohmann, Erwin Van Eyk, Nikolas Herbst, Samuel Kounev: “Predicting the Costs of Serverless Workflows”, in Proc. of the ICPE ’20: ACM/SPEC International Conference on Performance Engineering, Edmonton, AB, Canada, April 20-24, 2020, pp. 265–276, ACM, 2020.

**URL** <https://doi.org/10.1145/3358960.3379133>

In serverless computing, the responsibility for operation aspects, including application resource management, is offloaded to the Cloud provider. This includes, for example, managing virtual machines and containers, managing function execution runtimes (e.g., a Python runtime environment with respective libraries), elastic scaling, reliability/fault tolerance, monitoring, and logging. To manage such aspects, novel mechanisms for automated and proactive resource management are required. We focus on the development of techniques for self-aware platform operations including online learning and reasoning capabilities for efficient and scalable workflow execution.

### 3.20 From design to migration and management: FaaS platforms for application porting to optimized serverless implementation and execution

*Georgios Kousiouris (Harokopion University – Athens, GR)*

**License** © Creative Commons BY 4.0 International license  
© Georgios Kousiouris

**Joint work of** George Kousiouris, Dimosthenis Kyriazis

**Main reference** George Kousiouris, Dimosthenis Kyriazis: “Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay”, in Proc. of the 11th International Conference on Cloud Computing and Services Science, CLOSER 2021, Online Streaming, April 28-30, 2021, pp. 199–206, SCITEPRESS, 2021.

**URL** <https://doi.org/10.5220/0010412101990206>

The availability of decentralized edge computing locations as well as their combination with more centralized Cloud solutions enables the investigation of various trade-offs for application component placement in order to optimize application behaviour and resource usage. Key functionalities and operations needed by a middleware layer so that it can serve as a generalized architectural and computing framework in the implementation of a Cloud/Edge computing continuum are presented. As a primary middleware candidate, FaaS frameworks are taken under consideration, given their significant benefits such as flexibility in execution, event driven nature and enablement of incorporation of arbitrary and legacy application components triggered by diverse actions and rules. Gaps and enablers for three different layers (application design and implementation, semantically enriched runtime adaptation/configuration and deployment optimization) are highlighted. The goal is to enable abstracted application design and porting to the serverless paradigm, based on ready-made, reusable and self-regulating pattern prototypes, semantic annotation of functions in order to dictate deployment or runtime needs (based on goals and constraints), used by the underlying management mechanisms, as well as runtime optimization of the candidate services selection based on performance and configuration trade-offs. The talk highlights the main approach and goals of the H2020 PHYSICS project (<https://physics-faas.eu/>).



### 3.21 Software Development Using Serverless Systems

*Philipp Leitner (Chalmers University of Technology – Göteborg, SE)*

**License** © Creative Commons BY 4.0 International license  
© Philipp Leitner

Function-as-a-Service, and more generally serverless, is a massive area of research interest at the moment. Most of this research deals with how to build, maintain, and scale serverless infrastructure – a problem that few companies outside of a few large cloud providers actually have. However, orders of magnitude more, from small start-ups to billion-dollar industries, face the challenge of how to best make use of this new wave of cloud services to ideally serve their customers. My interest is studying how software engineering research can best support practitioners in this new world.

### 3.22 Running and Scheduling Scientific Workflows on Serverless Clouds: From Functions to Containers

*Maciej Malawski (AGH University of Science & Technology – Krakow, PL)*

**License** © Creative Commons BY 4.0 International license  
© Maciej Malawski

**Main reference** Krzysztof Burkat, Maciej Pawlik, Bartosz Balis, Maciej Malawski, Karan Vahi, Mats Rynge, Rafael Ferreira da Silva, Ewa Deelman: “Serverless Containers – rising viable approach to Scientific Workflows”, CoRR, Vol. abs/2010.11320, 2020.

**URL** <https://arxiv.org/abs/2010.11320>

Scientific workflows are an important class of applications, which consist of computing tasks and data transfers connected into a dependency graph. Traditionally, they are executed on HPC clusters, distributed infrastructures such as grids or clouds. Recent emergence of serverless infrastructures drives us to explore the applicability of these platforms to scientific workflows and associated research problems related to resource management.

Using HyperFlow, our workflow engine developed at AGH, we have evaluated the scientific workflow execution using FaaS (AWS Lambda, Google Cloud Functions) and CaaS platforms (AWS Fargate, Google Cloud Run). We have also performed performance evaluation of serverless cloud infrastructures with a focus on scientific workflows. Based on these experiences, we have recently started addressing scheduling challenges on highly-elastic infrastructures, including cloud functions and containers. Moreover, we have also approached solving simple scheduling problems using D-Wave quantum annealer, achieving quite promising preliminary results for small graphs of tasks fitting entirely in the computer architecture.

Current experience with severless platforms leads to the conclusion that they provide a viable solution for scientific applications, not only scientific workflows but also for large-scale data processing tasks, which come, e.g., from High Energy Physics domain. Serverless infrastructures provide excellent scalability, elasticity and high level of automation of resource management, but as there are many decisions regarding selection of function of container memory and CPU allocation, research on scheduling and performance optimization is still needed.

### 3.23 The case for a hybrid cloud model for serverless computing

*Vinod Muthusamy (IBM TJ Watson Research Center – Yorktown Heights, US)*

License © Creative Commons BY 4.0 International license  
© Vinod Muthusamy

Distributed applications have traditionally been architected to run on a single cloud vendor, using a combination of compute, storage, messaging, load balancing, orchestration, monitoring, authentication, analytics, and numerous other platform capabilities offered by the cloud vendor. Relying on a single vendor's platform has the benefits of tight integration of these capabilities but leads to vendor lock-in, making it difficult for application owners to migrate to another cloud vendor, and challenging for new cloud vendors to compete without building their own portfolio of services.

Hybrid cloud or multi-cloud architectures address the drawbacks of single-vendor cloud platforms, building applications and tooling to allow distributed application components to run on a mixture of private, on-premise, dedicated, and public cloud environments. Application developers have the flexibility to easily migrate their entire applications to another cloud vendor, or make fine-grained deployment decisions based on the performance, cost, regulatory compliance, security policies, and other capabilities of the cloud vendor, matched with the requirements of each application component.

Seen in this light, serverless computing is still in its infancy, with most serverless applications developed for and run on a single serverless platform. There are a class of enterprise applications that aren't amenable to run fully on a public cloud due to regulatory constraints, and the vendor and platform lock-in in today's most popular serverless platforms is holding back these applications from being rearchitected on serverless principles. As well, geo-distributed applications, such as those architected for edge computing platforms, will benefit from taking advantage of a variety of edge vendors; relying on a single vendor to offer edge servers at all desired locations severely constrains the choice of vendors.

A hybrid serverless model brings with it a number of challenges across the stack, including addressing the impedance mismatch when bridging across serverless platforms from multiple providers, including the non-functional properties such as latency, scalability, availability, and cost. For example, it is not clear what is the emergent cold-start behavior when a serverless function running on one platform calls a function on another. There are also functional mismatches, such as security policies, and messaging semantics that need to be reconciled.

As in conventional cloud applications, supporting serverless applications to run across a multi-cloud or hybrid cloud environment will give developers more flexibility, enable a new class of serverless applications held back by vendor lock-in constraints, support truly geo-distributed serverless applications, and offer an opportunity for new serverless platform vendors to compete with novel platform capabilities.

### 3.24 Performance Evaluation in Serverless Computing

*Alessandro Vittorio Papadopoulos (Mälardalen University – Västerås, SE)*

**License** © Creative Commons BY 4.0 International license  
© Alessandro Vittorio Papadopoulos

**Joint work of** Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina Abad, José Nelson Amaral, Petr Tuma, Alexandru Iosup

**Main reference** Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim Von Kistowski, Ahmed Ali-eldin, Cristina Abad, José Nelson Amaral, Petr Tuma, Alexandru Iosup: “Methodological Principles for Reproducible Performance Evaluation in Cloud Computing”, *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

**URL** <https://doi.org/10.1109/TSE.2019.2927908>

In the last few years, obtaining reproducible performance in distributed systems is gaining a lot of attention. This is due to the rapid adoption and diversification of cloud computing technology. The emergence of serverless computing poses additional challenges to such a problem.

Two opposite approaches can be adopted to assess the performance of these kinds of systems. On the one hand, empirical approaches focus on the analysis of the measurable performance of an existing system performing a series of experiments. In empirical studies, sound experimental methodology, and in particular reliable, consistent, and meaningful performance evaluation, is challenging but necessary [2]. On the other hand, theoretical approaches can create reliable models of the system under study, allowing for a deeper understanding of it [1]. Theoretical approaches typically require a design effort and may abstract from certain parts of the system that may be difficult to model.

I am interested in discussing what type of guarantees can be provided on serverless computing applications, and how such guarantees can be obtained through sound performance evaluation.

#### References

- 1 V. Gulisano, A. V. Papadopoulos, Y. Nikolakopoulos, M. Papatriantafidou, and P. Tsigas. Performance modeling of stream joins. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS)*, pages 191–202, New York, NY, USA, Jun. 2017. ACM.
- 2 A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. von Kistowski, A. Ali-Eldin, C. L. Abad, J. N. Amaral, P. Tuma, and A. Iosup. Methodological principles for reproducible performance evaluation in cloud computing. *IEEE Transactions on Software Engineering*, Jul. 2019.

### 3.25 Federated AI on Serverless Edge Clusters Powered by Renewable Energy

*Panos Patros (University of Waikato, NZ)*

**License** © Creative Commons BY 4.0 International license  
© Panos Patros

Artificial Intelligence (AI) applications for agritech, such as robotic harvest and pollination, cannot be implemented without reliable and secure access to computing power. Adding extra hardware on robots increases design complexity, power requirements and weight. Outsourcing to unreliable and off-shore cloud providers increases operational risk and threatens data and economic sovereignty.

The proposed solution is to offer AI services locally via interconnected clusters powered by locally generated renewable energy. Crucially, these Rural AI clusters will maintain a reliable connection with robots, and will leverage advanced federated-learning algorithms and a serverless architecture to store/compute sensitive data locally; thus, only connecting to the cloud for low-risk operations.

A serverless architecture for federated edge learning will provide a seamless transition between edge and cloud computation, while offering a much needed fine-grain allocation (and costing) of scarce edge resources. Because of the limited resources of edge systems, platform innovations will be required to enable these technologies, leveraging prior experience in cloud computing [1, 2, 3, 4, 5, 6].

## References

- 1 P. Patros, D. Dayal, K.B. Kent, M. Dawson, and T. Watson. *Multitenancy benefits in application servers*. Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering, 111-118, 2015
- 2 P. Patros, K.B. Kent, and M. Dawson. *SLO request modeling, reordering and scaling*. Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering, 180-191, 2017
- 3 P. Patros, K.B. Kent, and M. Dawson. *Mitigating garbage collection interference on containerized clouds*. 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 168-173, 2018
- 4 P. Patros, K.B. Kent, and M. Dawson. *Why is garbage collection causing my service level objectives to fail?*. International Journal of Cloud Computing, 7, 3-Apr, 282-322, 2018, Inderscience Publishers (IEL)
- 5 V. Podolskiy, Vladimir; M. Mayo; A. Koay; M. Gerndt; P. Patros. *Maintaining SLOs of cloud-native applications via self-adaptive resource sharing*. 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 72-81, 2019
- 6 V. Podolskiy, M. Patrou, P. Patros, M. Gerndt, and K.B. Kent. *The weakest link: revealing and modeling the architectural patterns of microservice applications*, Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering, 2020, ACM

### 3.26 Is serverless computing the holy grail of fog computing application design paradigms?

Guillaume Pierre (University & IRISA – Rennes, FR)

License © Creative Commons BY 4.0 International license  
© Guillaume Pierre

**Joint work of** Guillaume Pierre, Arif Ahmed, Ali Fahs, Hamidreza Arkian, Paulo Souza junior, Mulugeta Ayalew Tamiru, Mozhdeh Farhadi

**Main reference** Arif Ahmed, HamidReza Arkian, Davaadorj Battulga, Ali J. Fahs, Mozhdeh Farhadi, Dimitrios Giouroukis, Adrien Gougeon, Felipe Oliveira Gutierrez, Guillaume Pierre, Paulo R. Souza Jr., Mulugeta Ayalew Tamiru, Li Wu: “Fog Computing Applications: Taxonomy and Requirements”, CoRR, Vol. abs/1907.11621, 2019.

**URL** <http://arxiv.org/abs/1907.11621>

My research mostly focuses on the design of fog computing platforms. To process massive volumes of data being produced far from the data centers, fog computing extends cloud platforms with additional compute/storage/communication resources in the vicinity of the main sources of data, where these data can be (pre-)processed before reaching the cloud. Although this extension may seem trivial, it brings major new challenges in the way we

design these platforms. In particular, fog computing resources are located close to the main sources of data but necessarily far from each other. This means that it becomes much more difficult to share state between multiple fog nodes taking part in the same application. In this context, serverless computing provides an interesting programming paradigm which neatly separates stateless functions from stateful data services. In the Serverless workshop I tried to better understand the benefits and challenges brought about by this upcoming paradigm shift.

### 3.27 Performance Evaluation of Serverless Applications

Joel Scheuner (*Chalmers and University of Gothenburg, SE*)

**License** © Creative Commons BY 4.0 International license  
© Joel Scheuner

**Joint work of** Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup

**Main reference** Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup: “Serverless Applications: Why, When, and How?”, *IEEE Softw.*, Vol. 38(1), pp. 32–39, 2021.

**URL** <https://doi.org/10.1109/MS.2020.3023302>

Serverless applications typically combine event-triggered functions (i.e., FaaS) with scalable backend services (i.e., BaaS). However, such event-based integrations can lead to long delays that are difficult to debug in a distributed system. Therefore, my research aims to capture and explain application-level serverless performance through detailed tracing and reproducible experimentation.

My prior work consolidates 112 FaaS performance studies [1] and characterizes 89 serverless applications [2] both from academic and industrial sources. In the future, I am interested in performance-aware programming models where developers can indicate their performance-cost trade-off preferences and serverless applications optimize themselves accordingly.

#### References

- 1 J. Scheuner, P. Leitner, Function-as-a-Service Performance Evaluation: A Multivocal Literature Review. In *Journal of Systems and Software (JSS)*, Dec. 2020.
- 2 S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup. Serverless applications: Why, when, and how? *IEEE Software*, vol. 38, pp. 32–39, Jan 2021.

### 3.28 FaaS orchestration

Mina Sedaghat (*Ericsson – Stockholm, SE*)

**License** © Creative Commons BY 4.0 International license  
© Mina Sedaghat

Container orchestrators are often influenced by the application architectural models and their requirements. Modern applications (and their architectures) are getting more complex, often distributed geographically over a continuum of resources, and have stricter performance demands, i.e., on latency and data transfer. The evolution of the application architectures, from monoliths, to microservices and recently to Function as a Service (FaaS), puts new requirements on the orchestration systems and how the container deployment models should look like.

The current FaaS frameworks can only efficiently support a certain class of workloads, such as serving static content, time-based batch jobs, and ETL <sup>6</sup> jobs. They currently have a hard time supporting stateful applications with fine grain state sharing requirements. The basic assumption in the FaaS model is that functions are stateless, and if needed, they store their state using external storage. Therefore, stateful applications are currently constrained by limitations on existing cloud storage services, e.g. due to limited IO throughput and access latencies. I am, personally, interested in simplifying orchestration of Functions in a FaaS framework, providing support for a stateful applications, answering questions around data management, and finding solutions for a seamless orchestration of functions over a continuum of resources.

### 3.29 LaSS: Running Latency Sensitive Serverless Computations at the Edge

*Prashant Shenoy (University of Massachusetts – Amherst, US) and Ahmed Ali-Eldin Hassan (Chalmers University of Technology – Göteborg, SE)*

**License** © Creative Commons BY 4.0 International license  
© Prashant Shenoy and Ahmed Ali-Eldin Hassan

**Joint work of** Bin Wang, Ahmed Ali-Eldin, Prashant Shenoy

**Main reference** Bin Wang, Ahmed Ali-Eldin, Prashant J. Shenoy: “LaSS: Running Latency Sensitive Serverless Computations at the Edge”, in Proc. of the HPDC ’21: The 30th International Symposium on High-Performance Parallel and Distributed Computing, Virtual Event, Sweden, June 21-25, 2021, pp. 239–251, ACM, 2021.

**URL** <https://doi.org/10.1145/3431379.3460646>

Serverless computing has emerged as a new paradigm for running short-lived computations in the cloud. Due to its ability to handle IoT workloads, there has been considerable interest in running serverless functions at the edge. However, the constrained nature of the edge and the latency sensitive nature of workloads result in many challenges for serverless platforms. In this paper, we present LaSS, a platform that uses model-driven approaches for running latency-sensitive serverless computations on edge resources. LaSS uses principled queuing-based methods to determine an appropriate allocation for each hosted function and auto-scales the allocated resources in response to workload dynamics. LaSS uses a fair-share allocation approach to guarantee a minimum of allocated resources to each function in the presence of overload. In addition, it utilizes resource reclamation methods based on container deflation and termination to reassign resources from over-provisioned functions to under-provisioned ones. We implement a prototype of our approach on an OpenWhisk serverless edge cluster and conduct a detailed experimental evaluation. Our results show that LaSS can accurately predict the resources needed for serverless functions in the presence of highly dynamic workloads, and reprovision container capacity within hundreds of milliseconds while maintaining fair share allocation guarantees.

#### References

- 1 Bin Wang, Ahmed Ali-Eldin and Prashant Shenoy *LaSS: Running Latency Sensitive Serverless Computations at the Edge*. Proceedings of ACM Symposium on High Performance Distributed Computing (HPDC) 2021.

<sup>6</sup> Extract, Transform and Load

### 3.30 Fitting Serverless Abstractions and System Designs to Next-Generation Application Needs

*Josef Spillner (ZHAW – Winterthur, CH)*

**License** © Creative Commons BY 4.0 International license

© Josef Spillner

**Main reference** Josef Spillner: “Resource Management for Cloud Functions with Memory Tracing, Profiling and Autotuning”, in Proc. of the WoSC@Middleware 2020: 2020 Sixth International Workshop on Serverless Computing, Virtual Event / Delft, The Netherlands, December 7-11, 2020, pp. 13–18, ACM, 2020.

**URL** <https://doi.org/10.1145/3429880.3430094>

Are today’s serverless systems appropriate for emerging applications such as nation-scale digital services or massive IoT data stream processing? To answer that question, we need to reconsider system designs, programming abstractions and development tools.

On the system level, we investigate more light-weight isolation techniques including zero-coldstart microthreads. Software engineers can leverage these with syntactic constructs they already know in terms of coroutines, asynchronous processing and workers/tasklets. The aim is to reach beyond a few thousand instances per second to tens or hundreds of thousands of invocations, including light-weight state handling like with function-level ring buffers. We also study insights into application execution profiling and subsequent autotuning of memory allocation and other configuration parameters. Such techniques can help to reduce the overallocation of memory from the application engineer’s perspective, to some extent with current statically allocated function instances and to an even greater extent with container isolations permitting dynamic memory updates. This is technically possible even with Docker containers, however the necessary APIs are not exposed by commercial FaaS/CaaS providers.

On the abstraction and tooling level, we explore the use of declarative code annotations to extract functions suitable for offloading computation. The function requirements are then matched as part of a FaaSification process against the cross-provider deployment and execution constraints. Furthermore, we observe static and dynamic characteristics of software artefacts representing serverless software – such as AWS SAM – to convey to software engineers whether there will be any problems or flaws especially when the artefacts originate from third-party dependencies.

### 3.31 Architectural Patterns for Serverless-Based applications

*Davide Taibi (Tampere University, FI)*

**License** © Creative Commons BY 4.0 International license

© Davide Taibi

**Main reference** Davide Taibi, Nabil El Ioini, Claus Pahl, Jan Raphael Schmid Niederkofler: “Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Literature Review”, in Proc. of the 10th International Conference on Cloud Computing and Services Science, CLOSER 2020, Prague, Czech Republic, May 7-9, 2020, pp. 181–192, SCITEPRESS, 2020.

**URL** <https://doi.org/10.5220/0009578501810192>

Companies are increasingly adopting Serverless, by migrating existing applications to this new paradigm. Different practitioners proposed patterns for composing and managing serverless functions. However, some of these patterns offer different solutions to solve the same problem, which makes it hard to select the most suitable solution for each problem.

In this work, we aim at supporting practitioners in understanding the different architectural patterns adopted by different companies, reporting benefits and issues of their applications.

This work proposal was initiated by a previous literature review [1] and is aimed at collecting experiences directly from practitioners by means of interviews and surveys, and to validate the resulting patterns with different collaborative empirical studies.

## References

- 1 Taibi D., El Ioini N., Pahl C., Niederkofler J.R.S. Patterns for serverless functions (function-as-a-service): A multivocal literature review Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20) (2020), 10.5220/0009578501810192

### 3.32 Continuous testing of serverless applications

*André van Hoorn (Universität Stuttgart, DE)*

**License** © Creative Commons BY 4.0 International license  
© André van Hoorn

**Joint work of** André van Hoorn, Thomas F. Düllmann

**Main reference** Giuliano Casale, Matej Artac, Willem-Jan van den Heuvel, André van Hoorn, Pelle Jakovits, Frank Leymann, M. Long, V. Papanikolaou, D. Presentza, A. Russo, Satish Narayana Srirama, Damian A. Tamburri, Michael Wurster, Lulai Zhu: “RADON: rational decomposition and orchestration for serverless computing”, SICS Softw.-Intensive Cyber Phys. Syst., Vol. 35(1), pp. 77–87, 2020.

**URL** <https://doi.org/10.1007/s00450-019-00413-w>

Quality assurance is a key software engineering activity to deliver high-quality software. The way software is being developed has changed dramatically over the past years, due to emerging cloud-native architectural styles, such as microservices and serverless, in combination with modern software engineering paradigms such as DevOps. The frequency and velocity of changes impose challenges to quality assurance, particularly for assessing runtime quality attributes such as performance and resilience. On the other hand, the new developments provide opportunities for novel quality assurance approaches, e.g., due to established technologies, a high degree of automation, and operational feedback from production. We investigate the interplay of the mentioned topics in the DevOps Performance Working Group of the SPEC RG. Concerning the seminar topic, my particular interest is in the question of “How to seamlessly integrate quality-of-service assurance for serverless into the DevOps ecosystem?”.

Over the last year, I was involved in the EU Horizon 2020 project RADON on “Rational decomposition and orchestration for serverless computing”. RADON provides an end-to-end framework to develop serverless applications, building on the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA). To assess whether applications developed via the RADON methodology and framework meet their quality requirements, RADON includes the continuous testing workflow, which particularly aims to support software developers, QoS engineers, and release managers in producing high-quality applications. The core component implementing the continuous testing workflow is the Continuous Testing Tool (CTT). CTT enriches the TOSCA ecosystem by end-to-end support for continuous testing of microservice-based (including FaaS) and data pipeline applications in DevOps. CTT supports the whole workflow – from test specification over execution and reporting to automated updates based on production data – that is also extensible to custom needs, e.g., integrating other types of tests or tools. A particular innovation lies in the integrative test generation features for obtaining tailored tests, which fits into the constraints of DevOps-based development settings with separate teams and delivery pipelines, and the goal of fast and frequent releases.



### 3.33 Serverless Compute Primitives as a Compilation Target

Soam Vasani (*Stripe – San Francisco, US*)

License  Creative Commons BY 4.0 International license  
© Soam Vasani

FaaS is a compelling compute primitive: it has the best elasticity that cloud compute has so far offered, and it abstracts away more infrastructure than any other compute primitive has so far. However application developers must account for FaaS limitations on timing, networking, artifact size, etc; these limitations have fundamental effects on application architectures.

This raises the question: can we have the elasticity and abstraction of FaaS without having to learn new application architecture patterns? To this end I'm interested in borrowing ideas from compilers: can we use FaaS and other serverless technologies (such as object stores and workflow runtimes) as a compilation target? In other words, can we transform a source program that is not serverless-specific to a set of functions, objects, and workflows? If this is not universally possible, then is there a set of source programs for which this is both possible as well as useful?

As a prototype, I'm exploring this question for the specific technologies of Python and AWS serverless, transforming functions written in a subset of Python into a set of Lambdas, Step Functions and S3 buckets.

### 3.34 Network Challenges in Serverless Computing

Florian Wamser (*Universität Würzburg, DE*)

License  Creative Commons BY 4.0 International license  
© Florian Wamser

**Joint work of** Nguyen Huu Thanh, Nguyen Trung Kien, Ngo Van Hoa, Truong Thu Huong, Florian Wamser, Tobias Hofffeld

**Main reference** Nguyen Huu Thanh, Nguyen Trung Kien, Ngo Van Hoa, Truong Thu Huong, Florian Wamser, Tobias Hofffeld: "Energy-Aware Service Function Chain Embedding in Edge-Cloud Environments for IoT Applications", *IEEE Internet of Things Journal*, pp. 1–1, 2021.

**URL** <https://doi.org/10.1109/JIOT.2021.3064986>

The serverless computing paradigm promises a number of advantages over conventional cloud- or server-centered computing. Serverless computing offers the developer greater scalability and more flexibility at a lower cost. From the developer's point of view, one does not have to worry about the dimensioning, provision and administration of backend servers and hosts.


To provide flexibility, scalability, and developer-friendliness, a serverless platform typically manages and maintains the underlying resources. In addition to the computing resources, the *network* also plays a decisive role here, connecting the computing resources and transporting application requests to the serverless functions.

At the *University of Würzburg* we investigate the challenges for networks in connection with serverless computing. The most important points are:

1. Elasticity and scalability of network resources
2. Dynamic addressing and forwarding of requests to computing resources
3. Provision of network resources for functionality and adaptability

### 3.35 Decision Support for Modeling and Deployment Automation of Serverless Applications

Vladimir Yussupov (*Universität Stuttgart, DE*)

License  Creative Commons BY 4.0 International license  
© Vladimir Yussupov

The term “serverless” gains more and more attention in the context of cloud-native application development. Frequently being associated exclusively with the Function-as-a-Service (FaaS) cloud service model, the idea of what a serverless application is keeps evolving, resulting in more issues to decide on when engineering serverless applications. I am interested in the topic of decision support for modeling and deployment of serverless architectures comprising various kinds of components such as FaaS platforms, function orchestrators, serverless databases and message queues. In particular, I am investigating which decisions need to be considered (also, decisions captured in the form of patterns), and how to use them to support practitioners in transitioning from abstract serverless application models to refined, provider-specific deployment models that can be enacted using deployment automation technologies of choice.

Some related publications:

- Yussupov, V.; Soldani, J.; Breitenbücher, U.; Brogi, A. and Leymann, F. (2021). From Serverful to Serverless: A Spectrum of Patterns for Hosting Application Components. In Proceedings of the 11th International Conference on Cloud Computing and Services Science – CLOSER
- Yussupov, V.; Soldani, J.; Breitenbücher, U.; Brogi, A.; Leymann, F. (2021). FaaSten your decisions: A classification framework and technology review of function-as-a-Service platforms, In Journal of Systems and Software, Volume 175

## 4 Working groups

### 4.1 Design of Serverless Systems, Platforms, and Ecosystems (Topic 1)

*Samer Al-Kiswany (University of Waterloo, CA), Ahmed Ali-Eldin Hassan (Chalmers University of Technology – Göteborg, SE), André Bauer (Universität Würzburg, DE), André B. Bondi (Software Performance and Scalability Consulting LL, US), Ryan L. Chard (Argonne National Laboratory – Lemont, US), Andrew A. Chien (University of Chicago, US), A. Jesse Jiryu Davis (MongoDB – New York, US), Erik Elmroth (University of Umeå, SE), Alexandru Iosup (VU University Amsterdam, NL), Hans-Arno Jacobsen (University of Toronto, CA), Samuel Kounev (Universität Würzburg, DE), Vinod Muthusamy (IBM TJ Watson Research Center – Yorktown Heights, US), Guillaume Pierre (University & IRISA – Rennes, FR), Mina Sedaghat (Ericsson – Stockholm, SE), Prashant Shenoy (University of Massachusetts – Amherst, US), Davide Taibi (Tampere University, FI), Douglas Thain (University of Notre Dame, US), Erwin van Eyk (VU University Amsterdam, NL), and Soam Vasani (Stripe – San Francisco, US)*

License © Creative Commons BY 4.0 International license

© Samer Al-Kiswany, Ahmed Ali-Eldin Hassan, André Bauer, André B. Bondi, Ryan L. Chard, Andrew A. Chien, A. Jesse Jiryu Davis, Erik Elmroth, Alexandru Iosup, Hans-Arno Jacobsen, Samuel Kounev, Vinod Muthusamy, Guillaume Pierre, Mina Sedaghat, Prashant Shenoy, Davide Taibi, Douglas Thain, Erwin van Eyk, and Soam Vasani

### Organization

- Co-chairs: Alexandru Iosup and Samer Al-Kiswany
- Rapporteurs: Mina Sedaghat and Doug Thain

### Opening Statement

This topic focuses on the design of serverless systems, platforms, and ecosystems. We organized the discussion for this topic around sessions, aiming to first obtain a diverse set of sub-topics, then to refine our own views about a focused set of sub-topics, then to take in new perspectives and share our own in discussion with the groups working on other topics, and, last, to refine our views toward a vision. Our sessions proceeded as follows:

- First session: We focused on scoping, discussing possible sub-topics for the design topic and trying to have as many ideas represented as possible.
- Second session: We focused on choosing and discussing 3 sub-topics, and on finalizing the scoping effort with ideas that arrived from the cross-pollination with other topics. We discussed requirements, including geo-distributed operation, the impact of state and data streams, predictable performance, energy awareness, security, and auditability through provenance provisions; we further discussed the actual complexity when automating the operational concerns for the user. We further discussed operational techniques for workload and resource management at runtime, and ensuring SLAs and SLOs while still being able to “make it easy for the user”. We discussed lessons learned from cloud computing, especially from PaaS (e.g., “We wanted all applications to be equally supported, with one simple and unified interface, but every significant application has at least something different”).

- Third session: We analyzed existing serverless definitions, benefiting from discussion with other topics, and concluded on the key aspects of a good serverless definition. This allowed us to focus on a programming model, a reference architecture, non-functional requirements, patterns and anti-patterns in serverless applications in practice, and on toolchains. We also discussed programmability, portability, and interoperation.
- Fourth session: We had joint discussions with Topics 2 (software engineering) and 3 (application requirements). Main aspects discussed: What are the application domains and domain verticals? How to think about the user? What applications are the most important? What is the lifecycle of a serverless application? What sort of application architecture should be adopted? What is a good definition for serverless? How to think about managerial, policy, cloud, and resource-level metrics?
- Fifth and sixth sessions: concluding on the core definition, vision, challenges, etc.

### Link to other topics

There is a strong link to the other topics, both conceptually and, following the joint sessions, practically:

- Questions related to representative use cases and applications, and to requirements related to them, are essential for the design topic and link strongly to Topic 3.
- Questions related to implementing and realizing the software of serverless systems, platforms, and ecosystems, including both the software patterns and the engineering process, are linked strongly with Topic 2.
- Questions related to testing and evaluating serverless systems, platforms, and ecosystems are linked strongly to topic 4.

### A reference architecture for serverless computing

We discussed and agreed on a reference architecture for serverless computing. The reference architecture considers the following main layers:

1. Compute, memory, storage, and networking infrastructure, consisting primarily of (programmable) hardware devices and of corresponding virtualized devices.
2. Operating services, providing foundational services such as messaging, coordination, and authentication.
3. Resource managers, providing collections of (distributed) resources, physical and/or virtual, with pre-configured operating services, under a convenient programming interface.
4. Runtime engines, providing capabilities for executing simple and composite functions, up to orchestrating entire dataflows and workflows, and automating the back-end management of transient state and persistent data.
5. Front-end core, providing a programming model for serverless applications, specializations of this programming model for specific application domains, high-level programming languages for convenient programming, and portal and command-line high-level interfaces.
6. Across all layers, a toolchain of compilers, monitors, profilers, and benchmarks for serverless computing, helping optimize each aspect and making all levels observable.

Our main insight from the reference architecture is that the automated operation for serverless applications is an ecosystem, with many different parts developed and operated by different and autonomous organizations; this is very different from a single integrated system and leads to different design challenges and practices.

Another insight is that the designs of all the systems at different layers are highly influenced by the programming model, but currently no single programming model exists for serverless computing, and it is likely such a programming model will only be possible if it is very abstract and generic. We expect high levels of specialization and that serverless applications will use a special runtime stack as well as will often rely on back end services offered by the cloud providers.

We acknowledge that the serverless computing paradigm is in its infancy and many of the layers, especially the runtime engines, front-end core, and toolchains, will provide many radically different alternatives that will take time to mature and perhaps not converge.

### **Vision on the design of serverless systems, platforms, and ecosystems**

We envision that serverless systems, platforms, and ecosystems should aim to:

- Ensure (nearly) complete automation of operational concerns and
- high programmability, portability, and interoperability, for
- diverse application domains and use cases, where
- many parts are defined once but used many times, with
- on-demand deployment, (geo-)distributed operation, and utilization-proportional cost, by
- offering diverse operational techniques (Rethinking resource management and scheduling),
- supporting and enforcing non-functional requirements, controlling for variability,
- rethinking observability and providing serverless-related monitoring,
- rethinking the static and dynamic toolchain, and
- ensuring integration with a diverse, evolving technology ecosystem.

Next to many challenges, which we list in the following, there are also *uncomfortable questions*, such as:

1. The question is not can we make remote execution as easy as local execution, but can we make it easier and more beneficial?
2. Industry is ahead and facing immediate challenges, so how should academics engage so they have impact in this field?
3. What is new, over the problems of full automation of 70 years ago (utility computing), 20 years ago (grid), 10 years ago (cloud)? (Serverless is not the entire cloud and should not try to do everything.)

### **Challenges in the design of serverless systems, platforms, and ecosystems**

Related to the reference architecture:

**C1 Capturing the multi-level architectural features and emerging architectural patterns of this rapidly evolving serverless computing field**

**C2 Predicting which architectural features and patterns will succeed, and explaining why (and why not others)**

Related to full automation of operational concerns:

**C3 Agreeing on a serverless definition and making it operational**

Can we make it easy to run applications remotely? Can we achieve full transparency, in Coulouris' sense, a sort of “cloud button”? Can we achieve near-zero waste, scale to zero (cost)? (See also Section 5.1.)

**C4 Understanding system-level, operational requirements**

This includes understanding the stakeholders of serverless operations, the users of specific applications, and the systems-level requirements raised by industry verticals, application domains, and applications. Focus on both functional and non-functional requirements, and for non-functionals consider metrics at different levels of interest, from hardware resources to organization-wide managerial decisions. Focus on energy efficiency, but also on sustainability awareness (e.g., how the electricity used for serverless workloads is produced and consumed, how much greenhouse gas emissions and water consumption occurs here).

**C5 Programming model from a systems perspective**

What sort of application architecture should be adopted? What is the right granularity of the function? How to trade-off between providing control and simplicity? How to express and manage workflows (and is there a new way needed, or are workflow abstractions already sufficient)? What can we learn from decades of programming parallel and distributed systems?

**C6 Workload and resource management for serverless, and overall routing and scheduling**

How to extend and apply traditional techniques for workload and resource management? How to consider the full compute continuum (i.e., IoT/fog/edge/cloud)? How to replicate, cache, partition, consolidate, migrate, offload, etc. the functions and/or the data? How to provision, allocate, elastically scale, load-balance the resources? How to schedule and route across the whole (eco)system? How to consider resource provisioning over short periods of time (e.g., auto-scaling) and also long-term (e.g., capacity planning)?

**C7 Practical needs in serverless orchestration**

How to mix serverless with other computational models, i.e., run mixtures of workloads instead of merely serverless? How to achieve near-zero waste, even under complex deployment scenarios (e.g., geo-distributed scenarios)? How to reduce the serverless overhead added by the platform to the (commonly lightweight) functions representing the business logic of the application? How to ensure proper performance isolation while making efficient use of the shared infrastructure?

**C8 Manage ecosystem instability**

How to limit the impact of, e.g., performance variability, the impact of (correlated, even cascading) component downtime, multiple versions of the same service, service continuity under transience of various providers?

Related to the toolchain:

**C9 Create the serverless toolchain**

How does the traditional toolchain – Compiler, Linker, Loader Static Analysis, Dynamic Analysis, Dependency Detection, Testing, Debugging, Mocking, Tracing, Replaying – need to change for serverless, starting with FaaS? How to use self-descriptive metadata to improve (i) *safety* (type signature, semantics, version, dependencies, non-functional requirements, etc.), and (ii) *efficiency* (performance, resource requirements, co-location with other functions, etc.)? Process-wise, how to engage both toolchain and application developers, to motivate incremental deployment and interoperation of both metadata and tools, while accepting incomplete information?

**C10 Support for patterns and anti-patterns, both functional and non-functional**

What are the serverless patterns and anti-patterns, both functional and non-functional, that systems designers can work with? For example, what are the performance patterns and anti-patterns for serverless operations? How to support enterprise patterns, e.g., for integration or for distributed operation? How to support specific industry verticals or application domains, e.g., matching Topic 3: for scientific computing, for machine learning and artificial intelligence, for online gaming, and for mobile and telco operations?

**Next steps and takeaway for the community**

We have discussed the topic of design for serverless computing systems, platforms, and ecosystems. Linking to the other topics in this Dagstuhl Seminar, we have considered a definition for serverless computing, requirements from various application domains and use cases, software engineering concepts and processes, etc. We have provided in this section a summary of several critical aspects for design, including a reference architecture, a vision, and several uncomfortable questions and challenges.

The main takeaway for the community is that serverless computing poses hard, even grand challenges, related to full automation of operational concerns under hard constraints. The design of serverless systems, platforms, and ecosystems is an essential part of achieving the promise of serverless computing. The challenges we have listed here shape the task ahead, but there is more on the horizon.

As indicated by the value of the discussion we had with other topics in this seminar, designers should make sure the collaboration between computer systems, software engineering, performance engineering, and beyond to cross-disciplinary collaborations.

## 4.2 Software Engineering of Serverless Applications, but also Systems, Platforms, and Ecosystems (Topic 2)

*Simon Eismann (Universität Würzburg, DE), Robert Chatley (Imperial College London, GB), Nikolas Herbst (Universität Würzburg, DE), Georgios Kousiouris (Harokopion University – Athens, GR), Philipp Leitner (Chalmers University of Technology – Göteborg, SE), Pedro García López (Universitat Rovira i Virgili – Tarragona, ES), Bernard Metzler (IBM Research-Zurich, CH), Davide Taibi (University of Tampere, FI), Vincent van Beek (Solwinity, Amsterdam and Delft University of Technology, NL), André van Hoorn (Universität Stuttgart, DE), Guido Wirtz (Universität Bamberg, DE), and Vladimir Yussupov (Universität Stuttgart, DE)*

**License** © Creative Commons BY 4.0 International license  
© Simon Eismann, Robert Chatley, Nikolas Herbst, Georgios Kousiouris, Philipp Leitner, Pedro García López, Bernard Metzler, Davide Taibi, Vincent van Beek, André van Hoorn, Guido Wirtz, and Vladimir Yussupov

### Opening Statement

The group discussed the topic of serverless from the perspective of software engineers in DevOps teams that are responsible for the development and operation of software systems running on serverless cloud platforms. The group decided to use the established stages of the software life-cycle to guide the discussion. For each stage, the group discussed how this stage is different from traditional software development when building serverless applications and what the resulting challenges are.

### Changes to the Software Engineering Lifecycle

This section highlights the major differences in the engineering process when building serverless applications, compared to engineering traditional software architectures. Based on these differences, the group collected a number of software engineering challenges for serverless applications, which are discussed in the next section.

### Planning

During the planning phase, the requirements of the application need to be collected and based on them the fundamental decisions about the application are made. For serverless applications, two additional decisions need to be made during the planning phase. The first one is whether serverless is actually well suited for this task. As there are still several limitations to serverless, it is not a suitable solution for every application, yet. The second decision that needs to be made is the selection of a cloud provider. While this was also a decision for traditional cloud applications, its impact is far larger for serverless applications. The IaaS and container offerings of most cloud providers offer very similar features. However there are significant differences in the serverless offerings of different providers. While they all offer a function-as-a-service solution, there are large differences when it comes to the managed services. As cloud providers work to increase the number of specialized managed services, these differences will increase further.



### Design

The key objective of the design phase is to come up with a suitable software architecture for the planned application. In this phase, the serverless application is split into coarse-grained, individual units (called microservices, components, or service). Within such a serverless microservice, there is a second, explicit architecture layer that describes the separation of code into serverless functions, incorporated external services, and the triggers that define the control flow within the application. This architecture within a service also implicitly exists within a traditional application in the form of software classes. However serverless makes this architecture explicit and forces developers to think of this low-level architecture before the implementation. This change increases the awareness of developers for the architecture of their application, and makes architecture diagrams for this second architecture level commonplace (in contrast to the often neglected UML diagrams).

### Implementation

In the implementation phase, developers start implementing according to the requirements and the software architecture discussed in previous phases. A serverless application contains significantly less code than a traditional application, as much of the control flow and business logic is handled by managed services. However, these managed services and triggers need to be configured in the form of infrastructure-as-code (IaC) files. Therefore, developers spend a lot of time working on IaC files when building serverless applications. This means developers frequently need to context-switch between the actual code and the IaC file, as the functionality of the application is spread across both. While the tooling around code development is mature, the tooling around IaC and the integration of IaC and code is quite immature. This currently makes the development of serverless applications quite cumbersome.

### Testing

The testing phase for serverless architectures must cover both functional and non-functional aspects. Unit tests can be implemented with relative ease due to the smaller granularity of functions and tested as usual for the chosen programming language. Integration tests become more important for serverless applications as the majority of behavior to test is located outside of the functions. However, integration testing also becomes more difficult as serverless applications rely on integration of multiple fine-grained components hosted using provider-managed services. Integration tests can be executed in a local environment using service emulators and available tooling, testing remotely on the provider's side, or a combination of both options. In practice, the hybrid testing option is currently quite common, since the local testing is limited w.r.t. available tooling and is not representative enough, whereas only remote testing incurs additional costs and takes longer as applications need to be redeployed with each update.

### Deployment

During the deployment phase, all components and configurations of their interactions must be deployed to a target environment, meaning that not only the packaged source code has to be deployed, but also required event bindings need to be created, security policies configured, etc. As a result, a large part of the deployment requirements is at least partially addressed during the design and implementation phases: required component bindings are established either

in the source code or configuration files of the chosen deployment automation technology. The choice of the underlying deployment technology also defines which components can be deployed by it and in certain cases a combination of several technologies needs to be used, e.g., infrastructure deployment and configuration management using different automation tools.

## Identified Challenges

Based on the changes to the software engineering lifecycle, the group identified a number of software engineering challenges for serverless applications. The table below shows how the identified challenges map to the software engineering lifecycle phases.

■ **Table 1** Identified Challenges in the Lifecycle.

Lifecycle Phase	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Planning	X			X					X	X
Design				X	X	X	X		X	
Implementation			X			X	X	X		
Testing		X	X							
Deployment										

### C1 Identifying whether a use case is serverless-ready

Serverless is quickly evolving and more and more use cases are becoming suitable. However, there are still limitations to serverless, which means it is not a suitable solution for every application, yet. There are currently no guidelines on how to identify whether serverless is suitable for a specific use case, which hinders the adoption of serverless.

### C2 Testing serverless applications

Integration testing of serverless applications includes testing the configuration of managed services and function triggers. Local emulators for managed services and function integrations are difficult to build and maintain as serverless platforms are quickly evolving. Running integration tests directly on the cloud platform requires time-intensive deployments, which slows down the feedback cycle for developers.

### C3 Debugging serverless applications

Triaging the cause of bugs in serverless applications is currently quite difficult. As a feature is often implemented by multiple functions and managed services, understanding what happened for a single request requires the logs from the multiple functions and services involved in the processing of the request. The non-standardized logging schemas of managed services and immature observability tooling makes this quite cumbersome.

#### **C4 Predicting the costs of serverless applications**

On the surface, the serverless billing model – pay-per-use – seems predictable. However, estimating the cost per request for a serverless API requires developers to understand the pricing models of all involved services. This is further complicated by the fact that many of the costs are dependent on data volumes and execution times which are often challenging to estimate.

#### **C5 Determining function size**

With serverless computing, applications are broken down into many serverless functions that are connected via managed services and event triggers. Developers often need to make the decision if a function is too large and should be split into two or more functions. There are currently few, and often conflicting guidelines on how to determine the appropriate size of a serverless function.

#### **C6 Managing state in serverless functions**

In their current state, serverless functions are stateless, which means that applications that require state can not be built purely from serverless functions. Instead, stateful information is currently stored in managed services such as databases or messaging services. Enabling functions to have some fast, shared state would not eliminate the need for databases or messaging services, but simplify the development of serverless applications and enable new use cases.

#### **C7 Finding suitable abstract languages/models**

Serverless applications are currently designed as architecture diagrams and implemented in the form of code and infrastructure as code definitions. Serverless could benefit from an intermediate language or model that could bridge the gap between the very coarse-grained, non-standardized architecture diagrams and the hard to understand combination of code and infrastructure as code.

#### **C8 Reusing serverless functions**

Serverless applications are broken down into small parts (functions) which can enable the reuse of existing functions in new contexts. However, managing functions within an organization at scale is currently challenging. Open questions here include how to determine what requirements and assumptions an existing function makes and whether reusing a function should include a separate deployment or the routing of requests to the existing function deployment.

#### **C9 Migrating existing applications to serverless**

Many existing applications could benefit from a partial or full migration towards serverless. However, many of these applications are not migrated, as developers are unaware of how to structure the migration, which parts to migrate first, and how to manage a serverless/serverful hybrid application. Additionally, this poses the challenge of how to train developers that are used to the serverful model in the skill required to build serverless applications.

### 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)*

**License** © Creative Commons BY 4.0 International license  
© Josef Spillner, Bartosz Balis, Jesse Donkervliet, Nicola Ferrier, Ian T. Foster, Maciej Malawski, Panos Patros, Omer F. Rana, and Florian Wamser

### 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 ( $\lambda$ , 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.<sup>7</sup> 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;

<sup>7</sup> <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.<sup>8</sup> 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.<sup>9</sup> The popularity of games is difficult to

---

<sup>8</sup> <https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990>

<sup>9</sup> <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.,<sup>10</sup> 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

---

<sup>10</sup> <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,  $\mu$ -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 Shahradsad (University of British Columbia – Vancouver, CA), and Alexandru Uta (Leiden University, NL)*

**License**  Creative Commons BY 4.0 International license

© Cristina Abad, Kyle Chard, Pooyan Jamshidi, Alessandro Vittorio Papadopoulos, Robert P. Ricci, Joel Scheuner, Mohammad Shahradsad, and Alexandru Uta

#### 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.

**Application-level benchmarks:**

Application benchmarks focus on explaining the performance behavior of a known application under a serverless system. These benchmarks select representative applications motivated by real-world use cases and test them under realistic workloads. An example evaluation of a serverless application is presented in the ExCamera paper [7].

**Middleware/frameworks:**

Researchers and developers need a way to evaluate the performance of middlewares or frameworks, layers that are built on top of Function-as-a-Service platforms but are not user-focused (e.g., a workflow manager). The goal being, frequently, to preserve performance and reliability, while decreasing cost. An example of this type of evaluation can be seen in [8].

**Workload characterization:**

Like other cloud services, serverless offerings host a wide range of users running various applications. Characterizing the serverless workload enables discovering usage patterns, modeling resource consumption, and understanding the composition of serverless applications. For example, Microsoft's characterization study [9] came with open-source traces on invocation times alongside function duration and memory usage distributions for each application.

**Systems design / development / solution evaluations:**

Frequently, changes are made to the inner workings of serverless platforms like OpenWhisk or OpenFaaS. These studies seek to improve parts of the stack, like the scheduler (maybe while stubbing/simulating other parts); for example, as in [10]. Such inner working can be at different layers in the computer system stack including software, computer architecture, and hardware.

**Other:**

In the other category, we include any study that does not fit in the prior five categories; for example, software engineering papers decomposing monoliths, or papers that use serverless to test/validate something else. The former includes case studies showing how (typically monolithic) applications can be re-architected to work with a serverless design. For a concrete study that illustrates this category, consider [11].

**Performance evaluation approaches**

In addition to the classification of performance evaluation of serverless systems according to their goal, the group also discusses the different performance evaluation approaches that researchers can take. **Empirical** evaluations employ an observation-based approach in which the system is deployed in a testbed, a workload issued, and results observed and analyzed. **Theoretical** evaluations start with a model and try to reason about a system through analysis or simulations: What are the inputs and how do these inputs affect the performance? Theoretical approaches are particularly useful for predictions, and real-time decisions (e.g., scheduling, offloading). **Hybrid** approaches that combine the two prior approaches can be used for sensitivity analysis, to identify the top  $x$  parameters to configure or tune.

## Next steps and takeaway for the community

To enable better evaluations, industry should release traces that can be analyzed, modeled, and replayed. The community would also benefit significantly if cloud providers were to publish how their serverless systems work internally. We need platform and application benchmarks, and we need these to be based on a solid understanding of how actual applications use serverless frameworks. Less explored and equally important are analytical frameworks that can be used to explore pricing policies that can be beneficial for providers and consumers.

## References

- 1 X. Chen, S. Dallmeier-Tiessen, R. Dasler, S. Feger, P. Fokianos, J. B. Gonzalez, H. Hirvonsalo, D. Kousidis, A. Lavasa, S. Mele *et al.*, “Open is not enough,” *Nature Physics*, vol. 15, no. 2, pp. 113–119, 2019.
- 2 B. Haibe-Kains, G. A. Adam, A. Hosny, F. Khodakarami, L. Waldron, B. Wang, C. McIntosh, A. Goldenberg, A. Kundaje, C. S. Greene *et al.*, “Transparency and reproducibility in artificial intelligence,” *Nature*, vol. 586, no. 7829, pp. E14–E16, 2020.
- 3 E. van der Kouwe, G. Heiser, D. Andriess, H. Bos, and C. Giuffrida, “Sok: Benchmarking flaws in systems security,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 310–325.
- 4 A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-Eldin, C. Abad, J. N. Amaral, P. Tuma, and A. Iosup, “Methodological principles for reproducible performance evaluation in cloud computing,” *IEEE Transactions on Software Engineering*, 2019.
- 5 A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeier, C. Maltzahn, R. Ricci, and A. Iosup, “Is big data performance reproducible in modern cloud networks?” in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 513–527.
- 6 L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, “Peeking behind the curtains of serverless platforms,” in *USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 133–146.
- 7 S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, “Encoding, fast and slow: Low-latency video processing using thousands of tiny threads,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 363–376.
- 8 S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, “From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers,” in *USENIX Annual Technical Conference (USENIX ATC)*, 2019, pp. 475–488.
- 9 M. Shahradi, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in *USENIX Annual Technical Conference (USENIX ATC 2020)*, 2020, pp. 205–218.
- 10 E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “SOCK: Rapid task provisioning with serverless-optimized containers,” in *USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 57–70.
- 11 A. Goli, O. Hajihassani, H. Khazaei, O. Ardakanian, M. Rashidi, and T. Dauphinee, “Migrating from monolithic to serverless: A fintech case study,” in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2020, pp. 20–25.

## 5 Panel discussions

### 5.1 Toward a Definition for Serverless Computing

*Samuel Kounev (Universität Würzburg, DE), Cristina Abad (ESPOL – Guayaquil, EC), Ian T. Foster (Argonne National Laboratory – Lemont, US), Nikolas Herbst (Universität Würzburg, DE), Alexandru Iosup (VU University Amsterdam, NL), Samer Al-Kiswany (University of Waterloo, CA), Ahmed Ali-Eldin Hassan (Chalmers University of Technology – Göteborg, SE), Bartosz Balis (AGH University of Science & Technology – Krakow, PL), André Bauer (Universität Würzburg, DE), André B. Bondi (Software Performance and Scalability Consulting LL, US), Kyle Chard (University of Chicago, US), Ryan L. Chard (Argonne National Laboratory – Lemont, US), Robert Chatley (Imperial College London, GB), Andrew A. Chien (University of Chicago, US), A. Jesse Jiryu Davis (MongoDB – New York, US), Jesse Donkerkervliet (VU University Amsterdam, NL), Simon Eismann (Universität Würzburg, DE), Erik Elmroth (University of Umeå, SE), Nicola Ferrier (Argonne National Laboratory, US), Hans-Arno Jacobsen (University of Toronto, CA), Pooyan Jamshidi (University of South Carolina – Columbia, US), Georgios Kousiouris (Harokopion University – Athens, GR), Philipp Leitner (Chalmers University of Technology – Göteborg, SE), Pedro García López (Universitat Rovira i Virgili – Tarragona, ES), Martina Maggio (Universität des Saarlandes – Saarbrücken, DE), Maciej Malawski (AGH University of Science & Technology – Krakow, PL), Bernard Metzler (IBM Research-Zurich, CH), Vinod Muthusamy (IBM TJ Watson Research Center – Yorktown Heights, US), Alessandro Vittorio Papadopoulos (Mälardalen University – Västerås, SE), Panos Patros (University of Waikato, NZ), Guillaume Pierre (University & IRISA – Rennes, FR), Omer F. Rana (Cardiff University, GB), Robert P. Ricci (University of Utah – Salt Lake City, US), Joel Scheuner (Chalmers and University of Gothenburg, SE), Mina Sedaghat (Ericsson – Stockholm, SE), Mohammad Shahradsad (University of British Columbia – Vancouver, CA), Prashant Shenoy (University of Massachusetts – Amherst, US), Josef Spillner (ZHAW – Winterthur, CH), Davide Taibi (Tampere University, FI), Douglas Thain (University of Notre Dame, US), Animesh Trivedi (VU University Amsterdam, NL), Alexandru Uta (Leiden University, NL), Vincent van Beek (Solwinity, Amsterdam and Delft University of Technology, NL), Erwin van Eyk (VU University Amsterdam, NL), André van Hoorn (Universität Stuttgart, DE), Soam Vasani (Stripe – San Francisco, US), Florian Wamser (Universität Würzburg, DE), Guido Wirtz (Universität Bamberg, DE), and Vladimir Yussupov (Universität Stuttgart, DE)*

**License** © Creative Commons BY 4.0 International license

© Samuel Kounev, Cristina Abad, Ian T. Foster, Nikolas Herbst, Alexandru Iosup, Samer Al-Kiswany, Ahmed Ali-Eldin Hassan, Bartosz Balis, André Bauer, André B. Bondi, Kyle Chard, Ryan L. Chard, Robert Chatley, Andrew A. Chien, A. Jesse Jiryu Davis, Jesse Donkerkervliet, Simon Eismann, Erik Elmroth, Nicola Ferrier, Hans-Arno Jacobsen, Pooyan Jamshidi, Georgios Kousiouris, Philipp Leitner, Pedro García López, Martina Maggio, Maciej Malawski, Bernard Metzler, Vinod Muthusamy, Alessandro Vittorio Papadopoulos, Panos Patros, Guillaume Pierre, Omer F. Rana, Robert P. Ricci, Joel Scheuner, Mina Sedaghat, Mohammad Shahradsad, Prashant Shenoy, Josef Spillner, Davide Taibi, Douglas Thain, Animesh Trivedi, Alexandru Uta, Vincent van Beek, Erwin van Eyk, André van Hoorn, Soam Vasani, Florian Wamser, Guido Wirtz, and Vladimir Yussupov

A definition is the first principle of any field of human inquiry. As for many other complex issues, for serverless computing the semantics have become source of commentary and debate. So, what is the object of our inquiry, *what is serverless computing?*

Many of this Dagstuhl Seminar attendees engaged in early discussions around this question. Early definitions include aspects such as: the deployment model of Function-as-a-Service and Backend-as-a-Service being key to operate complex serverless applications [1]; granular billing matching actual use, event-driven operation, and (almost) complete lack of operational



logic [1]; (almost) no concerns about operation for the user, function lifecycle management including events as triggers, operations including performance isolation and prediction, operations to trade-off cost and performance under guidance from the user [2]; the details of FaaS operation that users can expect to encounter, as a reference model spanning functions to workflows [3]; etc.

With so many aspects to consider, a definition remained elusive. During the seminar, an intense discussion thread started resulting in an improved common understanding of the notion of “serverless computing”, around notions such as:

1. **NoOps:** Hiding/abstracting complexity of *execution environment* (physical and virtual machines, hypervisors, operating systems, containers, etc.) as well as *system/operation aspects*, such as resource management, component/instance deployment, instance lifecycle, elasticity/autoscaling, reliability/fault-tolerance, ...
2. **Utilization-based billing:** a billing model that only charges for the *resources actually used* both with respect to time and space, for example, for Function-as-a-Service (FaaS) this translates into “pay only for function execution (space dimension) in fine-granular time units (time dimension)”.

Based on this, we formulate the following definition of serverless computing:

**Definition:** *Serverless computing* is a cloud computing paradigm offering a high-level application programming model that allows one to develop and deploy cloud applications without allocating and managing virtualized servers and resources or being concerned about other operational aspects. The responsibility for operational aspects, such as fault tolerance or the elastic scaling of computing, storage, and communication resources to match varying application demands, is offloaded to the cloud provider. Providers apply utilization-based billing: they charge cloud users in proportion to the resources that applications actually consume from the cloud infrastructure, such as computing time, memory, and storage space.

Today, Function-as-a-Service (FaaS) platforms are the most prominent example of serverless computing offerings. Current FaaS platforms focus on the function as a unit of computation assumed to be small, stateless, and event-driven (i.e., executed asynchronously in response to certain triggers or events). The short runtime and stateless nature of FaaS functions makes it easier for FaaS cloud providers to implement autoscaling in a generic manner, while applying a fine-granular utilization-based cost model that bills customers based on the actual time functions are running. Given its popularity and rapid adoption, today FaaS is often used interchangeably with serverless computing.

We believe that the current assumptions of the FaaS model (small, stateless, and event-driven units of computation) might eventually be relaxed, as platforms evolve to support a wider set of applications. In addition to FaaS platforms, our broad notion of serverless computing also explicitly includes modern Backend-as-a-Service (BaaS) offerings, which are focused on specialized cloud application components, such as object storage, databases, or messaging. Finally, some Software-as-a-Service (SaaS) platforms support the execution of user-provided functions tightly coupled to the specific application domain. In summary, the serverless ecosystem includes a growing set of technologies and evolving programming models (e.g., FaaS, BaaS, some PaaS and SaaS), which, taken together, will provide the basis for building (end-to-end) next-generation serverless cloud applications.

**Please cite as:**

Samuel Kounev et al., Toward a Definition for Serverless Computing, in Serverless Computing (Dagstuhl Seminar 21201) (Cristina Abad, Ian T. Foster, Nikolas Herbst, and Alexandru Iosup, eds.), vol. 11(4), Chapter 5.1, p.34–93, Schloss Dagstuhl Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021.

or using

```
@Incollection{kounev_et_al:DagRep.11.4.34:ServerlessNotion,
  author = {Samuel Kounev and Cristina Abad and Ian T. Foster and
Nikolas Herbst and Alexandru Iosup and Samer Al-Kiswany and
Ahmed Ali-Eldin Hassan and Bartosz Balis and Andr'e Bauer and
Andr'e B. Bondi and Kyle Chard and Ryan L. Chard and
Robert Chatley and Andrew A. Chien and A. Jesse Jiryu Davis and
Jesse Donkervliet and Simon Eismann and Erik Elmroth and
Nicola Ferrier and Hans-Arno Jacobsen and Pooyan Jamshidi and
Georgios Kousiouris and Philipp Leitner and Pedro Garcia Lopez and
Martina Maggio and Maciej Malawski and Bernard Metzler and
Vinod Muthusamy and Alessandro V. Papadopoulos and
Panos Patros and Guillaume Pierre and Omer F. Rana and
Robert P. Ricci and Joel Scheuner and Mina Sedaghat and
Mohammad Shahradsad and Prashant Shenoy and Josef Spillner and
Davide Taibi and Douglas Thain and Animesh Trivedi and
Alexandru Uta and Vincent van Beek and Erwin van Eyk and
Andr'e van Hoorn and Soam Vasani and Florian Wamser and
Guido Wirtz and Vladimir Yussupov},
  title = {{Toward a Definition for Serverless Computing}},
  booktitle = {{Serverless Computing (Dagstuhl Seminar 21201)}},
  pages = {34--93},
  journal = {Dagstuhl Reports},
  year = {2021},
  volume = {11},
  issue = {4},
  editor = {Cristina Abad and Ian T. Foster and Nikolas Herbst and
Alexandru Iosup},
  publisher = {Schloss Dagstuhl -- Leibniz-Zentrum f{"u}r Informatik},
  address = {Dagstuhl, Germany},
  doi = {10.4230/DagRep.11.4.34},
}
```

**References**

- 1 Erwin Van Eyk, Alexandru Iosup, Simon Seif, Markus Thömmes: *The SPEC cloud group's research vision on FaaS and serverless architectures*. WOSC@Middleware 2017: 1-4
- 2 Erwin Van Eyk, Alexandru Iosup, Cristina L. Abad, Johannes Grohmann, Simon Eismann: *A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures*. ICPE Companion 2018: 21-24
- 3 Erwin Van Eyk, Alexandru Iosup, Johannes Grohmann, Simon Eismann, André Bauer, Laurens Versluis, Lucian Toader, Norbert Schmitt, Nikolas Herbst, Cristina L. Abad: *The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms*. IEEE Internet Comput. 23(6): 7-18 (2019)

## Participants

■ André Bauer  
Universität Würzburg, DE

■ Simon Eismann  
Universität Würzburg, DE

■ Nikolas Herbst  
Universität Würzburg, DE



## Remote Participants

■ Cristina Abad  
ESPOL – Guayaquil, EC

■ Samer Al-Kiswany  
University of Waterloo, CA

■ Ahmed Ali-Eldin Hassan  
Chalmers University of  
Technology – Göteborg, SE

■ Bartosz Balis  
AGH University of Science &  
Technology – Krakow, PL

■ André B. Bondi  
Software Performance and  
Scalability Consulting LL, US

■ Kyle Chard  
University of Chicago, US

■ Ryan L. Chard  
Argonne National Laboratory –  
Lemont, US

■ Robert Chatley  
Imperial College London, GB

■ Andrew A. Chien  
University of Chicago, US

■ A. Jesse Jiryu Davis  
MongoDB – New York, US

■ Jesse Donkervliet  
VU University Amsterdam, NL

■ Erik Elmroth  
University of Umeå, SE

■ Nicola Ferrier  
Argonne National Laboratory, US

■ Ian T. Foster  
Argonne National Laboratory –  
Lemont, US

■ Alexandru Iosup  
VU University Amsterdam, NL

■ Hans-Arno Jacobsen  
University of Toronto, CA

■ Pooyan Jamshidi  
University of South Carolina –  
Columbia, US

■ Samuel Kounev  
Universität Würzburg, DE

■ Georgios Kousiouris  
Harokopion University –  
Athens, GR

■ Philipp Leitner  
Chalmers University of  
Technology – Göteborg, SE

■ Pedro García López  
Universitat Rovira i Virgili –  
Tarragona, ES

■ Martina Maggio  
Universität des Saarlandes –  
Saarbrücken, DE

■ Maciej Malawski  
AGH University of Science &  
Technology – Krakow, PL

■ Bernard Metzler  
IBM Research-Zurich, CH

■ Vinod Muthusamy  
IBM TJ Watson Research Center  
– Yorktown Heights, US

■ Alessandro Vittorio  
Papadopoulos  
Mälardalen University –  
Västerås, SE

■ Panos Patros  
University of Waikato, NZ

- Guillaume Pierre  
University & IRISA –  
Rennes, FR
- Omer F. Rana  
Cardiff University, GB
- Robert P. Ricci  
University of Utah –  
Salt Lake City, US
- Joel Scheuner  
Chalmers and University of  
Gothenburg, SE
- Mina Sedaghat  
Ericsson – Stockholm, SE
- Mohammad Shahradsad  
University of British Columbia –  
Vancouver, CA
- Prashant Shenoy  
University of Massachusetts –  
Amherst, US
- Josef Spillner  
ZHAW – Winterthur, CH
- Davide Taibi  
University of Tampere, FI
- Douglas Thain  
University of Notre Dame, US
- Animesh Trivedi  
VU University Amsterdam, NL
- Alexandru Uta  
Leiden University, NL
- Vincent van Beek  
Solvinty, Amsterdam and Delft  
University of Technology, NL
- Erwin van Eyk  
VU University Amsterdam, NL
- André van Hoorn  
Universität Stuttgart, DE
- Soam Vasani  
Stripe – San Francisco, US
- Florian Wamser  
Universität Würzburg, DE
- Guido Wirtz  
Universität Bamberg, DE
- Vladimir Yussupov  
Universität Stuttgart, DE

