NSF Cloud 3.0 Workshop Report

January 18-19, 2018, Stanford, CA

Organized by: Aditya Akella (UW-Madison), George Porter (UC San Diego), Keith Winstein (Stanford)

Table of Contents

Introduction and Report Overview

Cloud computing has revolutionized the computing landscape in the last decade, turning what was once a specialized resource -- high-performance networked computing platforms -- into a convenient and efficient commodity. The ability to scale out computing workloads and network services has been an enormous driver of innovation in industry, academia, and beyond.

Despite this growth, cloud computing, and cloud-computing research, has largely been stuck in an archaic computational paradigm: the renting of whole servers (virtual or physical), each with an allocation of CPU, RAM, storage, and network connectivity. With this model comes a host of issues that cloud providers and cloud users have attempted to paper over, with mixed success: inefficient allocation of computing and network resources, budget overruns, unresponsive elastic scaling in response to varied demands, difficulty securing whole-server/whole-OS platforms, and clumsy and low-level programming environments.

Recognizing the above challenges, cloud providers have recently begun to offer "serverless" cloud computing, in which cloud users only need specify the application code to be run in response to some event; the cloud provider handles the rest (scaling, provisioning, security, resource allocation, etc.). An early version was Platform-as-a-service (PaaS), exemplified by Google's App Engine, which followed even earlier serverless models, such as shared web hosting and CGI scripting. A more recent popular recent variant of serverless computing is Function as a Service (FaaS), which forms the primary focus of this report. In the most common style of FaaS, compute tasks are specified as individual functions to be launched dynamically across available resources when triggered by external events. Some in the industry have started to refer to FaaS as Cloud version 3.0, or "Cloud 3.0".

In what follows, we will abuse terminology somewhat and use "serverless" and "FaaS" interchangeably. We will distinguish other forms of serverless computing where necessary.

In one view, FaaS may be the "packet switching" equivalent of cloud computing, with services composed of very granular, fine timescale tasks, compared with services that rent whole machines by the minute, which are more akin to "circuit switching." As with the

advent of packet switching more than a half-century ago, the true promise and downsides of FaaS computing may not be clear for many years, and incumbents who are wedded to older models could staunchly oppose it--likely with some merit. This new cloud computing model requires new techniques and enables new applications, but *which* applications are most able to benefit remains to be seen. Research has only just begun to emerge in this space.

The NSF Cloud 3.0 Workshop was convened to study new directions in FaaS/serverless cloud computing. One goal of the workshop was to understand existing offerings in FaaS, their capabilities, and which applications benefit most from them (and how). Another goal was to explore how FaaS might look like in the limit; for example, to what extent can advances in systems and networking support even more stringent performance than today's FaaS offerings? A related goal here was to explore hitherto unseen applications that might exploit such future FaaS platforms, and which applications or use-cases are fundamentally a poor fit for FaaS. The discussion was meant to be broad, to include various possible FaaS instantiations, e.g., third-party based serverless offerings vs. FaaS infrastructure put in place by an enterprise to supports its internal applications (akin to a private cloud today).

The participants were asked to identify themes and directions for new research in serverless computing, both at the infrastructure level (i.e., research on cloud platforms themselves) and at the application level (i.e., research on innovative ways of leveraging FaaS). The participants represented a wide range of expertise on systems, networking, and security from both academia and industry. Before the workshop, participants were asked to submit position papers describing their current research and their views on promising research avenues. The workshop was organized around numerous breakout sessions each of which focused on specific sub-areas of research in serverless computing. The breakout sessions then reported back to the larger group and the findings were used to guide subsequent breakout sessions. Two industry participants, one from Google and another from Microsoft, were invited to present to the group about their efforts in cloud computing.

This workshop report summarizes the findings of the workshop. We identify key research challenges in support for existing applications, as well as new hitherto unseen applications, and new grand research challenges both in core networking and systems, as well as cross-cutting challenges, aimed at future fundamental advances in FaaS. We conclude by outlining barriers to conducting research in this exciting space and offer our thoughts on overcoming them.

Applications

A key issue that arises is what scenarios is FaaS inherently suited or not suited for. FaaS can help improve flexibility in developing and deploying applications. It can also help offer

improvements in performance (latency, scale-in/out, etc) and availability compared to alternative ways of deploying applications. In what follows, we present example applications where FaaS could *potentially* offer benefits. This list is by no means exhaustive, but is merely meant as a starting point for exploration and discussion. We also outline challenges to realizing said benefits. Some of the benefits hinge on key advancements to the underlying platforms so as to ensure suitable performance, availability, security, and programming for current or future applications leveraging FaaS; we visit these issues in a subsequent section.

Edge cloud based network functions

Especially in the context of edge clouds, the combination of NFV and serverless computing can be quite powerful in enabling the seamless on-demand execution of specialized packet processing for end devices. For example, application-specific lambdas executed with an NFV framework in an edge cloud could be tailored to the specific needs of end-device traffic from sources such as video streaming, group collaboration, or security-sensitive applications. Service providers could offer such network-based enhanced service and end-users (or the applications they use) could selectively and dynamically invoke those in-network services as needed.

While serverless environments can be used very cost effectively for large-scale parallel processing data analytics applications, it is less clear if networked applications, such as middleboxes and routing/switching, can also benefit from these new computing services as they do not share the same characteristics. Performance targets for traditional cloud-based applications may not translate well to the needs of NFV. For instance, a key issue is obtaining high performance for network functions that must maintain state across packets, which requires accessing an external store. (State management in serverless computing is a fundamental research challenge in itself; we will visit this topic shortly.) Thus, a broad research area examining fundamental constraints existing platforms impose but also exploring how to offer the necessary architectural support for a range of network functionality, including routing, switching, middleboxes and application-level gateways. Exploring workloads such as these will be crucial for identifying if and how cloud-native designs can be realized for various important and “extreme” use-cases.

Large scale video and image analytics

These applications play a central role in a variety of tasks, ranging from broad questions such as image recognition challenges, to traffic and environment monitoring, and public safety. It is therefore important to consider how these applications can leverage the serverless world to achieve “burst-parallel” brief execution across thousands of nodes. FaaS services can offer much promise to such streaming analytics tasks. These applications can be particularly challenging because they use not just CPUs but GPUs and FPGAs as well,

and may stress many resource management and programming systems for serverless computing. Beyond today's video, it would be interesting to explore the processing of 3D sensors (LiDaR, stereo cameras) using serverless computing, under the assumption that streams of these sensors will become feasible in the near future. A key question here is to understand FaaS's suitability as a real-time execution platform. Subtly tied to this issue is the availability of data. If one has to upload a chunk of video, the time to do so may likely overshadow the analytics time. If it is a streaming live video that is being analyzed, the real-time nature of the FaaS platform will be stressed more.

Data Science in the serverless world

What is the role of FaaS in machine learning (ML) applications? There are two ways to consider this issue. ML applications can be run on platforms that offer "ML-as-a-service"; here FaaS plays the role of helping coordinate ML-as-a-service. Another option is to run ML applications atop FaaS -- where FaaS plays a role in implementing ML applications. Key issues arise in both scenarios. For example, in the latter case, we must ask: How do we enable new ML applications (e.g., large-scale and long running analytics over constantly evolving datasets) using serverless computing? More generally, how do we enable data science pipelines in the serverless world? Enabling such applications can have the road benefit of democratizing data analytics, or data science more generally. Two main challenges stand out. First, serverless computing, as of today, is priced much higher than doing the same unit of computing using rented VMs. If we intend to process live HD data streams 24x7, for example, this can be uneconomical. Second, 24x7 ML processing does not readily lend itself to the model of short-lived functions. A key research topic will involve how to build stateful and reliable long-standing computations out of serverless infrastructure, to support, for example, life-long learning applications. Another key question is building suitable scheduling and orchestration infrastructure for data science pipelines that is easy and flexible to program and control.

There are a number of other interesting systems-building challenges. As serverless functions are stateless there is a need for systems support to enable cross-function communication in the form of shuffles, broadcasts, reduces etc. Furthermore, there are a number of resource limitations such as fixed amount of memory, limited CPU time which restrict the kind of workloads that can be executed. Designing automatic ways to adapt existing workloads or developing new interfaces which can allow users to express applications like iterative ML algorithms would expand the class of workloads that can benefit from serverless technology.

Enabling domain scientists

While new computational methods create a multitude of data by-products, the root of the Big Data revolution is in the fact that instruments produce more data, more complex data, and that we build increasingly more interesting complex instruments. Working with those

online data producers typically means that timeliness of response is at a premium, both in order to “steer” an instrument effectively and to develop a feasible “conversation”. The ability to manage overall response times implies the delivery of an end-to-end quality of service which in turn implies controlled execution over various components of the system. A key challenge is enabling serverless execution with the property of timeliness (or even better the capability to trade-off timeliness against other qualities of service). This would revolutionize all the existing experimental and observational sciences, and enable new critical applications such as personalized medicine.

New security capabilities

Can serverless platforms enable new, and more cost-effective security solutions? For example, serverless computing can enable interesting new moving target defenses. Likewise, it can provide scalable and cost-effective DDoS defense systems. A broad question is understanding the envelope of security issues for which serverless provides “better” solutions than existing fixed function security solutions. A related question is whether FaaS may make it harder to have good performance and protect against compromises such as Spectre or Meltdown. Another question is: what can an attacker “learn” in a serverless world vis-a-vis today’s VM-driven or server-driven workloads?

Publish-subscribe applications

At one end of the spectrum, serverless computing enables programming with containers that are very similar to server environments with computation, storage, and network resources available as in a Linux server. At the other end of the spectrum, serverless computing enables each event to have its own computing, storage, and networking resources on an event basis. This is a natural fit for applications such as publish and subscribe systems.

Monitoring

Cloud1.0/2.0 has made it possible to talk about and offer Monitoring-as-a-Service (MaaS); that is, the ability for an enterprise to spin up virtual appliances (with monitoring capabilities such detailed packet capture) pretty much anywhere in its network (e.g., cloud and non-cloud portions) and for any desired duration. MaaS typically results in big data in the form of distributed streaming data, but their analysis in support of real-time network management solutions remains an open problem. Is serverless computing a part of a practically deployable solution to this problem? An interesting issue is to explore how such distributed streaming data can be queried in real time to develop novel approaches to cyber security and new solutions to network performance-related problems. In particular, what new capabilities serverless computing can offer to address the real-time constraints

and scalability issues (e.g., data rates, number of simultaneous queries) that make this a “hard” problem?

Laptop/desktop extensions

To move “traditional” applications to a serverless platform, we need mechanisms for running computations over FaaS that can automatically capture the inputs and processes of existing computational applications and run them faithfully with high parallelism. The broader vision can be achieved by doing this via a “laptop extension” where processes in the cloud see the environment as local ones. The low startup time of serverless platforms may let us transparently offload tasks from traditional computing environments.

Cloud-native applications

The next generation of services and systems must take a critical step forward to fully utilize emerging cloud platforms. Such “cloud-native systems” are designed not just to take advantage of the rentable nature of computing infrastructure, but intrinsically utilize now-standard as well as emerging cloud services to realize their end goals. For example, scalable, reliable distributed storage (e.g, Amazon’s S3, Google’s Cloud Storage, Azure’s Blob Storage) is now ubiquitous; these services form a strong storage base upon which to build applications and services, instead of simple collections of raw storage resources (e.g., disk drives). Similarly, new serverless compute platforms (such as Amazon’s Lambdas, Google Cloud Functions, or Azure Functions) enable users to launch small pieces of computation on demand, scaling up or down readily to take advantage of workload parallelism, all without considering issues such as server provisioning or maintenance. Cloud-native systems exploit base cloud services to realize new, more flexible, high-performance, reliable systems and services more readily than ever before. A key issue is outlining cloud-native principles that underlie this vision with a view on FaaS. We must understand how to align today's cloud application services / architectures with FaaS and develop the principles of this integrated approach. The principles, when applied correctly, can showcase new points in the systems and networking design space which are directly enabled by this modern version of the cloud.

FaaS Building Blocks

In what follows, we outline the research opportunities pertaining to improving FaaS offerings themselves, in terms of improving performance, availability, state management, programmability, moinitoring, etc. Opportunities for research into applications built on top of FaaS were presented in the previous section. We envision applications running atop FaaS to have a variety of service level objectives (SLOs), ranging from tight, low latency requirements on the one extreme to loose requirements that focus on the overall cost of

running the applications' computation. An ideal serverless platform should be able to optimally support different application SLOs.

Networking and systems

Low-latency execution and scheduling

Benchmarking of AWS Lambda and Google Cloud Functions shows that the latency for launching functions can be 10s to 100s of milliseconds. We need to bring this latency down, ideally to sub-10ms scales. Potential avenues of research include container reuse, very fast container spin up, and predicting/prefetching inputs over the network.

Fine-grained resource sharing

As serverless functions leverage computation resources beyond CPUs and moves to GPUs and FPGAs, fine-grained sharing of these resources will pose significant challenges. For example, existing GPU manufacturers provide limited isolation support between multiple application processes running on the same GPU and forces users to choose between high performance and low cost. Given that a function typically requires a fraction of a GPU's resources, allocating a complete GPU to one user is inefficient. We should explore how to perform fine-grained sharing of GPUs and FPGAs at the level of individual functions and users. Ensuring end-to-end isolation is another classic challenge that will only become more challenging in the serverless world when a function goes through a variety of resources with unique constraints.

Server-network interface

When multiple applications are multiplexed atop FaaS platforms, we can arrive at a situation where we need the underlying compute infrastructure to support flexible, hierarchical network policies; e.g., a hierarchy of rate-limits, priorities and weights specified across different applications, some of which have tight latency requirements, and other have elastic performance needs. Such requirements are difficult to support today. In particular, with current NICs, cloud operators must either 1) use a single NIC queue and enforce network policy in software, which incurs high CPU overheads and struggles to drive increasing linerates, or 2) use multiple NIC queues and accept that it is no longer possible to isolate competing applications or enforce network policy . These limitations particularly impact serverless applications with tight performance constraints. It also limits complex, multi-stage applications that need flexible, dynamically changing policies (e.g., dynamically re-prioritize some flows over others to meet a tight service-level objective). Existing solutions such as Flow Director are limited, e.g., they cannot support arbitrary hierarchical policies. Thus a case can be made for considering a new NIC design that supports both performance and the afore-mentioned flexibility in the context of serverless applications.

The undeniable reality of today's networking hardware and end-host stacks is that they are far more efficient when allowed to batch operations and transfer large blocks of data. The overheads involved in sending small chunks of data dominate the transfer times by orders of magnitude in existing networking stacks, and is only getting worse. Serverless computing is yet another reason why it is important to reconsider today's network stack, and the set of abstractions modern operating systems expose. We should explore whether providing a number of distinct network channels may be superior than today's model of a single network fabric.

Inter-Lambda/Thread communication

If serverless platforms outgrow their Web-based origins and eventually find broad adoption as general-purpose parallel computers, we may need IPC primitives similar to MPI or Unix domain sockets. Current methods for communicating between threads on serverless platforms are quite cumbersome and inefficient (e.g., using a rendezvous server to enable thread-to-thread communication).

Providing low-latency IPC primitives for serverless platforms could provide us a pathway to easily porting cluster-computing applications that previously relied on IPC primitives such as MPI. These include scientific computing applications that require coordination between threads such as particle simulations and PDE solvers. Recent work in datacenter networking reports RPC latency on the order of tens of microseconds by modifying both the end hosts and the network's switches. While more research is required to translate these networking results into end-to-end low-latency IPC primitives, they show that low-latency IPC might be feasible assuming the network's switches can be programmed, a capability that is emerging in production switches today.

Managing state

Embarrassingly parallel, stateless functions can do only so many things. In many cases, we may have to maintain states between iterations and stages. Existing serverless computing infrastructures only provide heavyweight solutions; e.g., storing states into Amazon S3 in case of Amazon Lambda. These storage systems were not designed for serverless use cases, where each function may deal with a small amount of data. Existing in-memory storage and caching solutions (e.g., Alluxio) are not fast enough for extremely low-latency accesses. Key-value stores lack common file system semantics and do handle large I/O well. We need a fast, in-memory storage solution that supports objects ranging from bytes to terabytes, is elastic, and acts as both a cache and a durable storage layer.

A concrete goal would be a decentralized, fault-tolerant, in-memory storage system, where individual bytes can be named and accessed using the same interface as that used to access large files. Serverless functions can store, retrieve, and discard their states in orders-of-magnitude faster rate than that using blob stores like S3 or existing in-memory storage systems like Alluxio. It would be expected to work on both DRAM and NVM.

Another possible direction would be exposing a shared disaggregated memory (DRAM and NVM) to all the functions in the same workflow in a transparent manner so that developers do not have to even reason about IPC or shared memory for inter-function communication. While there are recent solutions for memory disaggregation, sharing disaggregated memory and making it resilient to failures and load imbalance are all open challenges.

Tracing and debugging

In systems that permit serverless computing using short stateless functions, tracking the provenance of data through the system, and fine time-scale monitoring are important problems. Solving these may allow serverless applications--built out of small functions with documented inputs and outputs--to be more understandable, reproducible, and debuggable than more monolithic applications of today. A key challenge here is tracing indirect (storage-triggered) invocations.

Security

To provide such end-to-end timeliness of response we will need to explore system-level enforcement, potentially combined with accurate application characterization that can improve the delivery of controlled response times by pairing off non-interfering (or low-interfering) types of applications. This general principle applies across the spectrum of resource types that will need to be scheduled though will raise different specific challenges for each type of resource (e.g., program execution versus data transfer). Combining individual heterogeneous components into an end-to-end system is a challenge in itself and requires bridging different models, different levels of trust, and provider domains.

- Can we categorize and refine existing FaaS security architectures, point out deficiencies in their isolation models, and suggest near-term improvements?

- After decades of side-channel attacks, we've now seen the sky-is-falling version in the form of Meltdown/Spectre. What are good clean-slate designs of cloud hardware and software stacks that provide strong guarantees of side-channel-freeness, yet support the elasticity and agility of existing services?

- Lots of new software will be written for FaaS systems, and old security mistakes will be reinvented in new forms. How do we get ahead of this and provide FaaS design patterns, security frameworks, and APIs that guide developers towards getting right basic security issues (e.g., access control)?

Programming models

What are the right programming interfaces to serverless computing? Haskell-type model of compute over immutable named state? Constraining the dataflow with higher-level abstractions (MapReduce, SQL, etc?) A related, more general issue is that of having narrow, more optimizable programming models (e.g., JavaScript only or Haskell only) vs. more general models (native binaries, variety of languages/platforms) that affect the system design and optimizations available.

Lambdas/functions will permit “whitebox” resource management, placement and provisioning, as well as verification, by permitting static and dynamic program analyses of these lambdas. Two complementary research questions are the design of programming languages for lambdas that permits analysis, and the synthesis of lambdas from higher-level specifications. Finally, the design of highly-available datacenter fabrics and rack-scale systems for lambda execution that disaggregate compute and storage will likely become significantly important.

An open research question concerns the right level of isolation (containers, VMs, unikernels, etc.) and the proper abstraction boundary between user code and the platform. Should the details of compute *and* storage be abstracted from user code? Should all data access be declared to the platform, to allow efficient fine-grained scheduling by the provider? How can these abstractions be efficiently implemented? Can we design language-level mechanisms that offer these properties?

Barriers and Solutions

What are barriers to conducting research on serverless?

A key issue we face in our research is the opacity of existing serverless platforms. Very little is known about how they are designed and run, and as such it is difficult to reason about observed performance issues, and whether they are fundamental are not. Thus, determining how to make fundamental improvements is not easy. To address this, having open source serverless platforms would be invaluable; there is already work on this front, in the form of OpenWhisk and OpenLambda, but both are in very preliminary stages.

It would be useful to understand what real users are actually doing with serverless and what they could not have done/could have done but less efficiently with serverless vs. traditional cloud computing. Ultimately, understanding this may spur thinking into new, hitherto-unseen applications that benefit from future cloud computing offerings that could potentially offer much lower latencies and higher performance than today’s offerings.

It would likewise be useful to know what is going to change for an enterprise as it transitions from existing infrastructure as a service (IaaS) or other setups to Cloud3.0. How can we use the lessons learned from past experience on transitioning from “legacy” on-premise setups to IaaS to simplify the transition to Cloud 3.0? Will much of the change be opaque to the enterprise and only/mainly impact its cost structure (e.g., less cost due to more efficient utilization of resources) or are there obvious “killer apps” that require a closer look, both in terms of the functionality they require and capabilities they offer?

Serverless computing will clearly generate new workload patterns that systems and networking researchers will need to address. One of the most obvious of these is the fact that the offered workload will be much more dynamic and the control systems will need to be more responsive to deal with that, and optimize for the fluctuating demands. This new environment is likely to come with other requirements, e.g. low latency processing and efficient processing chains. However, until we have a better understanding of these workloads and associated requirements, it will prove difficult to produce effective designs.

To understand and innovate on systems issues such as resource allocation and fault tolerance, the academic community will require access to datacenter infrastructure traces that can be used to study the impact of the changes proposed. While some of the recent traces like the Azure VM trace provide statistics on VM allocations, there is no publicly available resource to understand the usage patterns in serverless computing. These shortcomings can also be addressed by adding serverless infrastructure to open source test-beds like CloudLab and using them to run scientific or other academic workloads.

A complete integrated platform for executing granular applications in modern datacenters is useful. The platform should enable experimentation -- it should not bake in assumptions about granularity of computing. Researchers should be able to bring up serverless frameworks that are completely under their control and can be arbitrarily modified. There are existing open-source systems (eg. OpenLambda) that fit this need from the software side, and cloud computing testbeds (eg. CloudLab and Chameleon Cloud) that provide the necessary hardware resources, but these pieces need to be better integrated, and the open serverless frameworks need to be set up so that they more directly model what is found in public clouds. Of course there is an inherent risk in the need for large engineering resources to keep up with the public clouds on their many capabilities.

The research community (perhaps in collaboration with industry) would do well to develop flexible yet robust open-source lambda frameworks designed, for example, for high-performance networking so that it is easy to develop new designs in a research setting and have those designs be adopted in practice.

What new testbeds, datasets, or tools could be provided to enable research in this area?

Testbeds

We outline specific thoughts not covered above on requirements for serverless testbeds to meet. In building a testbed that supports FaaS research, it is useful to understand experiments where the scale is really important: Would a scale of ~1,000 be sufficient (~1,000 cores) to explore a particular issue in serverless? How do we determine this? Instrumentation and monitoring are crucial. Experimenters may be interested in fine-grained customize data gathering.

Today, on CloudLab and Chameleon, experimenters often want to be able to download open source version of something that is popular, e.g., Eucalyptus + 2 racks and then do research (e.g., on applications running atop said platforms). It would be valuable to enable something similar for Cloud 3.0.

Data, broadly defined

In general, it is a lot more useful to share know-how than a testbed. It would thus help to enable a process by which best practices can be shared by the industry. Build canonical “test workloads” based on information and traces that can be gleaned from customer reports on lambda providers’ websites.

Resources

Many experiments in the serverless computing research area are increasingly looking into disaggregated hardware -- fast storage via very fast network connecting nodes with storage, memory, etc. It is worthwhile to to understand how best to build out such testbeds and enable sharing across experiments.

An additional barrier is access to physical machines (testbed/resources) that have the latest and state of the art instruction sets for virtualization and isolation. Also, machines that have accelerators (FPGAs) and a way to access and program the FPGAs. Having access to such resources at scale is crucial to understanding and developing future serverless architectures.

For much of the research in this space, it might be helpful to, rather than have shared testbeds, make available the know-how and associated software for building customized “microtestbeds” that are perhaps locally shared by a few institutions. This will help groups who find a single shared general testbed to be restrictive, e.g., toward conducting research

on highly performance-sensitive applications, to set up customized smaller versions that better match their research goals.

Closing

In summary, we believe that serverless computing offers exciting new challenges and opportunities. There is plenty of research to accomplish in core systems and networking, as well as at boundaries with related fields such as formal methods. However, we note that there are key barrier to conducting this research effectively. Overcoming these, with suitable infrastructure support and dataset exchange, would be important to allow research on serverless to come to fruition.

Workshop Participants

Adam Wierman	Caltech
Aditya Akella (Organizer)	UW-Madison
Albert Greenberg	Azure
Alex C. Snoeren	UCSD
Ali Ghodsi	Databricks
Anirudh Sivaraman	NYU
Ann Von Lehmen	NSF
Barath Raghavan	USC
Bruce Maggs	Duke
Changhoon Kim	Barefoot
Darleen Fisher	NSF
Ganesh Ananthanarayanan	MSR
George Porter (Organizer)	UCSD
George Varghese	UCLA
Hakim Weatherspoon	Cornell
Ion Stoica	UC Berkeley
Jeff Mogul	Google
John Brassil	NSF
John Ousterhout	Stanford
Kate Keahey	ANL
Keith Winstein (Organizer)	Stanford
Ken Calvert	NSF
Matei Zaharia	Stanford
Michael Swift	UW-Madison
Monia Ghobadi	MSR
Mosharaf Chowdhury	Michigan
Nate Foster	Cornell
Nick McKeown	Stanford

Parveen Patel	Azure
Peter Steenkiste	CMU
Rachit Agarwal	Cornell
Ramesh Govindan	USC
Remzi Arpaci-Dusseau	UW-Madison
Rob Ricci	Utah
Shivaram Venkataraman	UC Berkeley/UW-Madison
Srini Seshan	CMU
Sujata Banerjee	VMWare
Suman Banerjee	UW-Madison
Theophilus Benson	Brown
Tom Ristenpart	Cornell
Vyas Sekar	CMU
Walter Willinger	Niksun
Xiaowei Yang	Duke

Pointers to Full Workshop Resources

Full workshop information, including one-page write-ups, slides, and notes from breakout sessions and keynote talks may be found at https://sites.google.com/site/cloud3workshop/