DOI:10.1145/3406011

**The evolution that serverless computing represents, the economic forces that shape it, why it could fail, and how it might fulfill its potential.**

BY JOHANN SCHLEIER-SMITH, VIKRAM SREEKANTI, ANURAG KHANDELWAL, JOAO CARREIRA, NEERAJA J. YADWADKAR, RALUCA ADA POPA, JOSEPH E. GONZALEZ, ION STOICA, AND DAVID A. PATTERSON

# What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing

IN 2010, SOME of us co-authored a *Communications* article that helped explain the relatively new phenomenon of cloud computing.[4] We said that cloud computing provided the illusion of infinitely scalable remote servers without charging a premium for scale, as renting 1,000 servers for one hour costs the same as renting one server for 1,000 hours, and that economies of scale for the cloud provider allowed it to be surprisingly inexpensive. We listed challenges to cloud computing, and then predicted that most would be overcome so the industry would increasingly shift from computing inside local data centers to "the cloud," which has indeed happened. Today two-thirds of enterprise information technology spending for infrastructure and software is based in the cloud.[8]

We are revisiting cloud computing a decade later to explain its emerging second phase, which we believe will further accelerate the shift to the cloud. The first phase mainly simplified system administration by making it easier to configure and manage computing infrastructure, primarily through the use of virtual servers and networks carved out from massive multi-tenant data centers. This second phase hides the servers by providing programming abstractions for application builders that simplify cloud development, making cloud software easier to write. Stated briefly, the target of the first phase was system administrators and the second is programmers. This change requires cloud providers to take over many of the operational responsibilities needed to run applications well.

To emphasize the change of focus from servers to applications, this new phase has become known as *serverless computing*, although remote servers are still the invisible bedrock that powers it. In this article, we call the traditional first phase *serverful computing*.

Figure 1 shows an analogy. To attend a remote conference, you either rent a car or hail a taxicab to get from the airport to your hotel. Car rental is like serverful computing, where you must wait in line, sign a contract, reserve the car for your whole stay no

## » key insights

- **The cloud originally revolutionized system administration. This second phase of cloud computing simplifies cloud programming.**
- **Serverless computing encompasses much more than cloud functions, or Function-as-a-Service (FaaS)—other cloud programming abstractions such as object storage also hide the complexity of servers, and more are on the way.**
- **Serverless today works well in limited applications, so cloud providers will create new application-specific and general-purpose serverless products to enable more use cases.**
- **This next phase of cloud computing will change the way programmers work as dramatically as the first phase changed how operators work.**

IMAGE BY ALEXEY V SMIRNOV

matter how much you use it, drive the car yourself, navigate to the hotel, pay for parking, and fill it with fuel before returning it. The taxi is like serverless computing, where you simply need to give the hotel name and pay for the ride; the taxi service provides a trained driver who navigates, charges for the ride, and fills the gas tank. Taxis simplify transportation, as you don't need to know how to operate a car to get to the hotel. Moreover, taxis get higher utilization than rental cars, which lowers costs for the taxi company. Depending on the length of the conference, the cost of car rental, the cost of parking, the cost of gas, and so on, taxis are not only easier, they might even be cheaper.

In serverless computing, programmers create applications using high-level abstractions offered by the cloud provider. For example, they can define *cloud functions*[a] using functional-style "stateless" programming in the language of their choice, often JavaScript or Python, then specify how the functions should run, whether in response to Web requests or to triggering events. They may also use serverless object storage, message queues, key-value store databases, mobile client data sync, and so on, a group of services offerings known collectively as *Backend-as-a-Service* (BaaS). Managed cloud function services are also called *Function-as-a-Service* (FaaS) and collectively Serverless Cloud Computing today = FaaS + BaaS (see Figure 2).

The main innovation of serverless is hiding servers, which have an inherently complex programming and operating model. Server users must create redundancy for reliability, adjust capacity in response to changes in load, upgrade systems for security, and so on.[17] This often requires difficult reasoning about failure modes and performance in distributed systems. Tools can help, for example, by adjusting capacity heuristically, a form of *autoscaling*, but these too require detailed configuration and ongoing monitoring. By contrast, serverless hands these and other responsibilities to the cloud provider.

Three essential qualities of serverless computing are:

1. Providing an abstraction that hides the servers and the complexity of programming and operating them.

a Different cloud platforms have different names for their offerings: Azure Functions for Microsoft Azure, Cloud Functions for Alibaba Cloud, AWS Lambda for Amazon Web Services (AWS), Google Cloud Functions and Google Cloud Run for Google Cloud Platform (GCP), IBM Cloud Functions for IBM Cloud, and Oracle Functions for Oracle Cloud.

2. Offering a pay-as-you-go cost model instead of a reservation-based model, so there is no charge for idle resources (see Figure 3).

3. Automatic, rapid, and unlimited scaling resources up and down to match demand closely, from zero to practically infinite.

The cloud-based synthesis of *all* of these properties is substantially more transformative than previous environments that came close to providing them.[8,17] Returning to our analogy, a taxi service (serverless computing) must provide a cab with a licensed driver (hide operation), charge only when giving a ride (pay as you go), and schedule enough cabs to minimize customer wait time (autoscaling). If taxis don't reliably provide all three, then customers may instead rent and operate cars (serverful computing).

**Figure 1. Cloud computing approaches compared to rides from an airport: Serverful as renting a car and serverless as taking a taxi ride.**

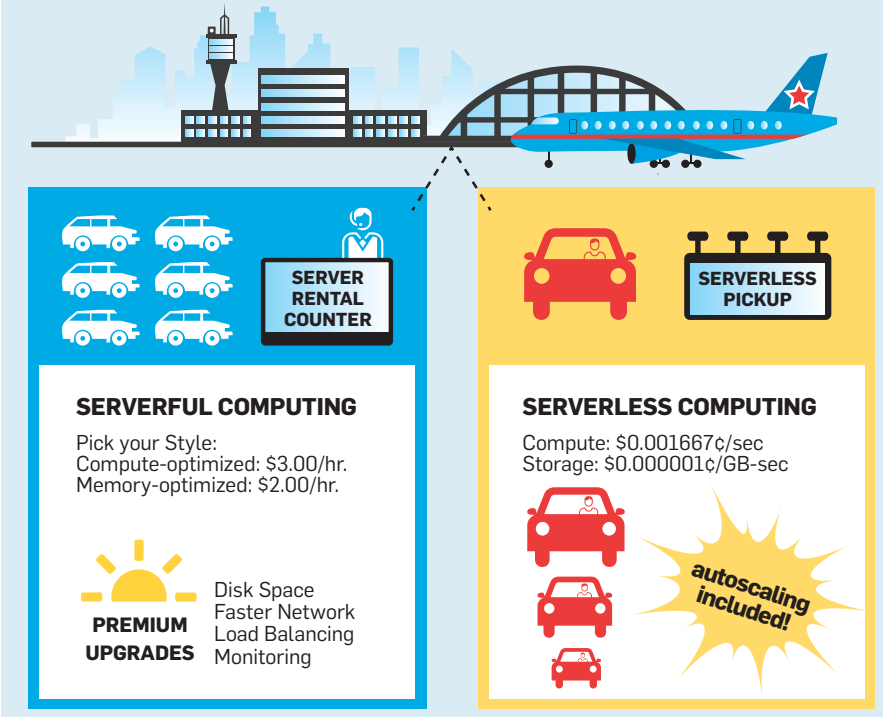
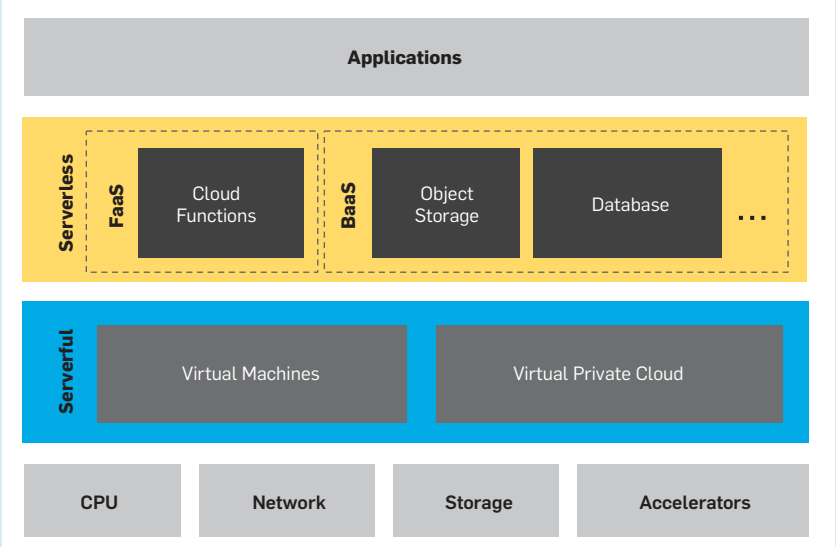



**Figure 2. Serverless vs. Serverful cloud computing: serverless provides an abstraction between applications and the underlying servers.**




A recent *Communications* article gave an excellent introduction to the current state of serverless computing, how it differs from Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS), its market share, example use cases, and its limitations.[8] In this article, we share our views on the evolution that serverless computing represents, the economic forces that shape it, why it could fail, and how it might evolve to fulfill its potential.

We project that the majority of data center computing will be dominated by serverless computing but we also believe that serverless computing will depart substantially from the serverless offerings of today. In particular, we believe that new general-purpose serverless abstractions will emerge, adding sophisticated state management and automatic optimization to enable many more use cases. Serverless now depends upon homogeneous CPUs, but in the future serverless will simplify use of hardware accelerators such as Graphical Processing Units (GPUs) or Tensor Processing Units (TPUs)[19] that support specific workloads—they offer the most likely path to higher performance as Moore's Law slows.[14] While there are concerns today about serverless security, we believe that a careful design could in fact make it *easier* for application developers to secure their software against external attackers.

As in 2010, we once again predict that these challenges will be overcome and this second phase will become the dominant form of cloud computing, accelerating its popularity by putting the power of the cloud in the hands of *all* application developers.

## Understanding What Serverless Is Today

Cloud functions[8] capture much of the mindshare in serverless computing, but they are one of many services in the serverless cloud. The excitement around FaaS is well justified because it offers a glimpse of what general-purpose serverless computing might look like, yet BaaS services comprise a much larger, and older, set of serverless services.

For example, AWS initially offered

their S3 object storage as a remote backup and archival service, years before announcing EC2 virtual machine rental. You can think of S3 as a precursor to serverless computing that offered "diskless storage," that is, providing storage but hiding the disks. Over time, cloud providers offered additional BaaS services to help serverful computing. Message queues (for example, AWS SQS, Google Cloud Pub/Sub) were another early service. Later came key-value databases (for example, Google Cloud Datastore, AWS DynamoDB, Azure CosmosDB) and SQL-based big data query engines (for example, AWS Athena, Google BigQuery).

When AWS Lambda launched in 2015 it was the first cloud functions product and offered something unique and compelling: the ability to execute nearly any code that runs on a server. It included support for several programming languages and for arbitrary libraries, all on a pay-as-you-go basis, operating securely and at any scale. However, it imposed certain limitations on the programming model that even today restrict it to certain applications. These include a maximum execution time, the lack of persistent state, and restricted networking.[13]

Today, several serverless environments can run arbitrary code, each catering to a particular use case. For example, Google Cloud Dataflow and AWS Glue allow programmers to execute arbitrary code as a stage in a data processing pipeline, while Google App Engine can be thought of as a serverless environment for building Web applications.

These many serverless offerings have in common the three essential qualities of serverless computing: an abstraction that hides the servers, a pay-as-you-go cost model, and excellent autoscaling. Taken together they offer a set of alternatives that may be combined to meet an ever-growing range of applications.

## Serverless Cloud Economics

Today's cloud has been shaped as much by business considerations as by technical progress, and its future will be as well. Cloud customers choose serverless computing because it allows them to stay focused on solving problems that are unique to their domain or business, rather than on problems in server administration or distributed systems.[6] The strength of this customer value proposition is a primary cause for optimism about the future adoption of serverless computing.

While serverless computing may appear more expensive since the unit prices of resources are higher (see sidebar "The Cost of Serverless"), customers only pay for resources that they are using, while the cloud provider bears the cost of idle resources. In practice, customers realize substantial cost savings when porting applications to serverless.[30] While this cost reduction could threaten cloud provider revenues, the Jevons Paradox[2] suggests that low prices can spark consumption growth that more than offsets the reduction in unit costs, leading to revenue growth. Cloud providers also gain a profit opportunity by helping customers meet variable and unpredictable resource needs, something they can do more efficiently from a shared resource pool than customers can do using their own dedicated resources.[16] This opportunity also exists in serverful computing but grows as resources are shared on a more fine-grained basis. Serverless computing also offers cloud providers opportunities to improve their margins because BaaS products often represent product categories traditional-

# The Cost of Serverless

**If you compare the per-minute cost of running an AWS Lambda cloud function with the cost of an AWS t3.nano VM with the equivalent 0.5 GB memory, it might look like serverless computing is 7.5x as expensive. Such a comparison is misleading, however.**

**The beauty of serverless computing is that it provides much more than servers, yet results in cloud bills that are often much lower. Included in the price is redundancy for availability, monitoring, logging, and automated scaling, all of which need to be provided separately in a serverful context. Cost comparisons must also factor in expected utilization, since serverless users pay only while their code executes. The users of the t3.nano VM must pay for the resources reserved, whether their code is running or not. Cloud providers claim that in practice customers see cost savings of 4x-10x when moving applications to serverless.[30]**

**While serverless often saves money, for some organizations the pay-as-you-go model is at odds with the way they manage their budgets. These may be fixed in advance, often annually. Planning to use a fixed amount of server capacity may seem easier, but managing to budget is challenging in practice, especially when many teams deploy cloud VMs, or when business needs are difficult to anticipate. We believe that as organizations use serverless more, they will be able to predict their costs based on history, similar to the way they do for other pay-as-you-go services, like electricity.**

**Figure 3. Serverless vs Serverful cloud computing: serverless users pay only for resources consumed, not for idle reserved capacity.**

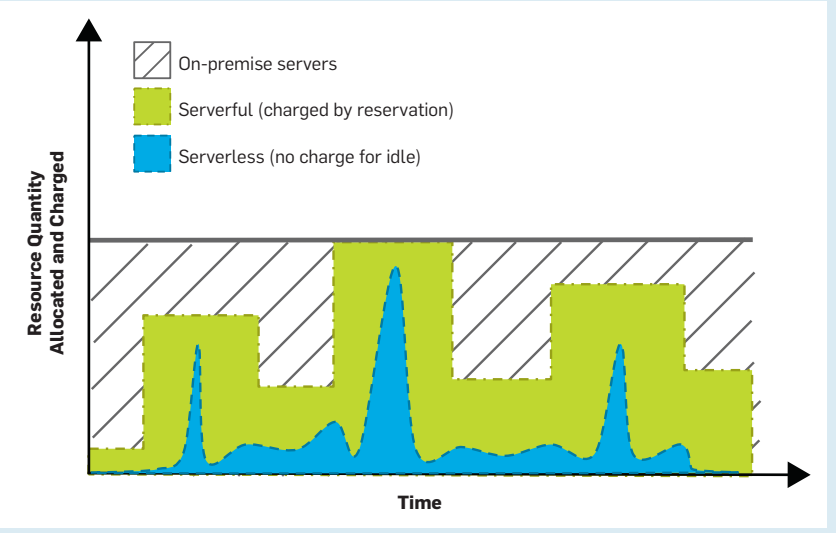


**Table 1. Alternative abstraction approaches.**

Cloud functions might appear to offer a general-purpose abstraction since they run arbitrary code, however due to their limitations they work only in some applications. More sophisticated derivatives might achieve the goal of general-purpose serverless computation.

| Serverless Abstraction Approach | | Big Data Example |
|---|---|---|
| **Application-specific** | Tool or component | AWS Athena |
| | Application framework | Cloud Dataflow |
| **General-purpose** | Hints to implementation | Affinity hints |
| | Automatic optimization | Communication-minimizing placement |

ly served by high-margin software products such as databases.

The serverless pay-as-you-go model has an important positive implication for the cloud providers' incentive to innovate. Before serverless, autoscaling cloud services would automatically provision VMs, that is, reserve resources, but the customer would then pay for this capacity even if it remained idle. With serverless, the cloud provider pays for idle resources, which creates "skin in the game" on autoscaling, and provides incentives to ensure efficient resource allocation. Similarly, as the cloud provider assumes direct control over more of the application stack, including the operating system and language runtime, the serverless model encourages investments in efficiency at every level.

More productive programmers, lower costs for customers, greater profits for providers, and improved innovation all create favorable conditions for serverless adoption. However, some cloud customers have raised concerns about vendor lock-in, fearing reduced bargaining power when negotiating prices with cloud providers.[16] The serverful VM abstraction is standardized—mostly on account of the Linux operating system and the x86 instruction set—but each provider's serverless cloud functions and BaaS APIs differ in both readily apparent and subtle ways. The resulting switching costs benefit the largest and most established cloud providers, and give them an incentive to promote complex proprietary APIs that are resistant to de facto standardization. Simple and standardized abstractions, perhaps introduced by smaller cloud providers, open source communities, or academics, would remove the most prominent remaining economic hurdle to serverless adoption.

## The Next Phase of Cloud Computing

Perhaps the best way to understand the shift that serverless computing represents is to focus on the first of the essential qualities (as noted previously): providing an abstraction that hides servers and thus simplifies the programming and operating model. From the outset, cloud computing provided a simplified operating model, but simplified programming comes from hiding servers. The future evolution of serverless computing, and in our view of cloud computing, will be guided by efforts to provide abstractions that simplify cloud programming.

It is striking how little cloud computing has changed how programmers work to date, especially when compared to the impact it has had on operators. Much of the software that runs in the cloud is the exact same software that runs in a traditional data center. Compare the programming skills most in demand today against those needed 10 years ago and you will notice that the core skill set has changed very little, even as specific technologies come and go. By contrast, the operator's job has changed tremendously. Installing and maintaining servers, storage, and networks are largely things of the past, replaced by a focus on managing virtualized infrastructure through cloud provider APIs, and by the DevOps movement, which emphasizes the technical and organizational aspects of change management.

What makes programming the cloud hard? While it is possible to use the cloud with just one server, this offers neither fault tolerance nor scalability nor pay-as-you-go, so most cloud programming quickly becomes distributed systems programming. When writing distributed systems, programmers must reason about the data center's spatial extent, its various partial failure modes, and all of its security threats. In the language of Fred P. Brooks, these concerns represent "accidental complexity," which arises from the implementation environment and stands in contrast to "essential complexity," which is inherent in the functionality that the application provides.[7] At the time of Brooks's writing, high-level languages were displacing assembly language, freeing programmers from reasoning about complex machine details such as register allocation or data layout in memory. Just as high-level languages hide many details of how a CPU operates, serverless computing hides many details of what it takes to build a reliable, scalable, and secure distributed system.

We next consider alternative approaches to serverless abstraction, including ones that exist today and ones that we imagine. These vie to answer the question, "if not servers, then what?" We group these alternative abstraction approaches into *application-specific* and *general-purpose* categories (see Table 1). Application-specific abstractions solve a particular use case, and several of them exist in products today. General-purpose abstractions must work well in a broad variety of uses and remain a research challenge.

Let us examine an illustrative example from big data processing. Consider a simple query that might arise in an e-commerce setting: computing an average over 10 billion records using weights derived from one million categories. This workload has the potential for a lot of parallelism, so it benefits from the serverless illusion of infinite resources.

We present two application-specific serverless offerings that cater to this example and illustrate how the category affords multiple approaches. One could use the AWS Athena big data query engine, a tool programmed using SQL (Structured Query Language), to execute queries against data in object storage. SQL is particularly well suited to analytics and can express this computation with a single statement. Alternatively, one could use a framework such as that which Google Cloud Dataflow provides. Doing so requires writing a simple MapReduce-style[11] program, for example, using Java or Python, with two func-

tions: one that computes a weighted average for some chunk of data, and another that combines weighted averages for separate chunks into one for their union. The framework takes care of piping data in and out of these functions, as well as autoscaling, reliability, and other distributed systems concerns. In contrast to the SQL-based tool, this abstraction can run arbitrary code, which can make it suitable to a wider range of analytics problems.

General-purpose serverless abstractions that offer a performant solution to our big data example do not yet exist. Cloud functions might appear to provide a solution since they allow users to write arbitrary code, and for some workloads they do,[28] but due to limitations they sometimes perform much worse than alternatives.[13,17] Figure 4 illustrates how network traffic could be much higher if we implement our example using cloud functions, rather than using an application-specific framework such as Cloud Dataflow. With cloud functions, the provider distributes work across various VM instances without regard to the application's communication patterns, which simplifies autoscaling but increases network traffic.

We suggest two paths to enhancing cloud functions so that they work well in a broader range of applications, potentially turning them into general-purpose serverless abstractions. First, we imagine that hints provided by the programmer might indicate how to achieve better performance. Hints might describe application communication patterns (for example, broadcast or all-reduce), or suggest task placement affinity.[25] Such an approach has precedent in compilers (for example, branch prediction, alignment, and prefetching hints).

Second, and more compellingly, we envision inefficiencies being removed by automatic optimization. In our example the cloud provider might promise to infer locality optimizations from observed communication patterns. In some cases, such inferences might also be made statically, based on an analysis of the program. In the single-machine context this has ample precedent in what modern compilers and language runtimes do, and one might think of this form of serverless computing as extending language support to distributed systems.

Figure 5 illustrates the difference between application-specific and general-purpose serverless abstractions. In the general-purpose case the cloud provider exposes a few basic building blocks, for example, an enhanced version of cloud functions and serverless storage of some sort. A variety of application-specific use cases can be built on top of these foundations. With application-specific serverless, cloud providers instead offer a proliferation of BaaS to meet the needs of an ever-greater number of applications.

Today, serverless computing remains entirely of the application-specific variety. Even cloud functions, which can execute arbitrary code, are popular mainly for stateless API serving and event-driven data processing.[27] We expect application-specific serverless computing to grow, but we are most excited about the potential emergence of general-purpose serverless abstractions, which could host software ecosystems catering to every need. In our view, only the general-purpose approach can ultimately displace servers to become the default form of cloud programming. However, general-pur-

**Figure 4. Increased communication for aggregation and broadcast patterns.**

Application-specific serverless frameworks (for example, Cloud Dataflow) can be implemented with serverful communication patterns. In this case (a) the fewer arrows indicate less network communication than in (b) the general-purpose serverless option. By packing $K$ tasks per VM instance, an application-specific serverless solution, like a serverful solution, is able to achieve a communication complexity of $O(N/K)$ for a job with $N$ tasks, as opposed to $O(N)$ for the cloud function based alternative which can not influence task placement. Typical values for $K$ range from 10 to 100, leading to an overall difference of one to two orders of magnitude.

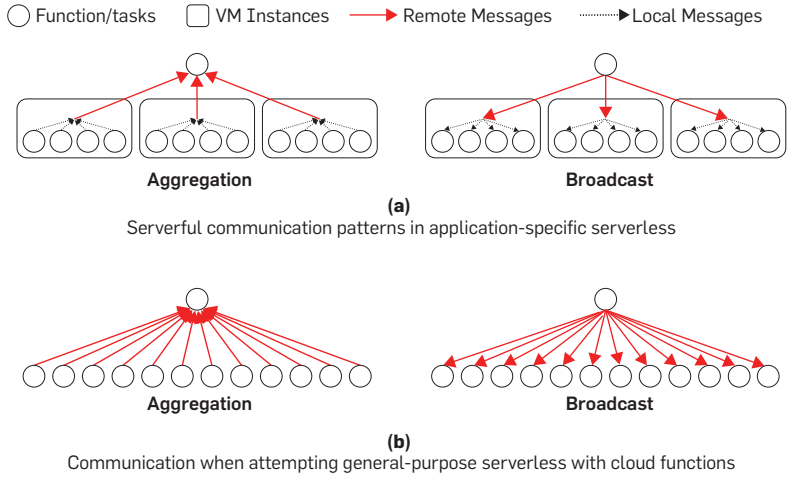
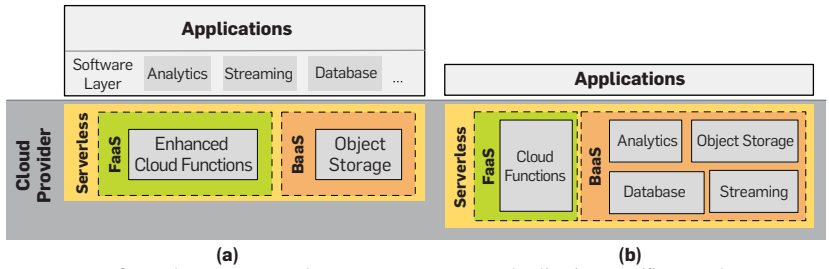



**Figure 5. Potential future directions for serverless.**

(a) General-purpose serverless abstractions support a wide range of needs, with application-specific functionality provided by software above the cloud provider interface, (b) application-specific serverless abstractions with many BaaS point-solutions.



pose serverless technology does not exist today, and developing it presents research challenges.

### Research Challenges

Serverless computing is evolving rapidly and offers various research challenges, many of them common to both application-specific and general-purpose serverless.

**State management.** Distributed cloud applications often need to exchange short-lived or ephemeral state between their component tasks. Examples include application-wide caches, indexes, and other lookup tables, or intermediate results of big data analytics. Cloud functions today allow applications to store ephemeral state locally at each function, which is useful for caching and as working memory for the program. Serverless shared state may be saved in object storage or key-value stores, but these do not simultaneously provide low latency, low cost, high throughput, and fine-grained access, as is possible with servers.[17] Approaches to addressing these challenges include temporary data storage for analytics[21] as well as stateful cloud functions that integrate caching and provide consistency guarantees.[29]

**Networking.** Cloud functions transfer the responsibility of scheduling work from the user to the cloud provider, which has several interesting consequences. Since users cede control over when functions run, passing state between cloud functions requires a trip through shared storage; direct network communication makes little sense and cloud providers block it. Accessing shared storage adds significant latency, sometimes hundreds of milliseconds. Users also cede control over where functions run, thus precluding optimizations common with servers, including sharing common inputs between tasks and combining outputs before sending them over the network (see Figure 4 and previous discussion). Attempts to overcome these challenges will highlight the tension between giving programmers more control and allowing the cloud provider to make optimizations automatically.

**Predictable performance.** Both FaaS and BaaS can exhibit variable performance which precludes their use in applications that must meet strict guarantees. Part of the reason for this is fundamental: serverless providers rely on statistical multiplexing to create the illusion of infinite resources, while denying users control over resource oversubscription. There is always some chance that unfortunate timing will create queuing delays. There is also a latency cost to reassigning resources from one customer to another, which in the cloud function context is known as a "cold start." Cold start latency has several components,[17] and significant among them is the time it takes to initialize the software environment of the function. There has already been progress in this area. Cloud function environments such as Google gVisor and AWS Firecracker[1] can now start in about 100 ms, whereas traditional VMs take tens of seconds to boot. It is also possible to accelerate application-level initialization such as loading libraries.[26] There is probably still much room for improvement in these areas, though there is also evidence that performance optimization and isolation for security are fundamentally at odds.[24] Customers of AWS Lambda can also avoid cold start latencies by purchasing "provisioned concurrency," which controversially reintroduces a form of resource reservation to the serverless model. We hope to also see pricing based on statistical guarantees, or Service Level Objectives (SLOs), which are absent in serverless today.

**Security.** Serverless computing leads to fine-grained resource sharing and so increases the exposure to *side-channel attacks*, whereby attackers exploit subtle behaviors of real hardware that differ from either specifications or programmer assumptions (see sidebar "Serverless and Security"). Threats range from Rowhammer attacks on DRAM[20] to those exploiting microarchitectural vulnerabilities.[22] In addition to adopting mitigations developed for serverful computing, serverless might employ randomized scheduling to make it more difficult for an attacker to target a specific victim. Serverless computing also can incur greater information leakage through network communication because of the fine-grained decomposition of an application and physical distribution of its pieces. An attacker observing the size and timing of network traffic, even if it is encrypted, might make inferences about private data. Addressing these risks may be possible through oblivious computing.[12]

**Programming languages.** Simplified distributed systems programming is a core benefit of serverless computing,[18] and while much previous work in this area is relevant, the serverless setting calls for a new perspective and adds urgency. Traditional challenges include fault tolerance, consistency, concurrency, and the performance and efficiency that comes from locality. New challenges include first-class support for autoscaling, pay-as-you-go, and fine-grained multiplexing.

Fault tolerance concerns are elevated by attempts to extend serverless computing beyond stateless cloud functions. Azure Durable Functions uses C# language features to provide transparent checkpointing, which makes it easier to write stateful and resumable serverless tasks. Microsoft Orleans,[5] which implements an *actor model*,[15] similarly hides fault tolerance concerns from programmers. Actors

## Serverless and Security

**Today, serverless computing merely shifts some security responsibilities from the cloud customer to the cloud provider, just as it shifts other system administration responsibilities. With cloud functions, security updates to operating systems, language runtimes, and standard software packages are applied without customer involvement, usually quickly and reliably. For BaaS services, the cloud provider assumes responsibility for securing everything behind an API. This path may prove to be an important advantage because it allows developers to reason about security at a higher abstraction level. They do not need to implement lower-level security mechanisms, which could lead to fewer security mistakes. While this benefit must be weighed against the exposure to attacks through shared hardware, we believe that improved abstractions may eventually make application security easier to achieve with serverless computing.**

also provide a notion of locality, and could be a counterpart to cloud functions for stateful serverless computing. Ray[25] embodies elements of both. Approaches to consistency include language-integrated transactions, pioneered by Argus.[23] However, transactions are fraught with performance and scalability challenges, which an autoscaling serverless environment may exacerbate. An alternative approach lies in languages like Bloom,[3] which allows automated analysis to determine which parts of a program can run independently, without coordination, and thus scalably. Pay-as-you-go should encourage language developers to rethink resource management, for example, automated garbage collection might be adapted to metered memory pricing. Language approaches to cloud programming,[9] which address the complexity of distributed systems programming head-on, may represent the most direct and ambitious approach to simplifying cloud programming.

**Machine learning.** We believe that automatic optimization with machine learning will play an important role in all of the areas discussed above. It may help decide where to run code, where to keep state, when to start up a new execution environment, and how to keep utilization high and costs low while meeting performance objectives. It may also aid in identifying malicious activity that threatens security, or in automatically cutting up large programs into pieces that can execute in separate cloud functions. Machine learning can help optimize serverful computing too,[10] but serverless abstractions give cloud providers more control over the relevant knobs, as well as the visibility across many customers required to train robust and effective models.

**Hardware.** Current trends in hardware may be complementary to serverless computing. The x86 microprocessors that dominate the cloud are barely improving in performance; in 2017, program latency improved only 3%,[14] a trend that if continued implies that performance won't double for 20 years. Similarly, the ending of Moore's Law is slowing the growth of per-chip DRAM capacity. The industry response has been the introduction of Domain Specific Architectures (DSAs), which are tailored to a specific type of problem, offering significant performance and efficiency gains, but performing poorly for other applications.[14] GPUs have long been used to accelerate graphics, and we are starting to see DSAs for ML such as TPUs. GPUs and TPUs can outperform CPUs for narrow tasks by factors of 30x.[19] These examples are the first of many, as general-purpose processors enhanced with DSAs will likely become the norm.

**It is striking how little cloud computing has changed how programmers work to date, especially when compared to the impact it has had on operators.**

We believe serverless computing may provide a useful programming model for integrating diverse architectures, say with separate cloud functions running on separate accelerators. It also helps create room for innovation by raising the level of abstraction, for example, by allowing a cloud provider to substitute a DSA for a CPU when recognizing a workload that could benefit.

### Why Serverless Computing Might Still Fail

While we believe serverless computing can grow to become the cloud programming default, we can also imagine several scenarios in which serverful computing retains its dominance. First, serverful computing is a moving target, one that improves relentlessly, if slowly. Cloud VMs that once were billed by the hour now have a minimum billing increment of one minute, and charge by the second thereafter. Container and VM orchestration tools (for example, Kubernetes, Terraform) help streamline complex deployments, and increasingly automate administrative tasks such as taking backups. Programmers can rely on mature software ecosystems and strong legacy compatibility when building applications, while companies already have teams skilled in serverful cloud deployments. Server hardware also keeps getting bigger and more powerful, bringing CPU, memory, and accelerator power together in a closely coupled environment, a benefit for some applications.

Second, today's successful serverless products fall into the application-specific category and are narrowly targeted, whereas general-purpose serverless abstractions have a better chance of displacing serverful comput-

ing (which is also general-purpose). However general-purpose serverless computing faces hurdles: the technology that we envision does not exist yet, and it may be a less lucrative business for cloud providers.

Finally, even if our vision plays out, the brand of "serverless computing" might not survive. The temptation to label older products as the next new thing is strong and can create confusion in the marketplace. We have been happy to see products such as Google App Engine pick up the serverless moniker, and along with it features such as scaling to zero. However, if the term becomes diluted by half-hearted efforts, then perhaps general-purpose serverless computing will emerge under another name.

## Conclusion and Predictions

Cloud computing is both flourishing and evolving. It has overcome the challenges that faced it in 2010, as we projected.[4] Offering lower costs and simplified system administration, the business is growing up to 50% annually and proving highly profitable for cloud providers. Cloud computing is now entering a second phase in which its continued growth will be driven by a new value proposition: simplified cloud programming.

Analogous to how hailing a taxi simplifies transportation over renting a car (see Figure 1), serverless computing relieves programmers from thinking about servers and everything complicated that goes along with them. Following the same naming convention, you could classify a taxi service as *carless transportation* in that the passenger need not know how to operate a car to get a ride. Serverless raises the level of abstraction of the cloud, adopts pay-as-you-go pricing, and rapidly autoscales down to zero and up to practically infinite resources.

Serverless computing is still evolving, and many open questions remain, both in defining its abstractions and in implementing them. We (boldly) conclude this paper with five predictions for serverless computing in the next decade:

1. Today's FaaS and BaaS categories will give way to a broader range of abstractions, which we categorize as either *general-purpose serverless computing* or *application-specific serverless computing*. While serverful cloud computing won't disappear, its relative use in the cloud will decline as serverless computing overcomes its current limitations.

2. We expect new general-purpose serverless abstractions to support just about any use case. They will support state management, as well as optimizations—either user-suggested or automatically inferred—to achieve efficiencies comparable or maybe better than those of serverful computing.

3. We see no fundamental reason for the cost of serverless computing to exceed that of serverful computing. We predict that as severless evolves and increases in popularity almost any application, be it tiny or massive-scale, costs no more—and perhaps a lot less—with serverless computing

4. Machine learning will play a critical role in serverless implementations, allowing cloud providers to optimize execution of large-scale distributed systems while providing a simple programming interface.

5. Computer hardware for serverless computing will be much more heterogeneous than the conventional x86 servers that powers it today.

If these predictions hold, serverless computing will become the default computing paradigm of the Cloud Era, largely replacing serverful computing and thereby closing the Client-Server Era, just as the smartphone brought the end of the PC Era.

**Acknowledgments.** We thank the reviewers for their thoughtful comments, as well as the many friends who gave feedback on early drafts. This work was conducted at UC Berkeley RISELab and it was supported by a National Science Foundation Expedition Project, Alibaba Group, Amazon Web Services, Ant Financial, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Scotiabank, Splunk, and VMware.

**References**

1. Agache, A., et al. Firecracker: Lightweight virtualization for serverless applications. In *Proceedings of the 17th USENIX Sym. Networked Systems Design and Implementation* (2020), 419–434.
2. Alcott, B. Jevons' paradox. *Ecological Economics 54*, 1 (2005), 9–21.
3. Alvaro, P., et al. Consistency analysis in Bloom: A CALM and collected approach. *CIDR*, 249–260.
4. Armbrust, M., et al. A view of cloud computing. Commun. ACM 53, 4 (Apr. 2010) 50–58.
5. Bernstein, P., et al. Orleans: Distributed virtual actors for programmability and scalability. MSR-TR-2014–41, 2014.
6. Brazeal, F. The business case for serverless, 2018; https://www.trek10.com/blog/business-case-for-serverless
7. Brooks, F. No silver bullet: essence and accidents of software engineering. In *Information Processing*. IEEE, 1986.
8. Castro, P., et al. The rise of serverless computing. *Commun. ACM 66*, 12 (Dec. 2019), 44–54.
9. Cheung, A., Crooks, N., Milano, M., and Hellerstein, J. New directions in cloud programming. *CIDR*, 2021.
10. Dean, J. Machine learning for systems and systems for machine learning. In *Proceedings of the 2017 Conf. Neural Info. Processing System*.
11. Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.
12. Goldreich, O. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, (1987) 182–194.
13. Hellerstein, J., et al. Serverless computing: One step forward, two steps back. *CIDR*, 2019.
14. Hennessy, J. and Patterson, D. A new golden age for computer architecture. *Commun. ACM 62*, 2 (Feb. 2019), 48–60.
15. Hewitt, C., Bishop, P., and Steiger, R. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd Intern. JoinConf. Artificial Intelligence*. (1973), 235–245. Morgan Kaufmann Publishers Inc.
16. Irwin, D. and Urgaonkar, B. Research Challenges at the Intersection of Cloud Computing and Economics. National Science Foundation, 2018.
17. Jonas, E. et al. Cloud programming simplified: A Berkeley view on serverless computing. Tech. Rep. No. UCB/EECS-2019-3, 2019.
18. Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., and Recht, B. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the ACM SoCC*, 2017.
19. Jouppi, N. et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual Intern. Symp. Computer Architecture*. (2017), 1–12.
20. Kim, Y., et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceeding of the 42nd ISCA*. IEEE Press, 2014, 361–372.
21. Klimovic, A., et al. Pocket: Elastic ephemeral storage for serverless analytics. In *Proceedings of the 13th USENIX Symp. Operating Systems Design and Implementation* (2018), 427–444.
22. Kocher, P., et al. Spectre attacks: Exploiting speculative execution. *Commun. ACM 63*, 7 (July 2020), 93–101.
23. Liskov, B. Distributed programming in Argus. *Commun. ACM 31*, 3 (Mar. 1988), 300–312.
24. McIlroy, R., Sevcik, J., Tebbi, T., Titzer, B., and Verwaest, T. Spectre is here to stay: An analysis of side-channels and speculative execution. 2019; arXiv:1902.05178.
25. Moritz, P., et al. Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX Symp. Operating Systems Design and Implementation* (2018), 561–577.
26. Oakes, E., et al. SOCK: Rapid task provisioning with serverless-optimized containers. In *2018 USENIX Annual Technical Conf.* (2018), 57–70.
27. Passwater, A. 2018 serverless community survey: Huge growth in serverless usage; https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage/
28. Perron, M., Fernandez, R., DeWitt, D., and Madden, S. Starling: A scalable query engine on cloud functions. In *Proceedings of the 2020 ACM SIGMOD Intern. Conf. Management of Data* (2020), 131–141.
29. Sreekanti, V., et al. Cloudburst: Stateful functions-as-a-service. *Proc. VLDB 13*, 11 (2020), 2438–2452.
30. Wagner, T. Debunking serverless myths, 2018; https://www.slideshare.net/TimWagner/serverlessconf-2018-keynote-debunking-serverless-myths

**Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson.**

The authors are associated with the UC Berkeley RISELab (Real-Time Intelligence Secure Explainable Systems Lab).