Real-time Serverless: Enabling Application Performance Guarantees

Hai Duc Nguyen, Chaojie Zhang, Zhujun Xiao, and Andrew A. Chien*
University of Chicago, *Argonne National Lab
{ndhai,chaojie,zhujunxiao,achien}@cs.uchicago.edu

Abstract

Today's serverless provides "function-as-a-service" with dynamic scaling and fine-grained resource charging, enabling new cloud applications. Serverless functions are invoked as a best-effort service. We propose an extension to serverless, called *real-time serverless* that provides an invocation rate guarantee, a service-level objective (SLO) specified by the application, and delivered by the underlying implementation. Real-time serverless allows applications to guarantee real-time performance.

We study real-time serverless behavior analytically and empirically to characterize its ability to support bursty, real-time cloud and edge applications efficiently. Finally, we use a case study, traffic monitoring, to illustrate the use and benefits of real-time serverless, on our prototype implementation.

CCS Concepts • **Computer systems → Cloud**;

Keywords Serverless, Real-time, Bursty, Interface

ACM Reference Format:
Hai Duc Nguyen, Chaojie Zhang, Zhujun Xiao, and Andrew A. Chien*. 2019. Real-time Serverless: Enabling Application Performance Guarantees . In *5th Workshop on Serverless Computing (WOSC '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3366623.3368133

1 Introduction

Serverless has seen huge growth in usage [18] and received much attention from the research community [3, 7]. The core technology behind serverless computation is *cloud functions* written to perform specific tasks when some conditions are met. With serverless, users implement functions and associate them with events. No server deployment and management are needed. Furthermore, these functions scale to hundreds of simultaneous invocations. Serverless employs millisecond billing so application cost scales with activity. Applications built upon serverless can be developed and deployed quickly and support a wide range of load. Some predict serverless will become a dominant cloud service [8].

Serverless' dynamic scaling is extremely useful for "bursty", low-duty factor applications such as music recognition, information access, intelligent assistants, internet-of-things sensor data, or any of a wide range of smartphone applications. In such applications, demand bursts arise from external world activity with short-lived computation requirements.

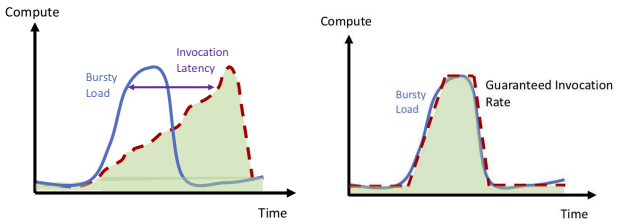
Some bursty applications need more than dynamic scaling; they have real-time requirements of guaranteed response latency and application quality. We term these applications *bursty, real-time*, and they include an important class of cloud and edge computing applications. Specific examples include data network monitoring, augmented reality, video monitoring, and public safety applications such as traffic monitoring. In such applications, load changes are driven by external events (e.g. cyber-attacks, car accidents), giving rise to burstiness. Such applications may have a hard deadline for data analysis to enable subsequent actions. Failure to meet a deadline may be unacceptable (death, system failure, ...) or make the results less valuable (can't block DoS attack, missed the criminal's car, ...) regardless of its quality. Current serverless system provide no means for application to guarantee latency, and thus meet real-time constraint. Serverless invocations can fail or be delayed arbitrarily, any applications guarantees about decision (or computation) quality.

We propose an extension of the serverless interface to increase its range of applications to include those with bursty, real-time requirements. We call this extension *real-time serverless.* It adds both an service-level objective (SLO) for function invocation rate to the interface, and delivers this SLO. We show how applications can use real-time serverless to guarantee real-time performance, including latency and quality. We use an analytical model to study benefits, and finally explore a case study implementing a traffic monitoring application on real-time serverless. Specific contributions of the paper include:

- Definition of the *real-time serverless* interface and guaranteed invocation rate
- Modeling studies that reveal real-time serverless benefits for real-time bursty application
- A traffic monitoring case study that shows how real-time serverless can be applied

WOSC '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7038-7/19/12…$15.00
https://doi.org/10.1145/3366623.3368133



(a) Serverless (best-effort) **(b)** Real-time Serverless

Figure 1. Best-effort and Guaranteed Invocation rate

- A prototype implementation that is being deployed to support a number of new applications (video monitoring, real-time instrument control)

Paper organization: Section 2 outlines real-time serverless interface, explaining how applications can achieve guaranteed performance. In Section 3, we evaluate benefits quantitatively and demonstrate on a traffic monitoring. We describe our initial implementation efforts in Section 5, and close with related work (Section 6) and summary (Section 7).

2 Real-time Serverless

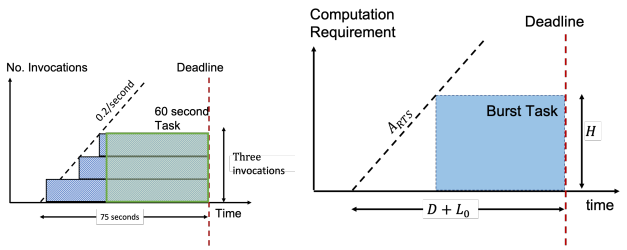
Serverless computing cannot guarantee performance for bursty, real-time workloads. We describe the reasons for this, and then propose the real-time serverless interface providing a guaranteed invocation rate. Finally we describe how it can be used to deliver real-time and quality guarantees.

Consider a real-time bursty application (Figure 1a (blue line)). Its computational requirements to meet its real-time or quality requirements vary with time. With serverless computing, as defined by all of the major cloud vendors [2, 9, 15], the application's ability to acquire compute resources depends on function invocation latencies (each invocation can viewed as an allocation of compute resources). If the invocation latencies lag the application requirement, then real-time and quality requirements (SLO) will not be achieved.

We propose *real-time serverless*, an interface for applications to advertise a guaranteed invocation rate, and a service-level objective (SLO) to deliver that rate. As Figure 1b (right), with a guaranteed invocation the real-time requirement can be met. Each real-time serverless function is associated with the following attributes (YAML format):

```
<function-name>
  lang:     <Language of function body>
  handler:  <Location of function body>
  image:    <Docker image reference>
  realtime: <Guaranteed invocation rate>
  timeout:  <Runtime limit>
  limits:   <Max resource use>
  requests: <Min resource use>
```

Most of the attributes are compatible to the serverless interface. We add `realtime` that allows users to specify the required invocation rate (in invocations/second). If a function



(a) Real-time Example **(b)** General Deadlines

Figure 2. Real-time application tasks mapped onto RT Serverless

declares realtime = A_{RTS}, the number of invocations per period, *Invokes*, must satisfy:

$$Invokes \geq \min[A_{RTS}, Requests] \quad (1)$$

where *Requests* is the number of invocation requests in the period. For example, if an IoT service for a sensor with a data reporting interval of 100ms wanted to ensure no data was lost, and all data was processed in a timely fashion, it might add "realtime: 10". A correct real-time serverless implementation meets all of guaranteed invocation rates. A guaranteed invocation rate of zero corresponds to commercial serverless offerings.

2.1 Using Real-time Serverless

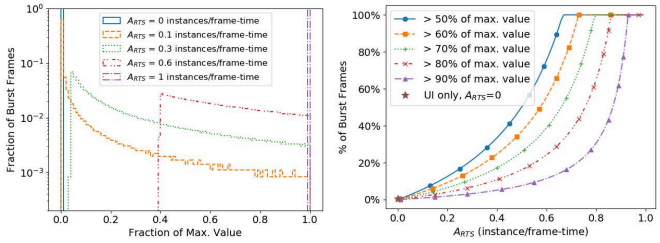
Let's begin with an example. Consider a real-time task with a burst compute requirement of three function invocations for 60 seconds as shown in Figure 2a. The task has a deadline of 75 seconds after release (the triggering event). A guaranteed rate of 0.2 invocations/second ensures that three invocations will be available within 15 seconds to provide the computation required for the 60-second execution to meet the real-time deadline.

An application can plan its resource requests to meet requirements based on the size and duration of its tasks and their deadlines because a guaranteed invocation rate ensures access to resources growing with slope A_{RTS}. Thus, real-time serverless enables applications to acquire resources as a *deterministic* and *linear* function of time. Higher compute demands can be met with proportionally increased A_{RTS}.

Management for real-time tasks in the general case is shown in Figure 2b; each task with maximum compute requirement, H, that lasts, D seconds and a latency requirement of $D + L_0$. A guaranteed invocation rate as below will meets the latency requirement.

$$A_{RTS} = \frac{H}{L_0} \quad (2)$$

Increased invocation rate can meet tighter latency bounds, and serve applications with quality requirements (e.g. soft real time). Tuning is easy – guaranteed invocation rate can simply be increased until real-time guarantees are met.



(a) Frame value distribution vs. A_{RTS}

(b) Fraction of burst frames achieving a specific fraction of maximum value

Figure 3. Guaranteed frame value at various guaranteed invocation rates (D=3,600 and τ=2,607 (1/2 per minute))

3 Bursty Application on RT Serverless

We construct a model of a real-time video analytics application. This application performs a low-cost screening of frames to detect potentially interesting events. When such events occur, the application triggers a deeper analysis – a "burst" of video frames – analyzed with a higher compute requirement, and where timely results are critical.

3.1 Modeling

We construct an analytical model for the traffic monitoring application running on real-time serverless. Each burst increases compute demand 10-100x for duration D frame-times (framerate = 1/30 second). Bursts arrive as a Poisson process with rate λ.

The video application is a serverless function that processes a single burst frame. It is deployed on real-time serverless with guaranteed invocation rate A_{RTS}. For each burst frame, an invocation is triggered in FIFO order, returning an instance used for invocation until burst frame processing finishes. Each frame can be thought of as a real-time task, and thus if A_{RTS} matches the framerate, then all frames are processed immediately (in effect, they meet a deadline with no slack). For simplicity, we use the interframe interval as the time unit (frame-time).

Many video analysis applications are not hard real-time, but rather have a softer notion deadline where delayed results are still of some value. We can model such a property with a value function that decays with increased delay (or slack in a real-time terminology), as a function of delay $L(i)$ for frame i in a burst. We use a simple decaying value function as an exemplar:

$$V_{frame}(i) = V_{max} e^{-\frac{L(i)}{\tau}} \tag{3}$$

where $V_{max} = 1$ is the maximum frame value and τ is time constant of value decay with latency[1]. $L(i)$ is the frame's waiting time before processing. For burst, the overall value is

$$BurstValue = \sum_{i=0}^{D-1} V_{frame}(i) \tag{4}$$

The FIFO model implies that for a burst at $t = 0$, the frame arriving at $t = i$'s invocation will begin at latest at $\frac{i}{A_{RTS}}$ as

$$L(i) = i(\frac{1}{A_{RTS}} - 1) \tag{5}$$

Current serverless offerings (no SLO) can be modeled with $A_{RTS} = 0$, which gives a worst case is a value of zero.

3.2 Designing for Application Quality

To illustrate how real-time serverless enables designing applications with guaranteed quality, we use the above analysis, consider a single burst of duration D, vary A_{RTS}, and plot the distribution of frame values in Figure 3a.

With $A_{RTS} = 1$ the maximum value is achieved for all frames – invocations are granted fast enough to keep up with the burst. As A_{RTS} decreases, the distribution shifts to lower value (left), with all frames achieving at least 40% of the maximum value with $A_{RTS} = 0.6$, and decaying as decreases to $A_{RTS} = 0.1$. At $A_{RTS} = 0$ (no invocation rate guarantee), provided by current commercial serverless systems, invocations can be held up for long periods of time, producing low application value guarantee (blue, distribution at left).

By choosing a guaranteed invocation rate, an application can ensure a guaranteed application quality for each frame, as shown in Figure 3b. At a given A_{RTS}, we show the fraction of burst frames achieving a particular fraction of maximum value. To achieve 50% of max value for even 50% of frames requires A_{RTS} of 0.5 instances/frame-time. At the high end, achieving 90% of max value 50% of the time requires A_{RTS} = 0.85, and 0.9 for 100% of the frames. With proper choice of A_{RTS}, applications are able to meet *any* target quality, unlocking simple, rational design for quality.

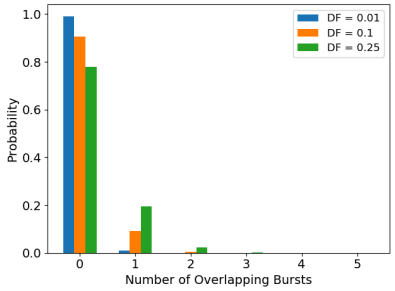


Figure 4. Prob. of # of Overlapping Bursts (vs Duty Factor).

3.3 Overlapping Bursts

For higher burst rates overlapping bursts can present new challenges. For example, a "person of interest" appears in one camera while another camera views a traffic accident.

[1] As value function is used to represent application quality, we use the term value and quality interchangeably.

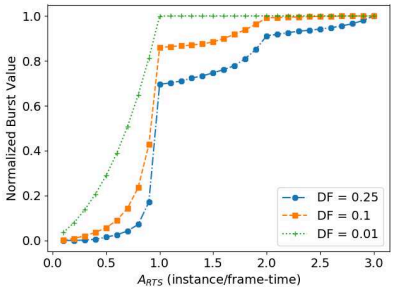


Figure 5. Guaranteed Invocation Rate to achieve Burst Value Fraction (varied Duty Factor).

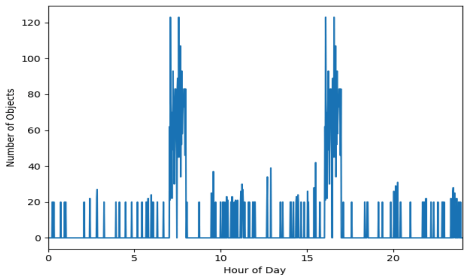


Figure 6. Traffic Monitoring trace (objects of interest): two rush hour periods, and higher daytime load.

We call such cases *burst interference*, and study how they can impact quality.[2]

Burst interference reduces achieved application quality. To maintain quality, bursty, real-time applications must increase their requested A_{RTS}. The duration (D) and arrival rate (λ) drive this increase, so we vary duty factor ($DF = D \cdot \lambda$) to explore increases in required guaranteed invocation rate (see Figure 4). For many monitoring systems a duty factor of 1% may be typical, probability of interference is extremely low (0.005% for two and 0.00002% for three bursts). Even with duty factors 10-25 times higher, the probability of > 2 simultaneous bursts is very low.

In Figure 5, we plot the value achieved from for various A_{RTS}. Increasing guarantee invocation rate can provide effective service for multiple concurrent bursts. At $A_{RTS} = 2$, two concurrent bursts can be handled with full value. And, at $A_{RTS} = 3$ invocations per frame-time is sufficient to guarantees maximum value for the three-burst cases. So, maintaining quality (100% of burst value) for a 25x increase in duty factor requires only a 3-fold increase in a guarantee A_{RTS}. So, even for multiple concurrent bursts, an application is still able to achieve desired quality by simply adjusting A_{RTS}; and, it appears due to statistical multiplexing, high value can be maintained at a low-cost!

4 A Traffic Monitoring Example

We study a real-world bursty application, real-time video traffic monitoring. The application does video counting and tracking, processing real traffic videos [19] using a Glimpse-like [4] pipeline. Each vehicle entering initiates a burst of analysis, producing characteristics captured in Table 1.

	Burst Duration (frames)		Burst Demand per Frame	
	Mean, StDev	Min–Max	Mean, StDev	Min–Max
Night	116, 186	30–2,445	21, 3	20–80
Day	120, 216	30–2,323	20, 3	20–80
Rush hours	917, 1293	30–7,464	48, 23	20–200
Overall	197, 503	30–7,464	24, 11	20–200

Table 1. Burst Statistics for Traffic Monitoring Example

Trace activity varies with time of day and other traffic fluctuation. Our 24-hour trace has 3 distinct periods: rush hour (high burst demand), daytime (medium burst demand), and nighttime (low burst demand) as illustrated in Figure 6.

We rerun experiments in Section 3.2 using simulation on the trace and plot the results in Figure 3. Clearly, due to the complicated statistics properties, the distribution of frame value in Figure 7a is much noisier than the results we get in Figure 3a. However, the effect of adding the guaranteed allocation rate still remains. At $A_{RTS} > 0$, the application can ensure many frames to achieve high value. Guaranteed value also increases as A_{RTS}, although comparing to the analytical data, it is much slower due to extreme high computation demand during rush hours. Figure 7b reveals application value is a monotonically increasing and continuous function of guaranteed allocation rate. Thus, the traffic monitor can use the guaranteed allocation rate as a tuning parameter to meet desired object counting quality. This confirms real-time serverless' capacity of providing a mean for quality design as we introduce in Section 3.2

5 Implementing Real-time Serverless

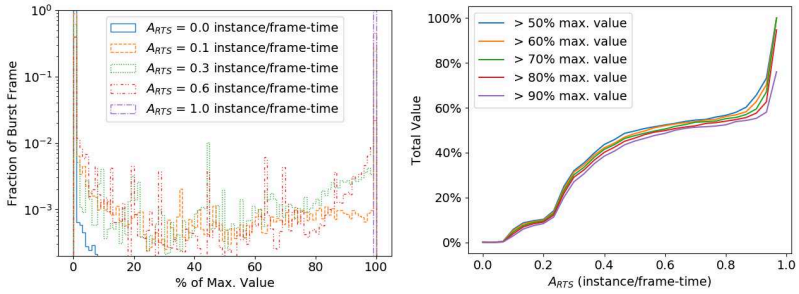
5.1 Worst-case Resource Approach

A strong invocation rate guarantee has significant implications for resource management. The fact that serverless invocations have bounded, short durations means that serverless invocations can always be reclaimed predictably. Specifically, from the real-time serverless interface, a function can be characterized by a tuple (realtime, timeout) = (A_{RTS}, R_{RTS}) indicating its guaranteed allocation rate A_{RTS} and invocation maximum runtime R_{RTS}. All instances allocated for function invocation at the time t can be reused no later than $t + R_{RTS}$. Given application an A_{RTS} guarantee, we can bound the resource requirement as

$$C_{A_{RTS},R_{RTS}} = A_{RTS} \cdot R_{RTS} \tag{6}$$

Thus, the maximum resource cost for a real-time serverless function grows linearly with allocation rate and invocation duration. This confirms basic feasibility of real-time serverless implementation. Further efficiency gains are possible by exploiting statistical workload characterizations to reduce resource requirements [11, 21, 23]. We study how to do this for real-time serverless extensively in [16].

[2]Bursts are independent, identically distributed Poisson arrivals for convenience. We leave more complicated cases for future work.



(a) Frame value distribution vs. A_{RTS} **(b)** Fraction of burst frames achieving a specific fraction of maximum value (FIFO)

Figure 7. Traffic Monitoring application Value on RT Serverless (simulation)

5.2 Real-time Serverless Prototype

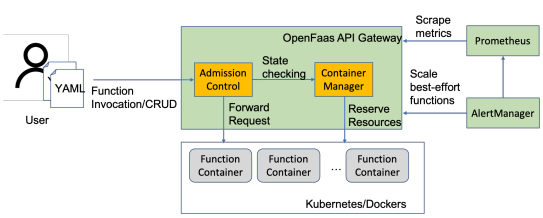


Figure 8. Real-time Serverless Implementation. Modules (orange) are added to OpenFaaS.

We have built a real-time serverless prototype as an extension of OpenFaaS, an open-source implementation of serverless platform [17]. The primary extensions are *admission control* and *predictive container management*. The admission control unit accepts or rejects the deployment of functions with guaranteed invocation rate, based on the system state and requested guarantees. The predictive container manager provisions containers to meet the guarantees. The system is depicted in Figure 8.

To demonstrate the prototype, we ran the Traffic Monitoring application discussed in Section 3. Each video frame is processed by a single cloud function invocation. The Traffic Monitoring application is run with different invocation rate guarantees in the real-time serverless prototype, and report achieved throughput and application value (see Figure 9). To demonstrate performance isolation, a competitive background best-effort cloud function workload large enough to fill the system is running. The prototype achieves the guaranteed invocation rate in all cases. Experiments were run on the UChicago RIVER system, which includes 2x Intel Xeon Gold 6138 20-core processors (80 threads total), 2.0GHz, 37.5MB cache, 512GB DRAM and a complement of IO devices.

Figures 9a,9b, and 9c show how the function request and invocation rates vary as the guaranteed invocation rate is increased. With no guaranteed rate ($A_{RTS} = 0$), the system fails to provide enough invocations for the application to catch up with computation bursts, especially during rush hour periods. At ($A_{RTS} = 0.3$), shown in Figure 9b, the situation improves, and finally in Figure 9c, with $A_{RTS} = 1$, the SLO of 1 invocation per frame-time, application can adapt to burst demand (the invocation and request curves are nearly identical), enabling to higher application quality.

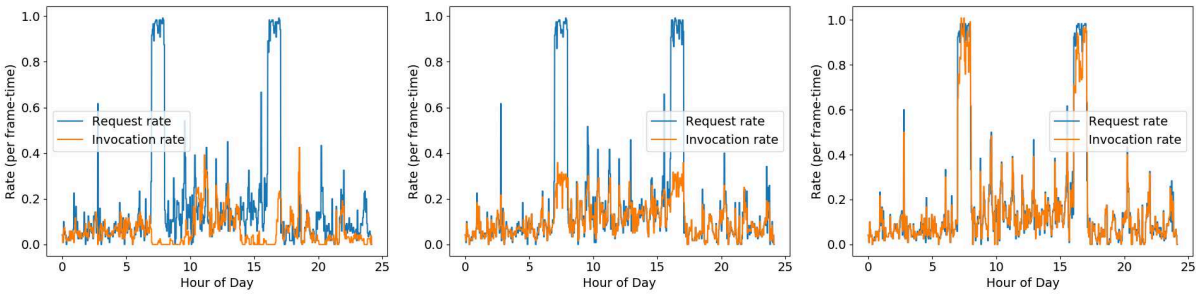
Figure 10, shows how our real-time serverless prototypes delivers on the promise of manageable quality. Despite a competitive background load, the prototype allows application to increase their quality by increasing A_{RTS}.

Our prototype exposed one interesting challenge - dealing with ad hoc invocation request loss. Requests can be dropped when various resources and wait timeouts (reflected in difference between Figure 10 and 7b). Such discarding a problem for real-time serverless, producing an ad hoc priority and scheduling outcome that can violate application guarantees. We are solving these problems in future implementations

6 Related Work

Serverless is growing rapidly with numerous commercial offerings [2, 9]. Many new applications explore how to use serverless [5, 7, 24]. There are also many Open-source projects such as Knative [14] and OpenFaaS [17]. There are efforts to reduce FaaS overhead [1, 6, 22], but generally such efforts increase variability of performance and we know of none that provide SLO's for invocation latency or rate. The existing systems all deliver invocations on a best-effort basis with significant documented variability [20], and cannot provide application real-time guarantees.

Numerous efforts to extend the serverless model focus on how to support traditional 3-tier elements that enable business logic (distributed computing or systems) and databases [10, 13]. These efforts propose richer naming and communication services for invocations and efficient data access but no performance guarantees. Other efforts seek to add more general, flexible programming to cloud functions [12] These important directions are distinct from our goal to enable guarantees of performance for bursty, real-time applications built on function-as-a-service, benefiting from the benefits of rapid dynamic scaling and fine-grained resource billing.



(a) A_{RTS} = 0 invocation/frame-time **(b)** A_{RTS} = 0.3 invocation/frame-time **(c)** A_{RTS} = 1 invocation/frame-time

Figure 9. Real-time serverless prototype with Traffic Monitoring application

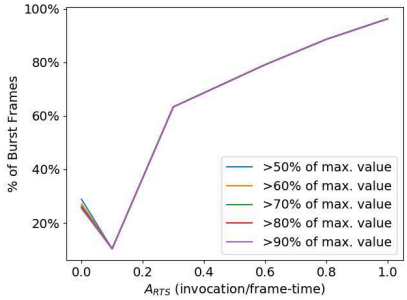


Figure 10. Frame value distribution vs. A_{RTS} on OpenFaas RTS Prototype

7 Summary and Future Work

The serverless computing model has enabled a broad range of new applications and provides new levels of application convenience. However, today's serverless gives best-effort service, making it unable to support bursty, real-time because it cannot guarantee application quality.

We have shown that the addition of guaranteed function invocation rate, producing real-time serverless can solve this problem. Enabling applications to provide quality guarantees, and conveniently tune their quality. Further, our analysis of real-time serverless for a variety of burstiness properties highlights its scalability and promising cost-effectiveness. We have built a prototype system, and are working with leaders on a number of application.

We have shown the power of the real-time serverless model. Open questions remain about the cost of its implementation. What are the best approaches? How do their costs and properties vary as a function of the underlying system and user applications? These interesting questions provide an exciting future research space.

Acknowledgements This work supported by National Science Foundation Grants CNS-1405959, CMMI-1832230, and CNS-1901466. We gratefully acknowledge support from Intel, Google, and Samsung.

References

[1] Istemi Ekin Akkus and et. al. 2018. SAND: Towards High-Performance Serverless Computing. In *USENIX ATC*.
[2] Amazon 2015. AWS Lambda. https://aws.amazon.com/lambda/.
[3] Lixiang Ao and et. al. 2018. Sprocket: A Serverless Video Processing Framework. In *SoCC '18*.
[4] Tiffany Yu-Han Chen and et. al. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *13th Conference on Embedded Networked Sensor Systems*.
[5] Daniel Crankshaw, Gur-Eyal Sela, Corey Zumar, Xiangxi Mo, Joseph Gonzalez, Ion Stoica, and Alexey Tumanov. 2019.
[6] Oakes Edward and et.al. 2018. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *USENIX ATC*.
[7] Sadjad Fouladi and et. al. 2017. Encoding, Fast and Slow: Low-latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX NSDI'17*.
[8] Gartner. 2018. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019. https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019.
[9] Google 2016. Google Cloud Functions. https://cloud.google.com/functions/.
[10] Joseph M. Hellerstein and et. al. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *CIDR*.
[11] Wilkes John. [n. d.]. Keynote: Google Flex. In *JSSPP 2018*.
[12] Eric Jonas and et. al. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*.
[13] Eric Jonas and et.al. 2019. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Technical Report UCB/EECS-2019-3. University of California, Berkeley.
[14] Knative 2018. Knative. https://cloud.google.com/knative/.
[15] Microsoft 2010. Microsoft Azure Cloud. https://azure.microsoft.com/.
[16] Hai Duc Nguyen. 2019. *Cloud Resource Management for Bursty, Real-time Workloads*. Master's thesis.
[17] OpenFaaS 2017. OpenFaaS. https://docs.openfaas.com.
[18] Andrea Passwater. 2018. 2018 Serverless Community Survey: Huge Growth in Serverless Usage. https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage Visited May, 2019.
[19] Twin Forks Pest Control. 2019. Southampton Traffic Cam. https://www.youtube.com/watch?v=rpbkCUbWVio.
[20] Liang Wang and et. al. 2018. Peeking behind the Curtains of Serverless Platforms. In *2018 USENIX ATC*.
[21] Rich Wolski, John Brevik, Ryan Chard, and Kyle Chard. 2017. Probabilistic Guarantees of Execution Duration for Amazon Spot Instances. In *Supercomputing (SC) '17*.
[22] Ki Kim Young and et. al. 2018. Dynamic Control of CPU Usage in a Lambda Platform. In *IEEE CLUSTER 2018*.
[23] Chaojie Zhang, Varun Gupta, and Andrew A. Chien. 2019. Information Models: Creating and Preserving Value in Volatile Cloud Resources. In *IEEE International Conference on Cloud Engineering*. IC2E.
[24] Miao Zhang and et. al. 2019. Video Processing with Serverless Computing: A Measurement Study. In *29th NOSSDAV*.