Real-time Serverless: Cloud Resource Management for Bursty, Real-time Workloads

Hai Duc Nguyen  
University of Chicago  
dndai@cs.uchicago.edu

Chaojie Zhang  
University of Chicago  
chaojie@uchicago.edu

Zhijun Xiao  
University of Chicago  
zhujiunxiao@uchicago.edu

Andrew A. Chien  
University of Chicago and Argonne National Lab  
achien@cs.uchicago.edu

Abstract

Today’s cloud resource offerings provide no guarantees for resource allocation, so bursty application must reserve, and pay for resources they do not use - to achieve real-time guarantees. We propose a new type of cloud resource, Real-time Serverless (RTS) with a new service-level objective – guaranteed allocation rate. This guarantee enables timely resource allocation, enabling applications to achieve real-time performance efficiently. With a simple burst model, we study real-time serverless analytically, exploring their effect on application quality, guarantees, and cost. Next, we simulate statistically varying bursts and higher loads (multi-application), to study the impact of real-time serverless.

In both analytic and simulation studies, adding guaranteed allocation rate enables bursty, real-time applications to achieve guaranteed high quality cost-effectively. Specifically, for a desired application quality, the required allocation rates can be determined. In addition, for duty factors from 0.025 to 0.25, the value of real-time serverless to the application is > 4x than traditional. Further results show that multiple applications can share real-time serverless efficiently, supporting duty factor increases of 25x with only a 1.6x increase in allocation rate (provider resource cost). Finally, we present a case study of a traffic monitoring application. Despite more complex burst statistics, our results show major benefits for cost and application quality. Application benefit makes real-time serverless worth nearly 16x virtual machine resources for delivering application value.

1 Introduction

Cloud computing has grown steadily in the dozen years since Amazon introduced Elastic Compute Cloud (EC2) and AWS in 2007, and is capturing a growing portion of both Internet and enterprise IT computing services. This success has fueled the rapid growth of cloud computing companies in both resources and revenue [11], and shaped the primary resource offerings – virtual machines with hourly accounting, indefinite lifetime, and high reliability [2, 15, 25]. These offerings cannot respond in guaranteed timely fashion to bursty loads, as requests of “on-demand” virtual machines can be delayed arbitrarily, or even rejected.

Growth of the “internet of things” in and consumer devices [26, 28, 29], over 100 million intelligent assistants [4], urban and rural monitoring systems [6, 22], and even expensive large equipment [9, 24] exceeds 20 billion devices, and is driving the creation of new cloud resource models. These IoT devices exhibit bursty computational requirements, requiring many small invocations, and rapid scaleup and scaledown of total computational resources to achieve both responsive performance and acceptable cost.

In response, cloud providers created a new resource model – “cloud functions” (aka Serverless or function-as-a-service – FAAS) that provide a best-effort invocation service [3, 14]. FAAS supports many concurrent invocations with at best effort within strict concurrency limits [3, 10, 19]. Resources are allocated and released per-invocation, with execution as short as a few seconds. In contrast to a virtual machine service, there is no notion of a continuous resource and no indefinite resource commitment; invocation execution is strictly limited (5 minutes, 15 minutes). The accounting model matches; cloud providers charge at fine-grain for resource use (gigabyte-seconds or finer) – so cost scales both down and up based on total use independent of its temporal (when, idle, burst). Consequently, despite much higher cost per unit resource (gigabyte-seconds) compared to virtual machine offerings, the serverless model is cost-effective and popular amongst new applications, and is reputed to be the “fastest growing element of cloud workload”. Serverless offerings cannot respond in guaranteed timely fashion to bursty loads, as startup latencies and throughput rates for serverless invocations are strictly best effort. And despite efforts at improvement [17, 21], invocations can be delayed arbitrarily.

We are interested in a new class of IoT-enabled applications for distributed intelligence that are characterized by bursty load and real-time requirements. Their bursty load typically arises from external real-world or cyberworld events. Their response is subject to real-time constraints with hard dead-
lines. Examples include distributed video analytics such as traffic or crowd control, network monitoring and response such as cybersecurity, real-time control such as autonomous drones, trucks, automobiles, and a large class of information integration and decision applications. These applications combine continuous sensing, with bursts of deeper computational analysis and timely decisions when interesting events occur. For example, failing to be timely (missing a critical event) might allow a fugitive to escape, a drone to collide with a bird, and so on. This class of applications cannot be served cost-effectively by today’s cloud (provisioning VM’s for max burst load is too costly, and on-demand allocation of VM’s and serverless cannot provide real-time guarantees).

To meet the needs of bursty, real-time applications, we propose real-time serverless, a new cloud resource type that provides a guaranteed allocation rate and thereby enables bursty applications to achieve real-time guarantees cost-effectively. To prove feasibility, we show one approach that realizes guaranteed allocation rates. Then, we systematically evaluate the benefits of guaranteed allocation rate resources for application quality guarantees and cost-efficiencies for bursty workloads using analytical modeling and simulation. Results show robust advantages of real-time serverless on a wide range of bursty properties, enabling cost-efficient high application value. Further resource efficiency scales well for multiple applications, suggesting this structure is attractive for cloud providers. Finally, we present a case study of a traffic monitoring application. Despite more complex burst statistics, our results show that major benefits for cost and application quality.

Specific contributions of the paper include:
1. A new cloud resource type, real-time serverless (RTS), that enables applications to achieve real-time guarantees for bursty workloads, and do so cost-effectively
2. Analytical and simulation evaluation for varying burst size and rate, as well as for multiple bursty applications that show the guaranteed allocation rate delivers responsive application performance efficiently. These real-time serverless are valuable for bursty, real-time applications – for duty factors from 0.025 to 0.25 the value of real-time serverless is 4x to nearly 16x greater than traditional cloud resource types.
3. Multiple applications can share real-time serverless instances efficiently, supporting duty factor increases of 25x with only a 1.6x increase in allocation rate (provider resource cost), suggesting the approach is attractive for a resource provider implementation.
4. A case study, using a Glimpse-like pipeline, on real traffic monitoring video that shows the benefit of a guaranteed allocation rate resources, demonstrating 16x value per unit real-time serverless compared to traditional.

The remainder of the paper is organized as follows. In

Figure 1: Real-world events such as traffic incidents give rise to burst computational requirements in video analytics applications.

Section 2, we describe the challenge of bursty, real-time workloads and our approach of guaranteed allocation rate resources. In Section 3, we create an analytical model for our new approach and explore its benefit for bursty, real-time applications. Section 4 broadens and deepens this study, via simulation of more complex scenarios. Section 5 applies the resource management ideas to a specific video analysis application. Finally, Sections 6 and 7 discuss related work and then summarize the paper with suggestions for future work.

2 Challenge: Bursty, Real-time Workloads

Bursty, real-time applications are characterized by unpredictable bursts of computing load, that have critical real-time deadlines that must be met. Bursts are often triggered by external events (e.g. a traffic incident, a flash crowd in the physical world, cyberattack, a trending meme in an app, etc.), and responses must trigger critical action (emergency response, shutdown) with hard, real-time deadlines. Bursts can increase application compute requirements by 10 or 100-fold, or even more and last for durations of seconds to minutes. These applications must deliver guaranteed high quality with real-time deadlines because application quality decreases rapidly with latency.

These bursty, real-time workloads are not a good match to traditional cloud resource models, as shown in Figure 2, where low allocations are compute limited (green shaded), incurring processing latencies that deliver poor quality, and high allocations incur unacceptable high resource waste (and corresponding cost). Throughout, we use the terms application quality and value synonymously.

1Throughout the paper, we use the terms application quality and value synonymously.

2Throughout the paper, we use the terms application quality and value synonymously.
2.1 Video Analytics Example

Consider a Video Analytics (VA) application that monitors traffic intersections in a city, and triggers in-depth analysis when upon observing an interesting event happens (a “wanted” car is seen, a traffic accident, a pedestrian falls down, etc.). On average over all intersections, events occur at an average rate of 2 per hour, and each initiates a burst of computation with a duration of 2 minutes. This is a duty factor of 6.6%.

While monitoring for interesting events, low-quality video (224x224 pixel resolution, 30 frames/second) is analyzed with a small neural net of 50 layers. When burst happens, full frames (resolution of 1024x1080), and a much wider and deeper neural net (150 layers) is used, producing burst computation height of 60x higher per frame compared to normal monitoring. Suppose we need 1 CPU instance for normal monitoring, then processing a burst frame requires 60 CPU instance-frame times of work. With a traditional cloud resource model (UI), reserving 60 CPU instances in advance would ensure each burst frame can begin processing right away at the cost of wasting 98.3% of the time. Such waste increases the cost of real-time video analytics and may render many applications infeasible.

One might suggest employing on-demand, spot, or serverless instances to service bursts, allocating them when a burst occurs, and then release them. However, none of these resource types provide any guarantee of allocation rate or allocation latency. So, a bursty application cannot guarantee any level of quality with these resource types. The problem is, that for all of these services, allocation is a best-effort activity. With today’s cloud, the only way to guarantee quality is to preallocate resources (e.g. reserved), paying for them even if they are idle.

2.2 Key Challenges to Support Bursty Applications

We divide challenges for responsive resource management to support burst, real-time workloads into three parts:

1. What new cloud resource types can enable bursty, real-time applications to guarantee quality?
2. How can these resource types be provided and scale to multiple applications?
3. Can these resource types be economically attractive in the highly-competitive cloud computing market?

2.3 Allocation Rate Guarantees for Bursty, Real-time Applications

To support bursty, real-time applications, we propose a new cloud resource type, real-time serverless, with allocation-rate service-level guarantees. This is distinct from existing cloud resource types – most of which can be classified as unreserved, indefinite duration resources (UI). That is, they are unreserved, so when a resource allocation request arrives, the system chooses whether or not to grant it. Consequently, the application cannot allocate resources reliably or in a timely fashion. UI resources are also granted to the application indefinitely, limiting the resource manager’s ability to reclaim resources. We reiterate the definition of UI resources and describe real-time serverless that is guaranteed rate, fixed duration (RTS).

- Unreserved, indefinite duration (UI): Resources are granted if available, and granted for indefinite duration. No allocation rate is guaranteed.
- Real-time serverless (RTS): An allocation rate is guaranteed (e.g. 10 instances/second), and each allocated instances is granted for a fixed duration (e.g. 5 minutes).

We depict our new approach in Figure 3, adding a real-time serverless with guaranteed allocation rate. To service bursty load, applications can request real-time serverless instances, and allocate resources with real-time response, and therefore is capable of achieving real-time guarantees for burst loads.
2.4 Realizing Guaranteed Allocation Rate Resources

Figure 4: A Real-time serverless resource pool (RTS) can be implemented with a fixed number of instances.

We describe a simple implementation of a real-time serverless (RTS) resource pool. As shown in Figure 4, given the fixed duration ($FD$), an instance allocated at time $t$ can be reclaimed by the resource manager at time $t + FD$. At that time, it can be reallocated, bounding the instances required to implement the guarantees. Given a guaranteed allocation rate, $A_{RTS}$, this means that the required size (number of instances) required to realize a real-time serverless pool with properties $A_{RTS}, FD$ is

$$Size(A_{RTS}, FD) = A_{RTS} \cdot FD$$

Thus the resource cost of these guarantees grows linearly with both the guaranteed allocation rate ($A_{RTS}$) and the finite duration ($FD$). There are many other ways to implement RTS pools, but we describe this simple implementation to demonstrate feasibility.

While we do not propose a particular deployment approach in this paper, there are two obvious approaches. First, a cloud user could statically allocate a set of virtual machines (UI resources), and run a resource manager as described in this section to create a RTS resource type. Second, a cloud provider could allocate a set of resources, and likewise, run the resource manager we have described.

2.5 Resource Pricing

For bursty applications, fine-grained resource pricing is attractive because of their low duty factor (ranging from a few to perhaps 20 percent). We assume that RTS resources can be priced at a granularity of a single second – similar to today’s serverless offerings [1]. We will consider attractive pricing for both applications and cloud providers in our evaluation.

2.6 Example: Real-time Serverless for Video Analytics

Revisiting our video analytics example, adding a RTS pool with guaranteed allocation rate of 30 instances per second, the
Figure 5: A burst computation requirement, allocation of resources, and the delay of each frame (e.g. \(I_2\) and \(I_7\))

### 3.1 Terms and Notation

First, in Table 1, we define terms for bursts \((H, D, \lambda)\) and application value decay \((\tau)\). We also define application metrics for latency and value. Finally, we define terms for resource management, including allocation rate and cost.

In real-time bursty applications, there is a loss of value (or quality), when processing is delayed. We model this as value that decays with delay. Specifically, for VA, we use an exponential decay with frame latency with \(\tau\) as the critical time constant. That is, the value of a single burst frame \(i\), delayed by \(l(i)\) frame times:

\[
V_{frame}(i) = V_{max}e^{-\frac{l(i)}{\tau}}
\]  

where \(l(i)\) is the latency of the \(i\)-th frame. If no instances are preallocated \((P = 0)\), the latency is infinite.

In Figure 5, we depict a burst computation requirement and a dynamic allocation response. If the response cannot keep up, the resource limit incurs frame processing delay. Latency compounds (e.g. latency of the 7th frame, \(I_7\), is higher than the second frame \(I_2\)) as burst frames continue to arrive (assumes FIFO processing).

### 3.2 Impact of Guaranteed Allocation Rate on Application Quality

Application quality is determined by the frame processing latency. The cause of processing latency is limited availability of resources. For an application using UI type resources, there is no guarantee of allocation, so \(A_{UI} = 0\). If the video analytics application pre-allocated \(P\) instances, then the latency for frame \(i\) can be expressed as

\[
l_{UI}(i) = \frac{i \cdot H}{P} - i \quad ; ; \quad \text{for } P > 0
\]

where \(i \cdot H/P\) reflects the processing time of preceding burst frames and \(i\) is arrival time for the \(i\)-th frame. If no instances are preallocated \((P = 0)\), the latency is infinite.

In contrast, RTS type resources can provide instances at the maximum guaranteed rate \(A_{RTS}\). If we assume that the application allocates instances at that maximum rate until all pending frames are being processed. For \(A_{RTS} \leq 1\), a burst frame that arrives at time \(i\), will be serviced starting at \(i/A_{RTS}\). Consequently,

\[
l_{RTS}(i) = i(\frac{1}{A_{RTS}} - 1)
\]

Where each frame is processed on a single instance with processing time \(H\) per frame.

Figure 6: Achievable application value achievable for UI and RTS with various \(A_{RTS}\). \((H=140, D=3,600, \tau=2,607 \text{ (1/2 per minute)})\)

The models can be used to generate distributions of frame values for a burst. We compare the achieved application value for UI and RTS, exploring a variety of guaranteed allocation rates. For the baseline, we use UI resource pool alone with \(P = 1\) instance, reflecting an acceptable cost, and \(A_{RTS} = 0\) (in blue in Figure 6). For UI, no additional resources can be allocated for each burst, so only a tiny fraction of maximum value is delivered (the frames mostly at left). For RTS, as the guaranteed allocation rate, \(A_{RTS}\), is increased, the application can service a burst by allocating resources more and more rapidly. Even a small \(A_{RTS}\) significantly increase the number of frames achieving close to maximum value (orange), and further increases in \(A_{RTS}\) (green, red) improve the situation dramatically. For example, with \(A_{RTS} > 0.6\), the application can ensure that all frames exceed 40% of maximum value. And as \(A_{RTS}\) increases towards 1 instance/frame time, a growing frame value guarantee can be achieved, reaching 100%.
This illustrates that RTS enables bursty applications to provide a guarantee of quality.

Another way to think about application quality is to ask what fraction of frames achieve a particular fraction of maximum quality. We plot this metric versus guaranteed allocation rate, $A_{RTS}$, in Figure 7. To achieve 50% of maximum value for even half of the frames, the video analytics application requires an $A_{RTS}$ of 0.5 instances/frame-time. To achieve 50% of maximum value for 100% of the frames, an $A_{RTS}$ of 0.67 instances per frame time is needed. At the high end, to achieve 90% of maximum value, 0.85 instances/frame-time are required for 50% of the frames, and 0.9 for 100% of the frames. At $A_{RTS} = 1$, the application can deliver 100% of maximum value because an instance can be allocated for each arriving frame. Again, this illustrates that RTS enables bursty applications to provide a guarantee of quality.

In summary, our analytical model shows that adding a guaranteed rate, fixed duration resource pool can enable bursty, real-time applications to achieve high value. In fact, the results show that allocation rate is the critical enabler of high value. This is striking as no major cloud services provide any resource types with guaranteed allocation rate.

### 3.3 Impact of Burst Shape on Application Quality

The benefit of RTS depends on burst shape ($H$: height, $D$: duration) and burst arrival rate ($\lambda$) in a workload. We consider various shapes, holding total computation demand (i.e. $H \cdot D$) constant. For simplicity, here we consider one burst at a time (no burst interference), and revisit this later via simulation in Section 4. For each burst, we compute the burst value, which is the sum frame values for each frame in the burst as below:

$$BurstValue = \sum_{i=0}^{D-1} V_{frame}(i) = V_{max} \sum_{i=0}^{D-1} e^{-\frac{i}{\tau}}$$

Critical factors are $D$ (duration), the resources available ($A_{RTS}$), and the value decay ($\tau$). Failing to keep up with burst demand (i.e. allocation delay) will directly lead to value deterioration.

In Equation 6, value deterioration due to limited resources can be seen in $e^{-\frac{i}{\tau \cdot A_{RTS}}}$ which decreases as a function of $i$, but at a rate determined by $A_{RTS}$. For longer durations ($D$) this produces more terms in the sum, but each is exponentially smaller, so the sum grows slowly with $D$. The only way to mitigate this effect, is to have larger $A_{RTS}$. Because we only process frames on a single instance, burst height $H$ does not affect value.

We plot these results in Figure 8. First, increasing $A_{RTS}$ always improves application value regardless of the burst shape. And a finite value of $A_{RTS} = 1$ is sufficient to achieve 100% of maximum application value. Further, our plot shows five different burst duration scenarios, varying from 900 frame-times (30 seconds) to 14,400 frame-times (8 minutes); holding total compute demand constant. At a guaranteed allocation rate, long bursts always achieve higher burst value than shorter bursts – because they have more frames. For example, at $A_{RTS} = 0.6$, value for $D = 1,800$ frame-times is about 1,400 smaller than 2,300 achievable by $D = 3,600$, 3,300 by $D = 7,200$ and so on. However, the longer bursts require a higher allocation rate in order to achieve the same fraction of maximum potential burst value. For example, $A_{RTS} = 0.3$ is needed to achieve 50% of maximum value at $D = 1,800$ frame-times but if $D$ increases to 7,200 frame-times, allocation rate of $A_{RTS} = 0.63$ achieves the same level of quality.

In summary, burst shape can affect the guaranteed allocation rate needed to achieve a high fraction of maximum value. And at a finite allocation rate $A_{RTS} = 1$, RTS can deliver maximum potential value regardless of the burst shape. These
studies confirm that RTS can robustly deliver high burst value for a wide range of burst shapes.

### 3.4 Impact of a Guaranteed Allocation Rate on Cost

Now we develop analytical model to assess whether RTS are attractive from a cost point of view. For comparison purposes, we define the cost of RTS relative to UI resource with ratio $k$ in cost/instance-second as below:

$$C_{RTS} = k \cdot C_{UI}$$  \hspace{1cm} (7)

If the RTS instances have fixed duration greater than $H$, the cost of serving a burst, $Cost_{RTS}$, can be written in terms of the total demand, $H \cdot D$, as below:

$$Cost_{RTS} = H \cdot D \cdot C_{RTS}$$  \hspace{1cm} (8)

For the UI pool, if we assume pre-allocation of $H$ instances, that maximizes value (as in Figure 2). Allowing for charges for the idle periods between bursts ($1/\lambda$), produces the cost for a single burst:

$$Cost_{UI}(P) = \frac{1}{\lambda} H \cdot C_{UI}$$  \hspace{1cm} (9)

RTS instances are only allocated when bursts happen, so we need to factor in the workload’s duty factor ($DF = D \cdot \lambda$). Figure 9 shows the resource cost normalized to $Cost_{UI}$ for UI and RTS for two duty factors (fixed burst arrival rate $\lambda$, varied duration $D$). Fixing $D$ and varying $\lambda$ have the same effect. We set $P = H$ and $A_{RTS} = 1$ so both approaches achieve 100% burst value. For RTS, we plot total cost at various cost ratios ($k$).

Starting at Figure 9a at low duty factor such as $DF < 0.025$, RTS resource premiums of 10-20x can still be cost-effective for bursty applications. For moderate duty factors, RTS is more cost-effective for $k$ as large as 8. For high duty factors, approaching 0.25 (25%), the RTS resources are used more often, and can eventually approach and exceed that of UI. For instance, when $DF > 0.125$, RTS is less cost-effective for $k > 6$. This suggests that premiums of 6-8x may still be attractive for bursty applications – much as 6x premiums for serverless resources are still attractive [1].

In summary, the analytic model shows that adding guaranteed rate, fixed duration resources enables bursty applications to deliver application value computation at attractive cost.

### 4 Evaluation

Our analytical model suggests that adding a guaranteed rate resource pool enables bursty, real-time applications to not only achieve higher quality/value, but also may have some economic advantages. However, the analytic models simplified a number of aspects of resource management – in this section, we conduct empirical studies of a wider range of workloads and cloud resource settings. Our goal is to understand how robust the guaranteed rate pool benefits are across a range of different type of bursty workloads. First, we study the basic scenario used for the analytical model, using our simulator, and validate the analytical model results. Second, we study a variety of bursty workloads, varying burstiness (arrival rate), burst duration, and systems with multiple sources of bursts. These studies characterize how broadly beneficial the guaranteed rate pools can be. Finally, we consider a case study, which is consisted of several realistic workloads with video analytics pipeline, to show that the guaranteed rate pool enables higher application quality on video analytics applications while showing economic benefits.

#### 4.1 Simulation

We construct simulator as an extension of the analytical model. The terms and notation described in Section 3 are reused. We
add variability to burstiness properties to capture the uncertainty of bursty workloads. In particular,

- **Burst arrival rate** ($\lambda$) bursts arrive in a memoryless Poisson process. Burst interference is possible.
- **Burst duration** ($D$) varies according to a Gaussian distribution.

Adding variability to burstiness properties allows the simulation to cover a wide range of bursty workloads. For examples, we use low variance, low arrival rate to simulate a uniform single bursty source. For a collection of uncorrelated, varied duration sources, we use high variance and high arrival rate.

In resource management, RTS pool guarantees allocation rate by splitting the time into fixed $1/A_{RTS}$ frame-time slots. Within each time slot, the pool makes one instance available to VA. VA can consume this instance without delay. If VA asks for more, it has to wait until the next time slot. We capture the scenarios we analyze in Section 3 by letting VA allocate resource immediately for every incoming frame. If frame arrival rate larger than allocation rate, delay accumulation occurs (frame experience delay of frames arrive before) harming application quality severely.

### 4.2 Validating the Analytical Model

We first simulate a scenario similar to that studied with the analytical model (Section 3), a bursty workload. Parameters include burst height of 140x normal load, and burst duration of 3,600 frame-times. We use 0.01 duty factor to avoid burst interference assumed by the analytical model.

We vary guaranteed allocation rate and try to reproduce the analytical results from Figure 6, 7, via simulation. The results of one day simulation are shown in Figure 10. The simulation results are close to that of analytical modeling with some noisiness due to the discrete method of simulation. For example, Figure 10a exhibits similar results but small variation due to the uncertainty in arrivals of bursts and lead to slight reduction in achievable value comparing to analytical model. Similarly, Figure 10b shows the increase in application value while guaranteed allocation rate is increasing. Although results show less achievable value comparing to analytical model, RTS pool shows comparatively significant growth in slope. Therefore, RTS pool presents promising benefits in maximizing value under bursty workload simulations.

### 4.3 Burst Overlap and Interference

In the analytical model, we omitted the effects of overlapping bursts – burst interference. Here we explore via simulation how burst overlap and interference affects application value and RTS effectiveness. We vary burst duration from 900-14,000 frame-times with standard deviation of $0.1 \cdot D$, and resulting increased duty factor to induce increased burst interference (low ($DF < 0.01$), medium ($DF < 0.1$), and high ($DF < 0.25$)). For low burst interference, (see Figure 11a), the simulation results match the analytical model results in Figure 8, validating its accuracy for a single burst. As we increase duty factor (see Figure 11b and 11c), we see degradation of burst value, particularly for the higher duty factors (longer duration). For example, at $A_{RTS} = 1$, $D = 14,400$ for $DF = 0.1$ and $DF = 0.25$, burst value is reduced to 88% and 77% respectively.

Interference can also cause degradation a low $A_{RTS}$, as the RTS pool has a lesser ability to respond to a single, much less several concurrent bursts. For example, this interference causes burst value for $D = 14,400$, $A_{RTS} < 0.5$ drop below $D = 7,200$ in high duty factor scenarios (see Figure 11c).

### 4.4 Supporting Multiple Applications Efficiently

One can think of high duty factors as a single application with many more events or as a combination of multiple in-
dependent applications sharing a single RTS resource pool. Thinking of the latter, we explore how higher guaranteed allocation rates can increase burst value towards the potential maximum.

In Figure 12, we show the sensitivity of normalized burst value to $A_{RTS}$ at various duty factors. The sharp breaks at $A_{RTS} = 1$ and $A_{RTS} = 2$ indicate overlapping bursts, and that they impact 14% of the burst value for $DF = 0.1$, and 30% of the burst value for $DF = 0.25$. We further note that achieving 100% of burst value in face of a 25x duty factor increase only requires a 3-fold increase in $A_{RTS}$.

We examine the potential for multiple applications to share a single RTS resource pool efficiently. Consider 10 applications, each accounting for $DF = 0.01$ summing to $DF = 0.1$ and 25 applications, each accounting for $DF = 0.01$ summing to $DF = 0.25$, and so on as shown in Figure 13. For low burst value ($< 0.5$) there is little difference in the required $A_{RTS}$. For moderate values, the difference grows but at a deeply sublinear rate. For example, for value of 80% potential maximum value, an increase from 1 to 25 applications requires only a 2x increase in $A_{RTS}$, and based on our model implementation (see Section 2.4), only a 2x in cloud provider resource commitment.

The curves cover the allocation needed for a wide range of duty factors from 0.01 to 0.25 but they are very close to each other indicates that only a small increment of allocation rate is sufficient to deal with significantly increment of burst demand. At 90% max value, the multiple is even smaller. requiring a 1.6x $A_{RTS}$ increase for a 25x increase in number of applications. These results suggest not only that RTS resource types scale well – supporting a growing number of bursty, real-time applications at high quality with a slowly growing number of resources. Our results show this growth is deeply sublinear, suggesting that RTS may be best implemented as a shared cloud service (not privately by a single application), and that doing so may be quite profitable for cloud providers.

### 4.5 Impact of Variability on Cost

We now consider how variability of burstiness properties (burst shape, arrival rate) affects the cost of absorbing bursts. In Figure 14, we reproduce the results from Figure 9 using simulation. At low duty factor, Figure 14a shows that RTS is more cost effective than UI, even with RTS resources at a 10-
(a) Cost vs. Low duty factor $(DF < 0.025)$

(b) Cost vs. High duty factor $(DF < 0.25)$

(c) UI resource cost vs. Duty factor

Figure 14: Cost for maximizing value using UI and RTS at various cost ratio ($k$) and Cost of UI at different duty factors, $P =$ maximum observed computation height (several bursts).

<table>
<thead>
<tr>
<th>Burst Duration (frames)</th>
<th>Burst Height</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean, StDev</td>
</tr>
<tr>
<td>Night</td>
<td>116, 186</td>
</tr>
<tr>
<td>Day</td>
<td>120, 216</td>
</tr>
<tr>
<td>Rush hours</td>
<td>917, 1293</td>
</tr>
<tr>
<td>Overall</td>
<td>197, 503</td>
</tr>
</tbody>
</table>

Table 2: Burst Statistics for Traffic Video

At higher duty factor (Figure 14b), RTS cost grows as burst frequency grows, with UI eventually more cost-effective for $k = 24$. However, these results are much better than for the analytical modeling (Figure 9b) where $k = 6$ made RTS more expensive than UI for $DF > 0.125$. Here, RTS is still cheaper than UI even at $k = 12$ and $DF = 0.25$. UI must reserve enough resources for the highest burst peak, so its cost grows with duty factor and burst interference as shown in Figure 14c. We believe these large jumps are the cause of the spikiness in Figure 14b.

5 Case Study: Traffic Intersection Monitoring

In Section 4, we used simulation for a range of bursty workload parameters to show the benefits of RTS resource types for application quality and cost. Here, we use a traffic monitoring case study, derived from a real traffic video, a deep neural network that does object detection, and the induced bursty workload trace to evaluate the benefits of guaranteed allocation for real-time, online video analytics.

5.1 Methodology

We model a Glimpse-like pipeline with a client and server (our cloud) that processes frames considered interesting by the shallow processing at the client [7] (see Figure 15). Glimpse uploads frames to the server for object detection. Our empirical measurements, characterized the server-side frame processing cost at 20x for object detection, but for richer analytics, this ratio could be much higher. We use a rush-hour traffic video captured from a traffic cam in Southampton, NY at the intersection of County Rd. 39A and North Sea Rd. and available from [8]. The video is 30 frames per second at a resolution of $1920 \times 1080$ color pixels per frame.

Because we are interested in analyzing complex behavior such as erratic driving, reckless walking, or traffic incidents, we use an efficient model [18] with ResNet trained on KITTI dataset [12]) to process the video and annotate it with object appearance and departure intervals. These object intervals are combined and collectively create the bursts (see Table 2). To scale up to a full 24-hour from our short, rush-hour clip, replicating it to create two 60-minute segments (morning and afternoon rush hour). We scale the time base by 20x while holding object interval duration constant, creating an 8-hour segment of lower traffic (daytime). Finally, we scale the time base by 40x while holding object interval duration constant, creating a 14-hour segment of lowest traffic (nighttime). The total number of bursts is 2,311 and duty factor is 0.175 for the
5.2 Impact of RTS on Quality and Cost

We first explore the basic characteristics of the traces, as shown in Table 2. The burst durations are much shorter than those explored in Sections 3 and 4 with an average burst duration of 7 seconds (210 frame-times), as shown in Table 2. Moreover, both the burst duration and height are highly variable within each part of the day.

Figure 16a shows the value distribution of the video trace; while similar qualitatively to Figures 6 and 10a, the real burst trace is much noisier. The traffic analysis quality benefits from increasing $A_{RTS}$, are shown in Figure 16b and 16c. Three curved sections are clearly visible and correspond to the three different operating points – rush-hour, daytime, and nighttime. At $A_{RTS}$ as little as 0.25 instances/frame-time, all of the nighttime value is captured. At $A_{RTS}$ of 0.9, the daytime value is captured. In Figure 16c, we see flat curves, and very little separation by fraction of burst-frame value. This reflects a difficult workload for increases in $A_{RTS}$ to improve application quality. To achieve full quality on the intense activity during rush hour (10 objects in frame) requires $A_{RTS} = 1$.

Figure 17 shows how the burst load varies over the 24-hour period. To illuminate how the RTS system responds with time, we overlay the application cost of both a RTS implementation at various cost ratios ($k$) and an UI implementation achieving the same (100%) quality. The benefits of dynamic management are clear. Considering the full 24-hour day, the RTS approach is 8.3x less expensive for $k = 2$; 4x less expensive for $k = 4$, and 2x less expensive for $k = 8$. In short, the RTS resources are 16x more valuable than traditional UI resources.

As discussed in Section 2.4, guaranteed allocation rate can be achieved at cloud provider resource cost linear in $A_{RTS} \cdot FD$. However, we find that for our case study, the effective cloud provider resource requirement is much lower as shown in Figure 18. This is because the typical burst height is much less than $FD$, so many instances are returned long before the full $FD$ time. In this case, resource cost is reduced 70-fold to only 123 instances. We expect savings such as this to occur in many cases, but will vary in magnitude.
6 Related Work

To the best of our knowledge, no cloud resource management research that provides guaranteed allocation rate. A wide variety of research studies explore resource management in clusters and datacenters. We discuss these classes of resource managers and the resource models they support.

Commercial Serverless Deployments The serverless (also function-as-a-service or FAAS) event-driven model creates instances to process events. It provides no guarantees of invocation latency. Further AWS’s Lambda [3] and Google Cloud Function [14] limit execution duration to 15 minutes and 6 minutes respectively. Both of these services limit maximum concurrent invocations to 1000 within a region. In contrast, our RTS resources guarantee allocation rate, a stark contrast, while supporting a similar fine-grained “burst” computing model.

Job Scheduling for Heterogeneous workloads Mainstream resource management schedulers such as TORQUE [31], SLURM [34] support batch jobs that run for hours. Extensive research schedules jobs with heterogeneous resource requirements (including parallelism) and run times, optimizing for resource utilization, job wait time, and more. For example, a number of studies explore FCFS and backfill algorithms [30,32]. Further, Tang et al. [32] propose metric-aware scheduling that optimizes performance based on such as fairness and system utilization. However, these schedulers are not responsive to burst demands, do not provide guaranteed allocation rates, and do not support fine-grained (seconds) resource allocation.

Datacenter Resource Management formulates a more complex problem, mixing long-running processes with more typical jobs with dependent sets of short-running tasks. Mesos [16] uses a distributed two-level scheduling model, offering resources to computing frameworks, who in turn schedules tasks. Borg [33] schedules a mixed workload of end-user-facing service jobs (high priority) and batch jobs (low priority). Firmament [13] is a centralized scheduler desired to be scalable at low placement latency if only partial workload consists of short tasks. Sparrow [27] is a decentralized scheduler designed for datacenters that support jobs composed of very short, sub-second tasks. It achieves very high throughput rates. None of these systems provide guaranteed allocation rates.

Coping with Bursty Workloads Scalable internet services deal with bursty loads managing yield and latency by dropping requests [5]. For example, the WeChat microservice system has an elaborate system for load shedding that orders drops to minimize wasted work [35]. However, all of these approaches drop requests in order to maintain service quality for accepted requests. In contrast, because we assume the cloud has sufficient resources to service our bursty, real-time applications, we take the approach of guaranteed allocation rate to maintain quality for all of the received requests.

Closest to our study, one recent effort attempts to accommodate real-time advanced photon source (APS) experiment data analysis (a coarse-grained task), sharing resources on a batch job scheduled system [23]. This system dynamically shifts nodes from the batch schedulers resources into an on-demand allocation pool. This shifting is based on prediction of the bursty, real-time APS workload. Our work differs in application granularity (coarse-grained vs. fine-grained), and the predictability of their domain. Because of their dependence on prediction, their resulting system cannot provide real-time or application quality guarantees. Our approach creates a dependable, guaranteed allocation rate, upon which application guarantees can be built.

7 Summary and Future Work

Today’s cloud resource offerings provide no guarantees for resource allocation. We have proposed and evaluated a new type of cloud resource with a distinct service-level objective – guaranteed allocation rate for fixed duration instances (RTS). Using analytical modeling, simulation, and a case study, we have shown that this guarantee enables timely resource allocation for bursty needs, and empowers applications to provide quality guarantees. Further studies show that such resources can be cost-effective for applications and perhaps attractive, due to multi-application scaling, to cloud providers.

Our results open a number of interesting research questions: How should varying allocation rate requirements be met for diverse applications? Can they be in a shared resource pool, or need they be separated? For a given quantity of resources, how should a cloud provider divide across UI and RTS resources to best serve applications? Can it do so dynamically? Can RTS guarantees be added to serverless models? We look forward to future research by the community to address these questions.

Acknowledgments

This research was supported in part by the National Science Foundation under Awards CNS-1405959 and CMMI-1832230. Support from the CERES Center for Unstoppable Computing is also gratefully acknowledged.
References


