Real-time Serverless: Enabling Application Performance Guarantee

Hai Duc Nguyen¹, Chaojie Zhang¹, Zhujun Xiao¹, and Andrew A. Chien¹,²

¹University of Chicago ²Argonne National Lab
Serverless has Limitation

- Function-as-a-Service (FaaS) aka Serverless is the fastest growing element of cloud workload

But
- Best-effort invocations
- Long-tail latency

https://serverless-benchmark.com
Bursty, Real-time Applications

Computation demand surges when interest events happen

- A “wanted” person appears
- A cyber attack

Timely response to these events
Serverless vs. Bursty, Real-time Apps

Serverless invocations are best-effort

❌ No way to guarantee when an invocation will run
Real-time Serverless

Real-time Serverless (RTS) = Serverless + Guaranteed Invocation Rate

Function description:
- Maximum Runtime (timeout)
- Handler
- ...
- Guaranteed invocation rate ($A_{RTS}$): at least 1 invocation per period of ($1/A_{RTS}$) seconds.
- ...
Real-time Serverless

Resource

Bursty Load

Serverless

Time

Resource

Bursty Load

Real-time Serverless invocation

Timely resource access
Analytic Model: Video Monitoring

Value is represented as:

\[ FrameValue = MaxValue \cdot e^{-\frac{Response\ Delay}{\tau}} \]
RTS Guarantees Statistics for Frame Value

Guarantee Invocation Rates
- \( A_{RTS} = 0.0 \) invocation/\( ft \)
- \( A_{RTS} = 0.1 \) invocation/\( ft \)
- \( A_{RTS} = 0.3 \) invocation/\( ft \)
- \( A_{RTS} = 0.9 \) invocation/\( ft \)
- \( A_{RTS} = 1.0 \) invocation/\( ft \)

\( ft = \text{frame-time} = 1/30 \text{ sec} \)

- High guaranteed invocation rate \( \rightarrow \) high value
- Guarantee Statistics for Frame Value
Rational Design for Value

Application can adjust guaranteed invocation rate to meet any value target.

Higher is better

✓ Enable application to engineer the value distribution.
For realistic bursty applications, the interference probability is low.
RTS can support Multiple Bursts

Bursts can happen simultaneously

✓ Real-time Serverless can support multiple bursts
✓ Approach is simple – just increase the guaranteed invocation rate
Implementation

Real-time serverless interface
- Compatible with serverless

- <function name>
  - lang: <Language of function body>
  - handler: <Location of function body>
  - image: <Docker image reference>

  realtime: <Guaranteed invocation rate>

  timeout: <Runtime limit>
  limits: <Maximum resource use>
  requests: <Minimum resource use>

Working prototype
- Leverage OpenFaaS
- Admission control at function registration
Case Study: Traffic Monitoring

- Traces from real video over Glimpse
- Low-level monitor for vehicle presence
- Bursts arise when vehicles appear.

![Traffic Image]

![Graph Image]
Simple Frame Value Model (Success/Fail)

Vary guaranteed invocation rate (large background load)

- Application Requests
- Serverless/OpenFaaS ($A_{RTS} = 0$)
- Real-time Serverless, $A_{RTS} = 0.3$
- Real-time Serverless, $A_{RTS} = 1.0$

Serverless cannot respond to demand changes

- RTS’ guarantee invocation rate enables it to respond to application demand despite competition from background load
- Higher RTS invocation rate improves for success rate for multiple bursts
Related Work

- Traditional Serverless with fast, dynamic invocation
  - Amazon Lambda, Google Cloud Function, OpenFaaS, Knative, etc.

- Minimizing FaaS invocation overhead
  - SAND (ATC’18), SOCK (ATC’18), Kim et. al. (CLUSTER’18).

- Extension for improving FaaS performance
  - Jonas et. al. (SoCC’17), Hellerstein et. al. (CIDR), Jonas et. al. (Berkeley, 2019)

None focus on performance guarantees / real-time.
Summary

• Current serverless interface cannot support real-time, bursty applications.

• Real-time serverless = Serverless + Guaranteed invocation rate.
  • Guarantee statistics for value.
  • Enable rational design.

• A prototype shows timely response for a video monitoring application

• Future work
  • Efficient implementation for RTS interface
  • Explore the benefits of RTS interface for other application classes
Q&A

Acknowledgement. This work supported by National Science Foundation Grants CNS-1405959, CMMI-1832230, and CNS-1901466. We gratefully acknowledge support from Intel, Google, and Samsung.
Backup Slides
A Big Picture

Computing Infrastructure

Cloud Data Center

Edge Providers

Applications

IT Server

Real-time

Our focus!

Bursty
Validate Analytical Results with Simulation

![Graphs showing the fraction of burst frames vs. % of max. value and % of burst frames vs. $A_{RTS}$ for different $A_{RTS}$ values.](image-url)
Supporting Multiple Applications
Case Study: Statistics

<table>
<thead>
<tr>
<th></th>
<th>Burst Duration (frames)</th>
<th>Burst Height</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean, StDev</td>
<td>Min–Max</td>
</tr>
<tr>
<td>Night</td>
<td>116, 186</td>
<td>30–2,445</td>
</tr>
<tr>
<td>Day</td>
<td>120, 216</td>
<td>30–2,323</td>
</tr>
<tr>
<td>Rush hours</td>
<td>917, 1293</td>
<td>30–7,464</td>
</tr>
<tr>
<td>Overall</td>
<td>197, 503</td>
<td>30–7,464</td>
</tr>
</tbody>
</table>

Burst Statistics

Glimpse Pipeline Architecture

---

1 Tiffany Yu-Han Chen et. al., Glimpse: Continuous, Real-time Object Recognition on Mobile Devices, SenSys’15
RTS for Video Analysis

Guaranteed invocation rate enables value guarantee

Realistic Workload

Synthetic Workload
RTS for Video Analysis

Realistic Workload

Synthetic Workload

✓ Enable rational design for value guarantee
Robust against Burst Shape

- Fixed total demand per burst
- Vary burst duration (and height)

✓ Any value are achievable at an appropriate $A_{RTS}$

✓ Maximum value is achieved at $A_{RTS} = 1$, regardless burst shape
Robust against Burst Variability

Change variability by varying burst duration standard deviation

Variability causes value drop
Higher duty factor creates more damage
✓ Increase $A_{RTS}$ cancels variability effect
✓ RTS value can be maintained for wide burst variance
Multiple Real-time, Bursty Apps.

✓ RTS resource cost scales with actual demand
✓ RTS resource consumption is 2.2x to 5x lower than UI
✓ RTS helps cloud provider save resource to serve more applications
Application Cost: UI vs. RTS

- Resource cost for maximizing burst value with different duty factors
- Vary RTS vs. UI cost ratio

✓ RTS resource value per unit cost is 16-24x higher than UI
✓ RTS enables low cost solutions for real-time, bursty applications

Lower is better
UI Cost at Different Duty Factors
Resource Cost

✓ RTS is 2x to 8x cheaper than UI

✓ Actual resource requirement is 70x lower than the worst case
RTS Implementation Feasibility

RTS instances can be quickly reuse after reaching the max. runtime $R_{RTS}$

RTS pool capacity is bounded

$$C(A_{RTS}, R_{RTS}) = A_{RTS} \cdot R_{RTS}$$
RTS Interface (Lambda Extension)

• Function description in YAML format

```yaml
<function’s name>
lang: <prog. language>
handler: <refers to a folder where function body can be found>
image: <Docker image reference>
realtime: <minimum invocation rate>
environment:
  exec_timeout: <Hard processing timeout>
limits: # <----- Maximum resources used by the function
  memory: <max. memory>
  cpu: <max. cpu>
requests: # <----- Resource requested by an instance
  memory: <req. memory>
  cpu: <req. cpu>
```
Introduction

• Function-as-a-Service (FaaS) aka Serverless is the fastest growing element of cloud workload

• Expected to be the driving force for the future cloud computing

- Easy for development and deployment
- Dynamic resource scaling enables cost efficiency
- High resource management flexibility
Demonstration: Experiment setup

- RS
  - Real-time
  - Serverless
    - (best-effort)

- Serverless
  - (best-effort)

Driver

Image viewer

Streaming app

Background load generator

Driver

Image viewer

Streaming app