Real-time Serverless: Enabling Application Performance Guarantee

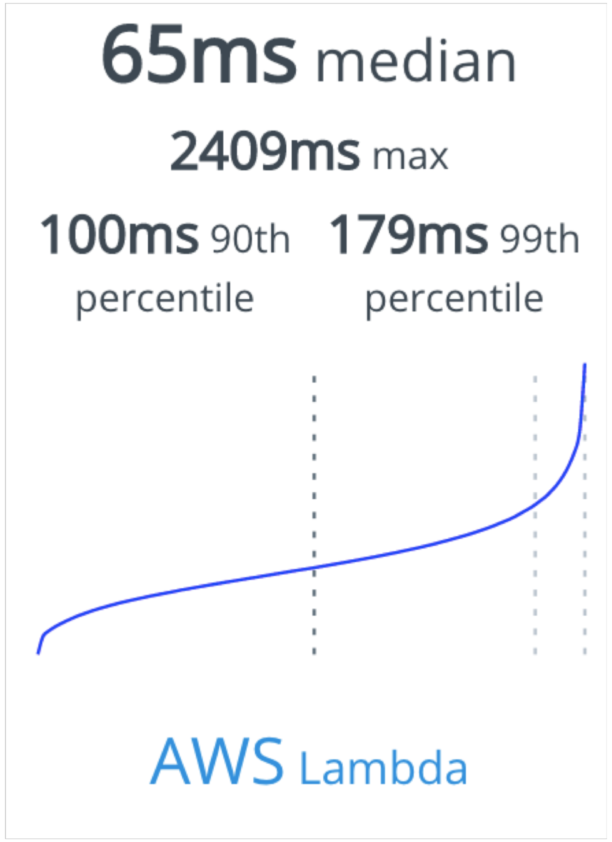
Hai Duc Nguyen[1], Chaojie Zhang[1], Zhujun Xiao[1], and Andrew A. Chien[1,2]
[1]University of Chicago [2]Argonne National Lab

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



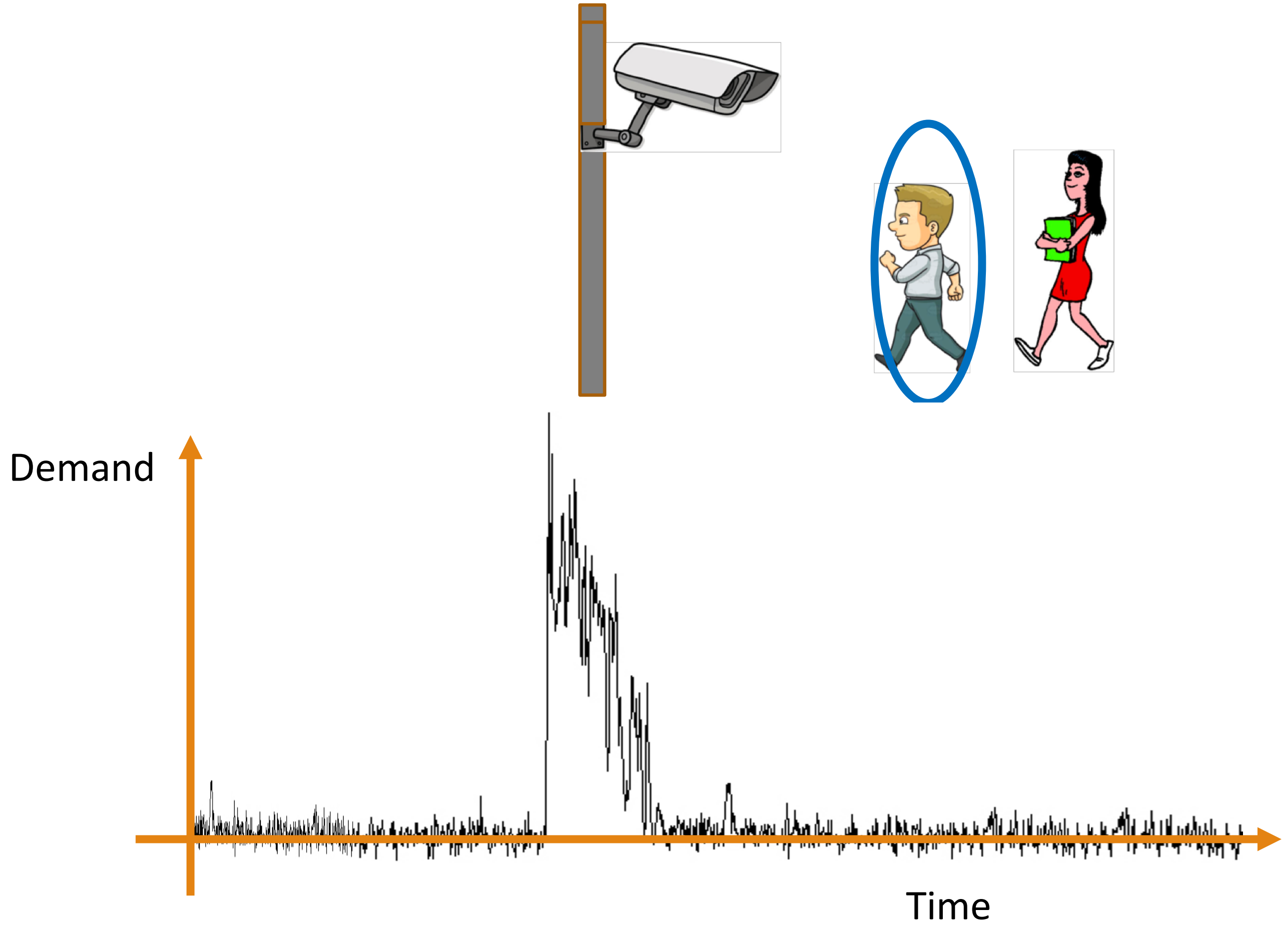
https://serverless-benchmark.com

Bursty, Real-time Applications

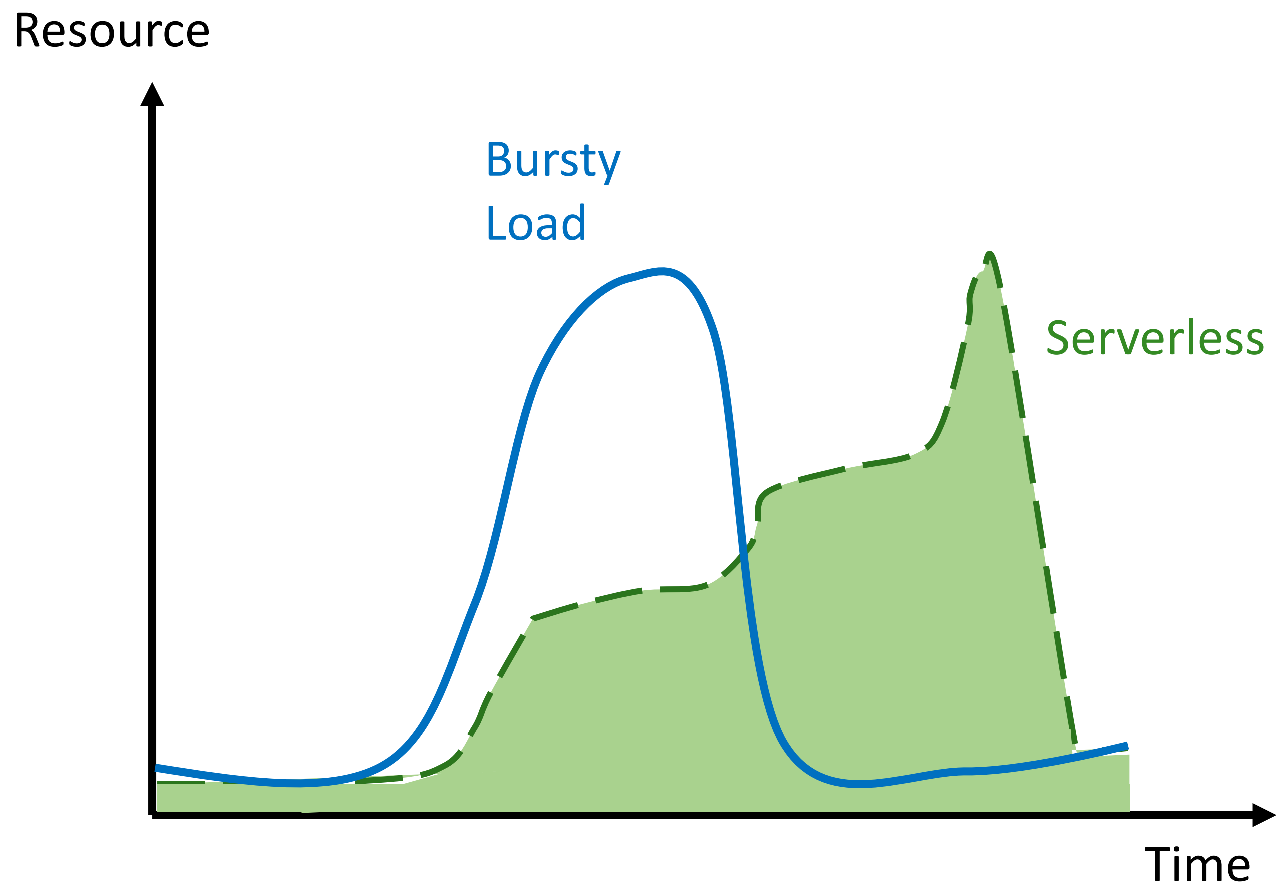
Computation demand surges when interest events happen

- A “wanted” person appears
- A cyber attack

Timely response to these events



Serverless vs. Bursty, Real-time Apps

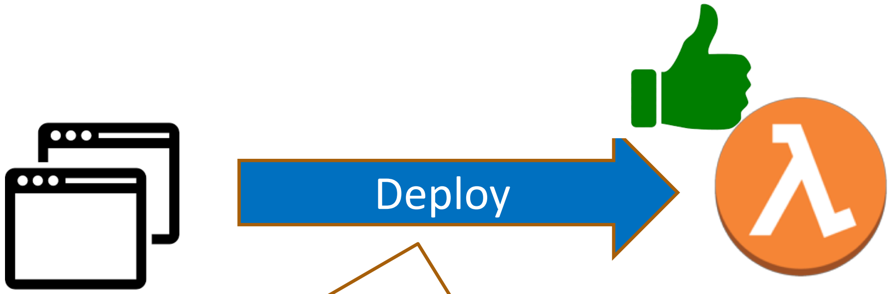


Serverless invocations are best-effort

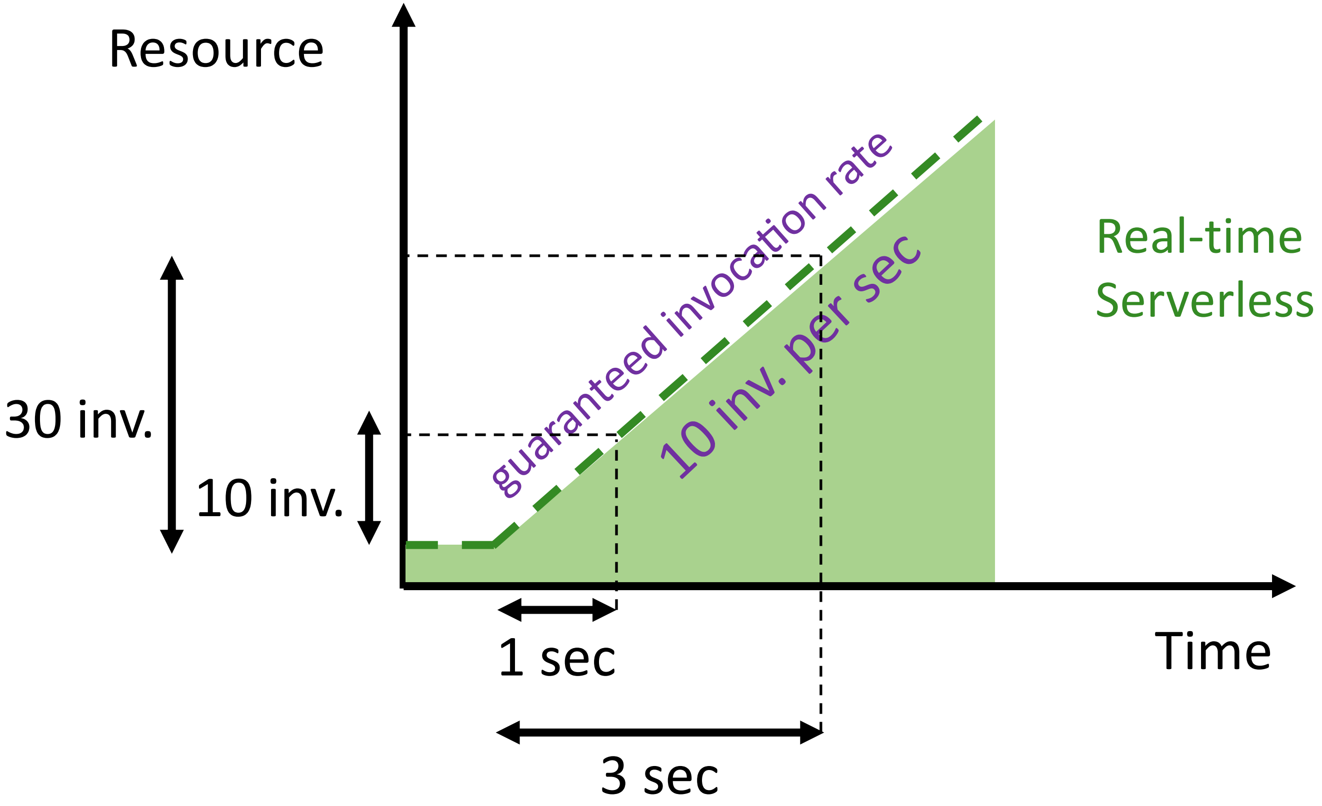
❌ No way to guarantee when an invocation will run

Real-time Serverless

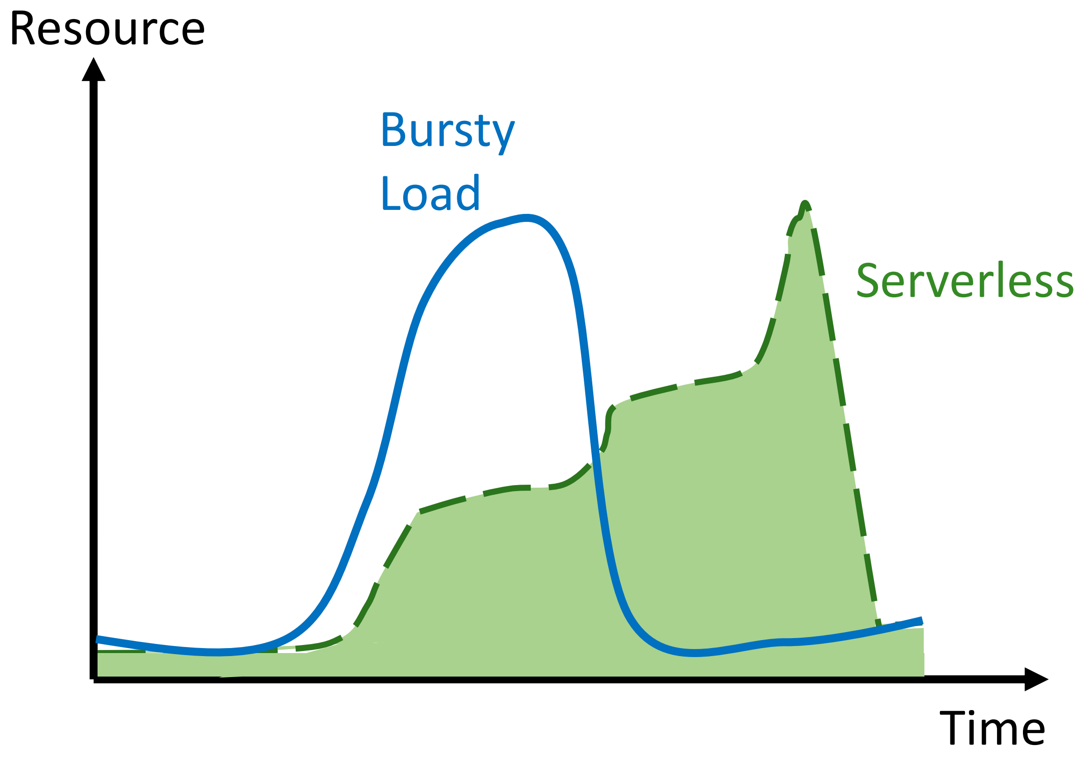
Real-time Serverless (RTS) = Serverless + Guaranteed Invocation Rate

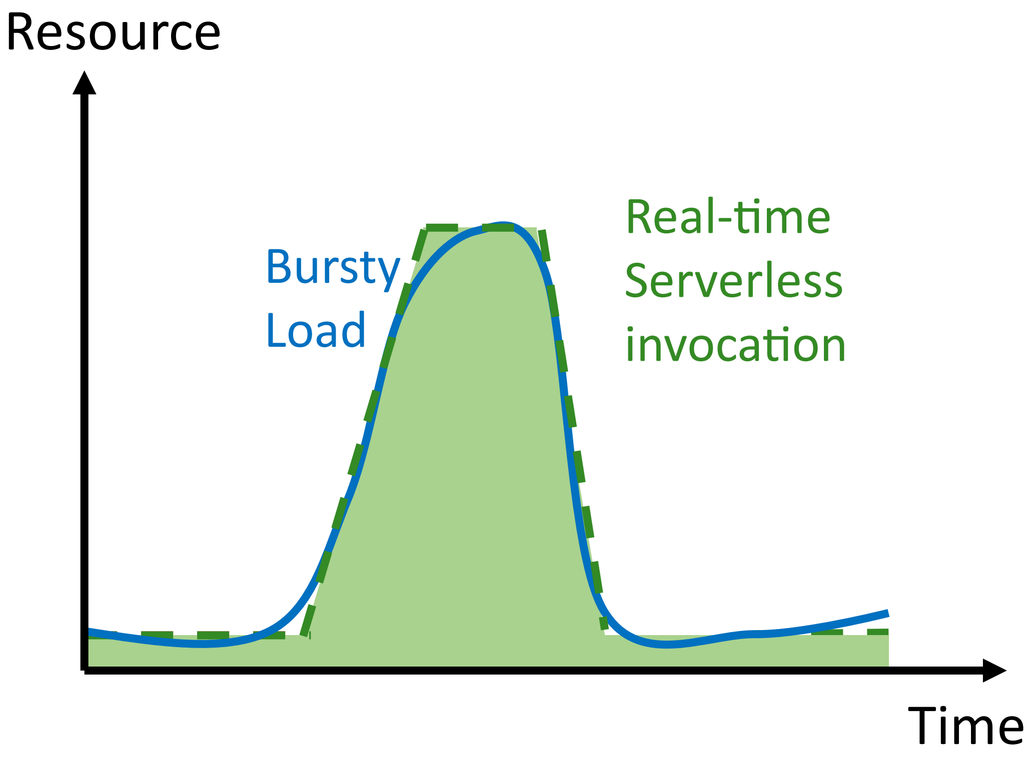


Function description:
- Maximum Runtime (timeout)
- Handler
- …
- **Guaranteed invocation rate (A_{RTS}): at least 1 invocation per period of ($1/A_{RTS}$) seconds.**
- …



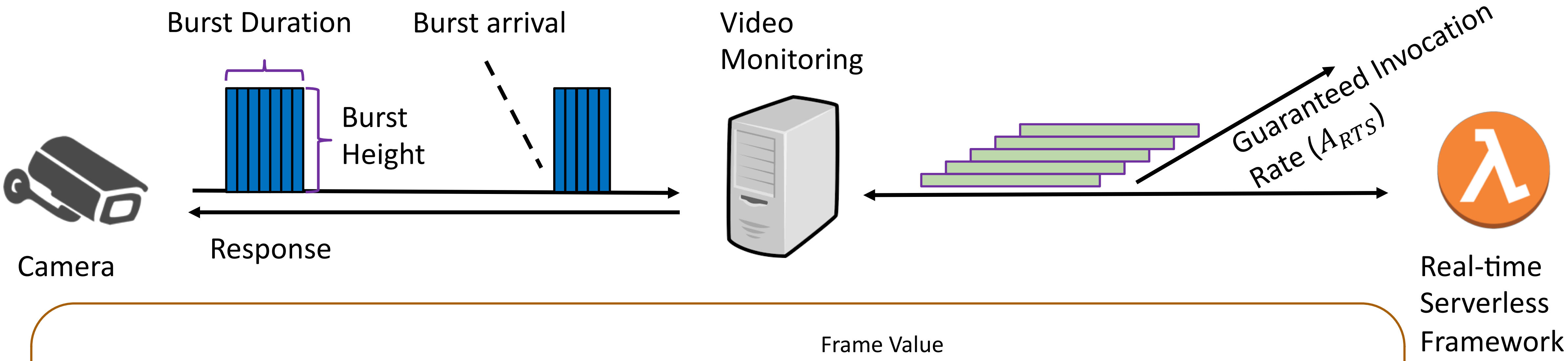
Real-time Serverless





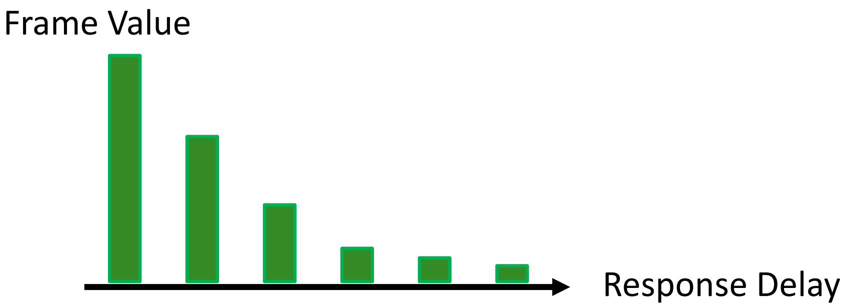
✓ Timely resource access

Analytic Model: Video Monitoring

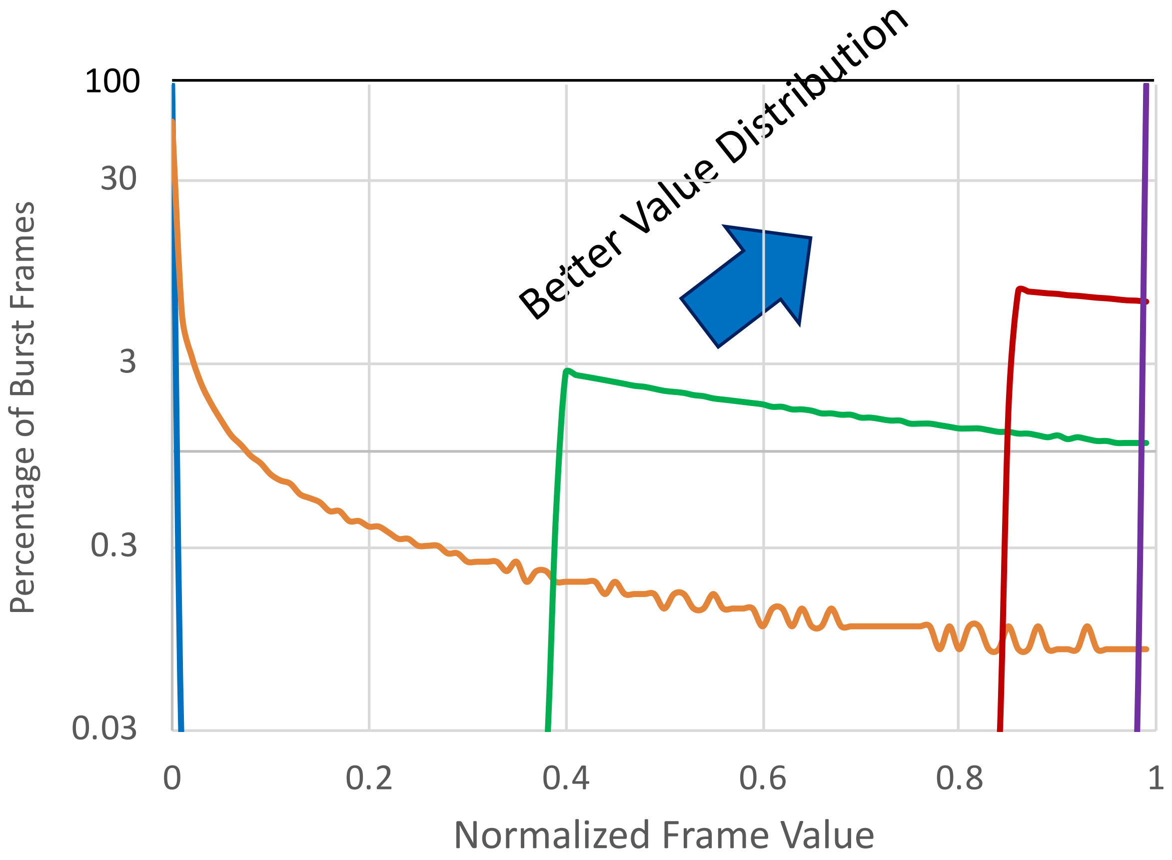


Value is represented as:

$$FrameValue = MaxValue \cdot e^{\frac{-Response\ Delay}{\tau}}$$



RTS Guarantees Statistics for Frame Value



Guarantee Invocation Rates

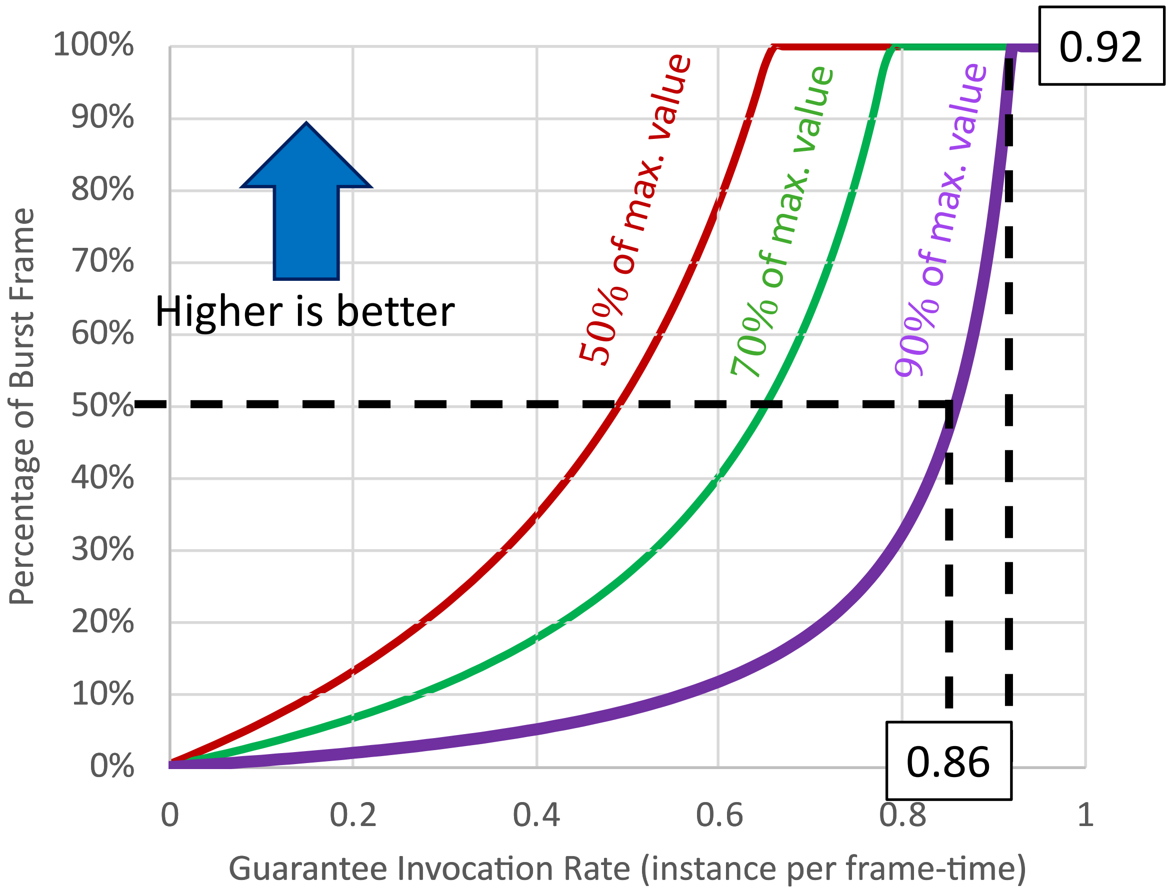
- $A_{RTS} = 0.0\ invocation/ft$
- $A_{RTS} = 0.1\ invocation/ft$
- $A_{RTS} = 0.3\ invocation/ft$
- $A_{RTS} = 0.9\ invocation/ft$
- $A_{RTS} = 1.0\ invocation/ft$

ft = frame-time = 1/30 sec

✓ High guaranteed invocation rate → high value

✓ Guarantee Statistics for Frame Value

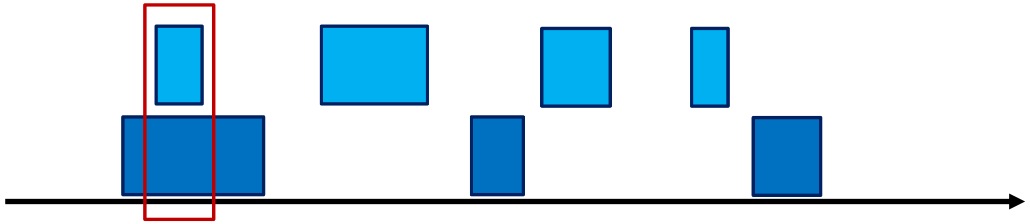
Rational Design for Value



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

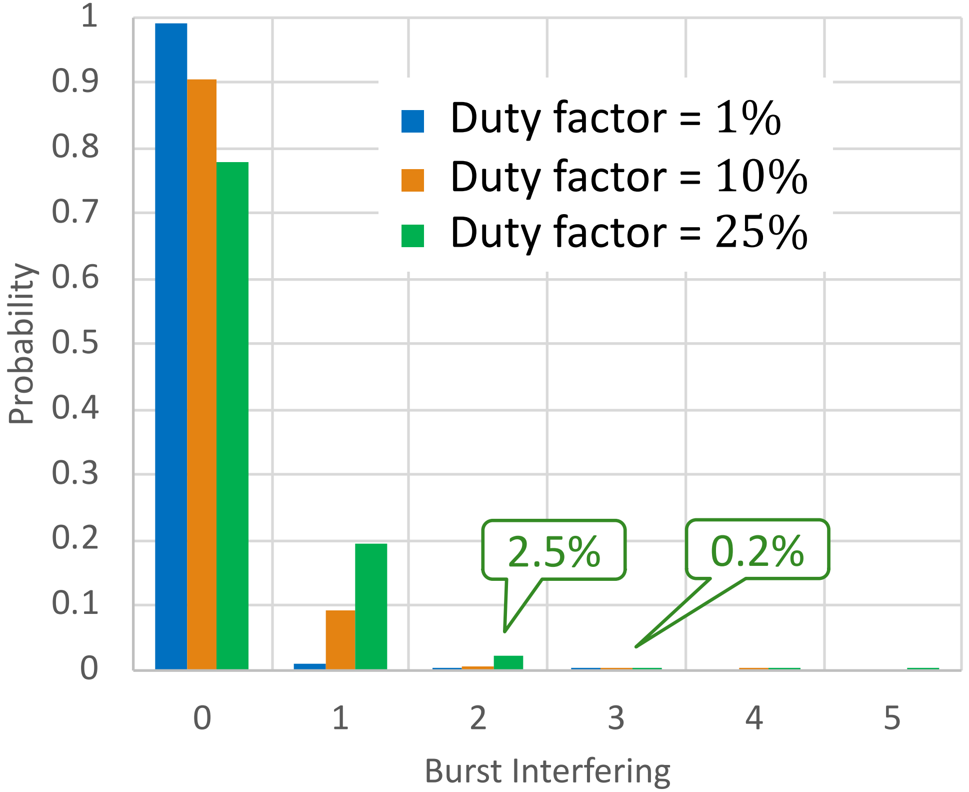
✓ Enable application to engineer the value distribution

RTS with Burst Interference

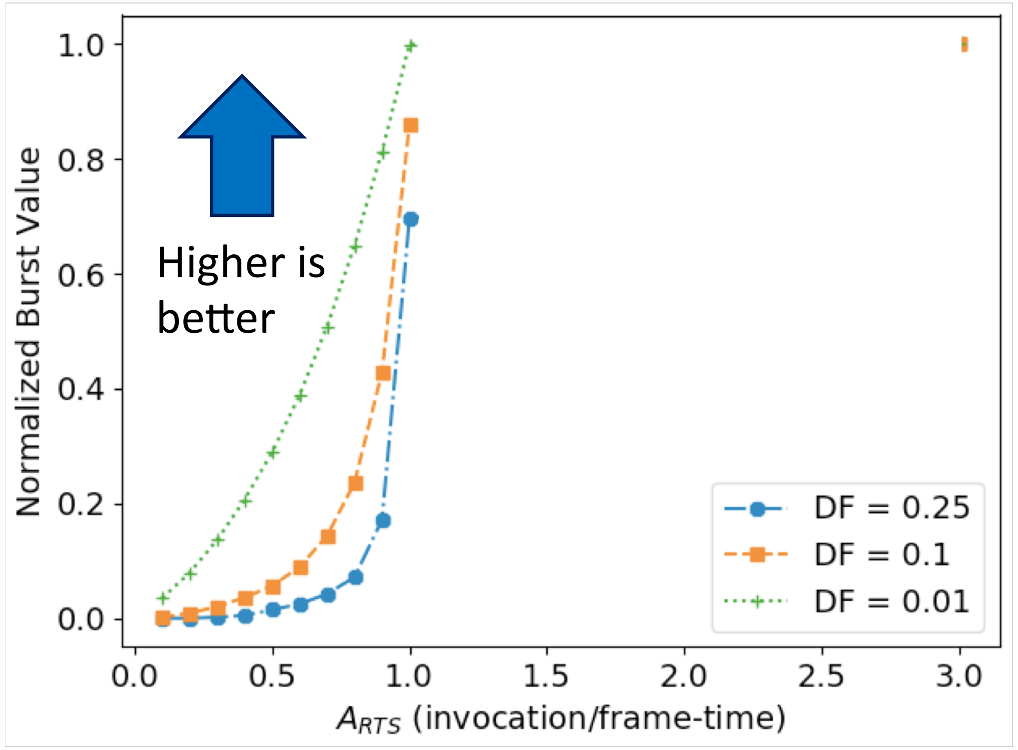


duty factor ↔ burst interference

✓ **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

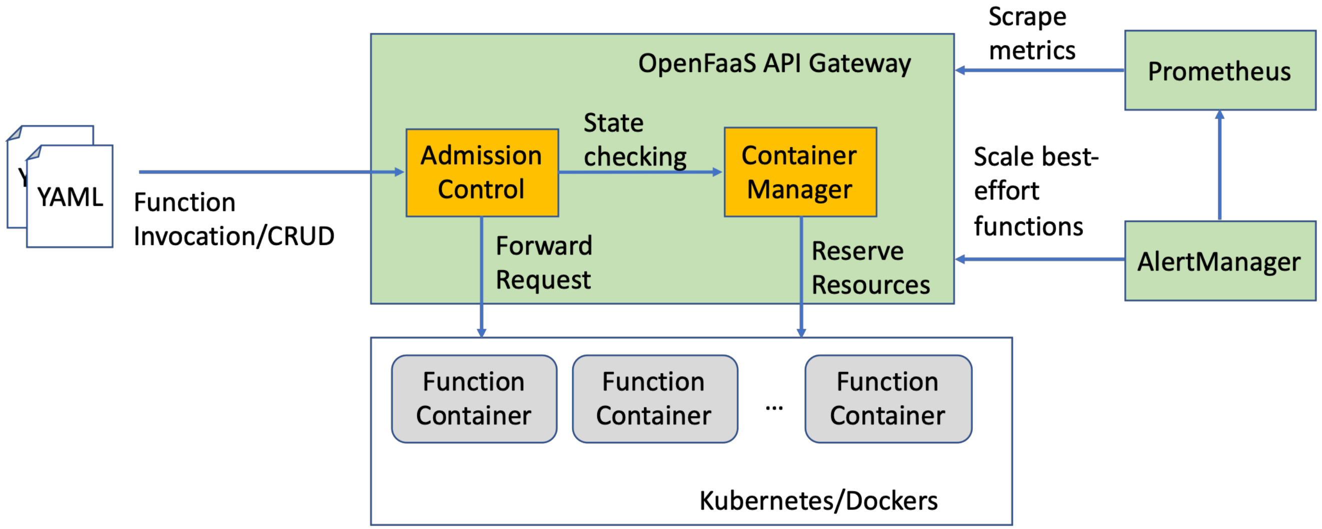
```
<function name>
    lang: <Language of function body>
    handler: <Location of function body>
    image: <Docker image reference>

    realtime: <Guaranteed invocation rate>

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

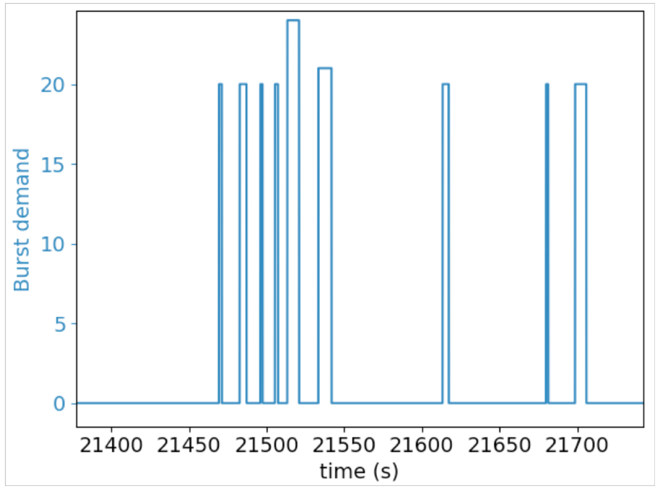
Working prototype

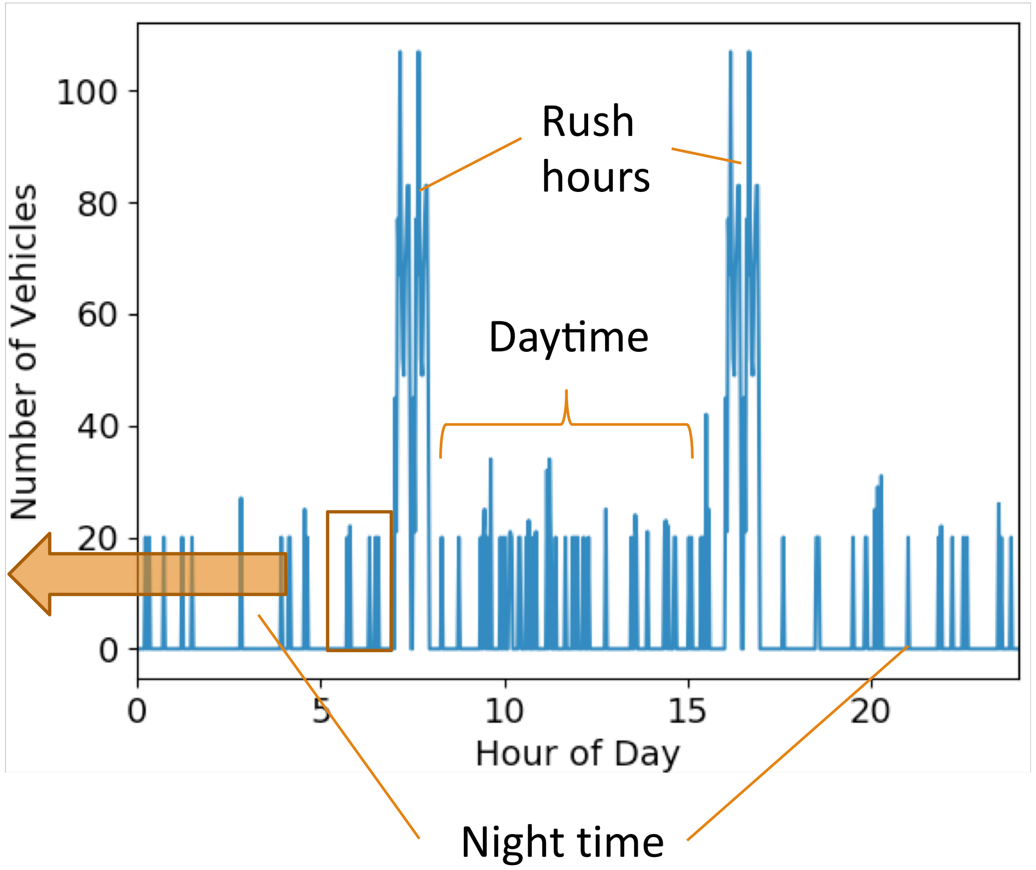
- Leverage OpenFaaS
- Admission control at function registration



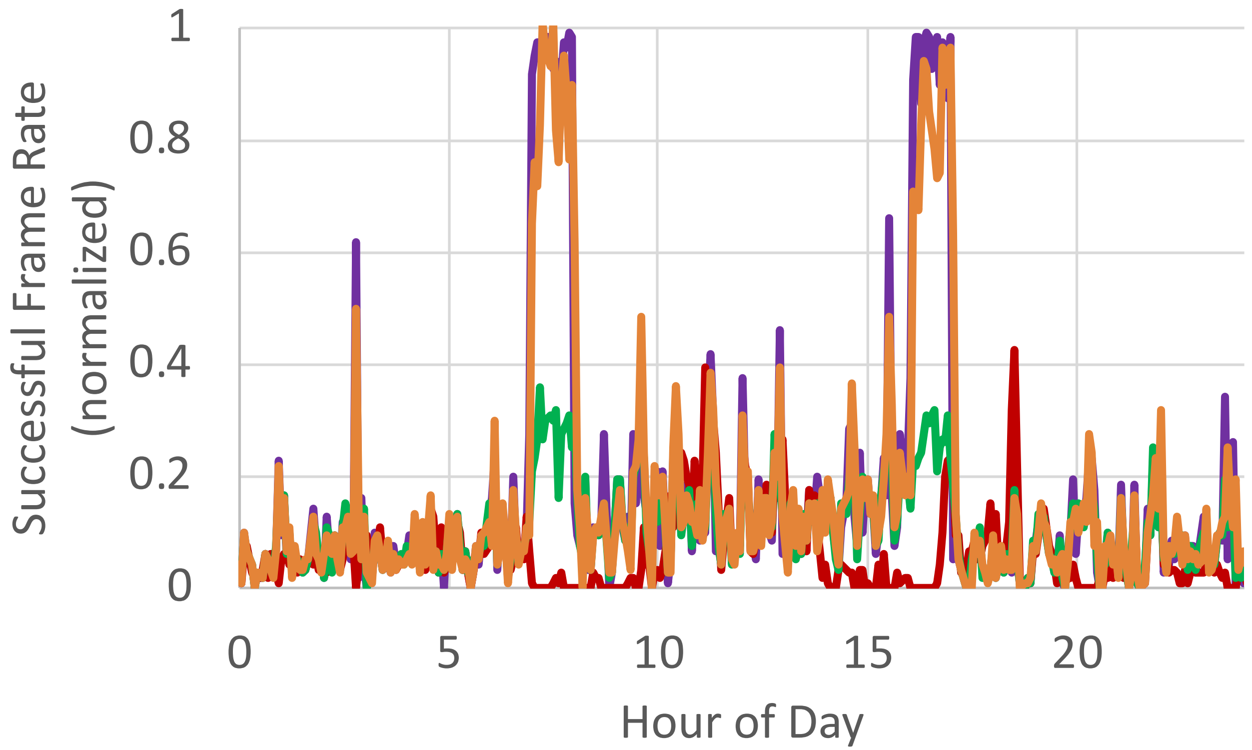
Case Study: Traffic Monitoring

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





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

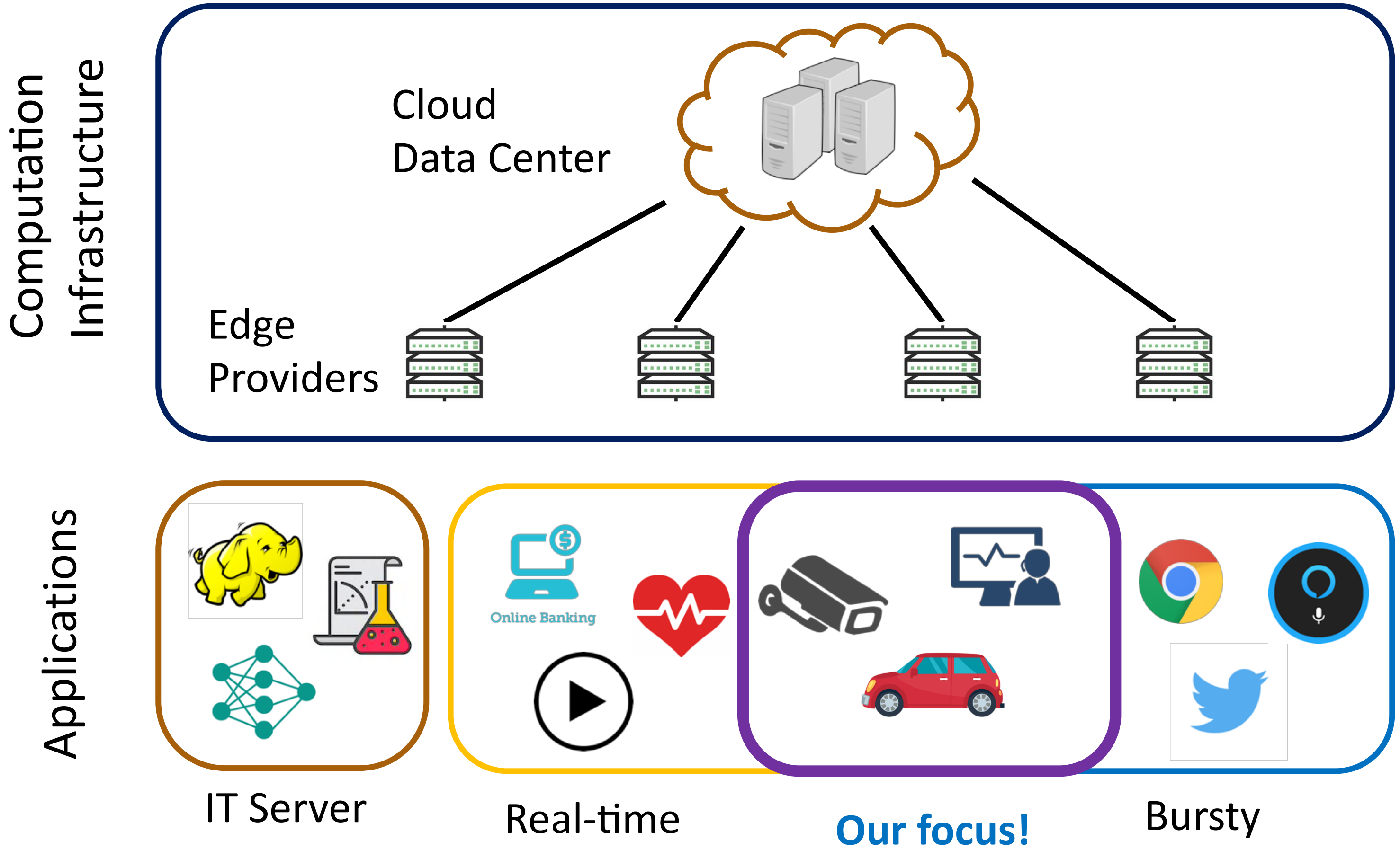
- Current serverless interface cannot support real-time, bursty applications.
- Real-time serverless = Serverless + Guaranteed invocation rate.
  - Guarantee statistics for value.
  - Enable rational design.
- A prototype shows timely response for a video monitoring application
- Future work
  - Efficient implementation for RTS interface
  - Explore the benefits of RTS interface for other application classes

Q&A

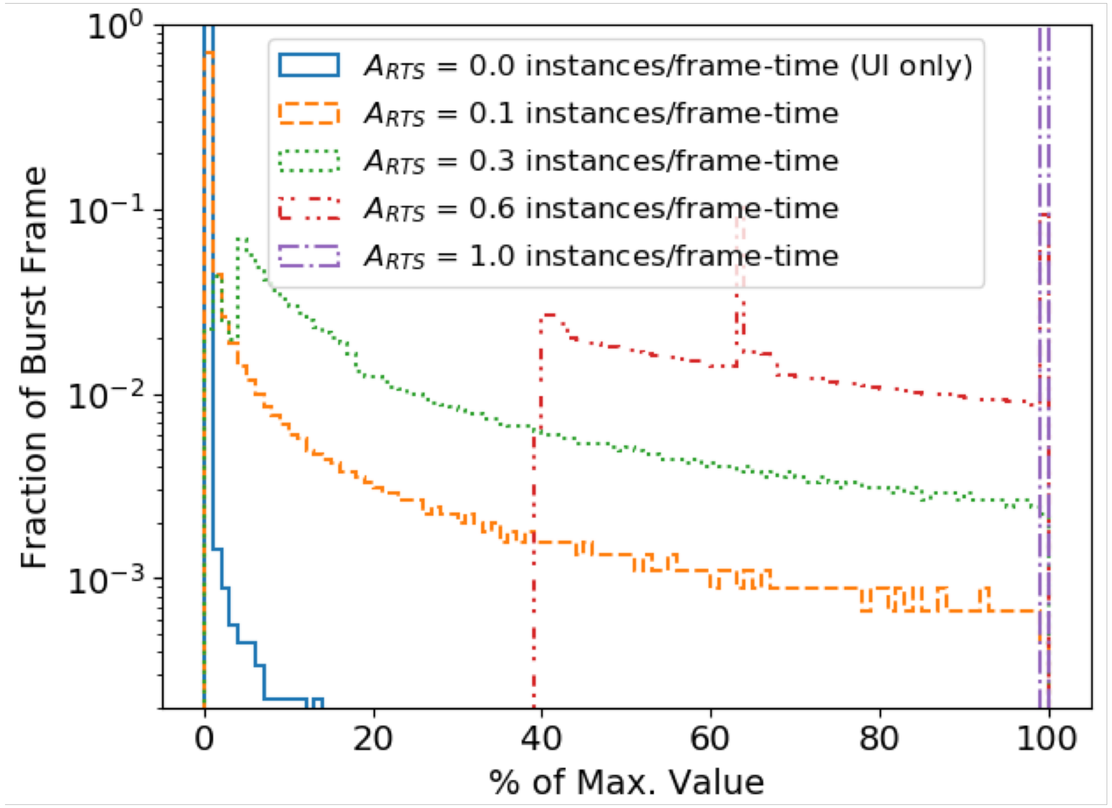
Acknowledgement. This work supported by National Sci- ence Foundation Grants CNS-1405959, CMMI-1832230, and CNS-1901466. We gratefully acknowledge support from Intel, Google, and Samsung.

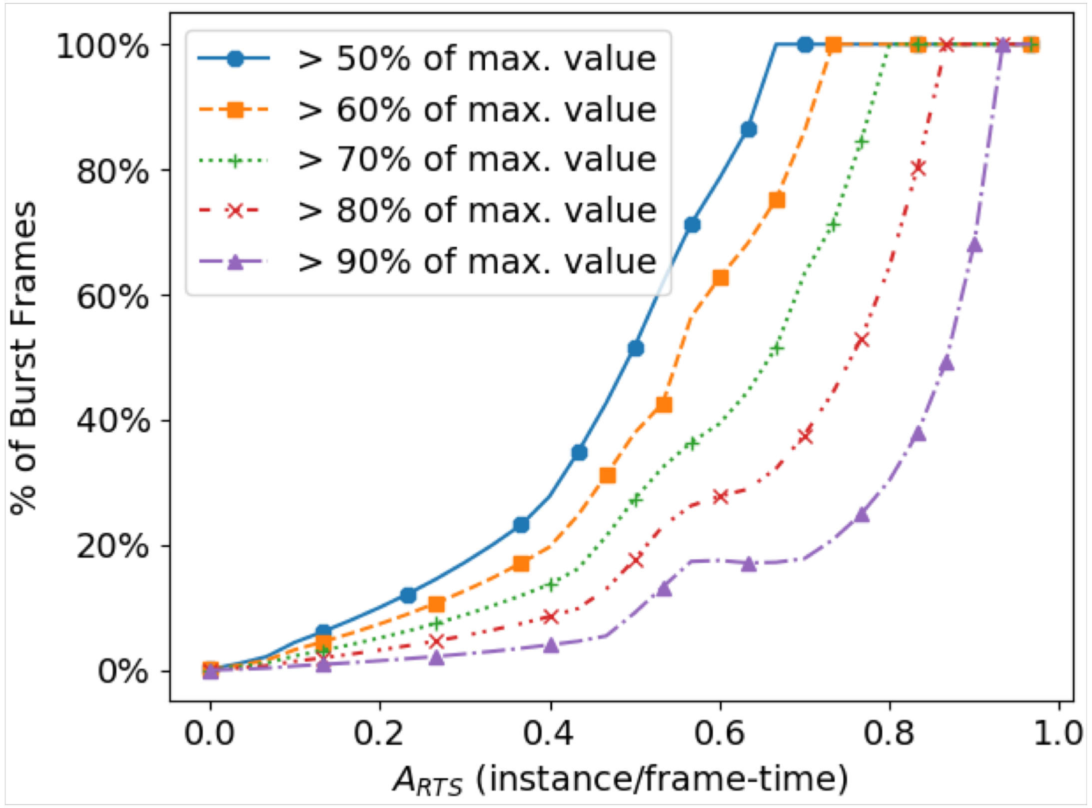
Backup Slides

A Big Picture

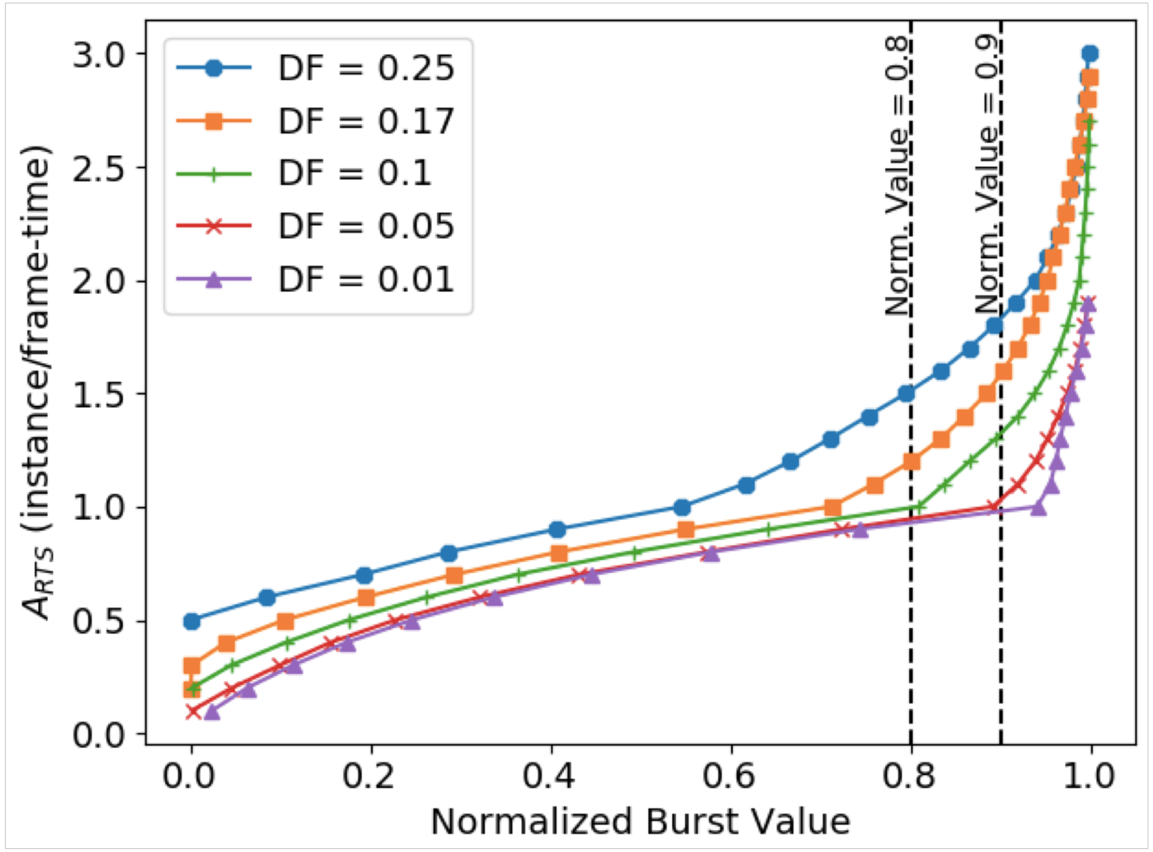


Validate Analytical Results with Simulation





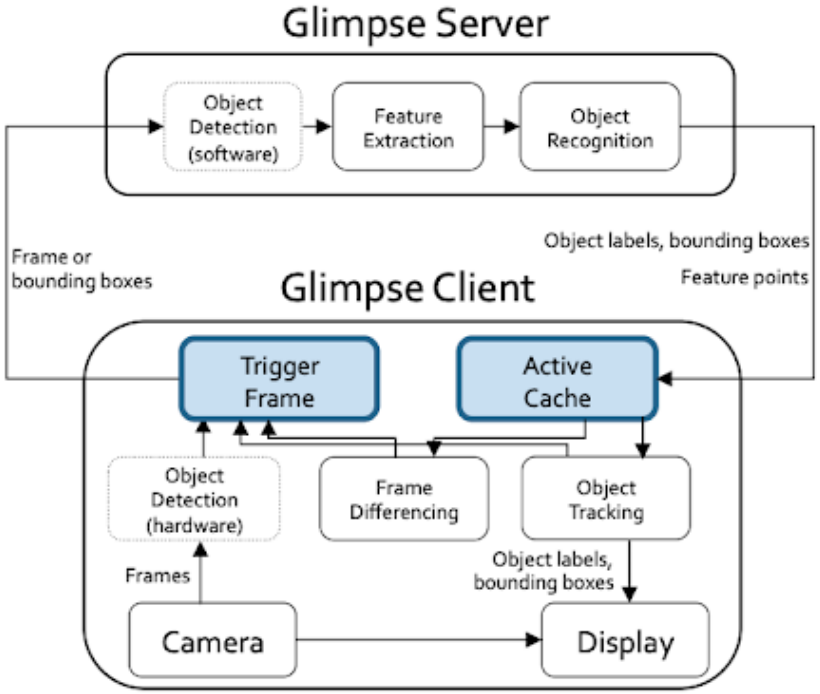
Supporting Multiple Applications



Case Study: Statistics

	Burst Duration (frames)		Burst Height	
	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

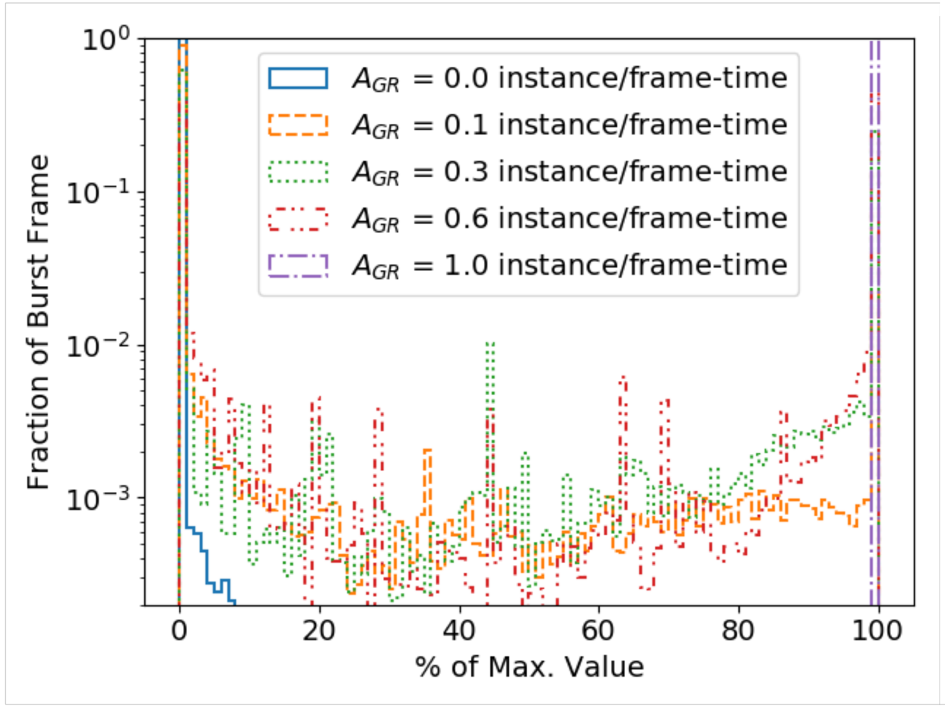
Burst Statistics



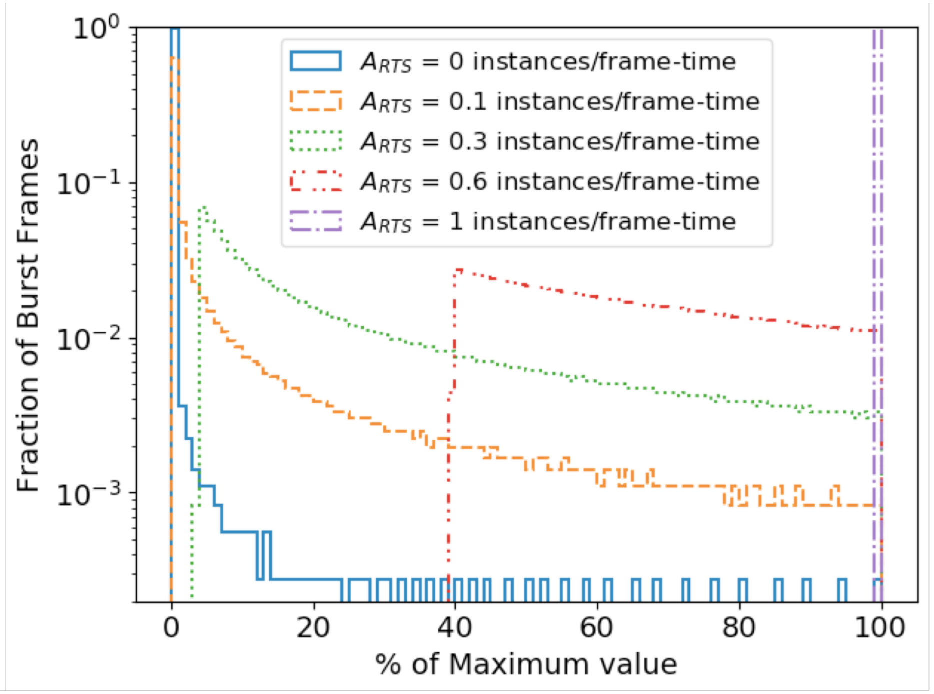
Glimpse Pipeline Architecture[1]

[1] Tiffany Yu-Han Chen et. al., Glimpse: Continuous, Real-time Object Recognition on Mobile Devices, SenSys'15

RTS for Video Analysis



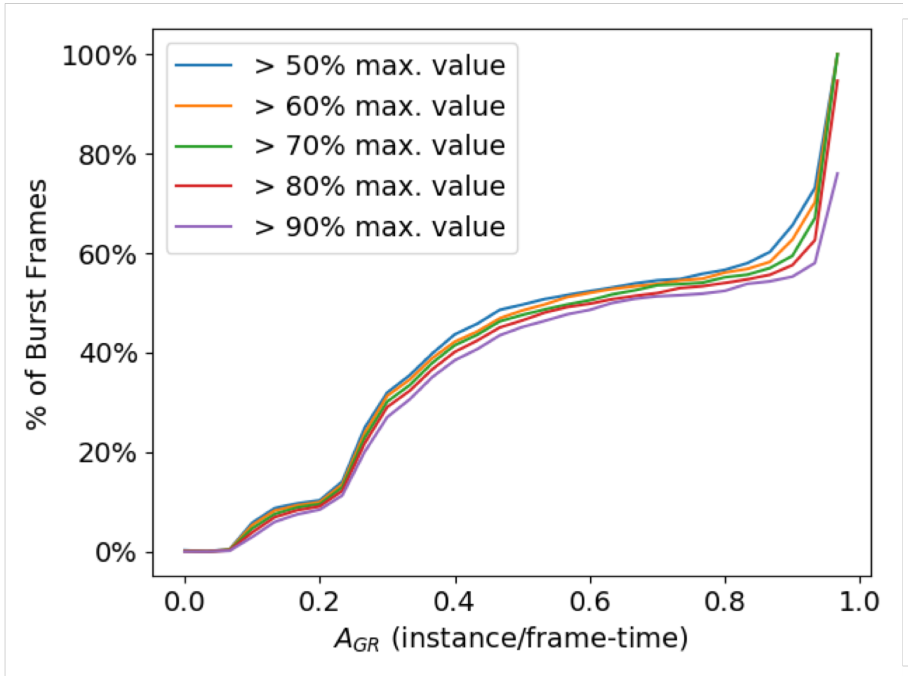
Realistic Workload



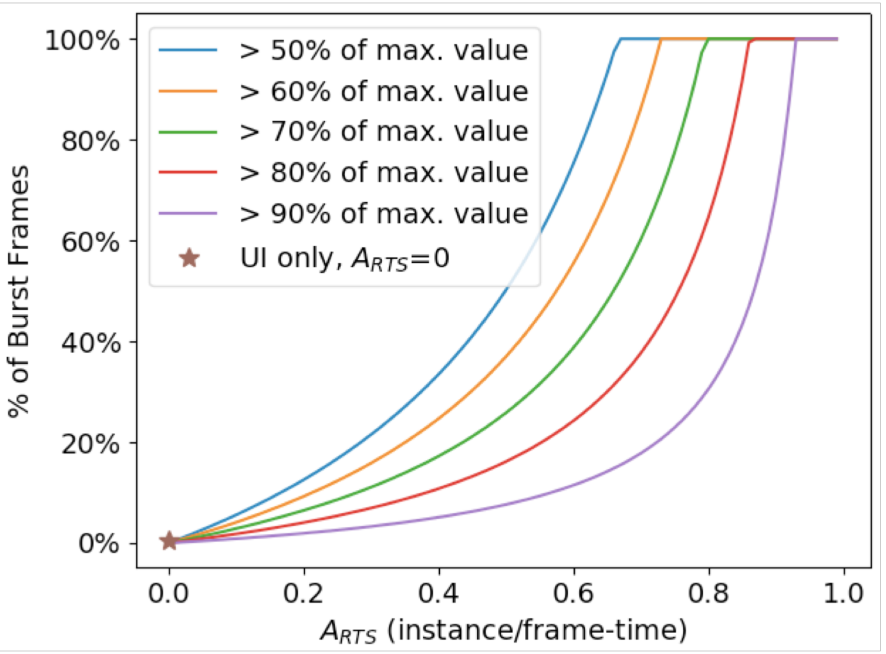
Synthetic Workload

✓ Guaranteed invocation rate enables value guarantee

RTS for Video Analysis



Realistic Workload



Synthetic Workload

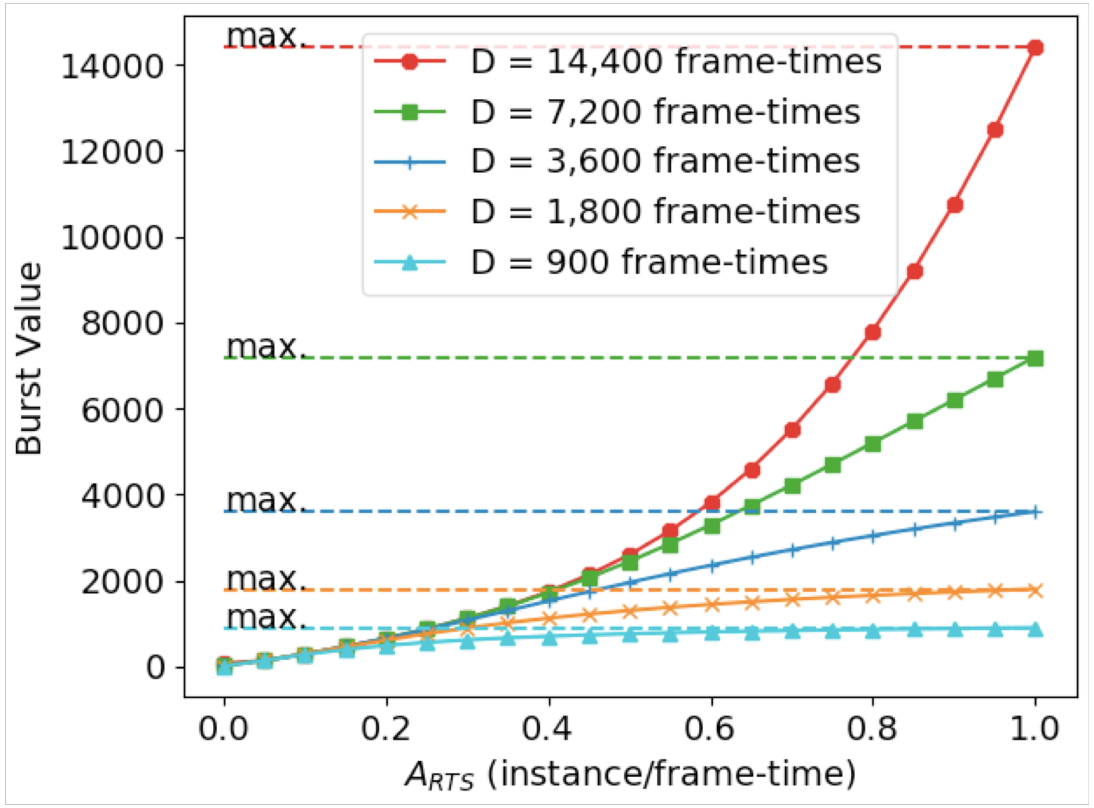
✓ Enable rational design for value guarantee

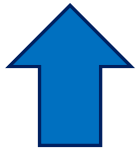
Robust against Burst Shape

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

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

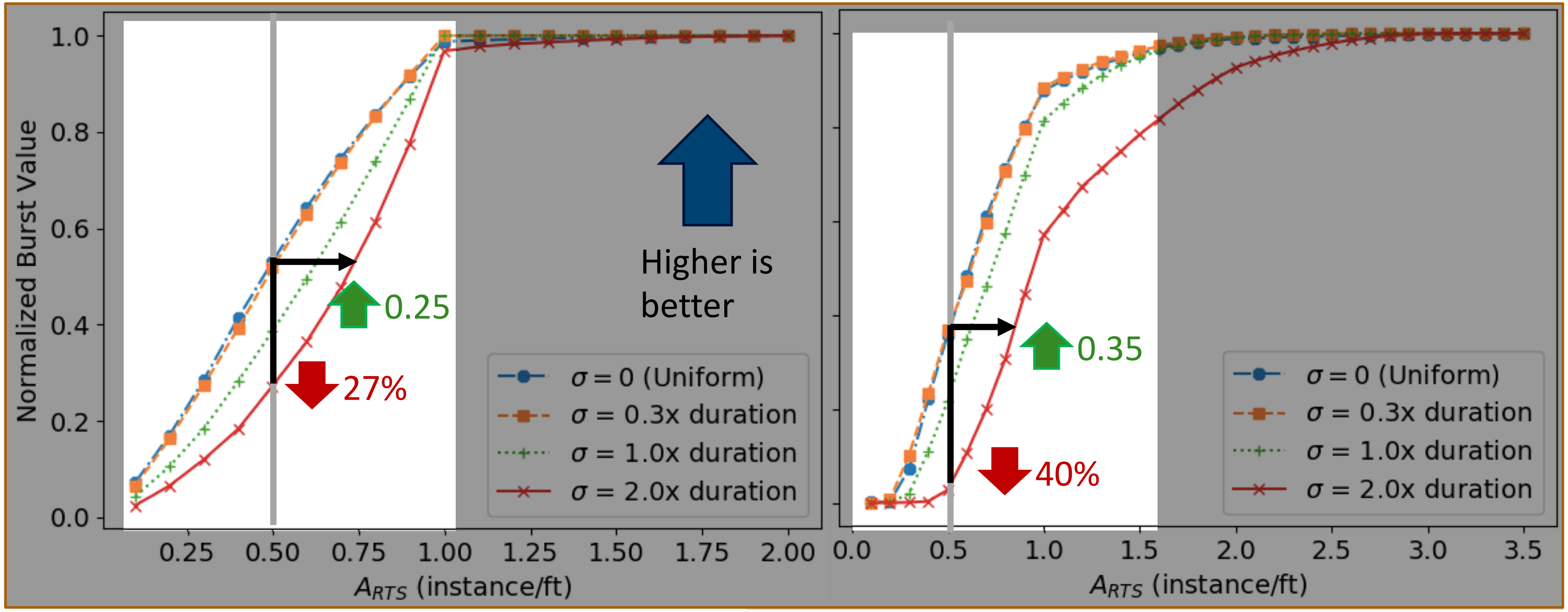
✓ Maximum value is achieved at $A_{RTS} = 1$, regardless burst shape





Robust against Burst Variability

Change variability by varying burst duration standard deviation



Duty factor = 1% Duty factor = 25%

Variability causes value drop
Higher duty factor creates more damage

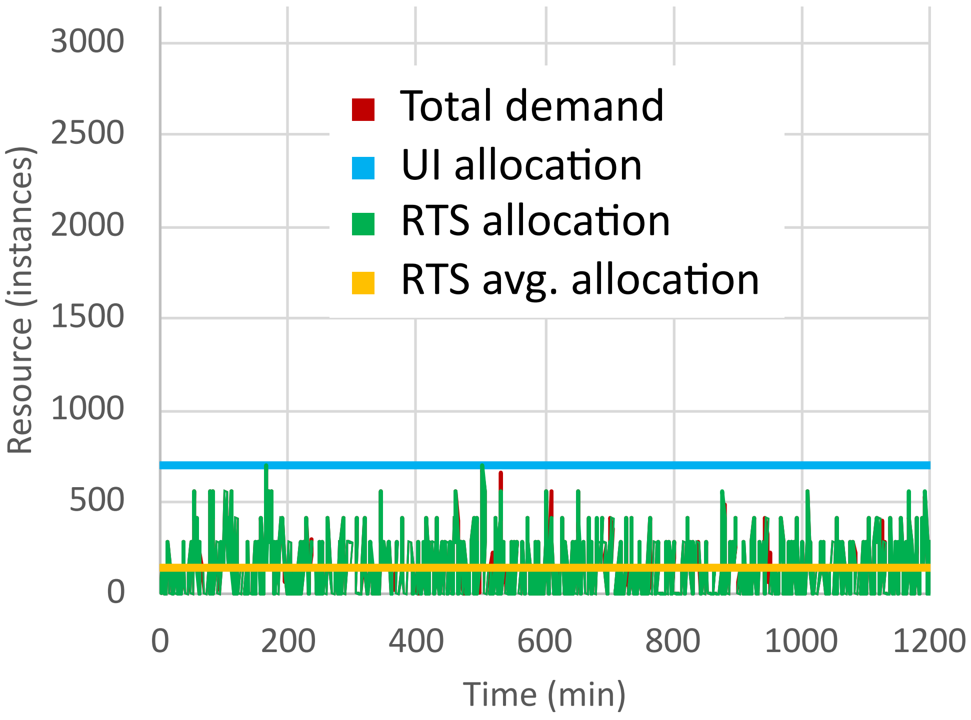
- ✓ **Increase A_{RTS} cancels variability effect**
- ✓ **RTS value can be maintained for wide burst variance**

Multiple Real-time, Bursty Apps.

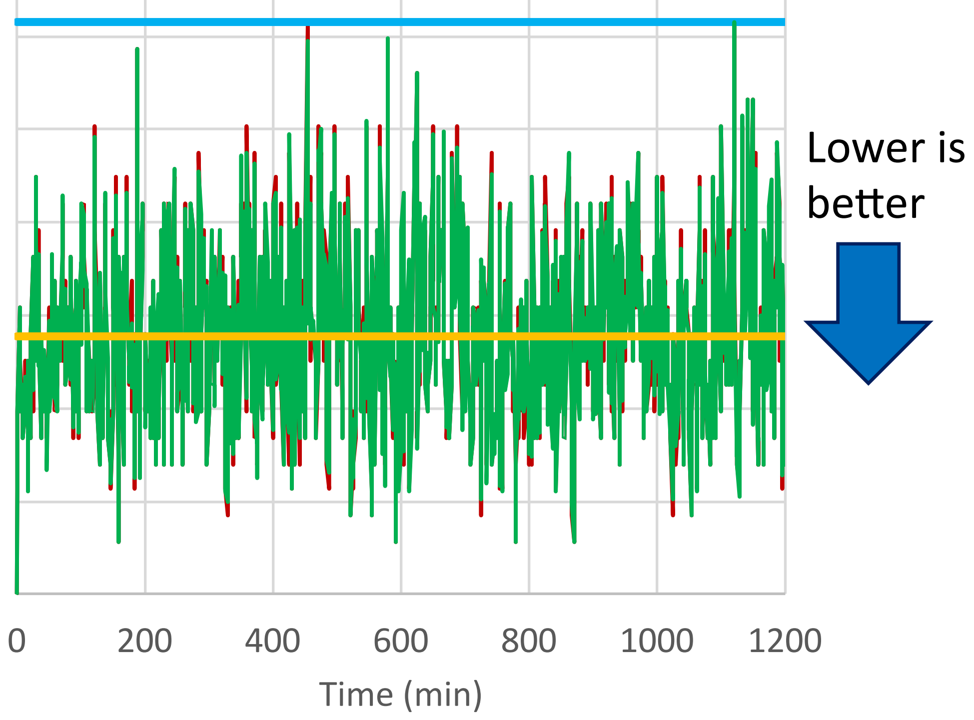
✓ RTS resource cost scales with actual demand

✓ RTS resource consumption is 2.2x to 5x lower than UI

✓ RTS helps cloud provider save resource to serve more applications



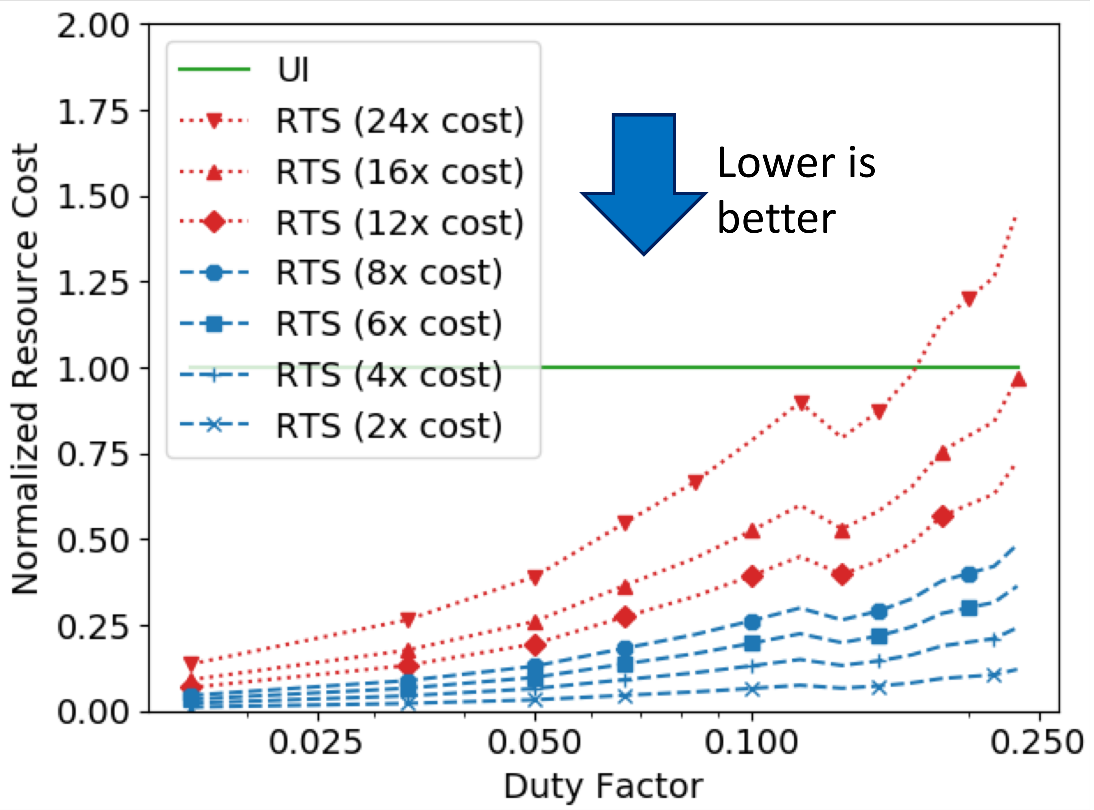
10 Apps



100 Apps

Application Cost: UI vs. RTS

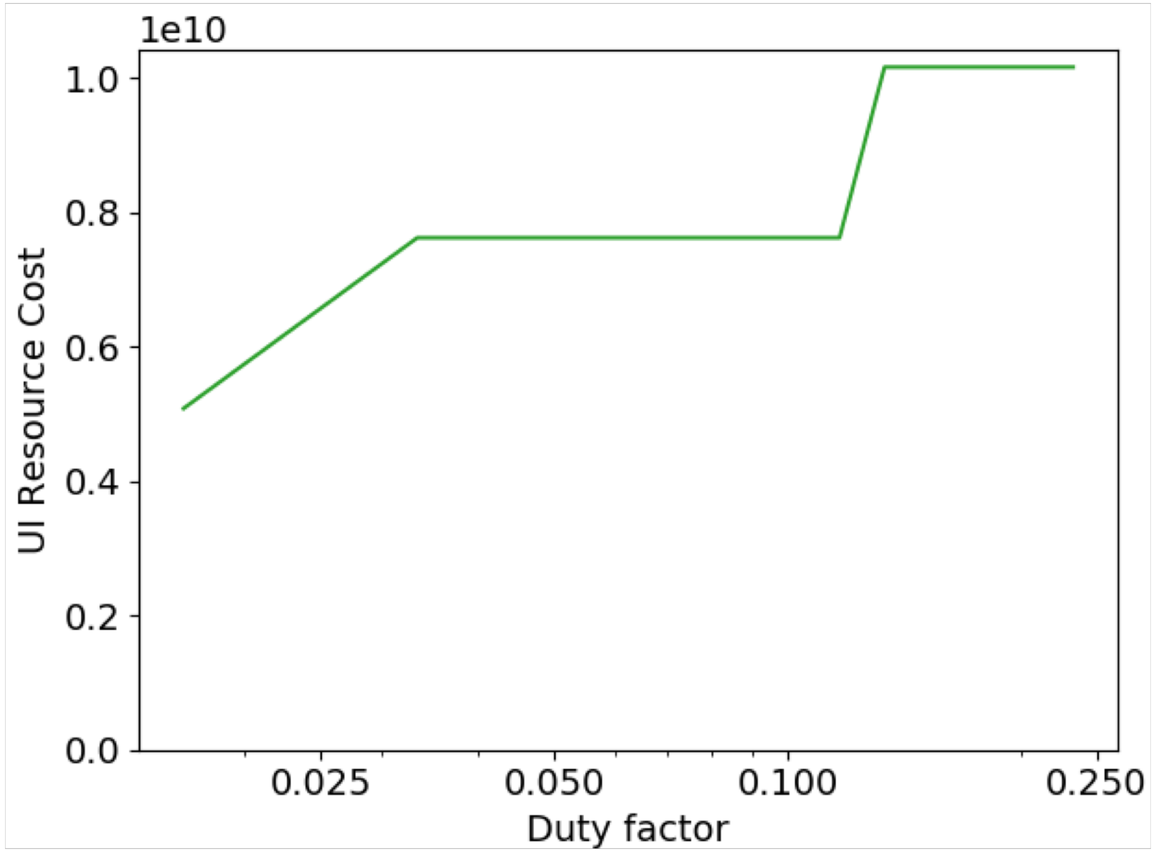
- Resource cost for maximizing burst value with different duty factors
- Vary RTS vs. UI cost ratio



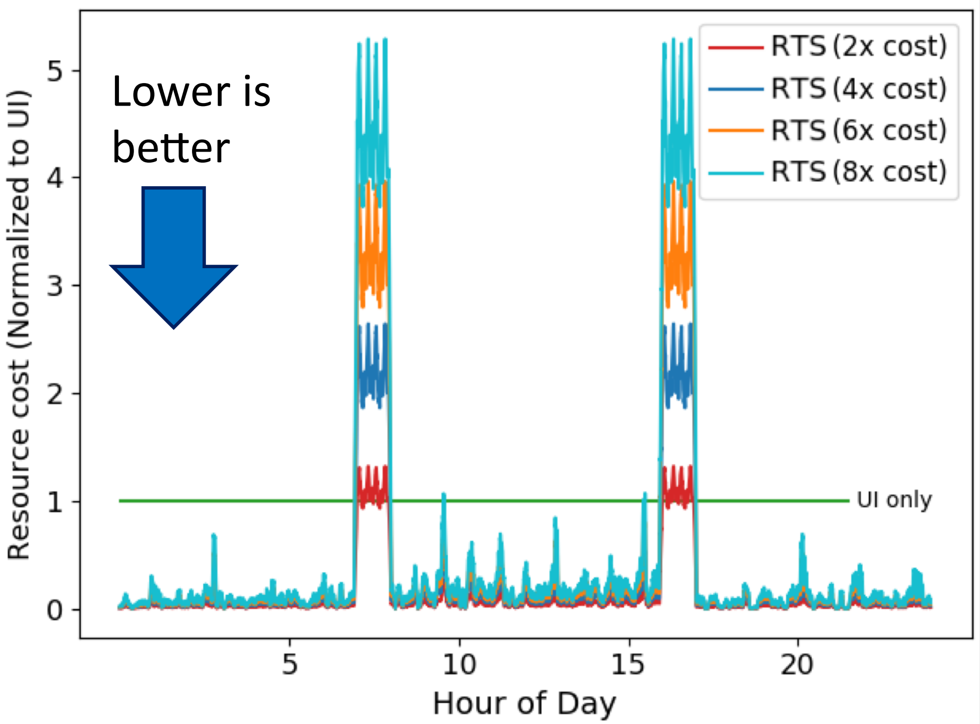
✓ RTS resource value per unit cost is 16-24x higher than UI

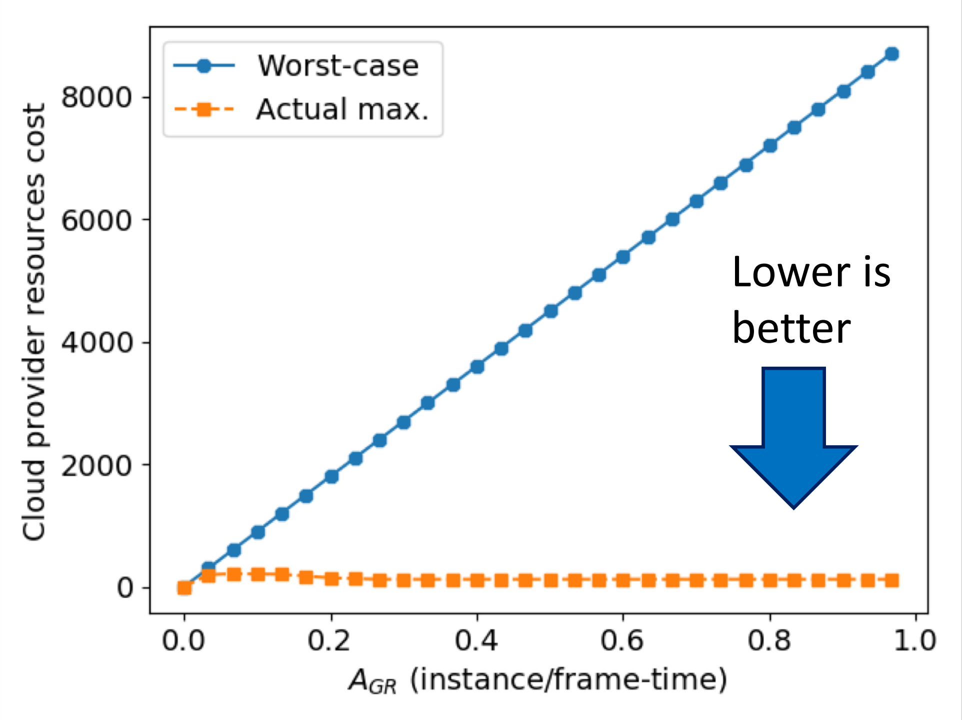
✓ RTS enables low cost solutions for real-time, bursty applications

UI Cost at Different Duty Factors



Resource Cost





✓ RTS is 2x to 8x cheaper than UI

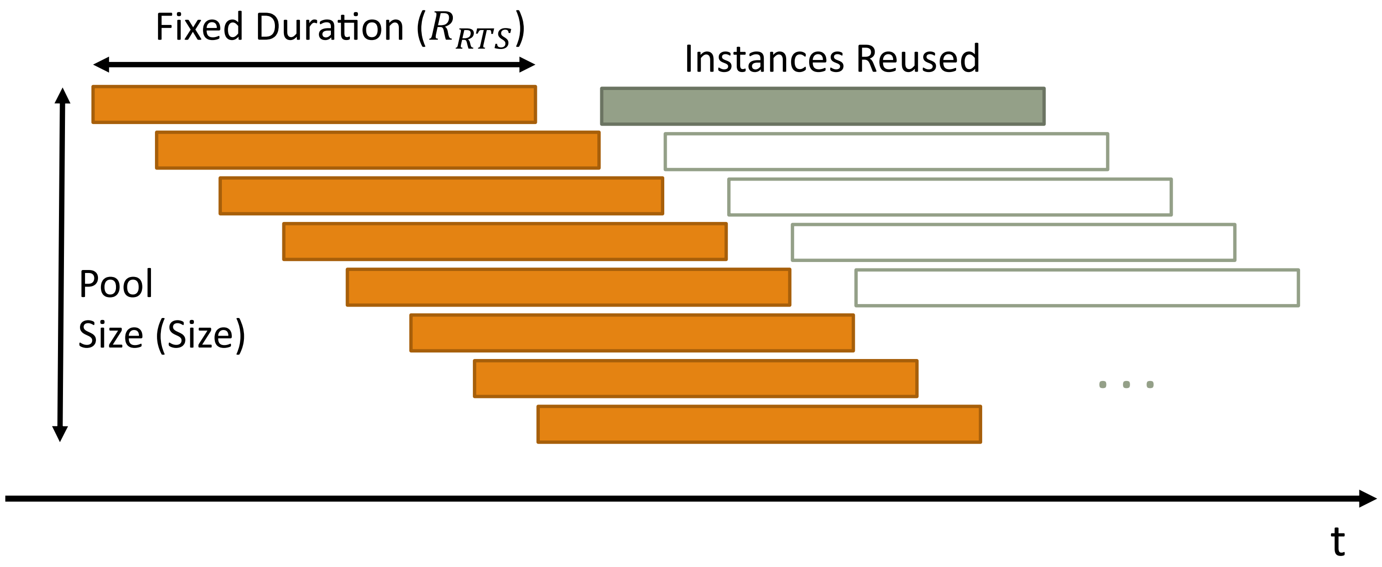
✓ Actual resource requirement is 70x lower than the worst case

RTS Implementation Feasibility

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

RTS pool capacity is bounded

$$\mathrm{C}(\mathrm{A}_{\mathrm{RTS}}, \mathrm{R}_{\mathrm{RTS}}) = \mathrm{A}_{\mathrm{RTS}} \cdot R_{RTS}$$



RTS Interface (Lambda Extension)

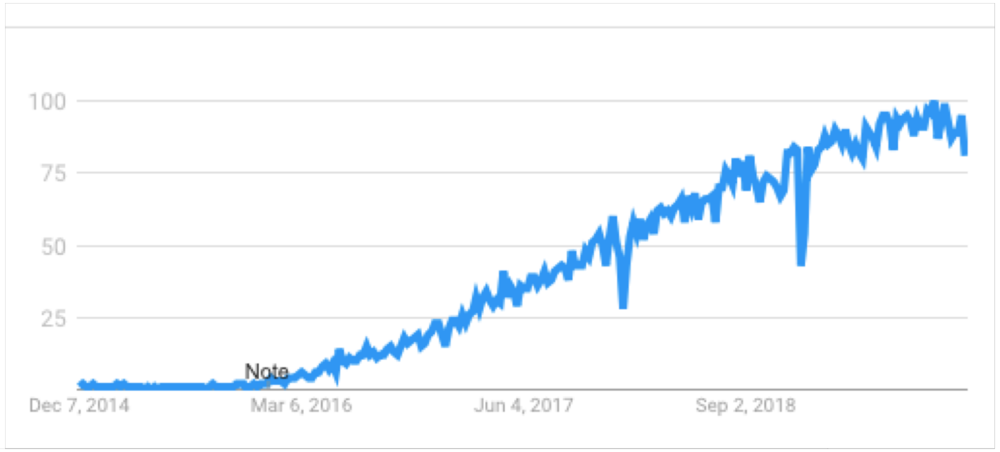
- Function description in YAML format

```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>
```

RTS Extension (annotates the `realtime` line)

Introduction

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