

Tackling Cold Start of Serverless Applications by Efficient and Adaptive Container Runtime Reusing

Kun Suo^{*}, Junggab Son^{*}, Dazhao Cheng[†], Wei Chen[‡], Sabur Baidya[§]

^{*}Kennesaw State University, [†]University of North Carolina at Charlotte

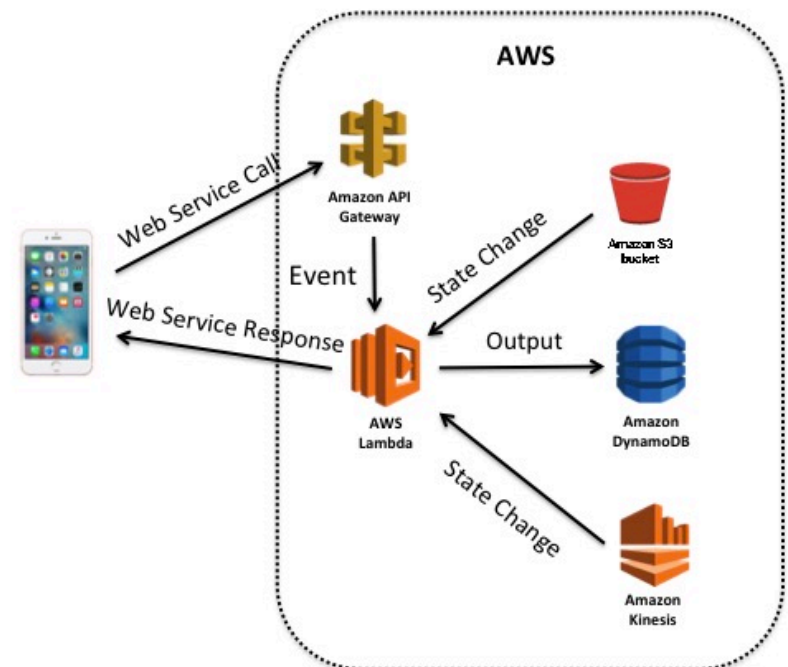
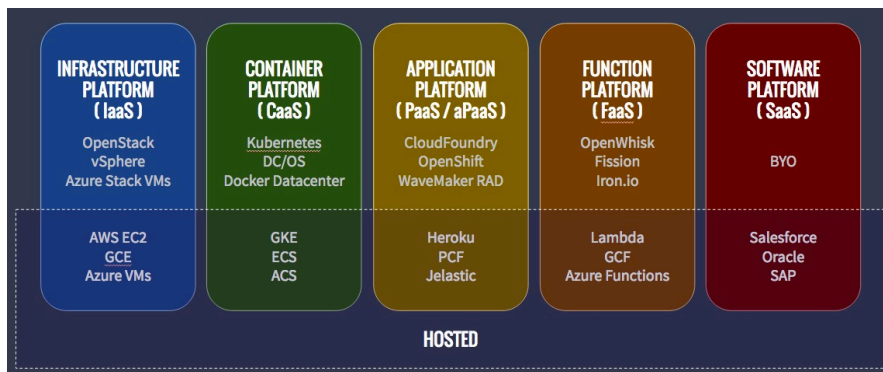
[‡]Automotive Vehicle Group, Nvidia, [§]University of Louisville



**IEEE
CLUSTER 2021**
Portland, OR, USA • 7–10 September

The Rise of Serverless Applications in the cloud

- **Serverless computing** is becoming increasingly important to on-demand, elastic, and cost-effective cloud services
- Key benefits of serverless applications
 - ✓ High performance
 - ✓ High scalability
 - ✓ Built-in availability
 - ✓ Fault tolerance



Serverless at the Edge



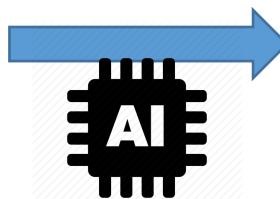
idle



Event
Trigger



Run ML/
DL code



Size

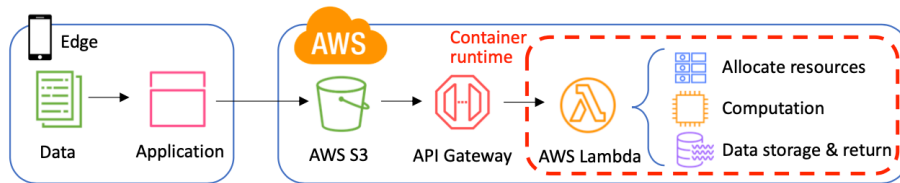
Speed

Plate

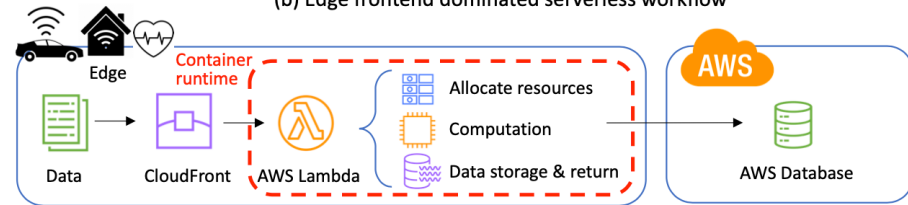
...

Serverless Pro vs Con

(a) Cloud backend dominated serverless workflow



(b) Edge frontend dominated serverless workflow



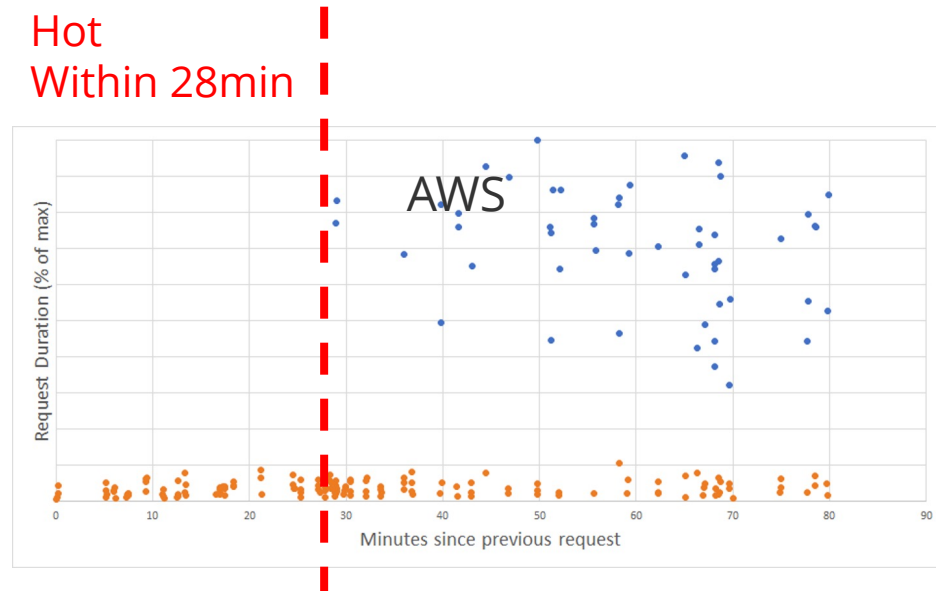
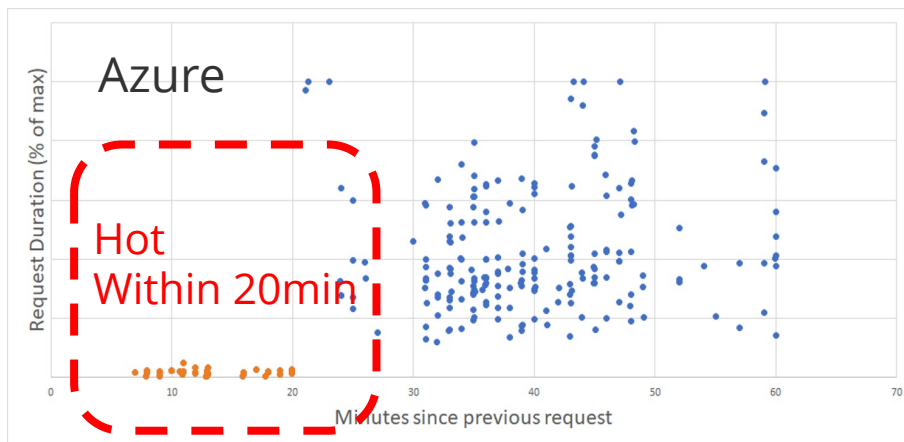
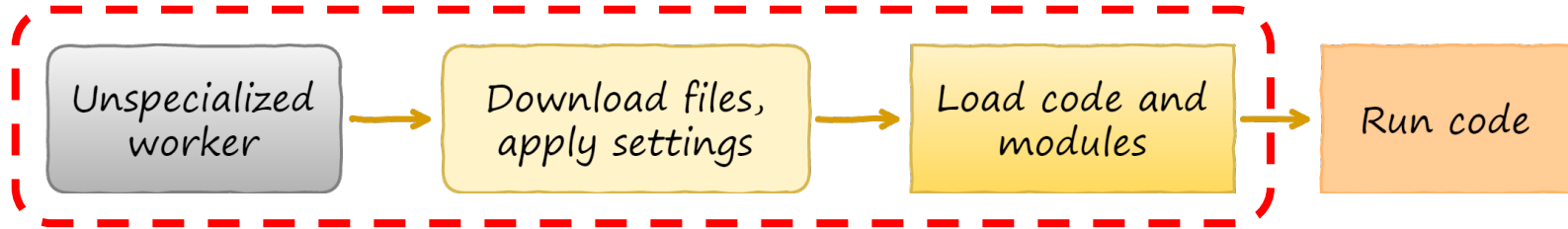
Benefits:

- Drop cost
- Fast Dev/deployment
- More security
- Microservice support
- Auto scale
- ...

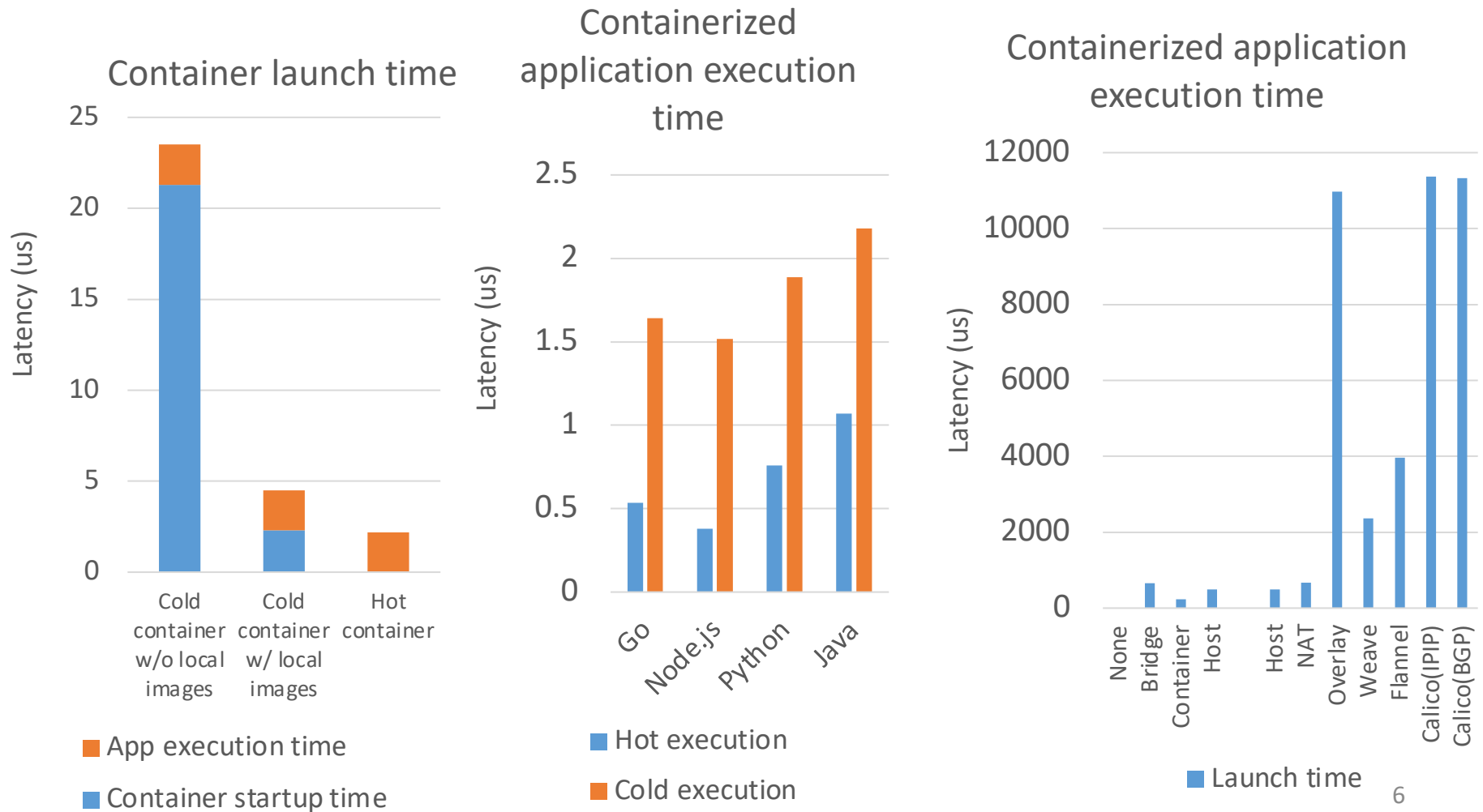
Drawbacks:

- Not good for long time apps
- Deep dependency on platform
- Cold start
- Lack of debugging or monitoring
- ...

Cold Start Impact



Motivation examples of cold start in serverless applications



AWS Lambda Cold Start Demo

Python:
http request return
random number

Measure the
request latency

The screenshot displays the AWS Lambda Management Console for a function named 'test'. The interface includes a top navigation bar with the AWS logo, 'Services', 'Resource Groups', and user information. The main content area shows the function's execution results. A 'Test' button is highlighted, indicating the function has been executed. The results section shows a JSON response: `{ "statusCode": 200, "number": 92 }`. Below this, a 'Summary' table provides details about the execution, including the Code SHA-256, Request ID, Duration (0.38 ms), Billed duration (100 ms), Resources configured (128 MB), and Max memory used (23 MB). The 'Log output' section shows the logging calls in the code, including the start, end, and report of the function execution, with the report showing the duration and memory usage.

test

Throttle Qualifiers Actions test Test Save

The section below shows the result returned by your function execution.

```
{
  "statusCode": 200,
  "number": 92
}
```

Summary

Code SHA-256	Request ID
sGK910fBuTtqRiLDvyMmBh5WRbACHBR3lwm4fy/EVEQ=	85bd5671-6a88-4efb-ba3a-4d677a1007df
Duration	Billed duration
0.38 ms	100 ms
Resources configured	Max memory used
128 MB	23 MB

Log output

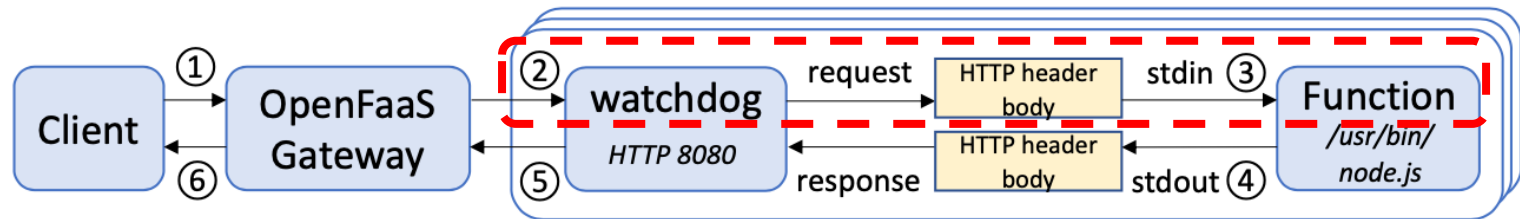
The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: 85bd5671-6a88-4efb-ba3a-4d677a1007df Version: $LATEST
END RequestId: 85bd5671-6a88-4efb-ba3a-4d677a1007df
REPORT RequestId: 85bd5671-6a88-4efb-ba3a-4d677a1007df Duration: 0.38 ms Billed
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 23 MB
```

00:01 Finish

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Analysis of cold start overhead



- OpenFaaS, an event-driven functions and microservices platform
- We added timestamps in source code and record six moments during the workflow path
- We found that function initiation time (2→3) dominates the total latency

Containerized Application Characteristics

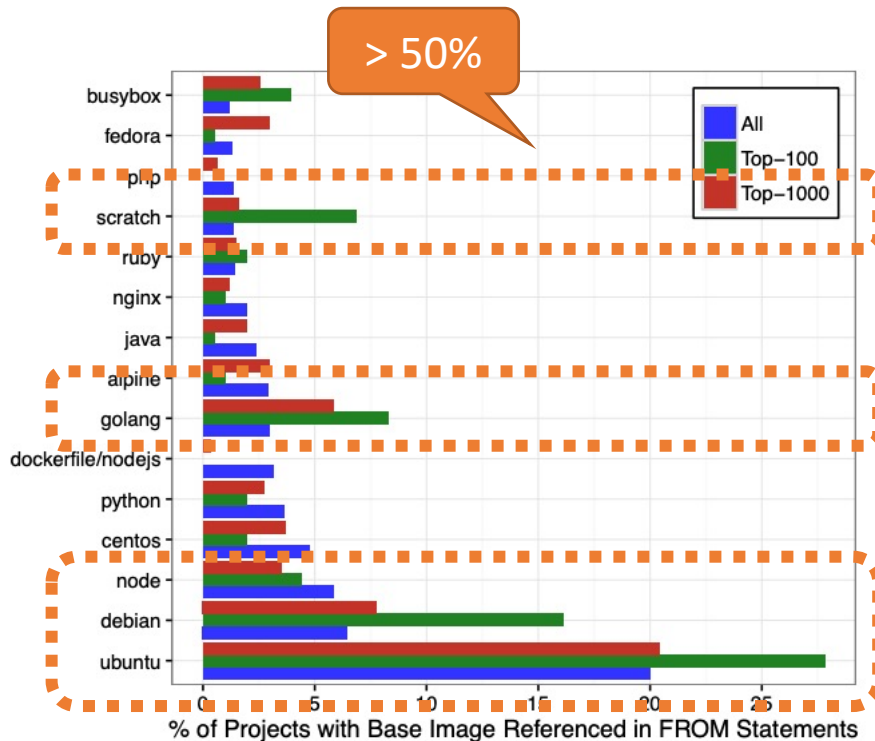


Fig. 5. Percentage of usage of 15 most commonly used base images

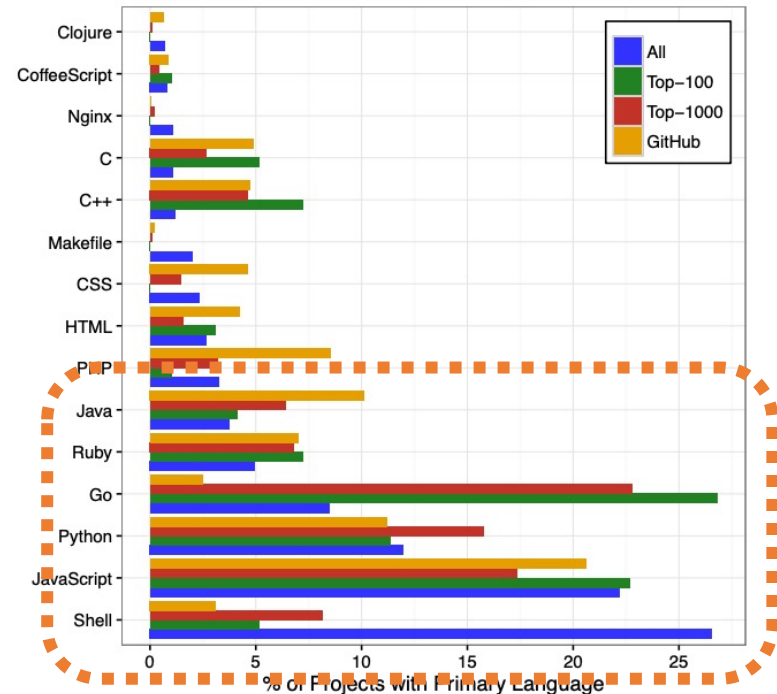


Fig. 4. Distribution of top 15 languages in our dataset

> 70%

Related Work

- **Accelerating Container Startup**

- ✓ Slacker [FAST-16], SAND [ATC-18], SOCK [ATC-18], Catalyzer [ASPLOS-20]
- ✓ Industry: periodically booting the containers, reducing artifact program size, prefetching the hot data

✗ Not focus on container runtime

- **Resource Management and Allocation**

- ✓ Elastic resource management: Huang [HPDC-19], X-containers [ASPLOS-19], Mohan [Hotcloud-19]
- ✓ Resource prediction and dynamic allocation: Emars [Cloud-18]

✗ Only effective within a protected domain, not serverless

- **Optimizing Container Architecture**

- ✓ Seuss [EuroSys-20], Iron [NSDI-18], Lloyd [IC2E-18], Nguyen [Serverless-19]

✗ Require changes To applications

Industry Practices

- **Alibaba:**

- ✓ New image format
- ✓ P2P network pair for data distribution
- ✓ Efficient decompression algorithm

- **Tencent**

- ✓ Deploys their functions based on lightweight virtualization
- ✓ New scheduling policies for active functions

- **AWS**

- ✓ Keep-alive policy that retains the resources in memory for 10x minutes after a function execution
- ✓ AWS Hyperplane create interfaces in advance for each serverless application instead of creating Virtual Private Cloud (VPC) for each function during execution

Our Approach: HotC

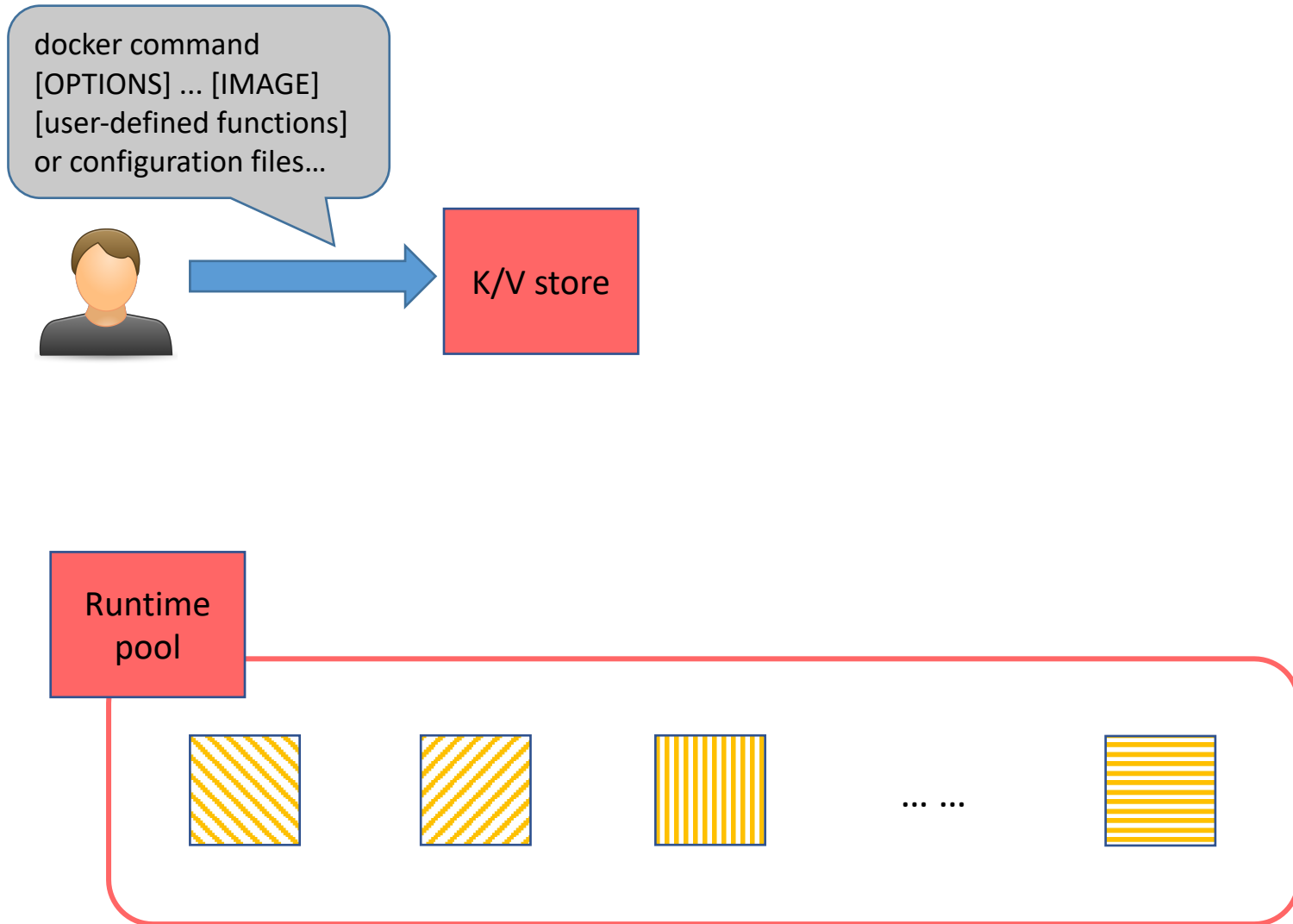
- **What is HotC?**

- ✓ An approach to leveraging lightweight containers and an efficient reuse mechanism to prevent the unnecessary cold start, excessive resource allocation, and reclaim

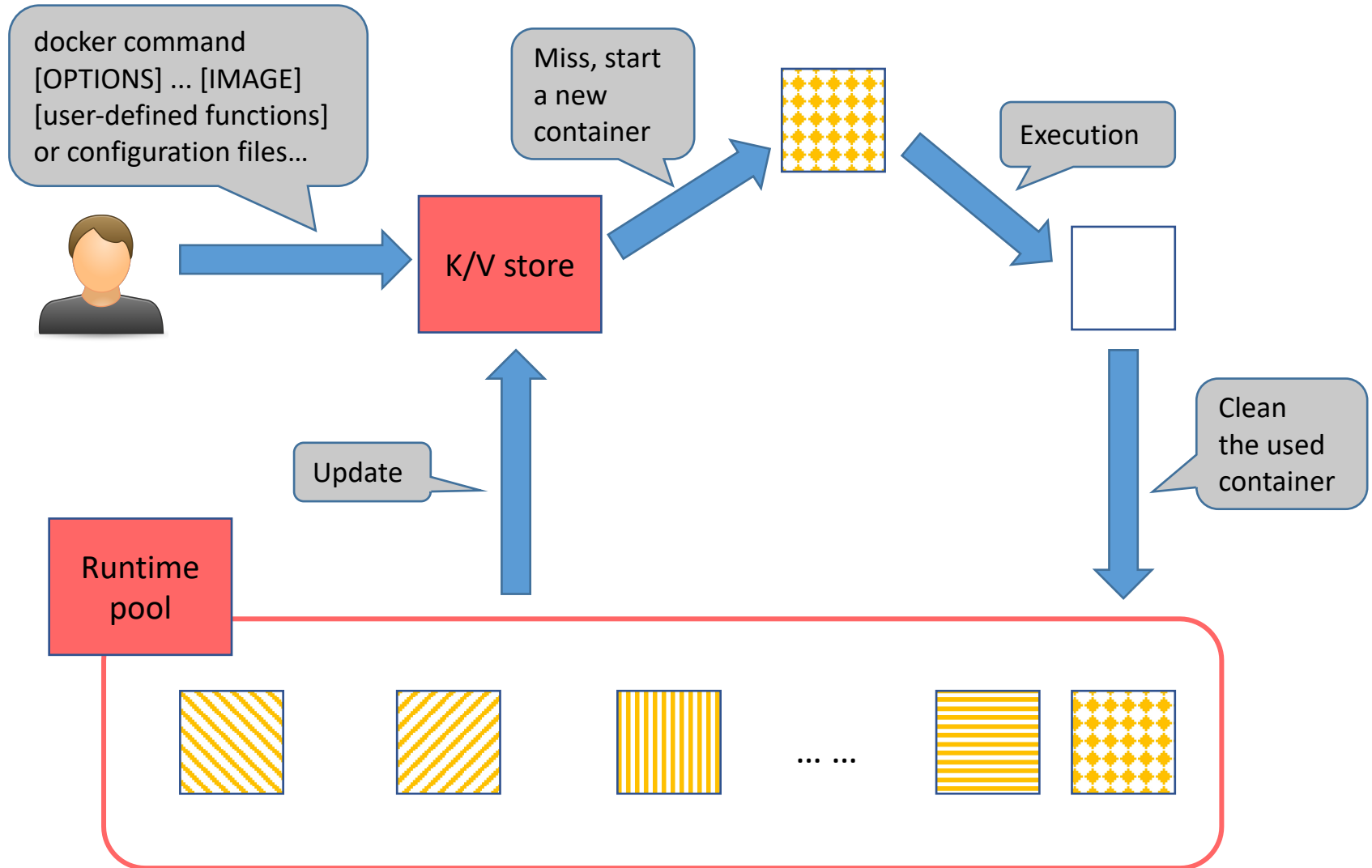
- **Key designs of HotC**

- ✓ A middleware between clients and backend servers
- ✓ Maintain a live container runtime pool and adaptively update the pool over time
- ✓ Reuse the same type of available container runtime to new requests

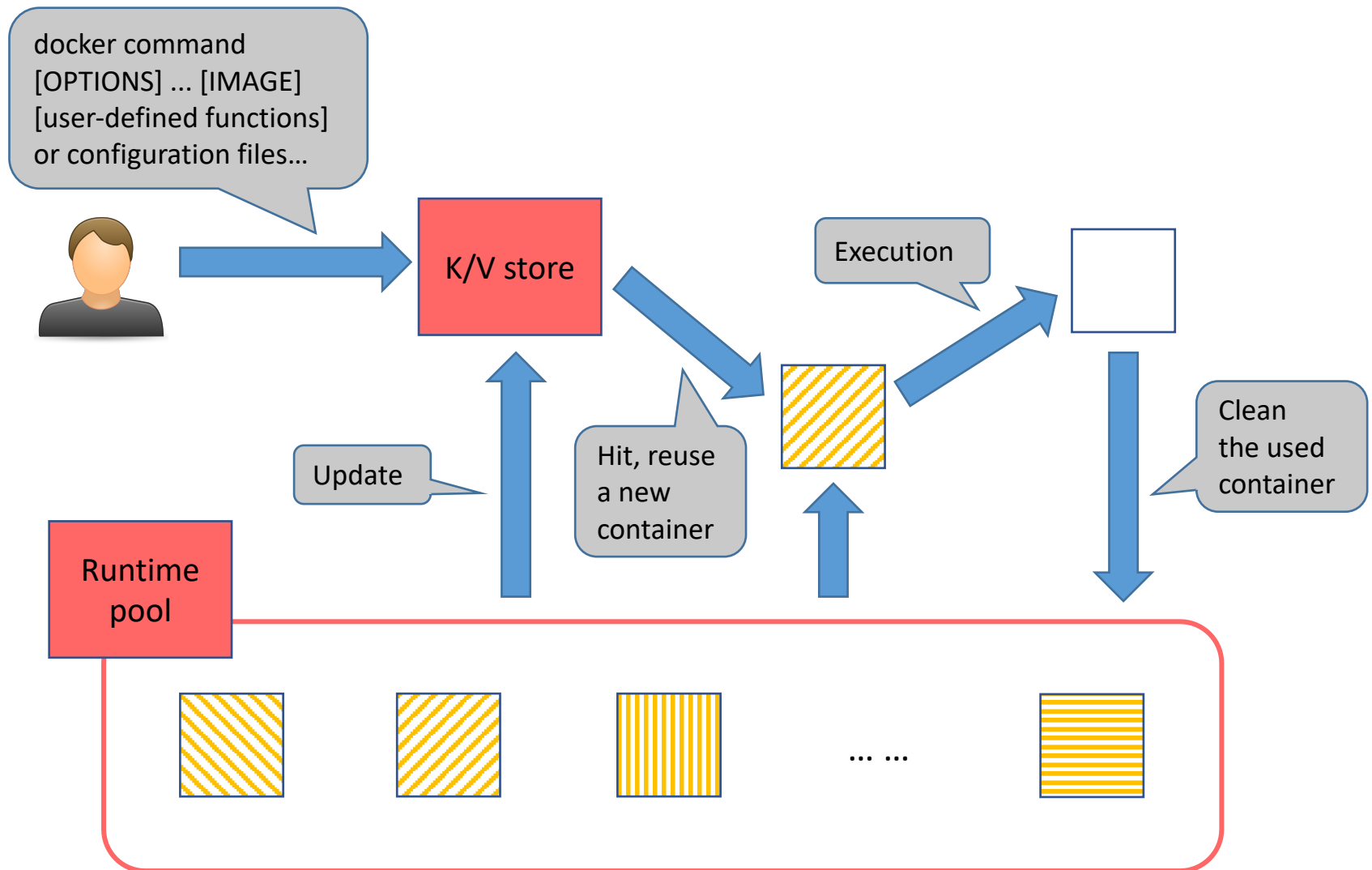
HotC Overview



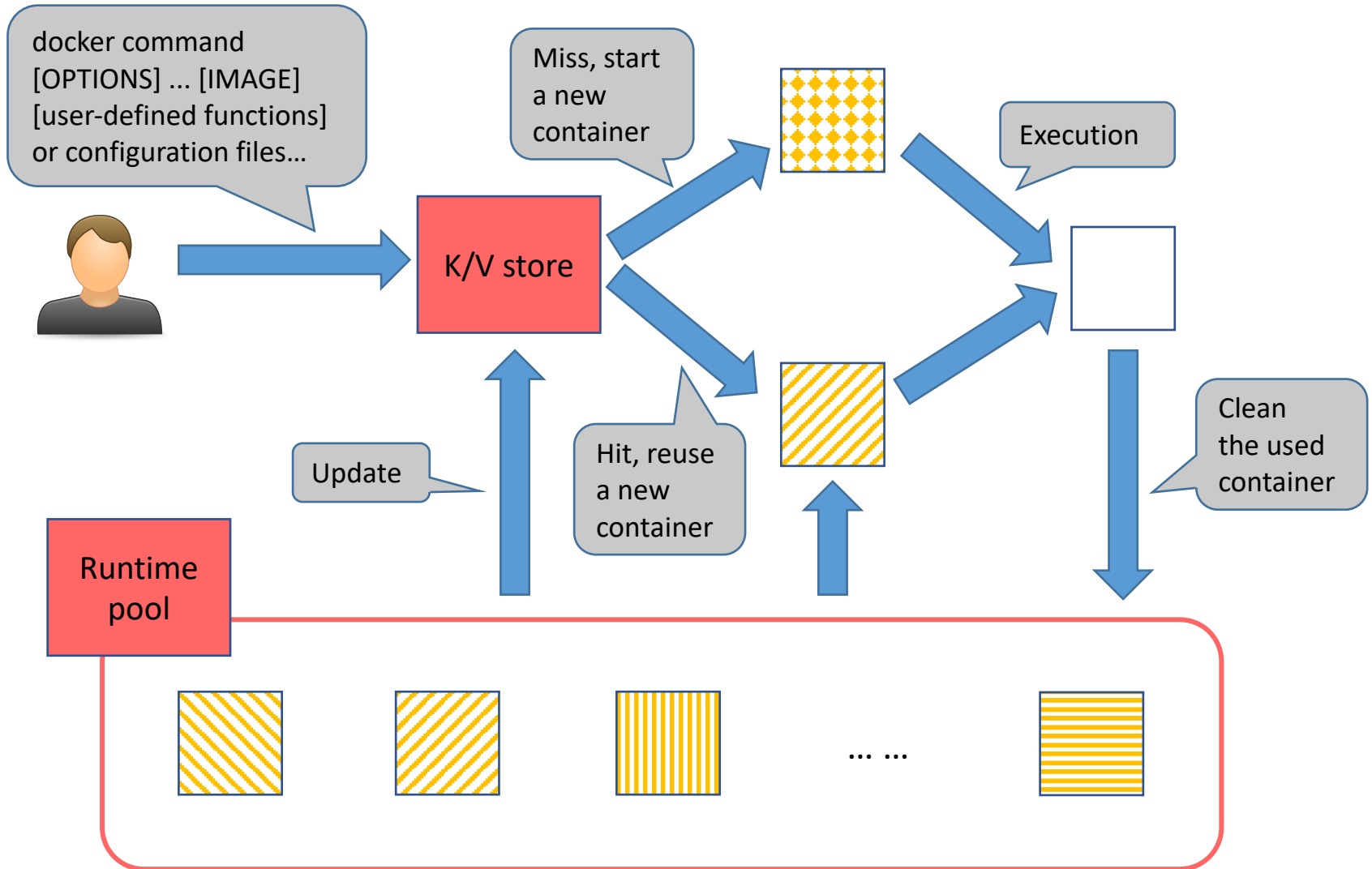
HotC Overview



HotC Overview



HotC Overview



Parameter Analysis

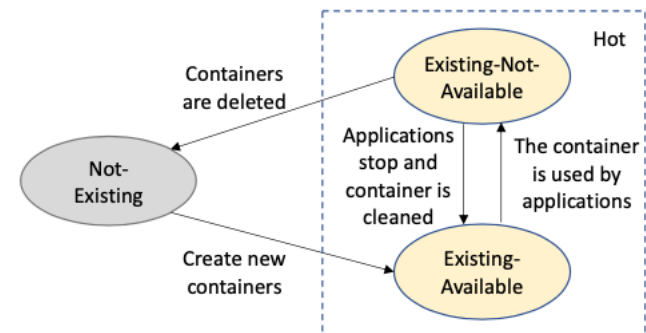
- **Analyze user command or configuration file**

- ✓ Parameters: container images, network configuration, UTS settings, IPC settings, execution options, etc.

- **HotC treats containers with identical parameter configurations as the same**

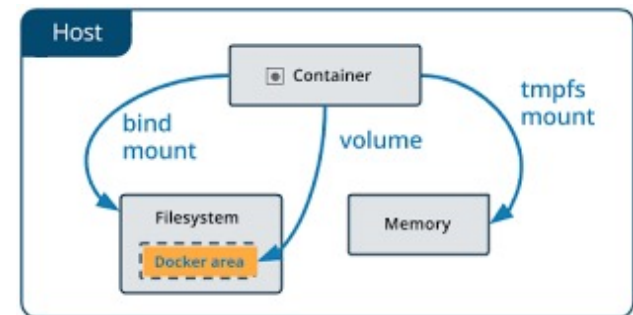
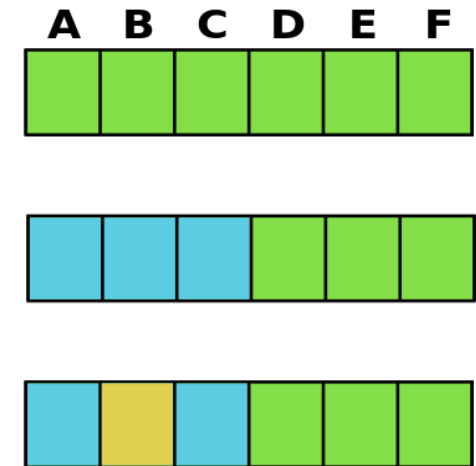
- ✓ HotC maintains a key value store to track the available containers
- ✓ Three states for the container: *Not-Existing*, *Existing-Not- Available* and *Existing-Available*
- ✓ Client reuses the first available container

```
23 ✓ nginx:
24     image: waldemarnt/cakephp-nginx
25     ports:
26         - 80:80
27     links:
28         - php
29     volumes_from:
30         - application
31     volumes:
32         - ./logs/nginx:/var/log/nginx
33
```



Container Runtime Pool & Cleanup

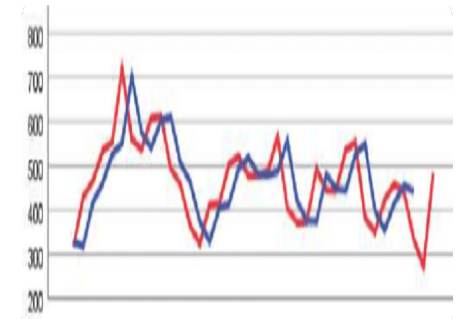
- HotC maintains a container pool inside the memory
- Maximum number of live containers to 500 and the memory usage threshold as 80% in the host
- Heuristically identify the memory pressure and terminated the oldest if necessary
- Cleanup the used containers using mounted volumes



Adaptive Live Container Management

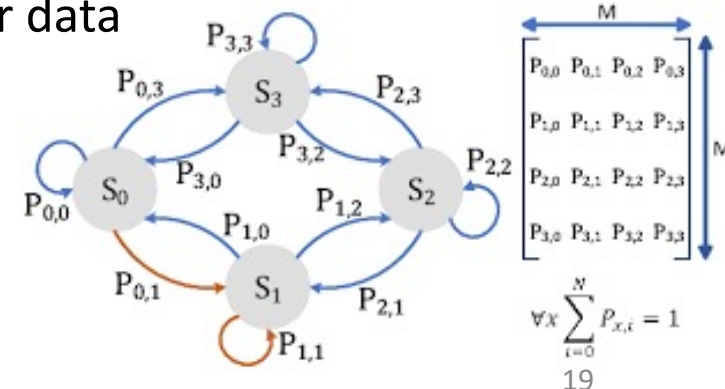
- Container prediction: whether one container will be reused

- ✓ The number of specific type of container runtimes $e_{k_i,t}$ with configuration k_i at time $t = \alpha * history[k_i][t] + (1 - \alpha) * e_{k_i,t-1}$, where $history[k_i][t]$ are time series data of how many specific type of container runtime in the pool and α is the exponential smoothing coefficient



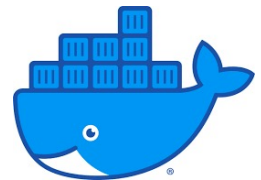
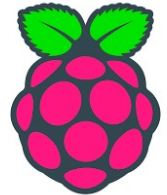
- Markov Prediction

- ✓ Overcome short-term forecasting with fewer data and volatility
- ✓ Predict the results through the transition probability between states and can better compensate for above limitations

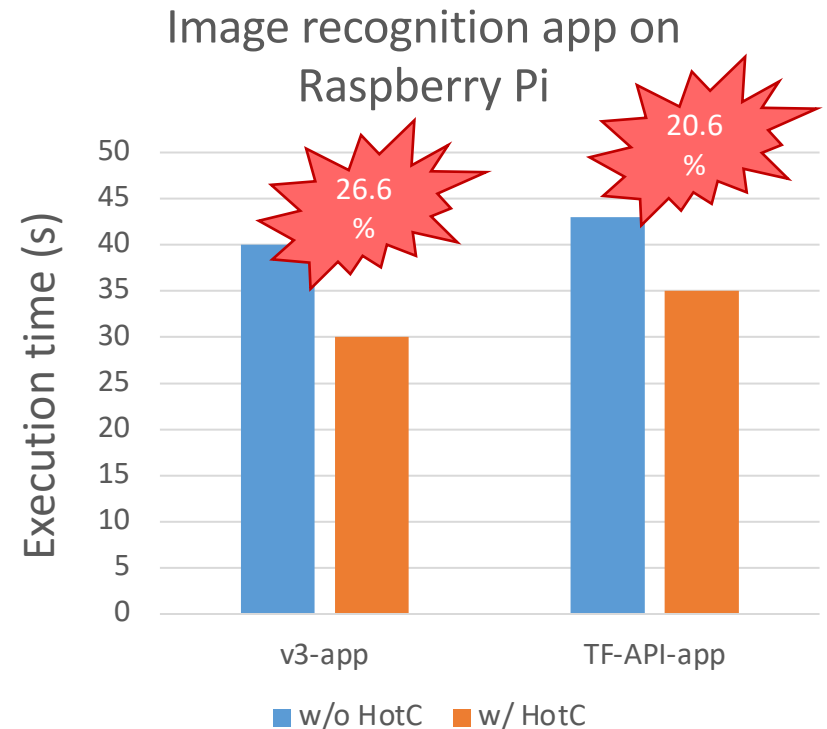
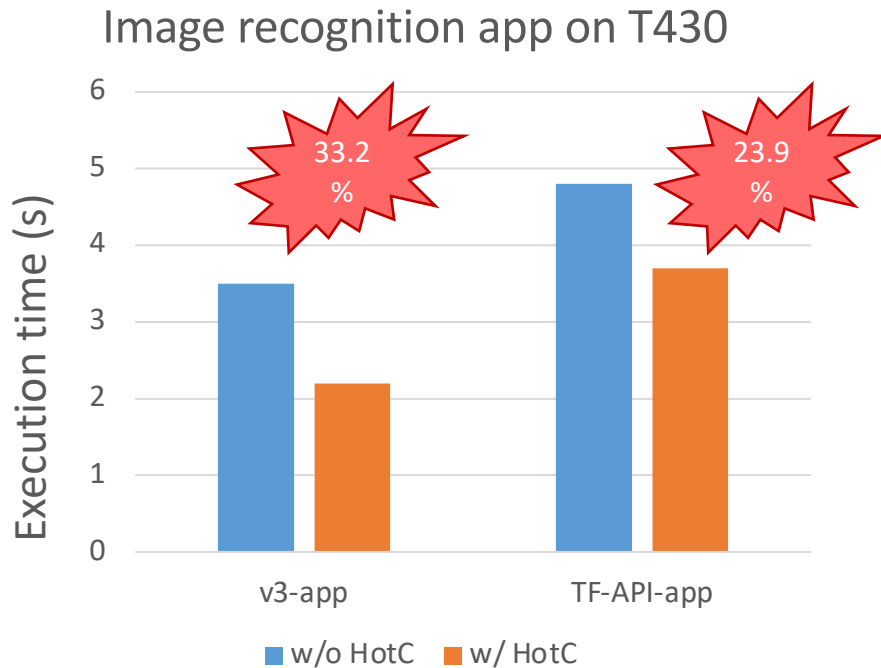


Evaluation Settings

- DELL PowerEdge T430 server:
 - ✓ Dual ten-core Intel Xeon E5-2640 2.6GHz processors, 64GB memory, Gigabit Network and a 2TB 7200RPM hard drive
 - ✓ Ubuntu 16.04 and Linux kernel version 4.4.20
- Raspberry Pi 3:
 - ✓ Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB memory and 32GB storage
 - ✓ Linux Raspberrypi 4.14
- Docker 1.17, OpenFaaS 0.8.5

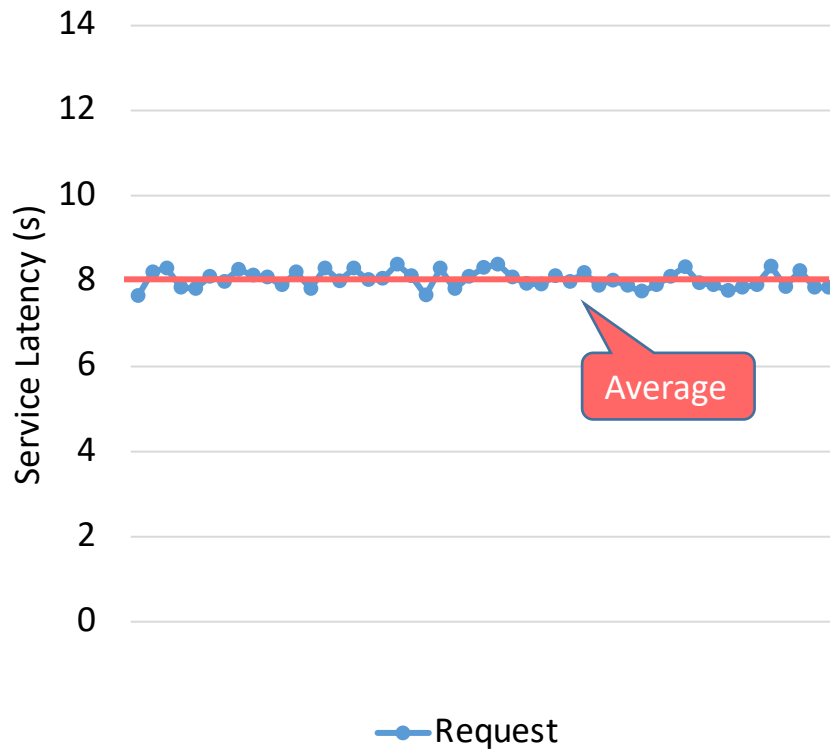


The image recognition application execution time

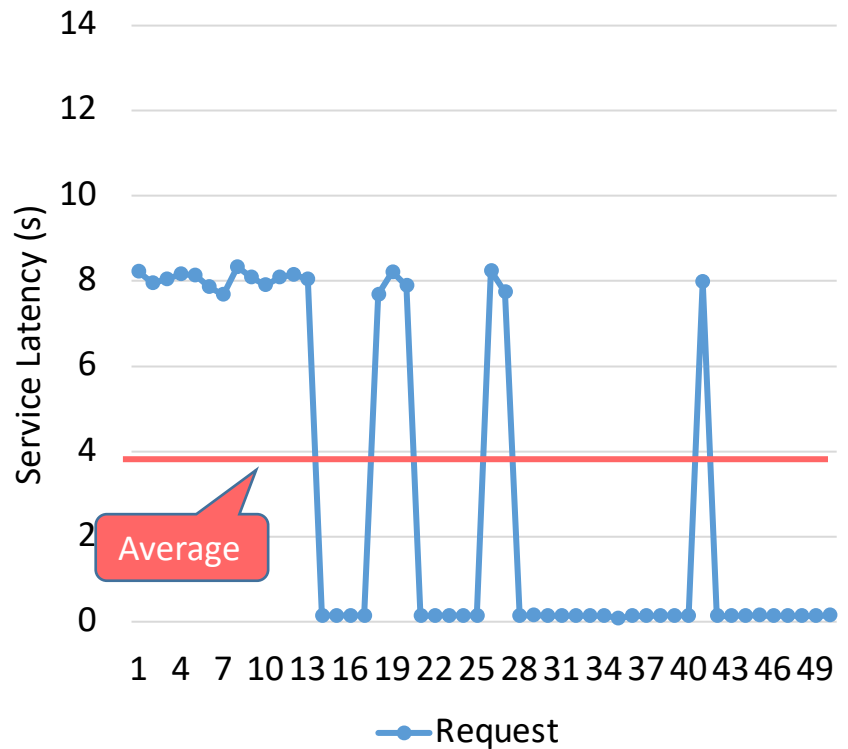


Web Application Latency

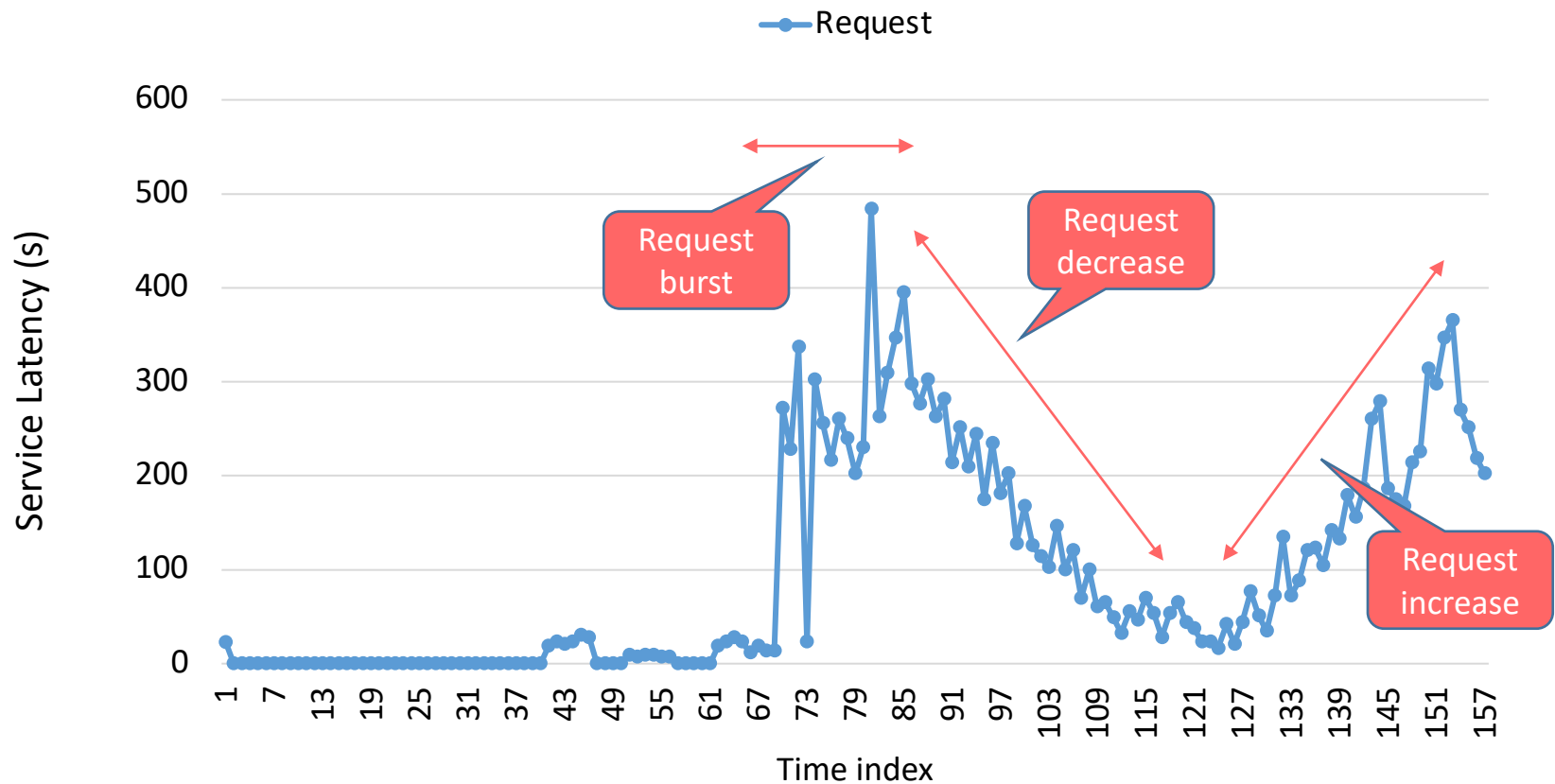
The web application request latency w/o HotC



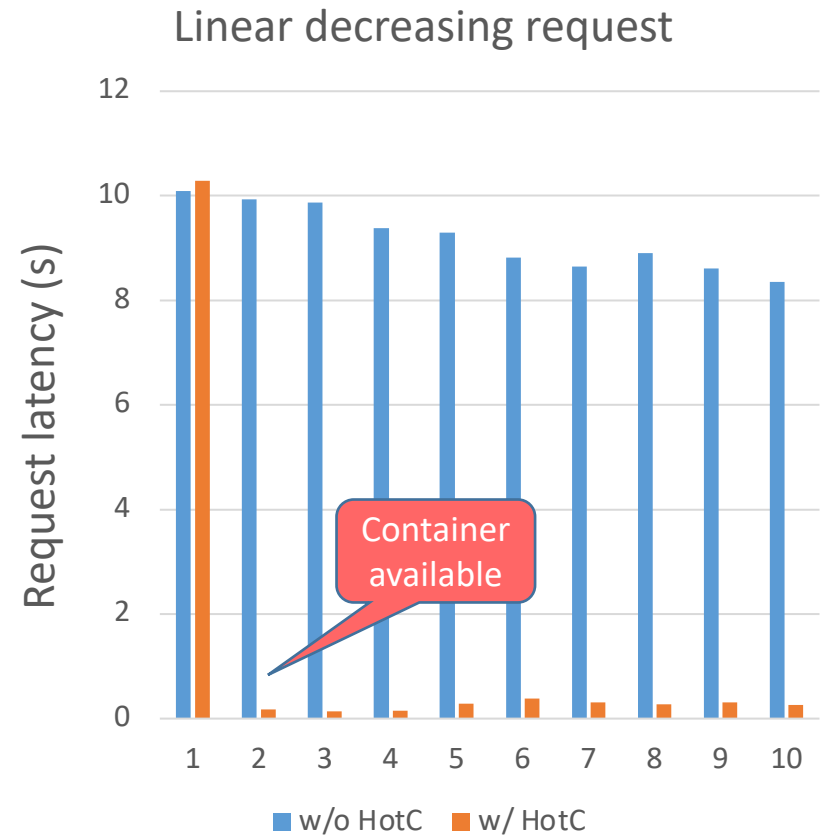
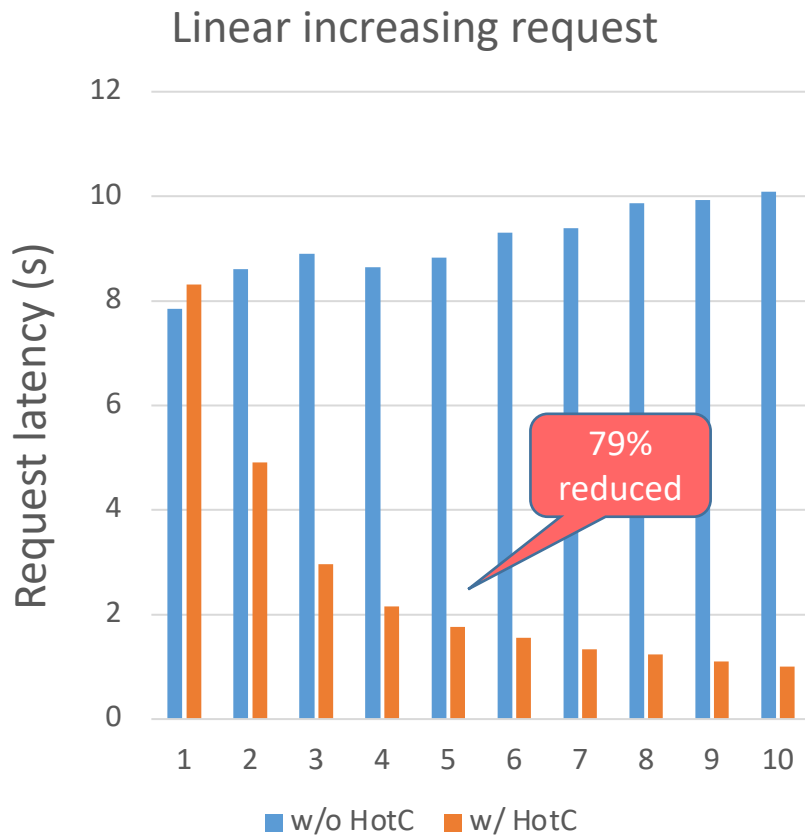
The web application request latency w/ HotC



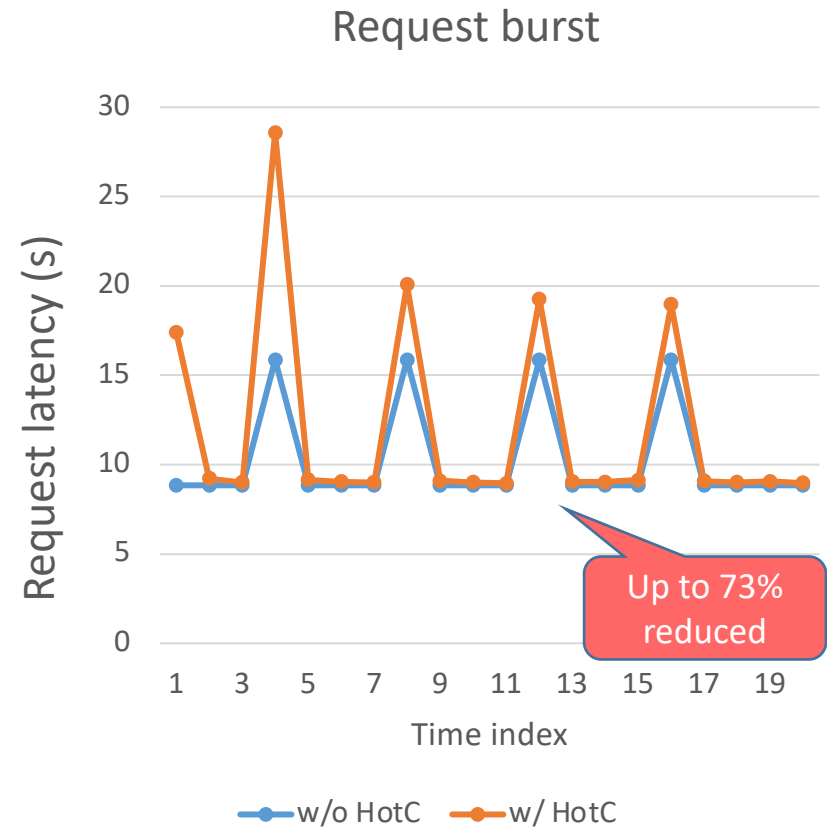
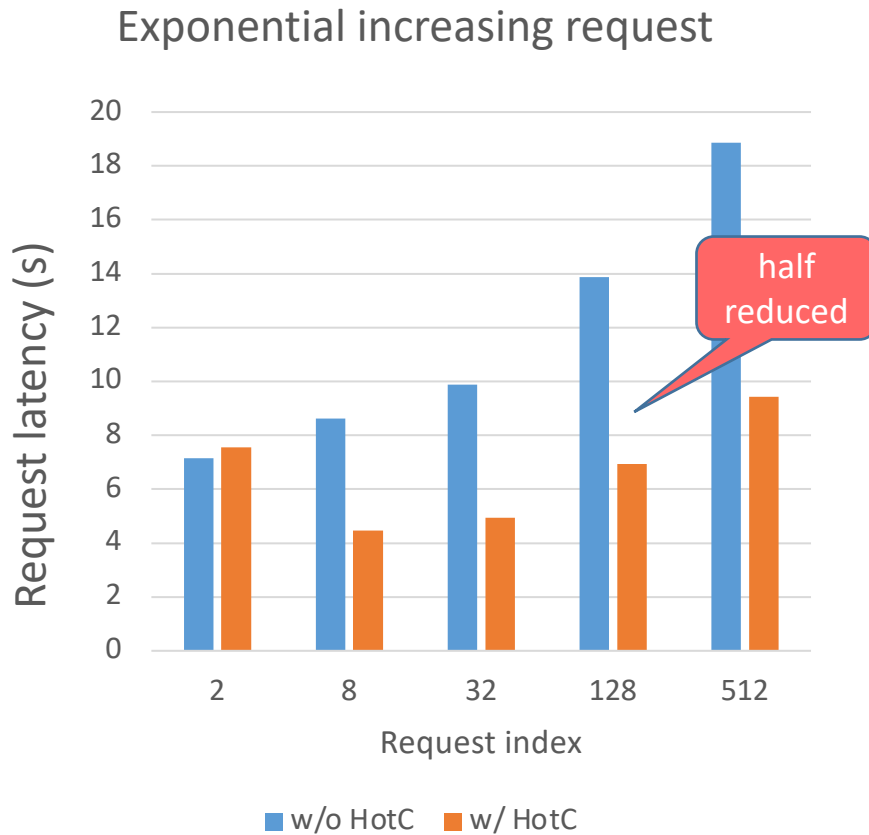
Youtube request statistics of Umass



Linear network throughput

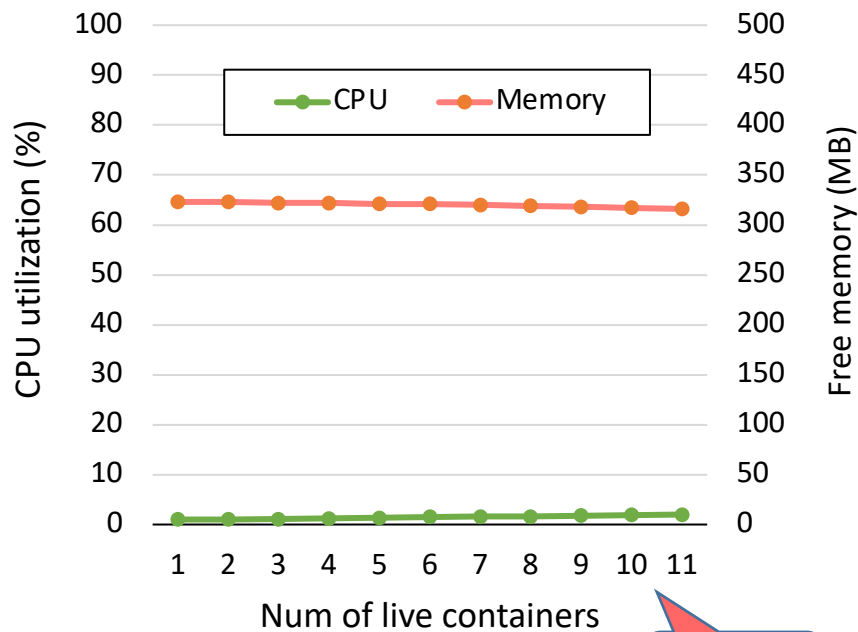


Exponential and burst request



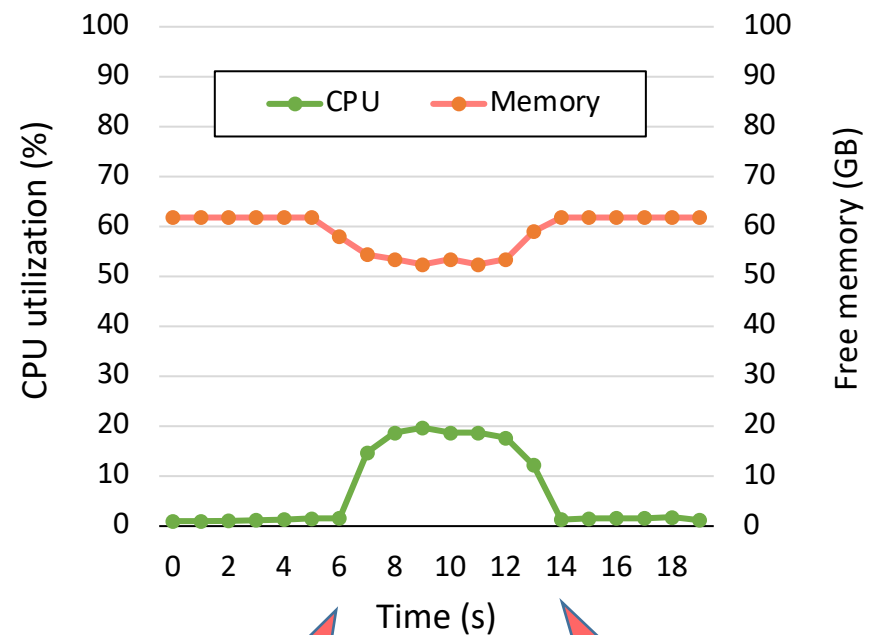
The resource consumption of live containers

Resource monitoring on Raspberry Pi 3



Less than 1%

Resource monitoring on T430 server



Container start

Container stop

Conclusions

- **Background & Challenges**

- ✓ Serverless computing is widely used but facing challenges such as long latency due to the container cold start.

- **HotC**

- ✓ A container-based runtime management framework which leverages the lightweight containers to mitigate the cold start and improve network performance.

- **Results**

- ✓ Our evaluation results show HotC introduces negligible overhead and can efficiently improve the performance of various applications in both cloud servers and edge devices.

Thank you !

Questions?