Fifer: Tackling Resource Underutilization in the Serverless Era

Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan Chidambaram, Mahmut Kandemir, Chita Das

ACM/IFIP Middleware’21
Dec 10, 2020
# Executive Summary

## Tenants

Faster Response Times

## Providers

Serverless Functions
# Executive Summary

## Tenants

- Faster Response Times
- SLO violations
- Cold-starts

## Providers

- Serverless Functions
- Over Provisioning & Underutilization
# Executive Summary

## Tenants
- Faster Response Times
- SLO violations
- Cold-starts
- Guarantee SLOs

## Providers
- Serverless Functions
- Over Provisioning & Underutilization
- Leverage Application Info Fully Utilize

---

**PennState College of Engineering**

*High Performance Computing Lab*
Serverless Function Chains

Stages in Intelligent Personal Assistant (IPA)

Containers for Each Microservice

Request

1500ms

Response

Users

Image Classification

Natural Language Processing

Question Answering
Serverless Function Chains

Stages in Intelligent Personal Assistant (IPA)

Image Classification  
Natural Language Processing  
Question Answering

Containers for Each Microservice

Users  
Request  
1500ms

Response

Container Cold Starts
Serverless Function Chains

Stages in Intelligent Personal Assistant (IPA)

- Image Classification
- Natural Language Processing
- Question Answering

Containers for Each Microservice

Request

1500ms

Response

Users

Container Cold Starts
- Container creation
- Model fetch time
Impact of cold-starts on performance?

- Model fetch time
- Container creation
- Model fetch time

Serverless Function Chains

Stages in Intelligent Personal Assistant (IPA)

Containers for Each Microservice
Why Cold Starts are bad?

Cold Start (First invocation)

Warm Start (Concurrent Invocations)
Why Cold Starts are bad?

Cold starts contribute ~2000 to 7500 ms on top of execution time.

Cold Start (First invocation)

Warm Start (Concurrent Invocations)
Why Cold Starts are bad?

Cold starts contribute \( \sim 2000 \text{ to } 7500 \text{ ms} \) on top of execution time.

How providers handle cold starts?

Cold starts contribute \( \sim 2000 \text{ to } 7500 \text{ ms} \) on top of execution time.
Current Serverless Platforms

Wang et al, Peeking behind the curtains of Serverless Platforms in ATC’18
Current Serverless Platforms

• Spawn new containers if existing containers are busy.
  ➡ Leads to SLO violations due to cold-starts.
  ➡ Many idle containers. Wasted power and energy.

Wang et al, Peeking behind the curtains of Serverless Platforms in ATC’18
Current Serverless Platforms

• Spawn new containers if existing containers are busy.
  ➡ Leads to SLO violations due to cold-starts.
  ➡ Many idle containers. Wasted power and energy.

• Employing static queuing of requests on fixed pool of containers
  ➡ Leads to SLO violations due to queuing.

Wang et al, Peeking behind the curtains of Serverless Platforms in ATC’18
Current Serverless Platforms

• Spawn new containers if existing containers are busy.
  ➡ Leads to SLO violations due to cold-starts.
  ➡ Many idle containers. Wasted power and energy.

• Employing static queuing of requests on fixed pool of containers
  ➡ Leads to SLO violations due to queuing.

• Not aware of application execution times and response latency requirements.
  ➡ Colossal container overprovisioning.

Wang et al, Peeking behind the curtains of Serverless Platforms in ATC’18
Current Serverless Platforms

• Spawn new containers if existing containers are busy.
  ➡ Leads to SLO violations due to cold-starts.
  ➡ Many idle containers. Wasted power and energy.

How can we do better?

• Not aware of application execution times and response latency requirements.
  ➡ Colossal container overprovisioning.

Wang et al, Peeking behind the curtains of Serverless Platforms in ATC’18
Application Characterization

![Graph of Microservices Latency](image)

*Djinn and Tonic: DNN Inference Benchmark Suite ISCA’15*
SLO’s usually within 1.5s

Djinn and Tonic- DNN Inference Benchmark Suite-ISCA’15
Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services, Middleware’17
Multi-staged applications have ample slack.

SLO's usually within 1.5s

Djinn and Tonic- DNN Inference Benchmark Suite-ISCA’15
Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services, Middleware’17
Multi-staged applications have ample slack.

Execution times of each function is predictable.

SLO’s usually within 1.5s.
How to exploit the slack and execution time predictability?

- Multi-staged applications have ample slack.
- Execution times of each function is predictable.

Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services, Middleware’17
Djinn and Tonic- DNN Inference Benchmark Suite-ISCA 15
Slack aware queuing
Slack aware queuing

(a) Baseline RM

Overheads

Total = 24 Containers

Stage 1

Stage 2

Stage 3

8 Containers

Cold starts

Time (ms)

(a) Baseline RM

PennState
College of Engineering
Slack aware queuing

(a) Baseline RM

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overheads</td>
<td>8 Containers</td>
<td>Cold starts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Overheads</th>
<th>8 Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>400</td>
<td>500</td>
<td>600</td>
</tr>
<tr>
<td>700</td>
<td>800</td>
<td>900</td>
</tr>
<tr>
<td>1000</td>
<td>1100</td>
<td>1200</td>
</tr>
</tbody>
</table>

Total = 24 Containers

Cold starts

8 Containers

Slack
Slack aware queuing

(a) Baseline RM

(b) Request-Batching RM

Overheads

8 Containers

Cold starts

8 Containers

Total = 24 Containers

Stage 1

Stage 2

Stage 3

Overheads

Cold starts

SLA

Time (ms)

100
200
300
400
500
600
700
800
900
1000
1100
1200

(b) Request-Batching RM

Total = 10 Containers

Cold starts

SLA

Time (ms)

100
200
300
400
500
600
700
800
900
1000
1100
1200

PennState High Performance Computing Lab
Slack aware queuing

(a) Baseline RM

(b) Request-Batching RM
Exploiting Slack to Queue requests can save up to **14 containers**.
Slack aware queuing

Exploiting Slack to Queue requests can save up to **14 containers**.
Slack Allocation
Slack Allocation

Slack = 700ms

- IMC (45ms)
- NLP (2ms)
- QA (51ms)
Slack Allocation

Slack = 700ms

- IMC (45ms)
- NLP (2ms)
- QA (51ms)
Slack Allocation

Slack = 700ms

IMC (45ms)
NLP (2ms)
QA (51ms)

Overheads

IMC
NLP
QA

Proportional Slack = Equal Containers
Slack Allocation

Slack = 700ms

What about Cold Starts?

Proportional Slack = Equal Containers
Reactive Scaling + Load Prediction
Prediction Model

![Graph showing RMSE vs. Latency for different prediction models]

- RMSE
- Latency (ms)
- MWA
- EWMA
- Linear R.
- Logistic R.
- Simple FF.
- WeaveNet
- DeepArEst
- LSTM
Prediction Model

![Graph showing RMSE and Latency for different models: MWA, EWMA, Linear R., Logistic R., Simple FF., WeaveNet, and LSTM. The graph illustrates the performance comparison of these models with LSTM showing the lowest RMSE and latency.](image)

![Graph showing time series of requests with varying latencies.](image)
Prediction Model

LSTM is the best with least RMSE

RMSE

Latency (ms)

Requests(s)

Time(s)
Proactive container provisioning using LSTM

LSTM is the best with least RMSE
Proactive container provisioning using LSTM

LSTM is the best with least RMSE
Fifer: Stage-aware Proactive container provisioning and management of function chains
Fifer: Stage-aware Proactive container provisioning and management of function chains

Stage-aware proportionate queuing
Fifer: Stage-aware Proactive container provisioning and management of function chains

Stage-aware proportionate queuing

Queuing delay-based Reactive container scaling
Fifer: Stage-aware Proactive container provisioning and management of function chains

Stage-aware proportionate queuing

Queuing delay-based Reactive container scaling

Proactive container scaling
IMPLEMENTATION

<table>
<thead>
<tr>
<th>Worker-1</th>
<th>Worker-2</th>
<th>Worker-3</th>
<th>Worker-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Image Repository</td>
<td>Container Image Repository</td>
<td>Container Image Repository</td>
<td>Container Image Repository</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>Kubernetes</td>
<td>Kubernetes</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>Fifer Management Framework</td>
<td>Fifer Management Framework</td>
<td>Fifer Management Framework</td>
<td>Fifer Management Framework</td>
</tr>
<tr>
<td>BRIGADE</td>
<td>BRIGADE</td>
<td>BRIGADE</td>
<td>BRIGADE</td>
</tr>
<tr>
<td>Request arrival rate</td>
<td>Request arrival rate</td>
<td>Request arrival rate</td>
<td>Request arrival rate</td>
</tr>
<tr>
<td>Stage-1</td>
<td>Stage-2</td>
<td>Stage-3</td>
<td></td>
</tr>
<tr>
<td>Schedule Containers</td>
<td>Schedule Containers</td>
<td>Schedule Containers</td>
<td>Schedule Containers</td>
</tr>
<tr>
<td>Query every heartbeat</td>
<td>Query every heartbeat</td>
<td>Query every heartbeat</td>
<td>Query every heartbeat</td>
</tr>
<tr>
<td>Load Balancer</td>
<td>Load Balancer</td>
<td>Load Balancer</td>
<td>Load Balancer</td>
</tr>
</tbody>
</table>

**Cluster Statistics**

- B_Size
- #Containers
- Create_time
- Comp_Time
- Sched_time

**Hardware**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon gold</td>
</tr>
<tr>
<td>Sockets</td>
<td>2</td>
</tr>
<tr>
<td>Core</td>
<td>16</td>
</tr>
<tr>
<td>Threads</td>
<td>2</td>
</tr>
<tr>
<td>Clock</td>
<td>2.8Ghz</td>
</tr>
<tr>
<td>DRAM</td>
<td>192GB</td>
</tr>
</tbody>
</table>
SLO violations and Containers
SLO violations and Containers

Better

Workload

SLO Violations

Heavy Medium Light
SLO violations and Containers

- #Containers normalized to baseline
- Fifer is spawns 20% less containers

Better

![Bar charts showing SLO violations and Containers](image-url)
SLO violations and Containers

- #Containers normalized to baseline
- Fifer is spawns **20%** less containers

- SLO violations normalized to baseline
- Fifer is similar to baseline with fewer containers
Utilization and Energy
Utilization and Energy

- Average #Requests executed per container (RPC).
- Fifer improves container utilization by 34%
Utilization and Energy

- Average #Requests executed per container (RPC).
- Fifer improves container utilization by 34%.
- Energy consumption normalized to Bline.
- Fifer is 31% more energy efficient.
• Details of the workload used.
• Evaluated schemes and policies.
• Details about LSTM training.