Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud

Jashwant Gunasekaran, Prashanth Thinakaran, Mahmut Kandemir, Bhuvan Urgaonkar, George Kesidis, Chita Das

Computer Science and Engineering
The Pennsylvania State University
Outline

• Elastic Web Services.

• VM-based Resource Procurement.

• Serverless Functions.

• Cost of VMs vs Cloud Functions.

• Spock Hybrid Elastic Scaling.

• Implementation and Evaluation.

• Results.
Elastic Web Services

• Short lived queries
  • Strict SLO.

Resources Required
• acquired/released on demand.
• Average to Peak ratio is high.

Typical example.? ML based web services
ML Inference Engine

How to provision resources?
Outline

• Elastic Web Services.

• VM-based Resource Procurement.

• Serverless Functions.

• Spock Hybrid Elastic Scaling.

• Implementation and Evaluation.

• Results.
VM-based Procurement
VM-based Procurement

- Initial warm pool of active VMs.
- Procure more VMs on demand.
- Autoscaling during request surge.

![Graph showing resource demand over time with VM, SLA, and arrival rate lines.](chart)
Disadvantages

• Very long VM setup times.

Possible alternative.

• Under-provisioned during sudden surge.
Outline

• Elastic Web Services.
• VM-based Resource Procurement.
• Serverless Functions.
• Spock Hybrid Elastic Scaling.
• Implementation and Evaluation.
• Results.
Serverless Functions
Serverless Functions

- Pay per second.
- Cost efficient.
- Scale instantaneously.

But, is serverless a replacement for VMs? Let's see an example.
Constant arrival rate

- Constant arrival rate.
- Cost compared under iso-performance.
- All requests have similar SLA compliance.
- VMs are 100% utilized.
Varying arrival rate

Cost-effective Solution?

- Each request is an ML inference for caffenet-model
- Cost compared under iso-performance.
- All requests have similar SLA compliance.
- VMs are provisioned for the peak request rate.
Why SPOCK.? 

- Use serverless functions along with VMS
- Reduce SLO violations during request surge
- Reduce intermittent over-provisioning VMs
Key Motivation

• It is non-trivial to predict the peak request rate at any given time period.

• Provisioning VMs for the peak demands would always lead to higher cost of deployment. While, under provisioning VMs leads to severe SLO violations for queries.

• Using serverless functions would overcome the SLO violation problem. However, it is not cost effective.
Outline

• Elastic Web Services.

• VM-based Resource Procurement.

• Serverless Functions.

• Spock Hybrid Elastic Scaling.

• Implementation and Evaluation.

• Results.
Spock Scheme

• Schedule queries on VM’s if available.
• If VM’s are fully utilized, redirect queries to lambda functions.
• Spawn a new VM in the meantime.
• After spin-up incoming requests are sent to new VMs.
• Scale down VMs after three minutes of inactivity.
Two Scaling Policies

- **Reactive**
  - Spin-up new VMs as when request surge occurs.

- **Predictive**
  - Using moving window linear regression to predict request every 1 minute.
  - Spin up new VMs based on prediction.

Let's see an example.
Spock resource procurement

![Diagram showing resource procurement over time with scale out and scale in arrows.]
Overall Design of Spock

User Applications

Resource Manager

Load Monitor

Scaling Policy

Load Balancer

Resource Required

Predicted Load

Query Complete

Resource Status

Queries

Instance Created

Query Assigned

VM

MODEL 1

VM

MODEL 2

VM

MODEL 3

VM

MODEL 4

Predictive

Reactive
Outline

• Elastic Web Services.
• VM-based Resource Procurement.
• Serverless Functions.
• Spock Hybrid Elastic Scaling.
• Implementation and Evaluation.
• Results.
Evaluation

• Two traces used to generate ML inference workload.

![Request rate per sec vs Time (minutes) for WITS and Berkeley](chart.png)
Evaluation

- Mxnet Framework.
- AWS resources.
- Pretrained ML models on imagenet dataset.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Memory Required (GB)</th>
<th>Memory Allocated (GB)</th>
<th>Average Execution (ms)</th>
<th>Requests vCPU for VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffenet</td>
<td>1.024</td>
<td>3.072</td>
<td>300</td>
<td>4</td>
</tr>
<tr>
<td>Googlenet</td>
<td>0.456</td>
<td>2.048</td>
<td>450</td>
<td>3</td>
</tr>
<tr>
<td>Squeezenet</td>
<td>0.154</td>
<td>2.048</td>
<td>130</td>
<td>6</td>
</tr>
<tr>
<td>Resnet-18</td>
<td>0.304</td>
<td>3.072</td>
<td>320</td>
<td>3</td>
</tr>
<tr>
<td>Resnet-200</td>
<td>1.024</td>
<td>3.072</td>
<td>956</td>
<td>1</td>
</tr>
<tr>
<td>Resnext-50</td>
<td>0.645</td>
<td>3.072</td>
<td>560</td>
<td>2</td>
</tr>
</tbody>
</table>
Evaluation

- Two scaling policies
  - Predictive
  - Reactive
- Three resource procurement schemes
  - Autoscale
  - X-autoscale
  - Spock
Outline

• Elastic Web Services.

• VM-based Resource Procurement.

• Serverless Functions.

• Spock Hybrid Elastic Scaling.

• Implementation and Evaluation.

• Results.
Berkely Trace Results

![Bar Chart 1](image1.png)

![Bar Chart 2](image2.png)
WITS Trace Results

![Normalized Cost vs. Autoscale and Spock](image1.png)

- **Mix-1**
- **Mix-2**
- **SLO Violation**

![SLO violations (%) vs. Autoscale and Spock](image2.png)

- **Mix-1**
- **Mix-2**
- **SLO Violation**
Spock Prediction Accuracy

![Graph 1: Request rate per sec vs Time (hundreds of sec)]

- **Actual**
- **Predicted**

![Graph 2: Request rate per sec vs Time (hundreds of sec)]

- **Actual**
- **Predicted**
Spock Resource Procurement

![Chart showing resource procurement over time](chart.png)

- **Request rate per sec**
  - Scale out
  - Scale in
  - --- Request rate

- **Time (hundreds of sec)**
  - 0
  - 1
  - 4
  - 10
  - 15
  - 30

- **Request rate (VMs)**
  - 0
  - 125
  - 250
  - 375
  - 500

---

PennState
www.cse.psu.edu/hpcl

Jashwant Gunasekaran
Prashanth Thinakaran

30