MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving

Chengliang Zhang†, Minchen Yu†, Wei Wang†, Feng Yan‡

†Hong Kong University of Science and Technology
‡University of Nevada, Reno
Machine Learning Serving - MLaaS

Deploy a trained model on cloud for user requests

• Highly dynamic demand
• Stringent Service Level Objectives on latency

Objectives of serving on public cloud

• Scale to dynamic queries
• SLO-aware: e.g. 98% of the requests must be served under 500ms
• Cost-effective

= “tabby cat”
Conventional Autoscaling – AWS SageMaker

- **Reactive scaling**: based on current load

<table>
<thead>
<tr>
<th>Provisioning Time (minutes)</th>
<th>Execution Time (&lt; 1s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Hide provisioning time -> over-provisioning**

e.g., in AWS EC2, serving an inception-v3 query is 20,000 times more expensive than redis query

Sagemaker suggests to adjust over-provisioning factor from 2

ML accelerators: GPU, TPU, FPGA

- Mass parallel support
- Essential for training complex models
- Expensive

Inference
- Run comfortably without them
- Way less parallelism

CPU: m5.xlarge: $0.192 per hour
GPU: p2.xlarge: $0.9 per hour
TPU v2: $4.5 per hour

- Choose between CPU and accelerators
- Justify the price tag
Characterization: CPU vs. GPU vs. TPU

- CPU: no significant benefits for small instances
- GPU & TPU: benefit substantially
- GPUs can be cheaper, but only with batching and high utilization

CPU: 1 vCPU, 2 GB mem; GPU: K80; TPU: TPU-v2

![Graph showing cost and latency for different batch sizes for CPU, GPU, and TPU](image)

- **Cost**:
  - CPU: lowest cost for small batch sizes
  - GPU: higher cost but decreases with larger batch sizes
  - TPU: very high cost across all batch sizes

- **Latency**:
  - CPU: highest latency for small batch sizes
  - GPU: decreases with larger batch sizes
  - TPU: lowest latency across all batch sizes

**Tradeoff**
- Larger batch size: Better cost-effectiveness
- Longer queuing delay: Tradeoff
Numerous Choices on Cloud

- Infrastructure as a Service (VMs)
- Container as a Service (Containers)
- Function as a Service (serverless comp.)

- **Large configuration space:** AWS offers more than 200 instance types in EC2 alone
- **Cost-performance trade-offs**
  - Preemptable instances (spot market)
  - Burstable instances

- The right service
- Appropriate configuration
- Exploit the discounts without sacrificing SLO
Cloud Services for Model Serving

**Infrastructure as a Service (IaaS)**

**Container as a Service (CaaS)**

**Function as a Service (FaaS, serverless comp.)**

<table>
<thead>
<tr>
<th>ML Model</th>
<th>EC2</th>
<th>ECS</th>
<th>Lambda</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>t (ms)</td>
<td>$</td>
</tr>
<tr>
<td>Inception-v3</td>
<td>5.0</td>
<td>210</td>
<td>9.17</td>
</tr>
<tr>
<td>Inception-ResNet</td>
<td>9.3</td>
<td>398</td>
<td>16.4</td>
</tr>
<tr>
<td>OpenNMT-ende</td>
<td>51.5</td>
<td>2180</td>
<td>96.3</td>
</tr>
</tbody>
</table>

EC2: c5.large; ECS: 2vCPU, 4GB mem; Lambda: 3008MB mem

Combining IaaS’s **cost advantage** with FaaS’s **scalability**
- Instead of overprovisioning IaaS, use FaaS to handle demand surge and spikes

Pay-as-you-go

Pay for what you use

- **FaaS**
- **CaaS**
- **IaaS**
IaaS: Instance Families and Sizes

There are 4 families of instance in EC2:

- general purpose m
- memory optimized r
- compute optimized c
- burstable t

- The bottleneck is CPU
- Performance grows sub-linearly with size

M1: Inception-v3, M2: Inception-Resnet, M3: OpenNMT-ende. Price and latency normalized by the value of c5.large
IaaS: Spot Instances

• **Discounted**: up to 75% off, dynamic pricing
• **Transient resource**: providers can take it back, interruptions

**ML serving is stateless**

• Requests are independent
• The response only depends on the requests

• No consistency requirement
• Cloud services: IaaS is cost-effective, FaaS has the best scalability
• With on-demand pricing, smaller CPU instances are preferable, cheaper, smaller scaling step size
• Accelerator batching: important control nob for cost and latency tradeoff
• Safe to use spot instances
Cost-effectiveness

• To maintain *high utilization* and hide *provisioning time*: workload prediction + proactive provisioning
• Use FaaS to reduce over-provisioning
• Adopt spot instances: online provisioning algorithm

Accelerator Support

• Use dynamic batching, batching requests according to arrival rate and SLO specification

*Batching guideline:*
• After batching, SLOs can’t be violated
• The overall throughput should be better than pre-bathing
Design Considerations Cont.

99% of the requests must complete under 1s

**SLO-awareness**
- Overall response time: no closed form solution
- ML inference execution time is deterministic

**Best effort solution:**
Monitor the queuing time for each request, direct requests to FaaS when necessary

Introducing MArk (Model Ark)

- Weighted round robin for load balancing
- Server front implemented with Sanic framework
- Support TensorFlow Serving, MXNet Model Serving, and other custom servables

- Nginx and Gunicorn for admission and parallelism control
- Support for spot instances
- Can be ported to all popular cloud platforms
Proactive Provisioning

- MArk: plug any predictive algorithm that best suits the workload

Heterogeneous Cluster
Deterministic processing time
Assume Poisson arrival

A compilation of M/D/c queues
No closed form solution

Heuristic: Greedy Provisioning

• Expose long-term trade-offs
• Find the cheapest instance: the # of requests to serve / (charge by the min. + launch overhead)
### Evaluations Setup

#### Test Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Type</th>
<th>Framework</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-v3</td>
<td>Image Classification</td>
<td>Tensorflow Serving</td>
<td>45MB</td>
</tr>
<tr>
<td>NASNet</td>
<td>Image Classification</td>
<td>Keras</td>
<td>343MB</td>
</tr>
<tr>
<td>LSTM-ptb</td>
<td>Language Modeling</td>
<td>MXNet Model Server</td>
<td>16MB</td>
</tr>
<tr>
<td>OpenNMT-ende</td>
<td>Machine Translation</td>
<td>Tensorflow Serving</td>
<td>330MB</td>
</tr>
</tbody>
</table>

#### Test Bed

**AWS**

Cluster size: up to 52 CPU instances, and 12 GPU instances
Cost Savings

Twitter workload
arrival pattern abstracted from real time tweets

- **MArk-ondemand**: up to **3.6×** savings
- **MArk-spot**: up to **7.8×** savings

- MO: MArk with only on-demand instances
- MS: MArk with spot instances
- SM: Sagemaker as a baseline
What if workload is unpredictable?

MMPP: unpredictable, highly dynamic workload
Unexpected Load Surge

75% surge

100% surge

Unexpectedly increase arrival rate
- Mark does not rely on prediction accuracy for SLO compliance
Conclusion

• Characterized ML model serving on cloud
  o Proposed combining IaaS and FaaS for ML serving
• Designed a cost-effective, SLO-aware system MArk
  o Predictive greedy provisioning
  o Dynamic batching to exploit accelerators
  o Support spot instances
• Implemented Mark, and evaluated it on AWS
  o Up to 7.8x cost reduction
Thank you for coming!

MArk is open sourced at
https://github.com/marcoszh/MArk-Project

Find me

Seeking internship opportunities