IRIS: Interference and Resource Aware Predictive Inference Serving on Cloud Infrastructures

IEEE International Conference on Cloud Computing 2023

A. Ferikoglou, P. Chrysomeris, A. Tzenetopoulos, E. Katsaragakis, D. Masouros, and D. Soudris
Microprocessors and Digital Systems Laboratory, ECE, National Technical University of Athens (NTUA), Greece

{aferekoglou, pchrysomeris, atzenetopoulos, mkatsaragakis, demo.masouros, dsoudris}@microlab.ntua.gr

4/7/2023
Rise of AI and SoA Model Complexity

- Ever-increased use of **Artificial Intelligence**

- Ever-increased **complexity of deployed models** in terms of:
  - Computation
  - Memory
  - Storage

- E.g., GPT-3 $\approx$175B parameters

Prohibitive for end-user to support Inference

MLaaS for DL Inference

- Cloud “as a solution” to the resource wall challenge of DL inference

  Inference takes **90% of total infrastructure cost** [1]

  Serves **tens-of trillions inference tasks** a day [2]

- To further exploit the trend **MLaaS** was introduced
  - End-user provide **pre-trained model** along with **throughput-/latency- QoS**

---


MLaaS Challenges

Resource Interference

Model Variants

Diverse QoSs/SLOs

Guarantee the user-defined QoS/SLO requirements and maximize resource efficiency
Inference Serving Testbed

- **HW/SW Infrastructure**
  - 2 VMs serving as **master** and **worker** of a **Kubernetes** cluster

<table>
<thead>
<tr>
<th>VM</th>
<th>vCPUs</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>4</td>
<td>8 GB</td>
</tr>
<tr>
<td>Worker</td>
<td>8</td>
<td>16 GB</td>
</tr>
</tbody>
</table>

- **Inference Engine Workloads**
  - **Image classification** + **Object detection** inference engines from **MLPerf** [1]

- **Synthetic Interference**
  - Microbenchmarks that stress **CPU**, **L2/L3 cache** and **memory bandwidth/capacity** from **iBench** [2]

---

Characterizing Inference Serving

- Quantify the impact of the following to inference engine performance
  - Different model representation backends i.e., TensorFlow, ONNX Runtime
  - Vertical/Horizontal scaling
  - Interference
**Isolated Execution**

**Q1:** How do different model variants for the same task behave in terms of performance?

<table>
<thead>
<tr>
<th>Image Classification</th>
<th>ONNX-resnet50</th>
<th>TF-resnet50</th>
<th>ONNX-mobilenet</th>
<th>TF-mobilenet</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Workers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>12.4</td>
<td>45.6</td>
<td>57.5</td>
</tr>
<tr>
<td>2</td>
<td>18.3</td>
<td>13</td>
<td>43.6</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>16.4</td>
<td>12.7</td>
<td>41.8</td>
<td>56.3</td>
</tr>
<tr>
<td>8</td>
<td>16.4</td>
<td>12.9</td>
<td>38</td>
<td>55.9</td>
</tr>
</tbody>
</table>

**TF-MobileNet 4.6x higher QPS than TF-ResNet50**

**No clear dominance of TensorFlow over ONNX Runtime and vice versa**
**Isolated Execution**

**Q2:** How does vertical scaling (i.e., #Threads) of resources affect performance?

<table>
<thead>
<tr>
<th>Image Classification</th>
<th>ONNX-resnet50</th>
<th>TF-resnet50</th>
<th>ONNX-ssd-mobilnet</th>
<th>TF-ssd-mobilnet</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Threads</td>
<td># of Workers</td>
<td># of Workers</td>
<td># of Workers</td>
<td># of Workers</td>
</tr>
<tr>
<td>8</td>
<td>101 44 45 50</td>
<td>55.2 53.9 54.1 53.7</td>
<td>27.4 22.1 22 22</td>
<td>73.8 67 61.9 74.3</td>
</tr>
<tr>
<td>4</td>
<td>62.7 60 52.5 52.9</td>
<td>31.6 31.7 31.3 31.4</td>
<td>26.8 26.7 27.1 26.3</td>
<td>41.5 46.4 46.7 40.5</td>
</tr>
<tr>
<td>2</td>
<td>33.8 33.1 33 31.4</td>
<td>24.4 24.5 24.3 24.5</td>
<td>23.3 23.1 21.7 22</td>
<td>26.9 26.8 28.2 27</td>
</tr>
<tr>
<td>1</td>
<td>19 18.3 16.4 16.4</td>
<td>12.4 13 12.7 12.9</td>
<td>17.9 18.6 17 15.9</td>
<td>18.7 20.2 19.4 20.3</td>
</tr>
</tbody>
</table>

#Threads = 8 compared to #Threads = 1

- 2.8x higher QPS on average for ONNX Runtime
- 3.8x higher QPS on average for TensorFlow
Q3: How does horizontal scaling (i.e., #Workers) of resources affect performance?

TensorFlow not affected due to CPython’s GIL

ONNX Runtime presents a 14% QPS drop on average
Q4: How does resource interference affect the performance of the inference engines?

#iBench = 16 on Object Detection

11.7x lower QPS on average stressing CPU

12.5x lower QPS on average stressing Memory BW
Q5: Do different backends reveal different performance sensitivity w.r.t resource interference?

**Image Classification**
- ONNX-resnet50: 1.5x higher QPS than TF-ResNet50
- TF-ResNet50

**Object Detection**
- ONNX-SSDMobileNet
- TF-SSDMobileNet: 2.5x higher QPS than ONNX-SSDMobileNet
Q6: How do different resource allocations affect the performance of the inference engines under the presence of interference?

**ONNX Runtime**

- #Threads = 2 or 4

**TensorFlow**

- #Threads > 5
Based on our observations we design IRIS, an interference- and resource-aware predictive orchestration methodology

- Identifies interference effects by exploiting low-level performance events
- Provides accurate performance predictions
- Automatically applies horizontal/vertical scaling policies and chooses the appropriate model variant
IRIS Design – Offline Phase

Random Scenario Execution

1a. Random Interference
1b. Continuous Monitoring
1c. Inference Engine Monitoring

Training Dataset Formation, ML Model Selection and Tuning

2a. ML Model Pool
2b. Training Dataset Formation
2c. Model Training
2d. Model Evaluation
2e. Best Model

Model Parameter Tuning

Optimized QPS Prediction Model

Inference Engine

A. Ferikoglou
IRIS Design – Online Phase

Server Metrics Collection

1. Run-time Monitoring
2. Average Metrics Accumulation prior to Deployment

QPS Prediction & Parallelism Level Selection

1. Form Test Dataset
2. QPS Prediction
3. Predicted QPS per Parallelism Level
4. Sort per CPU utilization
5. Selection of min. CPU parallelism that satisfies QoS

Target Task

Target QoS

A. Ferikoglou

IRIS: Interference and Resource Aware Predictive Inference Serving on Cloud Infrastructures
Evaluation

- We evaluate IRIS approach using:
  - different interference scenarios of varying intensity
  - different QoS constraints per inference engine (Low, Medium, and High)

- The model-less version of IRIS is compared with:
  - All the interference-aware model-specific IRIS schedulers
Evaluation – Model-less (Image Classification)

57.6% less QoS violations on average compared to model-specific schedulers

21.3% CPU utilization on average

2x, 1.2x, and 1.6x higher QPS for Low, Medium, and High QoS constraint on average
Evaluation – Model-less (Object Detection)

23.9% less QoS violations on average compared to model-specific schedulers

34.2% CPU utilization on average

1.4x, 1.8x, and 1.5x higher QPS for Low, Medium, and High QoS constraint on average
Conclusion

- We presented IRIS an interference- and resource-aware predictive scheduling framework for ML inference serving
  - Guarantees the application-specific QoS constraints while minimizing resource utilization
- The model-less feature achieves:
  - 1.5x fewer violations on average compared to model-specific
  - ≈30% less CPU utilization on average compared to model-specific
Thank You 😊