Locality Aware Model Serving with Resource Constraints

Anthony Thomas\textsuperscript{1}, Niketan Pansare\textsuperscript{2} and Berthold Reinwald\textsuperscript{2}

\textsuperscript{1} University of California, San Diego
\textsuperscript{2} IBM Research – Almaden
ML Model Lifecycle

Data Sources → Framework → Model

Training (offline)

Focus of this talk: Limited resources High-throughput + Low latency

Feature store for incremental training

Serving (on cloud)

Other Serving Scenarios:
- Extremely low-latency on-device serving
- Offline batch prediction
  - No latency requirements

User requests predictions
System generates predictions
User Feedback (E.g. incorrect prediction)
Summary

• We implement model serving as a multi-producer multi-consumer queueing problem and present an on-line algorithm which:
  1. Automatically selects the throughput maximizing batch size while bounding mean latency
  2. Accounts for data locality in a three tier cache hierarchy
  3. Never over-provision resources – no OOMs and avoid thrashing on CPU/GPU
  4. Does not introduce substantial scheduling overhead

• Goals:
  • Ensure maximum possible cluster utilization
  • While meeting user defined latency thresholds
Existing Model Serving Systems

Currently only two widely used open source model serving systems:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Clipper</th>
<th>TensorFlow Serving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Frontend</td>
<td>REST</td>
<td>REST/gRPC</td>
</tr>
<tr>
<td>Batching Strategy</td>
<td>AIMD</td>
<td>Static</td>
</tr>
<tr>
<td>GPU?</td>
<td>Sort of</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource Manager?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi-Node?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Scheduling Algo.</td>
<td>None</td>
<td>Round Robin</td>
</tr>
<tr>
<td>Backend Agnostic</td>
<td>Sort of</td>
<td>Sort of</td>
</tr>
</tbody>
</table>

TensorFlow Serving: VGG16

OOM on GPU
Serving in Apache SystemML

Phase 1: Queueing

- Scheduler checks if queue contains a full batch of requests \( or \) if a request has violated latency constraint:
  \[
  |q(m_i)| \geq b^*(m_i, CPU) \quad \| \quad |q(m_i)| \geq b^*(m_i, GPU) \\
  \text{now} - a(q(m_i)) \geq 1.2 \times l(m_i)
  \]
- If answer is “yes” scheduler routes a batch to an “execution queue” (will be explained later)
Serving in Apache SystemML

Client Frontend

Or Embeddable Service

Request Queue

Models:
\{m_i \mid i = 1:M\}

Scheduler

Executors

- Executor is assigned to either 1 CPU core, or 1 GPU
- Executors: \{c_j \mid j = 1:C\}
- \(M \gg C\) in general

Model Manager

- Allocates memory to executors
- Maintains SoftReference cache for unused model weights
Scheduling

- Queue based algorithm motivated by MapTask (Wang et. al. 2013) and Sparrow (Ousterhout et. al. 2013)
- Two asynchronous phases: routing and task assignment

1. Phase 2: Routing
   - Scheduler routes a batch of requests to up to three “execution queues” corresponding to data locality:
     1. Local executor queues – models whose weights are cached on an executor are routed here
        - If a model is cached on multiple executors – pick lowest utilization queue
        - Balances load across executors
     2. Global memory queue – models whose weights are cached in memory are routed here
     3. Global disk queue – models whose weights are cached on disk are routed here
   - We use “soft reservations” on requests to:
     - Allow for changing data locality
     - Allow for different batch sizes by executor type (e.g. CPU vs. GPU)

2. Phase 3: Task Assignment
   - Executor polls scheduler for new work
   - Scheduler picks next task from highest utilization queue
     - Ensures no single model gets to hog cache
   - If task fits in memory budget – executor dequeues original requests and processes them
Summary

• We implement model serving as a multi-producer multi-consumer queueing problem and present an on-line algorithm which:
  1. Automatically selects the throughput maximizing batch size while bounding mean latency
  2. Accounts for data locality in a three tier cache hierarchy
  3. Never over-provision resources – no OOMs and avoid thrashing on CPU/GPU
  4. Does not introduce substantial scheduling overhead

• Goals:
  • Ensure maximum possible cluster utilization
  • While meeting user defined latency thresholds
Backup
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_i \in M$</td>
<td>Set of models</td>
</tr>
<tr>
<td>$q(m_i)$</td>
<td>FIFO Request queue for $m_i$</td>
</tr>
<tr>
<td>$a(q(m_i))$</td>
<td>Arrival time of first request in $q(m_i)$</td>
</tr>
<tr>
<td>$t$</td>
<td>Current time</td>
</tr>
<tr>
<td>$</td>
<td>q(\star)</td>
</tr>
<tr>
<td>$e_j \in E$</td>
<td>Set of executors</td>
</tr>
<tr>
<td>$p(e_j)$</td>
<td>Executor type of $e_j$</td>
</tr>
<tr>
<td>$q(e_j)$</td>
<td>Local queue for $e_j$</td>
</tr>
<tr>
<td>$\bar{q}(p)$</td>
<td>Global queue for executor type $p$</td>
</tr>
<tr>
<td>$\mathbb{E}_t[q(\star)]$</td>
<td>Expected time to process requests in $q(e_j)$</td>
</tr>
<tr>
<td>$\mathbb{E}_{mem}[m \mid b]$</td>
<td>Expected memory for model $m$ with batch size $b$</td>
</tr>
</tbody>
</table>
for $m$ in $M$ do
  $\overline{P} \leftarrow \{p \mid p \in P : |q(m)| \geq B(m, p)\}$

  if $\overline{P} \neq \emptyset$ or $t - a(q(m)) \geq 1.2 \tau(m)$ then
    $\overline{P} \leftarrow (\overline{P} = \emptyset) \land P \land \overline{P}$
    $Q \leftarrow \{q(e) \mid e \in E : m \in e.cache \land p(e) \in \overline{P}\}$
    $Q \leftarrow Q \cup \{q(p) \mid p \in \overline{P}\}$
    $q^* \leftarrow \text{argmin}(\{E_t[q] \mid q \in Q\})$
    $b \leftarrow \min(|q(m)|, B(m, p(q^*)))$
    priority $\leftarrow t - a(q(m))$
    enqueue($q^*$, $(m, b)$, priority)
    $E_t[q^*] += \tau(m)$
  end
end

Algorithm 1: Routing Phase
Data: e - an executor, S - scheduler instance

\[ q^* \leftarrow (E_t[\bar{q}(p(e))] \geq E_t[q(e)]) \ ? \bar{q}(p(e)) : q(e); \]

\[ m, b \leftarrow \text{dequeue}(q^*); \]

if \( b \geq |q(m)| \ or \ S.f\_freeMemory \leq E_{mem}[m \mid b] \) then

\[ \text{requests} \leftarrow \emptyset; \]

else

\[ \text{requests} \leftarrow \text{dequeue}(q(m),b); \]

\[ E_t[q^*] \leftarrow \tau(m); \]

end

\[ e.\text{execute(requests)}; \]

Algorithm 2: Task Assignment Phase
Model Serving in Enterprise

**WML Models/User**
Number of model deployments per user (p=0.995)

**WML Serving Requests**
Number of calls to deployed wml models per day since 2018-01-01

**BigHead Serving System**
- 100+ models in production
- Latency requirement: 5-13ms

**Michelangelo Serving System**
- 5 main types of model – GLM to CNNs
- Latency requirement: 5-10ms
- ~250,000 reqs/second on high traffic models

**LASER Serving System**
- Latency requirement: ~10ms