Serving DNNs like Clockwork
Performance Predictability from the Bottom Up
Serving DNNs like **Clockwork**

Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!

**Clockwork**

End-to-end predictable DNN serving platform for the Cloud

- Supports 1000s of models concurrently per GPU
- Mitigates tail latency, supporting tight latency SLOs (10—100 ms)
- Close to ideal goodput under overload, contention, and bursts
Background
Inference Serving at the Cloud Scale is Difficult
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

<table>
<thead>
<tr>
<th>ResNet-50</th>
<th>Latency</th>
<th>Throughput</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
<td>$</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
<td>$$$</td>
</tr>
</tbody>
</table>

HW accelerators are necessary!
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

Problem

How can cloud providers efficiently share resources while meeting SLOs?

Latency

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Throughput</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
<td>$</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
<td>$$$</td>
</tr>
</tbody>
</table>

HW accelerators are necessary!
Existing Systems Incur Very High Tail Latency
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

![Latency Graph]

Clipper 100ms SLO

200 ms
Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Clipper 100ms SLO

Concurrent DNN inference over GPU
High variance in latency
Throughput gains only 25%
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Concurrent DNN inference over GPU
- 100x
- High variance in latency
- Throughput gains only 25%

Clockwork adopts a contrasting approach!
- Single-thread latency is extremely predictable
- Preserves DNN predictability at every stage of model serving

Tail latency within SLO
- 95 ms
- Preserves DNN predictability at every stage of model serving

Clockwork 100ms SLO
- Tail latency within SLO
- 95 ms
- Tail latency >> SLO

Existing Systems Incur Very High Tail Latency
How does Clockwork Achieve End-to-End Predictability?
Design Principles

Goal: 1000s of models, many users, limited resources

Maximize sharing
Design Principles

Goal: 1000s of models, many users, limited resources

1. Predictable worker with no choices

2. Consolidating choices at a central controller

3. Deadline-aware scheduling for SLO compliance

Maximize sharing
Users upload pre-trained models in advance: ● △ □ ★ ⭐ ...
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ⬤ △ ■ ● ● ...  

Inference request for ★  
Cold

Allocate memory for ★ ...  

Inference request for ★ (execute, since already in GPU memory)  
Warm

Allocate memory for ★ ...

Execute inference

Worker Node

RAM

GPU Memory

Execute inference

GPU Exec

GPU

32 GB

4 TB
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ ▪ ✷ ... 

Inference request for ✷ (execute, since already in GPU memory)

Inference request for ✷ ...

Cold

Allocate memory for ✷ ...

Warm

Execute inference

Worker Node

RAM

GPU Memory

GPU Exec

4 TB

32 GB
Users upload pre-trained models in advance: ○△□★★...  

Queues

Inference request for ★  
Allocate memory for ★...  
Execute inference  

Inference request for ★ (execute, since already in GPU memory)  

Cold

Warm

Concurrent inferences

+ Proprietary & undocumented policies

Unpredictable response times

![Graph showing latency (ms) vs concurrency]
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ⬜️ ▲ □ ▫ ▪ ▪ ⬛ ... 

Inference request for ⭐ (execute, since already in GPU memory) 

Cold

Allocate memory for ⭐ ... 

Execute inference

Warm

Concurrent inferences

+ Proprietary & undocumented policies

→ Unpredictable response times

Managed memory can be unpredictable
- GPU memory (cache) hits & misses

ResNet-50 — Hit: 2.3 ms | Miss: 10.6 ms

Workers

RAM

GPU Memory

GPU Exec

GPU

Worker Node

Queues

4 TB

32 GB
Designing a Predictable Worker (2/2)

Predictable Clockwork worker process

Worker Node
Managed memory can be unpredictable

---

**Solution**
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

---

Predictable Clockwork worker process

---

**Concurrent inferences**

Proprietary & undocumented policies

Unpredictable response times

---

**Solution**
Execute inference one at a time
Designing a Predictable Worker (2/2)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Predictable Clockwork worker process

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Solution
Execute inference one at a time
Designing a Predictable Worker (2/2)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Predictable Clockwork worker process

- Earliest Deadline First
- LOAD/UNLOAD (△, Deadline)
- INFER (★, I/P, Deadline)

Choices outsourced via action APIs

Concurrent inferences
- Proprietary & undocumented policies
- Unpredictable response times

Solution
Execute inference one at a time
Consolidating Choices

Users → Centralized Controller → Worker processes → GPU Worker Node $W_1$

- **Centralized Controller**
- **Worker processes**
  - LOADs
  - INFERS

- **GPU Worker Node $W_1$**
  - **GPU**
  - **RAM**
  - **GPU Memory**
    - PageCache
    - GPU Exec

- **GPU Memory**
  - **RAM**
  - **GPU Memory**
    - PageCache
    - GPU Exec
Consolidating Choices
Consolidating Choices

Users → Centralized Controller → Worker processes

Global State Manager:
- Latency Profiles
- Pending Tasks
- Memory State

Smarter load balancing & scheduling decisions

GPU Worker Node $W_1$

Tasks: LOADs, INFERS

State:
- Global State Manager
- Memory
- Pending Tasks
- Latency Profiles

GPU Memory
PageCache
CPU Exec

RAM
SLO-aware Scheduling
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

Worker Node $W_1$

- RAM
- GPU Memory
- GPU Exec
- LOADs
- INFERs
- Page Cache
SLO-aware Scheduling

Users

Centralized Controller

Worker processes

RAM

GPU Memory

GPU Exec

GPU Node

W1

Pending Tasks

W1 GPU

Time

 tnw

 tfree
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

Worker Node $W_1$

Inference request for $W_1$'s GPU

Pending Tasks

Time

Since $t_{\text{deadline}} < t_{\text{free}}$, inference request for $W_1$ is cancelled
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

- Users submit tasks.
- Centralized Controller manages task allocation.
- Worker processes execute tasks.

Inference request for $\star$ is further away.

Deadline is further away.

From latency profiles.

Time:
- $t_{\text{now}}$: Current time.
- $t_{\text{deadline}}$: Deadline for task completion.
- $t_{\text{free}}$: Available time before deadline.

Pending Tasks:
- $W_1$: Task pool.

GPU Worker Node $W_1$
SLO-aware Scheduling

From latency profiles

Deadline is further away

Since $t_{free} + \Delta_{infer} < t_{deadline}$, inference request for ✴ is scheduled on $W_1$
SLO-aware Scheduling

Centralized Controller

Worker processes

Users

Worker Node $W_1$

GPU Exec

GPU Memory

LOADs

INFERS

RAM

$W_1$ GPU

Pending Tasks

$\Delta_{\text{infer}}$

Time

$t_{\text{now}}$

$t_{\text{free}}$

$t_{\text{deadline}}$

What if $\Delta$ does not finish on time?
SLO-aware Scheduling

Centralized Controller

User processes

Worker Node $W_1$

GPU Memory

GPU Exec

RAM

What if $\Delta$ does not finish on time?

Pending Tasks $\Delta_{\text{infer}}$

$W_1$ GPU

Time $t_{\text{now}}$, $t_{\text{free}}$, $t_{\text{latest}}$, $t_{\text{deadline}}$

Clockwork also tracks $t_{\text{latest}}$, and cancels $\star$ if it fails to start before $t_{\text{latest}}$
SLO-aware Scheduling

Many benefits
- Prevent wasteful work
- Manage LOAD → INFER dependencies
- Choosing the best batching strategy
Evaluation
Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?
Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?
Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

Are Clockwork workers predictable?

Does consolidating choice help achieve end-to-end predictability?

Can Clockwork controller Scale?

Workloads from production traces
<table>
<thead>
<tr>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>How does Clockwork compare to prior model serving systems Clipper and INFaaS?</td>
</tr>
<tr>
<td>Can Clockwork serve thousands of model instances?</td>
</tr>
<tr>
<td>How low can Clockwork go in terms of the latency SLOs it can satisfy?</td>
</tr>
<tr>
<td>Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?</td>
</tr>
</tbody>
</table>

**This talk**

- Are Clockwork workers predictable?
- Does consolidating choice help achieve end-to-end predictability?
- Can Clockwork controller Scale?

**Workloads from production traces**
Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory + 1 Controller + 1 Client

Microsoft’s Azure Functions

Shahrad et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.” USENIX ATC 2020

4026 model instances
- Saturates 768 GB RAM
- 61 different model architectures
- ResNet, DenseNet, Inception, etc.

Workload

46,000 functions, 2 weeks
- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling.

Overpredictions → Idle resources
Underpredictions → SLO violations

Clockwork consistently overpredicts more than its underpredicts.

Errors are significant only in extremely rare cases.

Underprediction error = 55us
Overprediction error = 144us
Does Consolidating Choice Help?

Offered load \(\sim 10,000\) r/s, periodic spikes \(\sim 12,000\) r/s

Latency SLO = 100 ms deadline for each request
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork
- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

Goodput = SLO compliant throughput
Latency of all completed requests

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Latency of all completed requests

Maximum 99th %ile
Median
Mean
Cold Warm
Coldstarts

Goodput = SLO compliant throughput
Offered Load

Goodput

SLO compliant throughput

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Latency of all completed requests

Maximum 99th %ile
Median
Mean
Cold Warm
Coldstarts

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Offered Load

Goodput

SLO compliant throughput

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Latency of all completed requests

Maximum 99th %ile
Median
Mean
Cold Warm
Coldstarts

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Offered Load

Goodput

SLO compliant throughput

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests

Latency of all completed requests

Maximum 99th %ile
Median
Mean
Cold Warm
Coldstarts

Batching prioritized, absorbs spikes
Many cold starts
Cold requests = 1.3% of all requests
Does Clockwork Controller Scale?

Methodology
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Does Clockwork Controller Scale?

Methodology
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Does Clockwork Controller Scale?

Methodology
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers

Maximum goodput: 103,387 r/s for 110 workers

Linear scalability until #workers = 110
Goodput limited by worker’s utilization

Bottleneck shifts to Clockwork

Peak Goodput (r/s)

0 20 40 60 80 100 120 140
Number of Workers
Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability
- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform
- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Outperforms state-of-the-art DNN serving platforms
- Efficiently fulfills aggressive tail-latency SLOs
- Supports 1000s of DNN models with varying workload characteristics concurrently on each GPU

https://gitlab.mpi-sws.org/cld/ml/clockwork