Serving DNNs like Clockwork
Performance Predictability from the Bottom Up
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!
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
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

1000s of trained models of different types and resource requirements
Requests arrive at different rates and regularity

1000s of trained models of different types and resource requirements
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

Sustained + High Rate
Requests arrive at different rates and regularity
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)

- ResNet-50
  - Latency: 175 ms
  - Throughput: 6 req/s
- GPU
  - Latency: 2.8 ms
  - Throughput: 350 req/s

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)

<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?

<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!
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

CDF
Existing Systems Incur Very High Tail Latency

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

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

Tail latency >> SLO

Existing Systems Incur Very High Tail Latency

Concurrent DNN inference over GPU
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

High variance in latency

Throughput gains only 25%
Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Existing Systems Incur Very High Tail Latency

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

Single-thread latency is extremely predictable
Existing Systems Incur Very High Tail Latency

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

Tail latency >> SLO

Preserves DNN predictability at every stage of model serving

Clockwork adopts a contrasting approach!

Single-thread latency is extremely predictable

Concurrent DNN inference over GPU

High variance in latency

Throughput gains only 25%
Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Preserves DNN predictability at every stage of model serving

Clockwork adopts a contrasting approach!

Single-thread latency is extremely predictable
How does Clockwork Achieve End-to-End Predictability?
Design Principles

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

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
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
Designing a Predictable Worker (1/2)

Worker Node

- **RAM**
- **GPU Memory**
- **GPU Exec**

- 4 TB
- 32 GB

Worker Node
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ □☆ □ ...
Users upload pre-trained models in advance: ● △ ■ ▲ ☆ ● ...  

Inference request for ★ ...  

Allocate memory for ★ ...  

Cold  

Worker Node

RAM

GPU Memory

GPU Exec

32 GB

4 TB
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ □ ★ ☆ ...  

Inference request for ★  
allocate memory for ★ ...  

Allocate memory for ★ ...  
Execute inference  

Worker Node  

RAM  
GPU Memory  
GPU Exec  

4 TB  
32 GB  

Users upload pre-trained models in advance: ● △ ■ □ ★ ☆ ...
Inference request for ★
Allocate memory for ★ ...
Execute inference

Worker Node

RAM
GPU Memory
GPU Exec

4 TB
32 GB
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ ⭐ ⭐ ⭐ ⭐ ... → RAM

Inference request for ⭐ ... → Allocate memory for ⭐ ... → GPU Memory

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

Worker Node → GPU Exec → GPU

4 TB

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

Inference request for ★

Cold

Allocate memory for ★ ...

Execute inference

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

Warm

Worker Node

RAM

GPU Memory

Execute

GPU Exec
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ⬤ △ ■ ▲ ⭐ ⧤ ...

Inference request for ⭐

Allocate memory for ⭐ ...

Execute inference

Cold

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

Warm

**Worker Node**

- **RAM**
- **GPU Memory**
- **GPU Exec**

- 4 TB
- 32 GB

Queue
Users upload pre-trained models in advance: ● △ □ ★ ☆ ...  

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

Warm  

Concurrent inferences  
+ Proprietary & undocumented policies  
Unpredictable response times  

Workers upload pre-trained models in advance:  

Queues  

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

Cold  

Proprietary & undocumented policies
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ □ ★ ⮞ ...

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

Warm

Inference request for ★ ...

Allocate memory for ★ ...

Execute inference

Queues

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

Worker Node

32 GB

4 TB

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Users upload pre-trained models in advance: ● △ □ ★ ⮞ ...

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

分配内存为 ★ ...

执行推理 (since already in GPU memory)

Warm

并发推理

专有及未公开政策

不可预测的响应时间
Designing a Predictable Worker (2/2)

Predictable Clockwork worker process
Designing a Predictable Worker (2/2)

Predictable Clockwork worker process

Concurrent inferences

Solution
- Execute inference one at a time

Proprietary & undocumented policies
- Unpredictable response times
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
Managed memory can be unpredictable

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

Predictable Clockwork worker process

- Earliest Deadline First
- PCI
- GPU
- Time

Managed memory

Concurrent inferences

Proprietary & undocumented policies
- Unpredictable response times

Solution
Execute inference one at a time
Managed memory can be unpredictable

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

Designing a Predictable Worker (2/2)

Choices outsourced via action APIs

Predictable Clockwork worker process

Concurrent inferences

Proprietary & undocumented policies

Solution
Execute inference one at a time

Unpredictable response times
Consolidating Choices

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

- RAM
- GPU Memory
- PageCache
- LOADs
- INFERs
- GPU Exec
- GPU Node
Consolidating Choices

Users ➔ Centralized Controller ➔ Worker processes

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

GPU Worker Node $W_1$
- RAM
- GPU Memory
- PageCache
- CPU
- GPU Exec
- LOADs
- INFERs

Profi les
Pending Tasks
Memory State
Global State Manager
Centralized Controller
Worker processes
Users
Consolidating Choices

Users → Centralized Controller

Worker processes

Global State Manager

Latency Profiles
Pending Tasks
Memory State

Smarter load balancing & scheduling decisions

Centralized Controller

Worker processes

RAM

GPU Memory
PageCache

GPU Exec

GPU Worker Node W₁
SLO-aware Scheduling

Users

Centralized Controller

Worker processes

Worker process

GPU Memory

GPU Exec

GPU Worker Node W₁
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

Pending Tasks

W₁

GPU

Time

t_now

t_free

CPU

GPU Memory

LOADEDs

INFERs

GPU Exec

GPU Worker Node W₁
SLO-aware Scheduling

Centralized Controller

Worker processes

Pending Tasks

W₁ GPU

Time

Inference request for

Since t_{deadline} < t_{free}, inference request for ⬤ is cancelled
SLO-aware Scheduling

Centralized Controller

Worker processes

GPU Worker Node $W_1$

Pending Tasks

Deadline is further away

Inference request for $\star$
SLO-aware Scheduling

Centralized Controller

Worker processes

From latency profiles

Pending Tasks

Deadline is further away

Inference request for

Time

W_1

GPU

Deadline is further away from latency profiles.

Workers processes

GPU Worker Node W_1
SLO-aware Scheduling

From latency profiles

Deadline is further away

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

Users → Centralized Controller → Worker processes

What if Δ does not finish on time?
SLO-aware Scheduling

Centralized Controller

Worker processes

Users

Worker Node $W_1$

GPU

RAM

GPU Memory

GPU Exec

Pending Tasks

$W_1$ GPU

Time

$t_{\text{now}}$

$t_{\text{free}}$

$t_{\text{latest}}$

$t_{\text{deadline}}$

$\Delta_{\text{infer}}$

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

Clockwork also tracks $t_{\text{latest}}$, and cancels $\star$ if it fails to start before $t_{\text{latest}}$
Many benefits

- Prevent wasteful work
- Manage LOAD $\rightarrow$ INFER dependencies
- Choosing the best batching strategy
Evaluation
Questions
How does Clockwork compare to prior model serving systems Clipper and INFaaS?
Questions

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

Can Clockwork serve thousands of model instances?
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?
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
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?

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
Experiment Setup

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

Workload

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

46,000 functions, 2 weeks
- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads

Rate
Time
**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.

**46,000 functions, 2 weeks**
- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads

**Workload**
Are Clockwork Workers Predictable?
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

- Overpredictions ➔ Idle resources
- Underpredictions ➔ SLO violations
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions ➔ Idle resources
Underpredictions ➔ SLO violations

Experiment duration = 6 hours,
Offered load ~ 10,000 r/s
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.
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 ~10,000 r/s, periodic spikes ~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

Goodput = SLO compliant throughput

\[
\text{Goodput} = \text{SLO compliant throughput}
\]

![Graph showing time vs. throughput with data points indicating offered load and goodput over time with SLO compliance.](image)
Does Consolidating Choice Help?

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

**Goodput** = SLO compliant throughput

- Latency of all completed requests

### Graph

- **Time (Minutes)**: 0, 60, 120, 180, 240, 300, 360
- **Offered Load Goodput**: Various lines indicating different load levels
- **Latency (ms)**: 0, 40, 80, 1200, 10000, 12000

**Conclusion**

- Goodput is SLO compliant throughout the test period.
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
Does Consolidating Choice Help?

- offered load \( \approx 10,000 \text{ r/s} \), periodic spikes \( \approx 12,000 \text{ r/s} \)
- Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork

- Goodput \( \approx \) offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
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

Offered Load
Goodput

Maximum
99th %ile
Median
Mean
Cold
Warm
Coldstarts
Does Consolidating Choice Help?

- Goodput $\approx$ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

Latency SLO = 100 ms deadline for each request

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

The workload is successfully scheduled by Clockwork
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 $\approx$ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
Does Clockwork Controller Scale?
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
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

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

Linear scalability until #workers = 110

Goodput limited by worker’s utilization

Bottleneck shifts to Clockwork

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

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