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

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

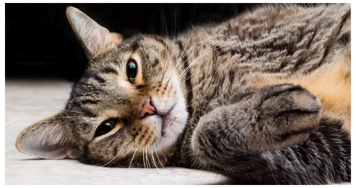
†Hong Kong University of Science and Technology

‡University of Nevada, Reno

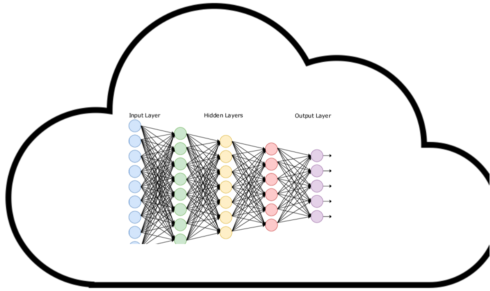
Machine Learning Serving - MLaaS

Deploy a trained model on cloud for user requests

- Highly dynamic demand
- Stringent Service Level Objectives on latency



+



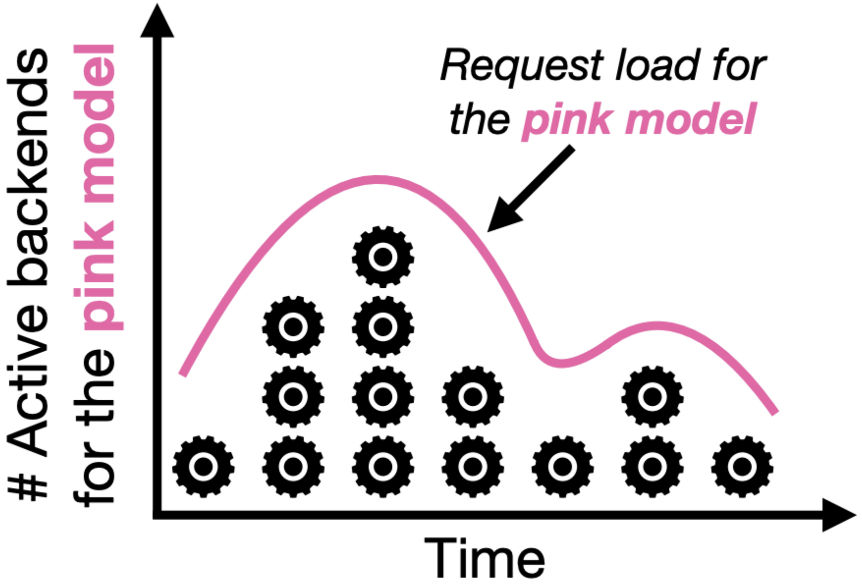
= “tabby cat”

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

Conventional Autoscaling – AWS SageMaker

Amazon SageMaker



- Reactive scaling: based on current load

Provisioning Time (minutes)	>>	Execution Time (< 1s)

Hide provisioning time -> over-provisioning

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

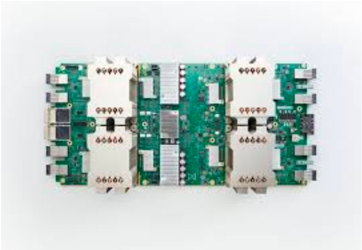
Sagemaker suggests to adjust over-provisioning factor from 2

[1] https://people.mpi-sws.org/~arpanbg/pdfs/middleware2017_slides.pdf

ML accelerators: GPU, TPU, FPGA

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

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



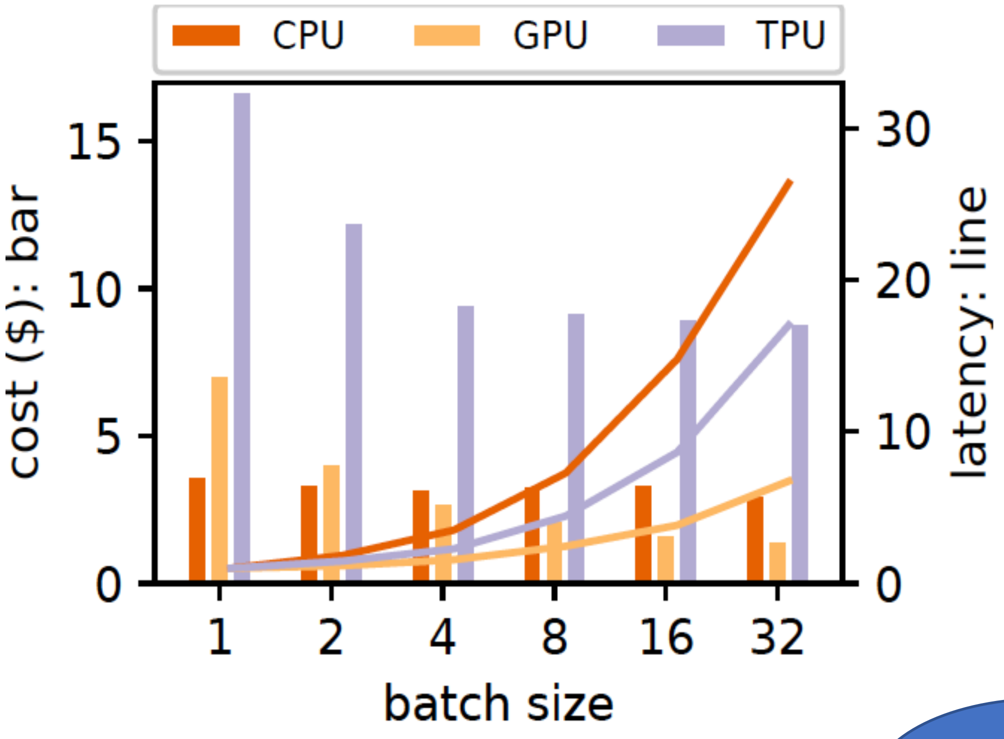
Inference

- Run comfortably without them
- Way less parallelism

- Choose between CPU and accelerators
- Justify the price tag

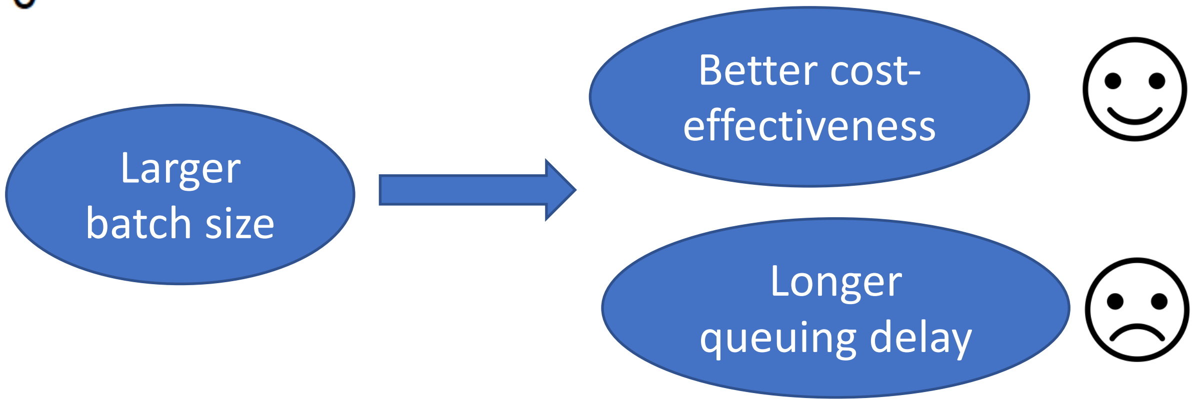
Characterization: CPU vs. GPU vs. TPU

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



(a) Inception-v3

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



- 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

Amazon EC2

Pay-as-you-go

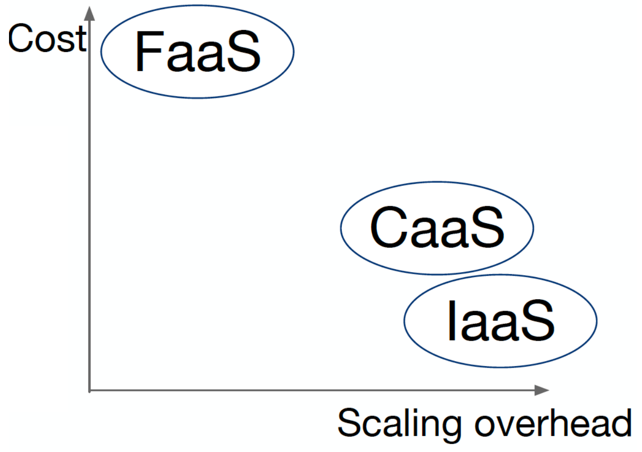
AWS Lambda

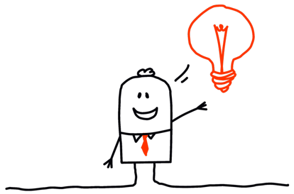
Pay for what you use

Infrastructure as a Service (IaaS)	**Container as a Service (CaaS)**	**Function as a Service (FaaS, serverless comp.)**

ML Model	EC2		ECS		Lambda	
	\$	*t* (ms)	\$	*t* (ms)	\$	*t* (ms)
Inception-v3	5.0	210	9.17	217	19.0	380
Inception-ResNet	9.3	398	16.4	411	39.3	785
OpenNMT-ende	51.5	2180	96.3	2280	155	3100

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

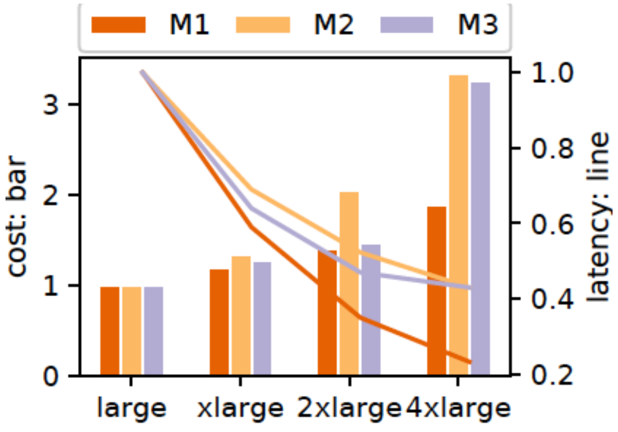




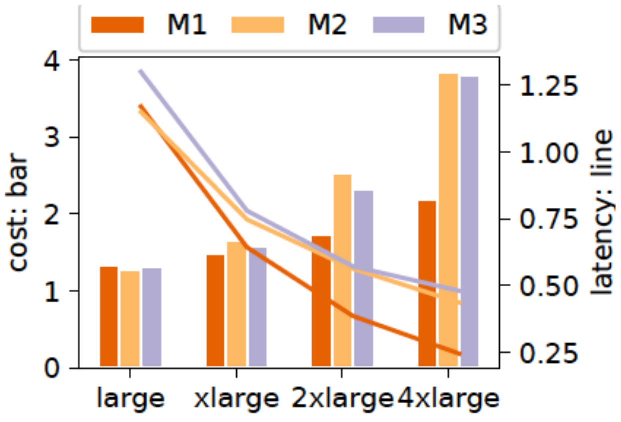
Combine IaaS's *cost advantage* with FaaS's *scalability*

- Instead of overprovisioning IaaS, use FaaS to handle demand surge and spikes

IaaS: Instance Families and Sizes



(a) c5 instances



(b) m5 instances

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

There are 4 families of instance in EC2 :

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

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

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

Characterization Summary

- 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

Design Considerations

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

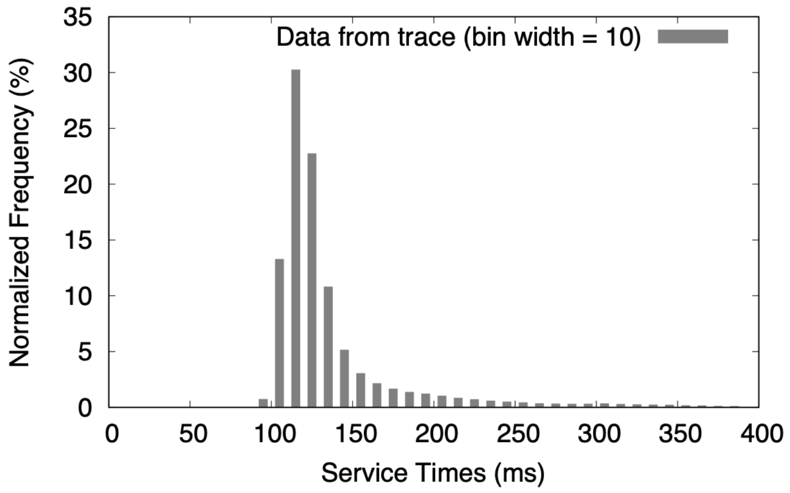
- Use dynamic batching, batching requests according to arrival rate and SLO specification

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

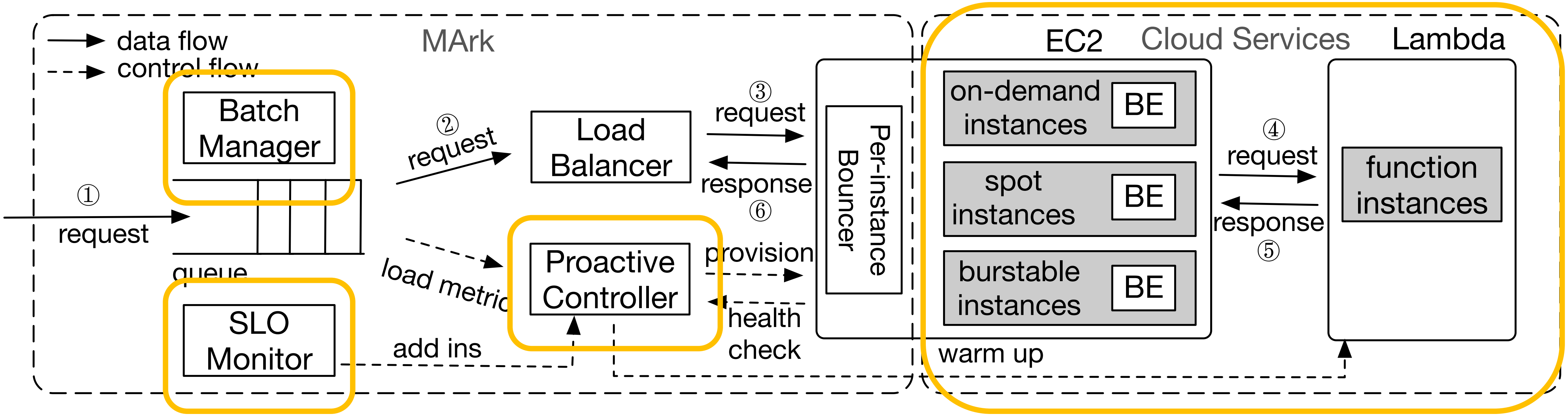
- Overall response time: no closed form solution
- ML inference execution time is <u>deterministic</u>

Best effort solution:

Monitor the queuing time for each request, direct requests to FaaS when necessary

[1] Gujarati, Arpan, et al. "Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency." Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. ACM, 2017.

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

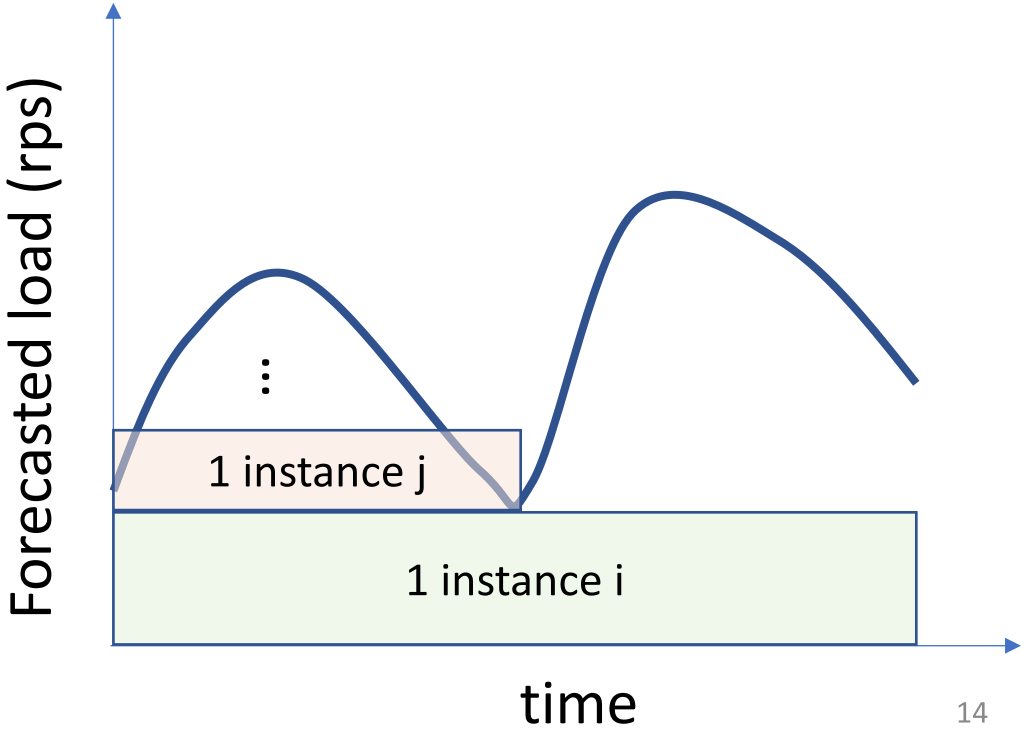
Heterogeneous Cluster
Deterministic processing time
Assume Poisson arrival

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

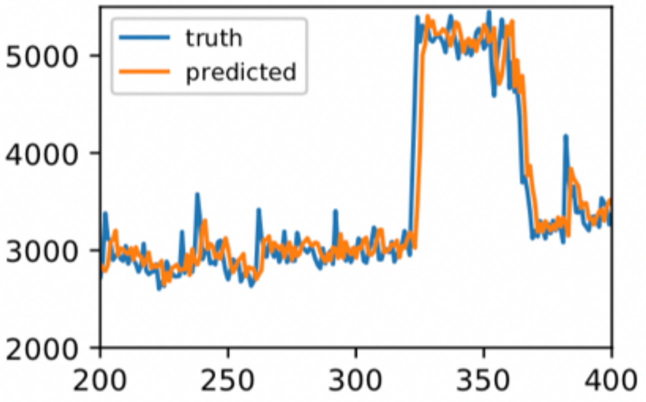
Model	Type	Framework	Size
Inception-v3	Image Classification	Tensorflow Serving	45MB
NASNet	Image Classification	Keras	343MB
LSTM-ptb	Language Modeling	MXNet Model Server	16MB
OpenNMT-ende	Machine Translation	Tensorflow Serving	330MB

Test Bed

AWS

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

Cost Savings

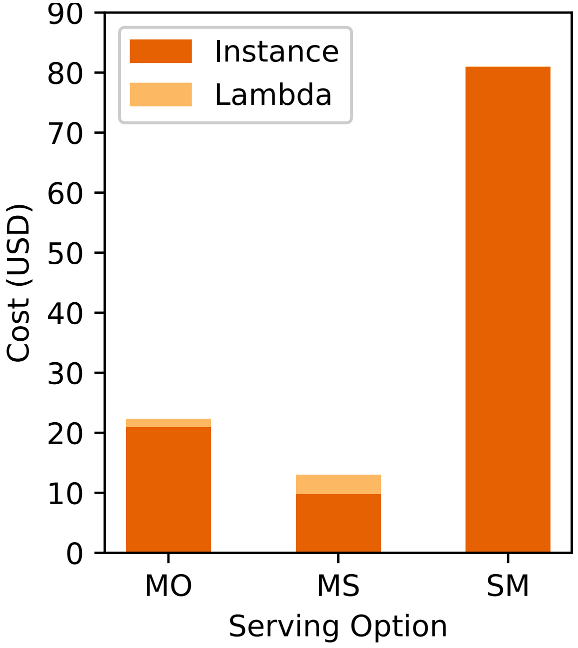


Twitter workload

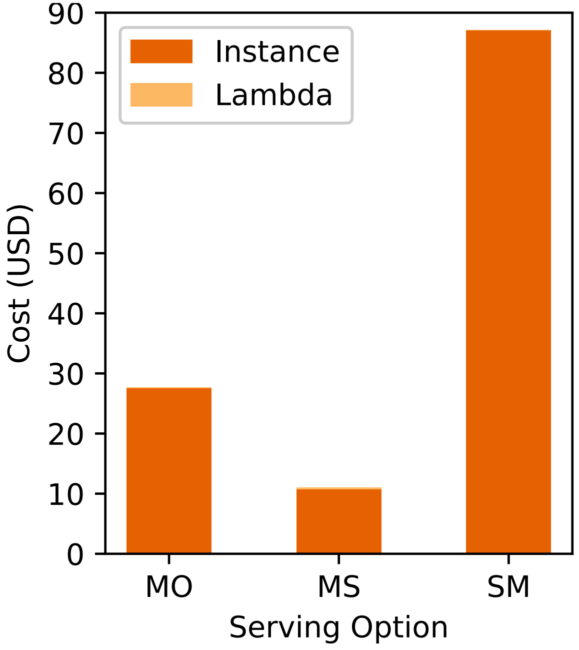
arrival pattern abstracted from real time tweets

- 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



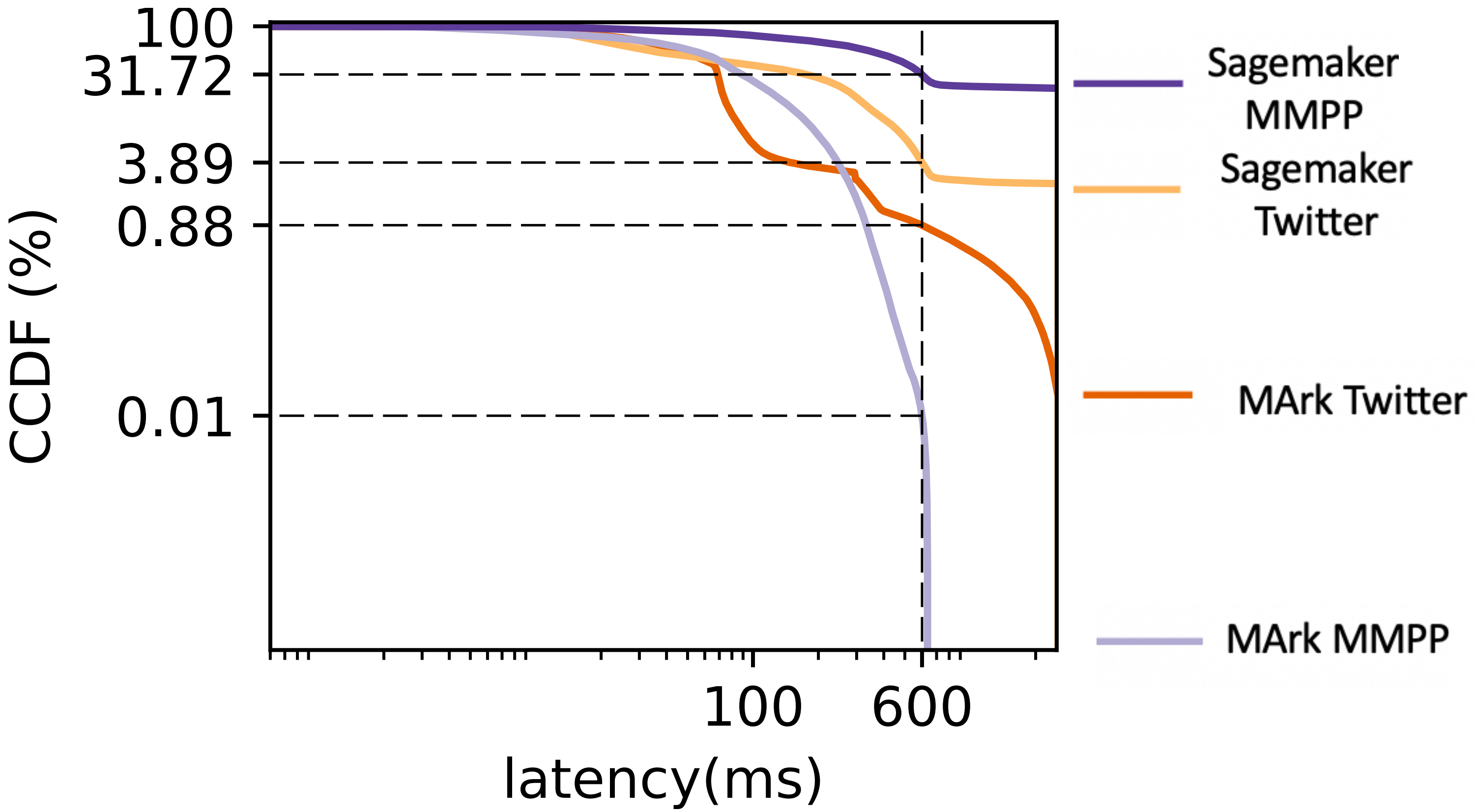
Inception-v3



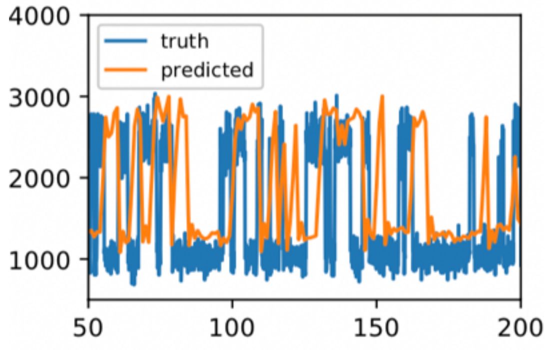
OpenNMT-ende

SLO Compliance

Latency complementary cumulative distribution function



Inception-v3

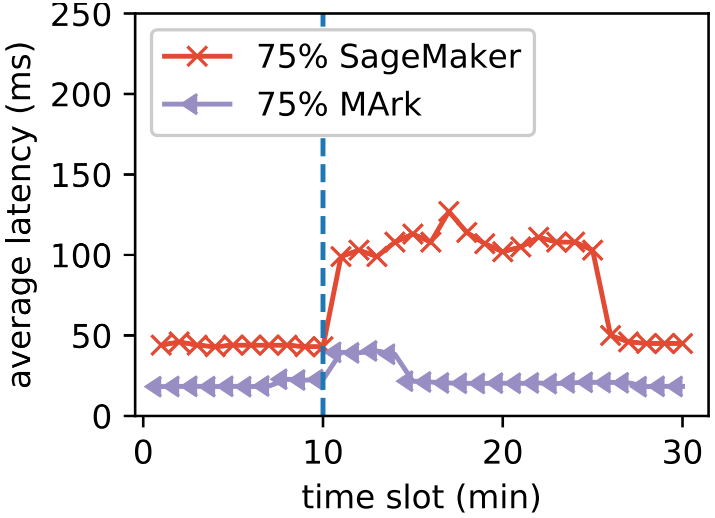


Markov-modulated Poisson Arrivals (MMPP)

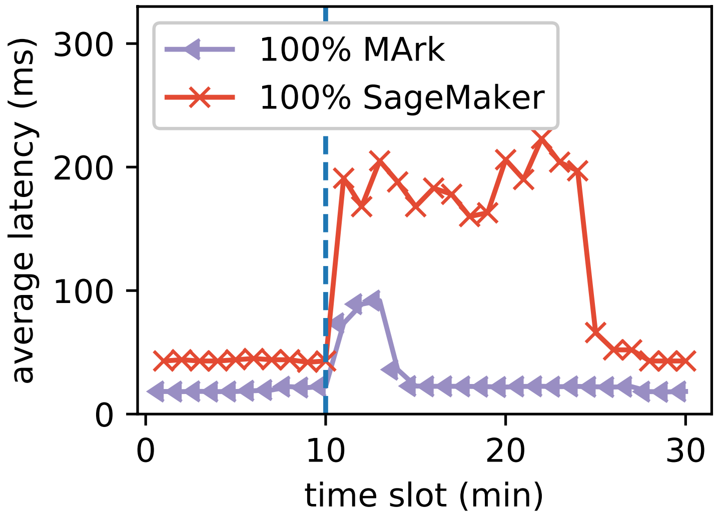
What if workload is unpredictable?

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
  - Proposed combining IaaS and FaaS for ML serving
- Designed a cost-effective, SLO-aware system MArk
  - Predictive greedy provisioning
  - Dynamic batching to exploit accelerators
  - Support spot instances
- Implemented Mark, and evaluated it on AWS
  - 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