Efficient Deep Learning Inferencing on Cloud Kubernetes Clusters using Smart Arm Node Provisioning

Anne Holler, Advisor, Elotl

Efficient DL Inferencing: Resource Mgmt Challenges

Deep Learning (DL) models are being successfully applied in many fields

- Most notably image and natural language processing

However, **managing DL inferencing at scale**, often run in the cloud for resource flexibility

- **Presents cost & operational complexity challenges**

Efficient DL Inferencing: Cost Challenges

DL models differ greatly in serving resources needed, particularly wrt system memory for weights
- Economical to use cloud **instance shapes that are right-sized** for models

In production, model's prediction load can vary significantly according to TOD & other factors
- Desirable to automatically adjust the number of cloud serving **instances to match current load**

x86+GPU compute instances, often used to serve DL models, are costly compared to CPU-only
- Worthwhile to evaluate if an inference use case can be handled well on **thriftier CPU-only instances**

Efficient DL Inferencing: Operational Complexity Challenges

Selecting currently-available minimum-cost cloud resources that match inferencing needs

- From **large and ever-evolving set of instance types**, whose **availability may vary over time**
- When **inferencing resource needs can change with model updates**

Efficient DL Inferencing: Talk Overview

Present approach to managing cost & operational complexity of DL inferencing at scale

Given cloud resources, organized in a Kubernetes (K8s) cluster for production container orchestration, the approach combines:

- **Right-sizing inference resources**
  - **Use Elotl Luna smart node provisioner**
  - Luna adds right-sized compute to cloud K8s cluster when needed and removes it when not

- **Right-sizing inference compute type**
  - Use CPU, can provide a price-performance advantage on DL inferencing relative to GPU for low-latency low-batch-size use cases.
  - **Use Arm64 CPU**, can yield price-performance advantage over x86 CPU

Efficient DL Inferencing: Talk Overview, Con't

Evaluate cost & operational complexity benefits of the right-sized approach

Compared w/2 common non-right-sized approaches running on:

- **3 kinds of public cloud K8s clusters**
  - Oracle Cloud (OCI) OKE K8s cluster
  - Google Cloud (GCP) GKE K8s cluster
  - Amazon Web Services (AWS) EKS K8s cluster

- **2 kinds of ARM compute**
  - Ampere A1 Arm compute
  - AWS Graviton2 Arm compute

AMPERE®

AWS Graviton2

Efficient DL Inferencing: Talk Outline

Efficient DL Inferencing: Resource Mgmt Challenges, Cost Challenges, Complexity Challenges

Efficient DL Inferencing: Talk Overview

Efficient DL Inferencing: Right-Sizing Approach

- **Right-sizing Inference Resources: Use Luna Smart Provisioner**
- **Right-sizing Inference Compute Type: Use Arm CPU Compute**

Experimental Setup for Evaluating Benefits of Right-Sizing Approach vs 2 other Sizing Approaches

Experimental Results for Sizing Approaches on 3 Cloud K8s Clusters

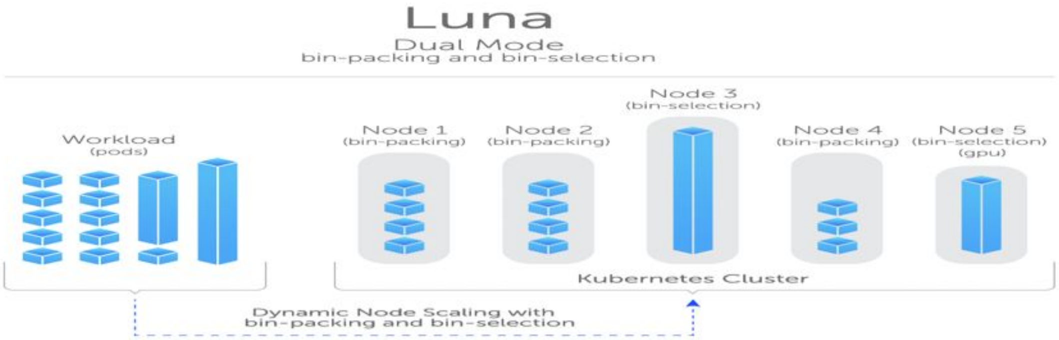
Conclusion: Summary and One More Thing

Right-Sizing Inference Resources: Use Luna Smart Provisioner

Right-sizing inference resources leverages Elotl Luna smart provisioner to add/remove nodes

Luna adds nodes to cloud K8s for pending placements, using bin-packing vs bin-selection policy

- **Uses bin-packing for pods w/smaller resource requests**
  - Node is allocated for hosting multiple pods
- **Uses bin-selection for pods w/larger resource requests or w/special requests such as GPU**
  - Node is allocated for hosting single pod, node is right-sized for the pod's resource request



Right-Sizing Inference Resources: Adapting Luna for DL Serving

Luna smart node provisioner suitable for handling any bursty workloads running on cloud K8s

- Previously reported on **Luna's mgmt of x86+GPU compute for DL training workloads**
  - Used Luna to validate Ludwig DL AutoML running on Ray Tune for tabular & text classification datasets

Simple to adapt Luna to managing right-sized compute for DL inference workloads

- Enabled Luna option to **consider Arm instances** for node provisioning in addition to x86
- Tuned **bin-packing vs bin-selection policy** to leverage granular shapes available for Arm
- Added support to run **Luna on Arm**, to allow Arm-only K8s clusters, when more efficient

Different Luna on Arm LOL

SCaLE

Right-Sizing Inference Compute Type: Use Arm CPU Compute

While the latest MLCommons datacenter inference benchmarks show GPUs delivering highest absolute performance for tested scenarios, YMMV

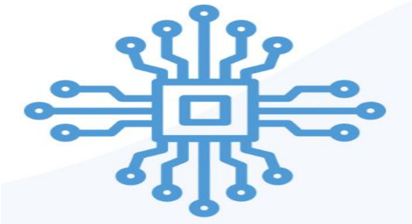
- **Low-latency small-batch-size cloud server scenarios can run more cost-efficiently on CPU systems** (see here)

Pre-deployment tests for our use case established that CPU-only deployment worked well

- **For right-size runs, no GPU resource requested; Luna chose Arm CPU as less expensive than x86 CPU**
- Arm can be more performant; e.g. Article shows Ampere A1 outperforming x86 CPU shapes on resnet_50_v15 model at fp32, same cost delivering >1.5x AMD's best E4 OCI shape & >2x Intel's Standard3 OCI shape

Right-sizing inference compute uses 2 kinds of Arm CPU compute

- **OKE,GKE: Ampere A1** + Ampere Optimized AI library is tuned for CPU DL inference w/o accuracy degradation.
- **AWS: Graviton2** is tuned for best price performance in AWS for variety of workloads, including DL inference.



Efficient DL Inferencing: Talk Outline

Efficient DL Inferencing: Resource Mgmt Challenges, Cost Challenges, Complexity Challenges

Efficient DL Inferencing: Talk Overview

Efficient DL Inferencing: Right-Sizing Approach

Experimental Setup for Evaluating Benefits of Right-Sizing Approach

- **Workload System, Models, Deployment, Load & Scaling Range**
- **Sizing Approaches (Right-sized, Max-sized, Dynamic fixed-size) Compared**

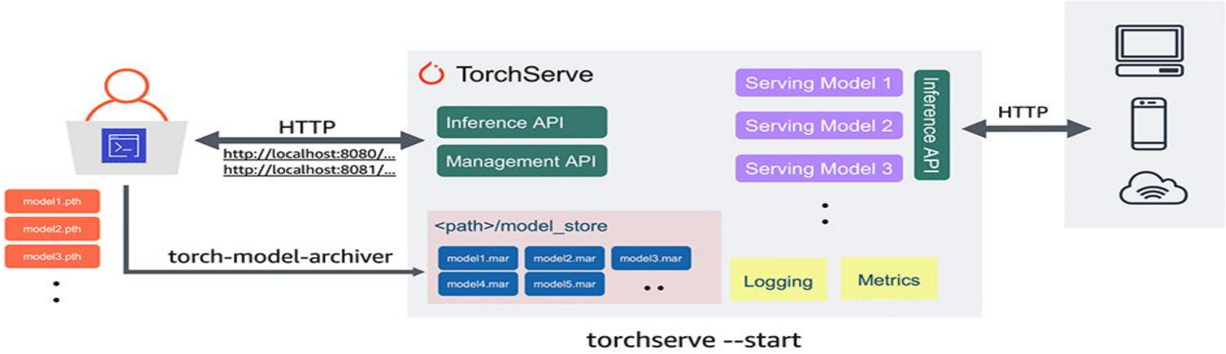
Experimental Results for Sizing Approaches on 3 Cloud K8s Clusters

Conclusion: Summary and One More Thing

Evaluating Benefits of Right-Sizing Inference: Workload System

Wanted to use tunable DL inference system, deployable on cloud K8s & including autoscaling

- **Chose TorchServe, high-performance full-featured open source tool for serving PyTorch models**
  - Supports tuning resource-related settings for each of the set of models being served
- **TorchServe supports deployment on Cloud-based K8s clusters**
  - Description of use on AWS here, which is source of architecture graphic shown below
- **TorchServe includes load-triggered AutoScaling, via a K8s Horizontal Pod Autoscaler**
  - HPA changes number of DL serving replicas behind TorchServe load-balancing endpoint



Right-Sizing Inference Expt: Workload Models

Run workload of serving 2 standard DL models, shown in table below

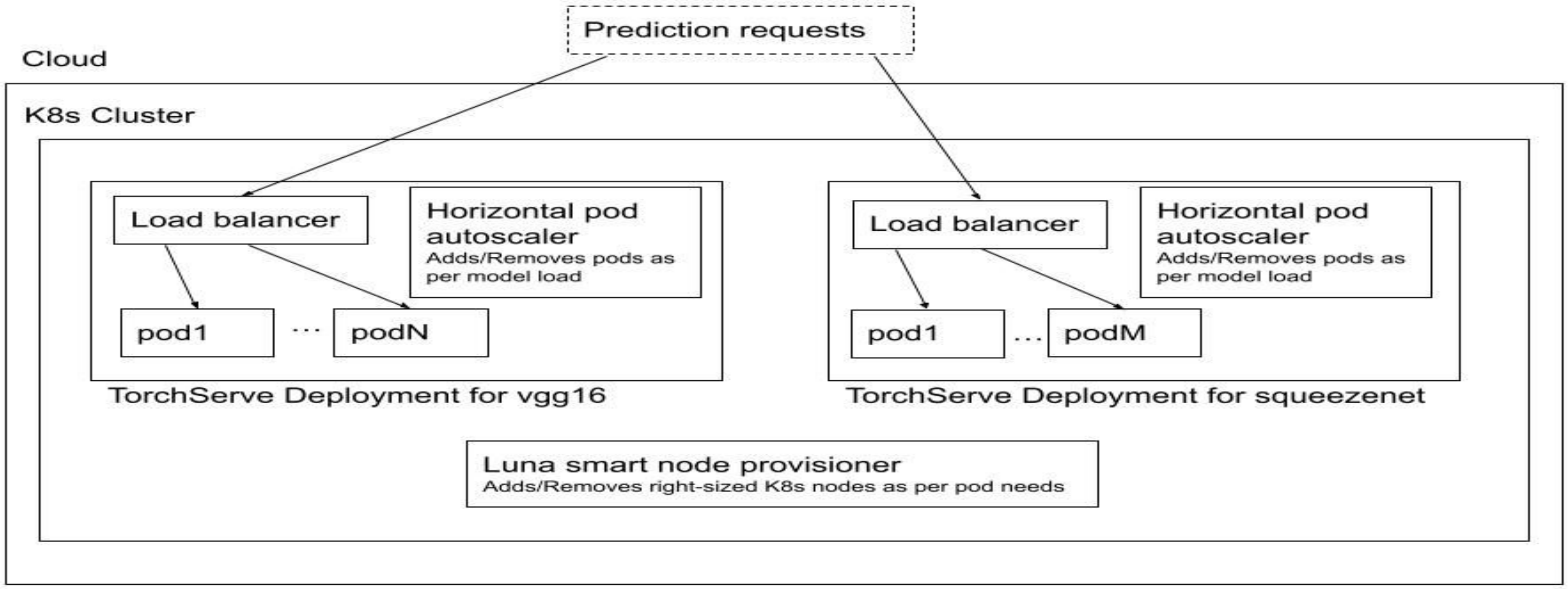
- Note **large difference in number of weights** between models

Model	Task	Weights
vgg16	Image Classification	138M
squeezenet1_1	Image Classification	1.2M

Right-Sizing Inference Expt: Workload Deployment

Run with each model handled by a separate TorchServe deployment to allow:

- Each **TorchServe worker replica size** to be **customized to model size**
- Each **TorchServe worker replica count** to **scale for model load independently**



Right-Sizing Inference Expt: Workload Load & Scaling Range

To generate each model’s peak load

- hey loadgen running 150 parallel threads requesting image classification on kitten_small.jpg

TorchServe is configured for low latency, w/its worker and netty threads set to 1

TorchServe Horizontal Pod Autoscaler

- Set to scale up replica count based on CPU utilization >50%
  - Note: CPU utilization trigger works for both CPU-only & GPU-enabled pods
- Set maxReplicas to maintain 95 percentile E2E latency <=~0.5 seconds at our peak load

```
anne@cloudshell:~/serve (elotl-dev)$ curl -X POST http://34.135.144.47:8080/predictions/vgg_16 -T docs/images/kitten_small.jpg
{
  "tabby": 0.530239462852478,
  "Egyptian_cat": 0.23884634673595428,
  "tiger_cat": 0.1343672275543213,
  "lynx": 0.06994840502738953,
  "Persian_cat": 0.009579462930560112
```

Right-Sizing Inference Expt: Sizing Approaches Compared

We compare 3 approaches listed below, w/resource configurations & operational models

- **Max-sized approach**
  - **Resource Configuration: Static with maximum-count maximum-sized x86+GPU compute nodes**
  - **Operational model: Kubernetes Cluster set up to handle peak inference load for both models**
- **Dynamic fixed-size approach**
  - **Resource Configuration: Dynamic with variable-count maximum-sized x86+GPU compute nodes**
  - **Operational model: K8s Cluster Autoscaler w/x86+GPU node pool defined**
- **Right-sized approach**
  - **Resource Configuration: Dynamic with variable-count right-sized Arm compute nodes**
  - **Operational model: Luna K8s smart node provisioner adding/removing right-sized cost-efficient nodes**

Efficient DL Inferencing: Talk Outline

Efficient DL Inferencing: Resource Mgmt Challenges, Cost Challenges, Complexity Challenges

Efficient DL Inferencing: Talk Overview

Efficient DL Inferencing: Right-Sizing Approach

Experimental Setup for Evaluating Benefits of Right-Sizing Approach vs 2 other Sizing Approaches

Experimental Results for Sizing Approaches on 3 Cloud K8s Clusters

- **OKE,GKE,EKS: Computing Resources Used, Accuracy & Latency Validation, Cost Comparison**
- **Operational Complexity for Sizing Approaches**

Conclusion: Summary and One More Thing

OKE Right-Sizing Inference Expt: Computing Resources

K8s has 2 statically-allocated Ampere Arm CPU nodes for cluster mgmt, including Luna

- Each is VM.Standard.A1.Flex shape w/2 OCPUs and 32GB [each: $0.0680/hr]

TorchServe

- Each model's TorchServe deployment set to create minimum of 1 worker pod
- Note: Squeezenet1_1 worker pod memory size needed increase when GPU requested

Configuration	Vgg16 worker pod size	Vgg16 node instance type	Vgg16 max worker pod count	Squeezenet1_1 worker pod size	Squeezenet1_1 node instance type	Squeezenet1_1 max worker pod count
Max-sized	400m, 4GB, 1GPU	VM.GPU2.1 (P100 w/16GB)	3	400m, 2GB, 1GPU	VM.GPU2.1	3
Dynamic fixed-size	400m, 4GB, 1GPU	VM.GPU2.1 [$1.275/hr]	3	400m, 2GB, 1GPU	VM.GPU2.1	3
Right-sized	400m, 4GB	VM.Standard.A1. Flex w/1 OCPU, 4GB [$0.0160/hr]	4	400m; 1GB	VM.Standard.A1. Flex w/1 OCPU, 1GB [$0.0115/hr]	4

OKE Right-Sizing Inference Expt: Accuracy & Latency Validation

Runs on right-sized and x86+GPU nodes produced same prediction results

Table shows 95 percentile E2E latency for models simultaneously handling peak load

- Run on right-sized and on x86+GPU configurations
- **Both models meet desired E2E latency target (<=~0.5 secs) for presented workload & set up**

Model	**Right-sized 95% latency seconds**	**x86+GPU 95% latency seconds**
vgg16	0.2243	0.2453
squeezenet1_1	0.2176	0.2975

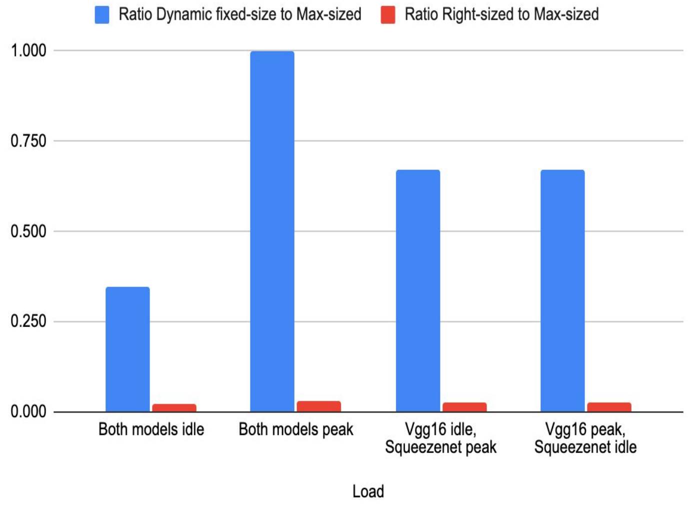
OKE Right-Sizing Inference Expt: Cost Compared at 4 op points

Table presents **costs per hour** for three approaches at **four operating points**

Figure shows ratios of Dynamic fixed-size and Right-sized costs to Max-sized costs

- When both models not at peak, Dynamic fixed-size configuration cost < Max-sized configuration cost
- **In all cases, Right-sized configuration << Max-sized configuration cost, by > an order magnitude**
- **Note: legacy VM.GPU2.1(P100)=$1.275/hr; current generation VM.GPU3.1(V100)=$2.950/hr**

Load	Max-sized $/hr	Dynamic fixed-sized $/hr	Right-sized $/hr
Both models idle	7.7860	2.6860	0.1635
Both models peak	7.7860	7.7860	0.2460
Vgg16 idle, squeezenet peak	7.7860	5.2360	0.1980
Vgg16 peak, squeezenet idle	7.7860	5.2360	0.2115



GKE Right-Sizing Inference Expt: Computing Resources

K8s has 2 statically-allocated Ampere Arm CPU nodes for cluster mgmt, including Luna

- Each is t2a-standard-2 shape w/2 VCPUs and 8GB [each: $0.077/hr]

TorchServe

- Each model's TorchServe deployment set to create minimum of 1 worker pod
- Note: Squeezenet1_1 worker pod memory size needed increase when GPU requested

Configuration	Vgg16 worker pod size	Vgg16 node instance type	Vgg16 max worker pod count	Squeezenet1_1 worker pod size	Squeezenet1_1 node instance type	Squeezenet1_1 max worker pod count
Max-sized	400m, 4GB, 1GPU	n1-standard-2 + T4 GPU (16GB)	7	400m, 3GB, 1GPU	n1-standard-2 + T4 GPU	8
Dynamic fixed-size	400m, 4GB, 1GPU	n1-standard-2 + T4 GPU [$0.445/hr]	7	400m, 3GB, 1GPU	n1-standard-2 + T4 GPU	8
Right-sized	400m, 4GB	T2a-standard-2 [$0.077/hr]	4	400m; 1GB	T2a-standard-1 [$0.0385/hr]	4

GKE Right-Sizing Inference Expt: Accuracy & Latency Validation

Runs on right-sized and x86+GPU nodes produced same prediction results

Table shows 95 percentile E2E latency for models simultaneously handling peak load

- Run on right-sized and on x86+GPU configurations
- **Both models meet desired E2E latency target (<=~0.5 secs) for presented workload & set up**

Model	Right-sized 95% latency seconds	x86+GPU 95% latency seconds
vgg16	0.3011	0.3142
squeezenet1_1	0.2996	0.3335

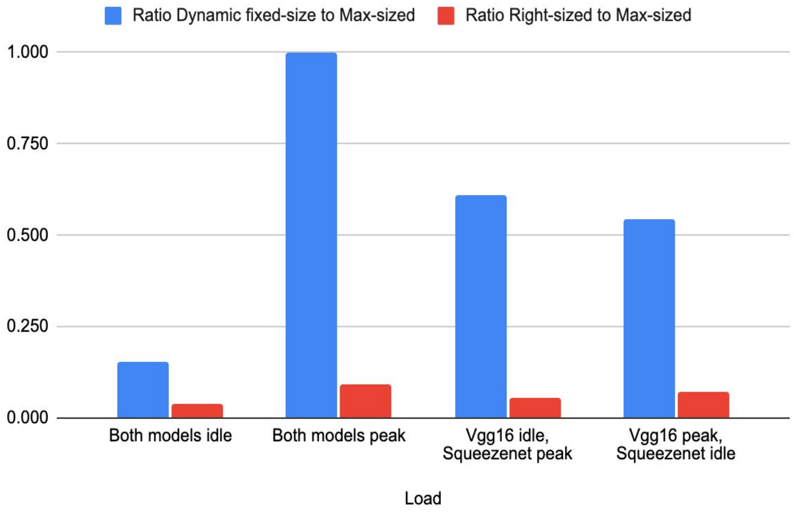
GKE Right-Sizing Inference Expt: Cost Compared at 4 op points

Table presents **costs per hour** for three approaches at **four operating points**

Figure shows ratios of Dynamic fixed-size and Right-sized costs to Max-sized costs

- When both models not at peak, Dynamic fixed-size configuration cost < Max-sized configuration cost
- **In all cases, Right-sized configuration << Max-sized configuration cost, by > an order magnitude**

Load	Max-sized $/hr	Dynamic fixed-sized $/hr	Right-sized $/hr
Both models idle	6.8290	1.0440	0.2695
Both models peak	6.8290	6.8290	0.6160
Vgg16 idle, squeezenet peak	6.8290	4.1590	0.3850
Vgg16 peak, squeezenet idle	6.8290	3.7140	0.5005



EKS Right-Sizing Inference Expt: Computing Resources

K8s has 2 statically-allocated Graviton2 Arm CPU nodes for cluster mgmt, including Luna

- Each is m6g.large shape w/2 VCPUs and 8GB [each: $0.077/hr]

TorchServe

- Each model's TorchServe deployment set to create minimum of 1 worker pod
- Note: Squeezenet1_1 worker pod memory size needed increase when GPU requested

Configuration	**Vgg16 worker pod size**	**Vgg16 node instance type**	**Vgg16 max worker pod count**	**Squeezenet1_1 worker pod size**	**Squeezenet1_1 node instance type**	**Squeezenet1_1 max worker pod count**
Max-sized	400m, 4GB, 1GPU	g4dn.xlarge (T4 w/16GB)	4	400m, 3GB, 1GPU	g4dn.xlarge	4
Dynamic fixed-size	400m, 4GB, 1GPU	g4dn.xlarge [$0.526/hr]	4	400m, 3GB, 1GPU	g4dn.xlarge	4
Right-sized	400m, 4GB	r6g.medium [$0.0504/hr]	4	400m; 1GB	t4g.small [$0.0168/hr]	4

EKS Right-Sizing Inference Expt: Accuracy & Latency Validation

Runs on right-sized and x86+GPU nodes produced same prediction results

Table shows 95 percentile E2E latency for models simultaneously handling peak load

- Run on right-sized and on x86+GPU configurations
- **Both models meet desired E2E latency target (<=~0.5 secs) for presented workload & set up**

Model	Right-sized 95% latency seconds	x86+GPU 95% latency seconds
vgg16	0.4305	0.4452
squeezenet1_1	0.4436	0.4522

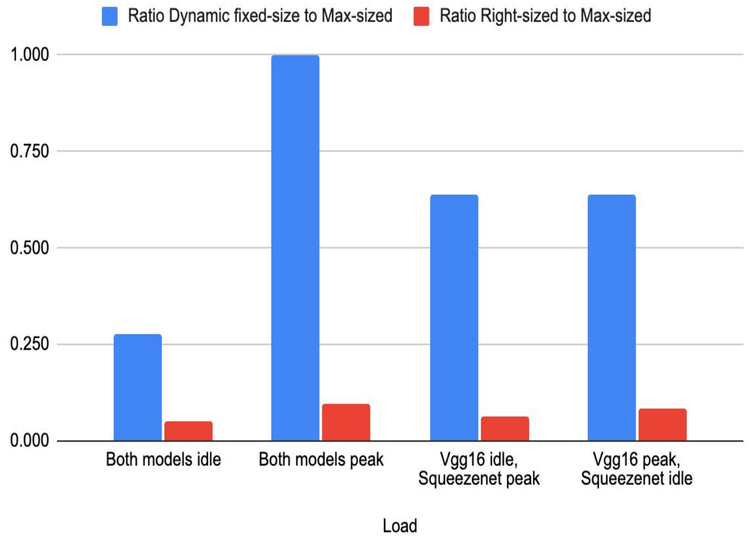
EKS Right-Sizing Inference Expt: Cost Compared at 4 op points

Table presents **costs per hour** for three approaches at **four operating points**

Figure shows ratios of Dynamic fixed-size and Right-sized costs to Max-sized costs

- When both models not at peak, Dynamic fixed-size configuration cost < Max-sized configuration cost
- **In all cases, Right-sized configuration << Max-sized configuration cost, by > an order magnitude**

Load	Max-sized $/hr	Dynamic fixed-sized $/hr	Right-sized $/hr
Both models idle	4.3620	1.2060	0.2212
Both models peak	4.3620	4.3620	0.4228
Vgg16 idle, squeezenet peak	4.3620	2.7840	0.2716
Vgg16 peak, squeezenet idle	4.3620	2.7840	0.3724



Right-Sizing Inference Expts: Operational Complexity Compared

Operational complexity analysis; entries marked “no” involve manual updates for changes

- Note: OCI discontinuing legacy VM.GPU2.1 is example of offerings update
- **Right-sized configuration handled all three kinds of system changes automatically**

Configuration	**Automatically adapts to TorchServe max worker count needed changes**	**Automatically adapts to TorchServe worker size changes**	**Automatically handles cloud instance availability & offerings changes**
Max-sized	no	no	no
Dynamic fixed-size	yes	no	no
Right-sized	yes	yes	yes

Efficient DL Inferencing: Talk Outline

Efficient DL Inferencing: Resource Mgmt Challenges, Cost Challenges, Complexity Challenges

Efficient DL Inferencing: Talk Overview

Efficient DL Inferencing: Right-Sizing Approach

Experimental Setup for Evaluating Benefits of Right-Sizing Approach vs 2 other Sizing Approaches

Experimental Results for Sizing Approaches on 3 Cloud K8s Clusters

Conclusion: Summary and One More Thing

Right-Sizing Inference: Summary

Have shown that Right-sizing

- **Inference resources via using Elotl Luna**
- **Inference compute shapes by choosing Arm CPU-only compute**

Reduces cloud resource costs significantly, measured at 4 operating pts for 3 vendors

- **While reducing operational complexity for changes in inferencing resources & cloud offerings**

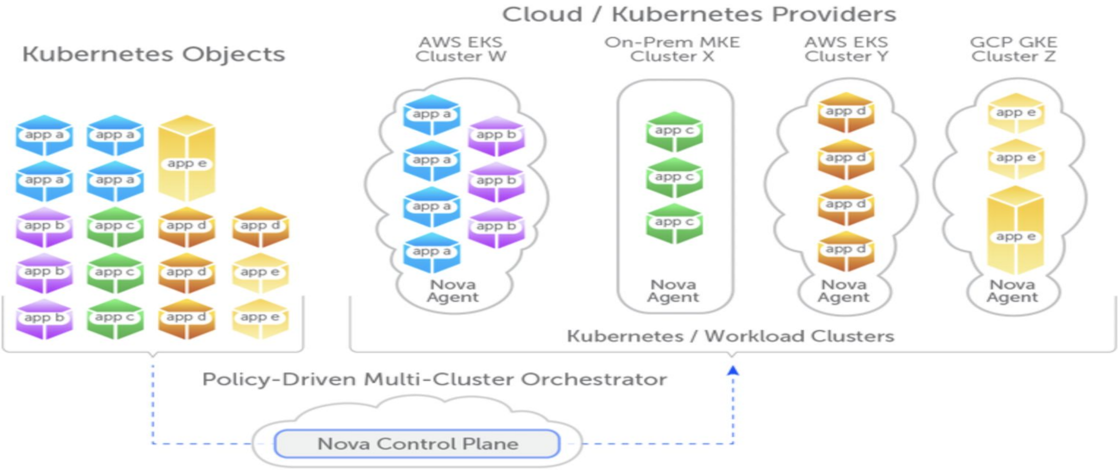
Right-Sizing Inference: One More Thing

K8s cluster that hosts DL serving workloads may be one of a K8s cluster set

- E.g., there may be separate K8s cluster w/GPU nodes that hosts DL training workloads

Elotl Nova, K8s cluster scheduler, places K8s workloads onto policy-selected cluster

- **Luna and Nova can be used together to simplify K8s resource management**



Please Give Right-Sizing a Try on Your Workloads!

Here are resources to get you started:

- Elotl Luna
- Ampere Computing
- OCI Ampere A1 Compute, OCI OKE
- GCP Ampere A1 Compute, GCP GKE
- EKS Graviton2 Compute, AWS EKS
- OCI Blog: Deep Learning inferencing at scale with Oracle Cloud A1 Compute with Elotl Luna

Thanks!