KubeCon | CloudNativeCon
North America 2024

Optimizing Load Balancing and Autoscaling for LLM Inference on Kubernetes

David Gray
Software Engineer – Red Hat Performance and Scale

Personal Introduction

Hi, I'm David!

- Based out of Toronto, ON, Canada
- Past: k8s operator development for out-of-tree kernel driver enablement and node tuning
- Present: LLM Inference performance on k8s

Part of the Performance and Scale for AI Platforms (PSAP) team at Red Hat

- Making AI applications run better and faster on Linux, Containers, and Kubernetes

Explore the Red Hat Performance and Scale Blogs

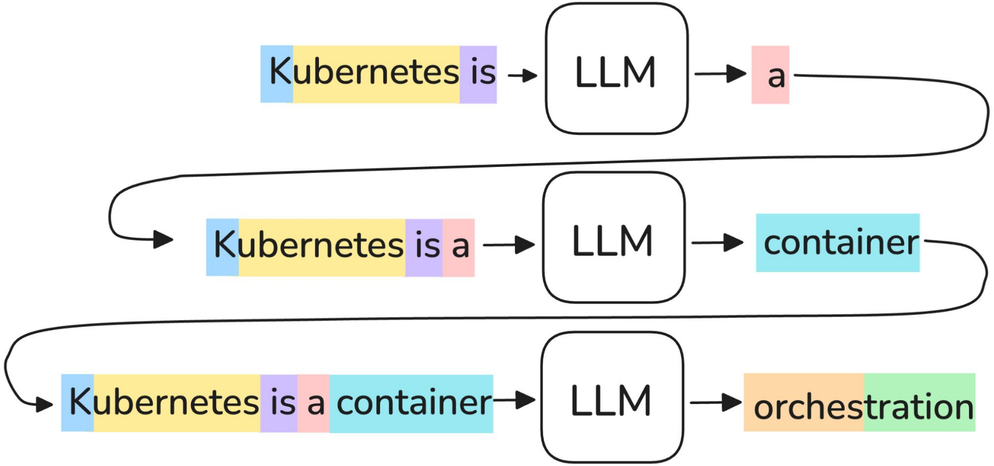
Agenda

What we'll discuss today

- Background
- Deploying LLMs on Kubernetes: vanilla and KServe
- Load-balancing for LLM Inference
- Pod autoscaling for LLM Inference
- Conclusion and future ideas
- Q&A

Background

Background and motivation



Why optimize inference performance?

- Cost-efficiency
- Energy efficiency
- Trend towards inference with long sequences:
  - RAG
  - Chain-of-thought

Model servers (inference engines / runtimes)

Examples:

- vLLM
- text-generation-inference 🤗
- TensorRT-LLM
- SGLang

Software for loading and running models

- HTTP / gRPC server
- Optimized accelerator kernels
- Batch processing
- Performance optimizations

Measuring LLM inference performance

Latency

- Time-to-first-token (TTFT)
- Inter-token latency (ITL)
- Response time: depends on the number of tokens

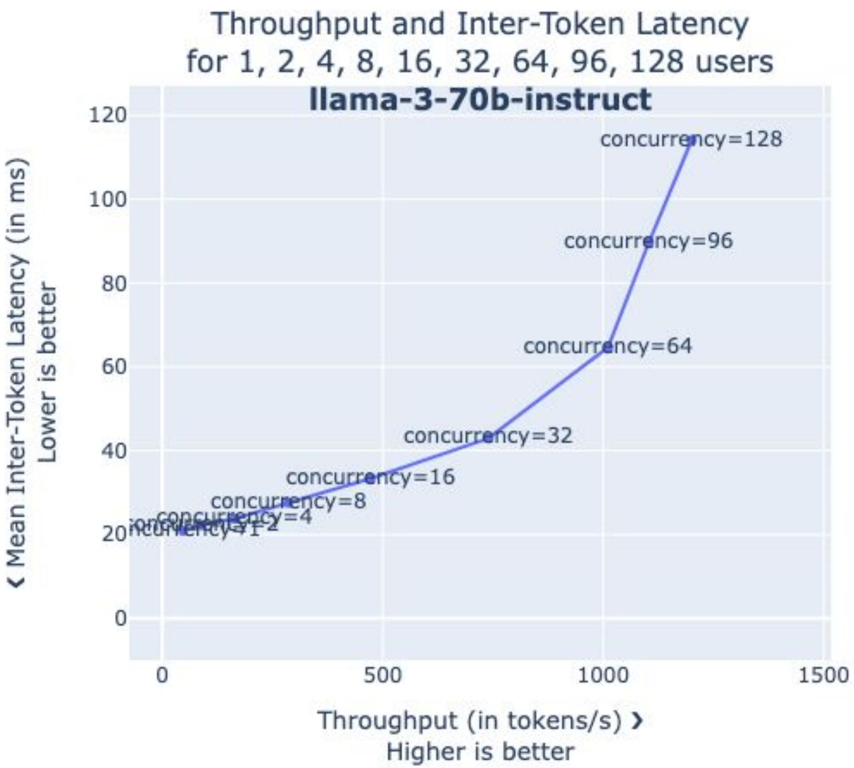
Throughput

- Tokens / second
- Across all concurrent requests

Benchmarking tools

- [llm-load-test]
- [fmperf]

Example data:

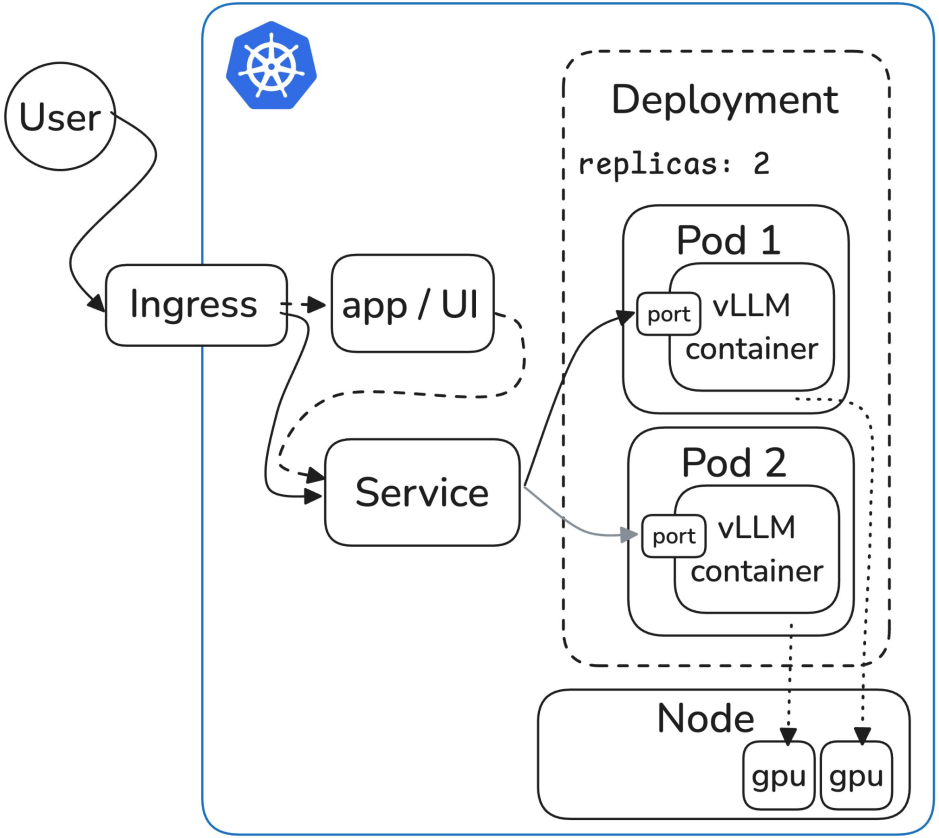


LLM Inference on Kubernetes

LLM Inference on Kubernetes

Deploying an LLM model on k8s the "vanilla way"

- Deployment + Service
- Part of another application, or behind a preprocessing app.



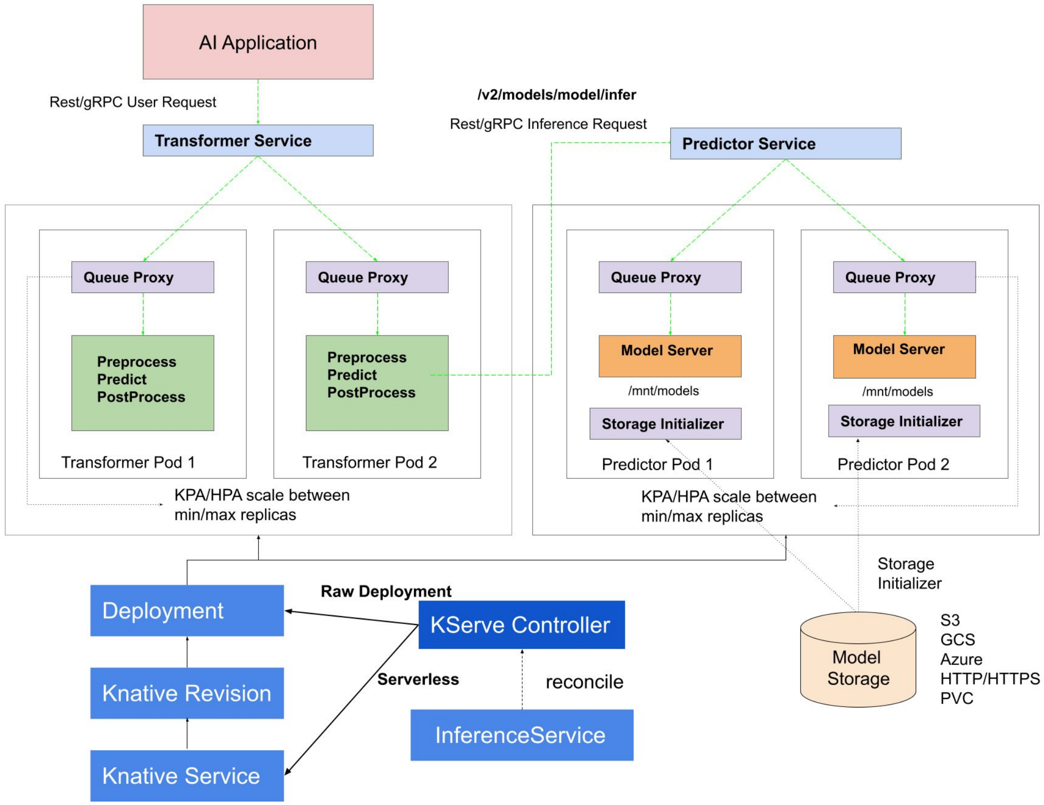
—

[More on load balancing with ClusterIP Service]

KServe Introduction

- CRDs
  - ServingRuntime
  - InferenceService
- Deployment modes
  - RawDeployment
  - Knative

- Knative load balancing
  - Least-requests based
- Knative Pod Autoscaler (KPA)



—

[KServe control plane documentation]
For more info, see [Yuan Tang's talk on KServe features]

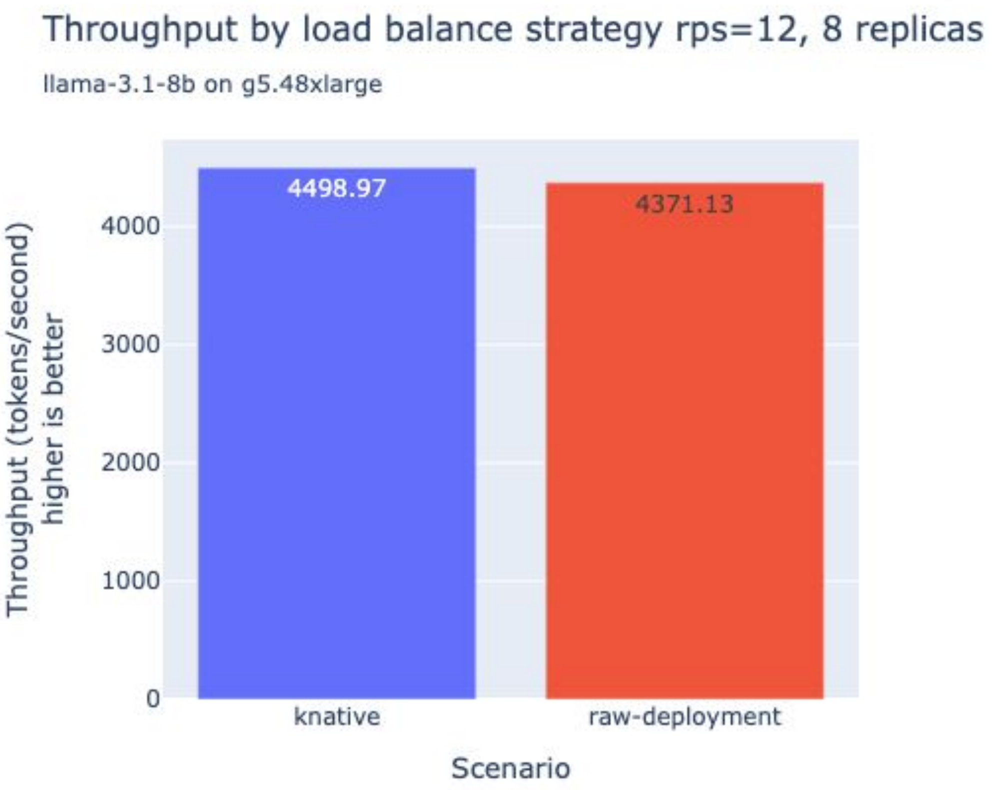
Load balancing for LLM inference

Load balancing: Test methodology

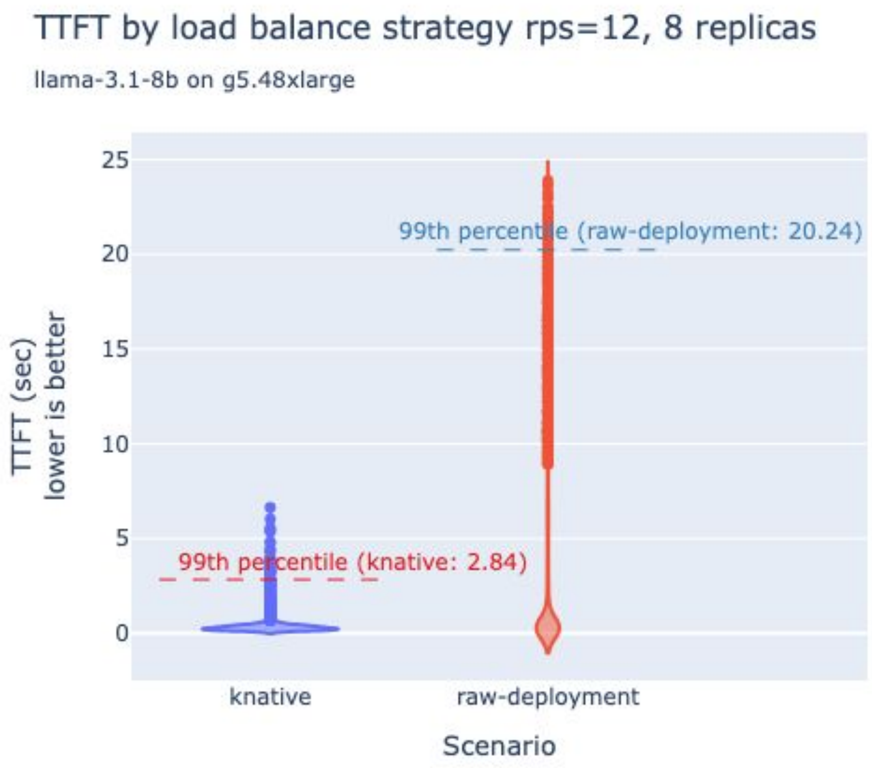
Model: Llama-3.1-8b
Hardware: AWS g5.48xlarge
8 replicas, each on 1xA10G GPU (24GB)

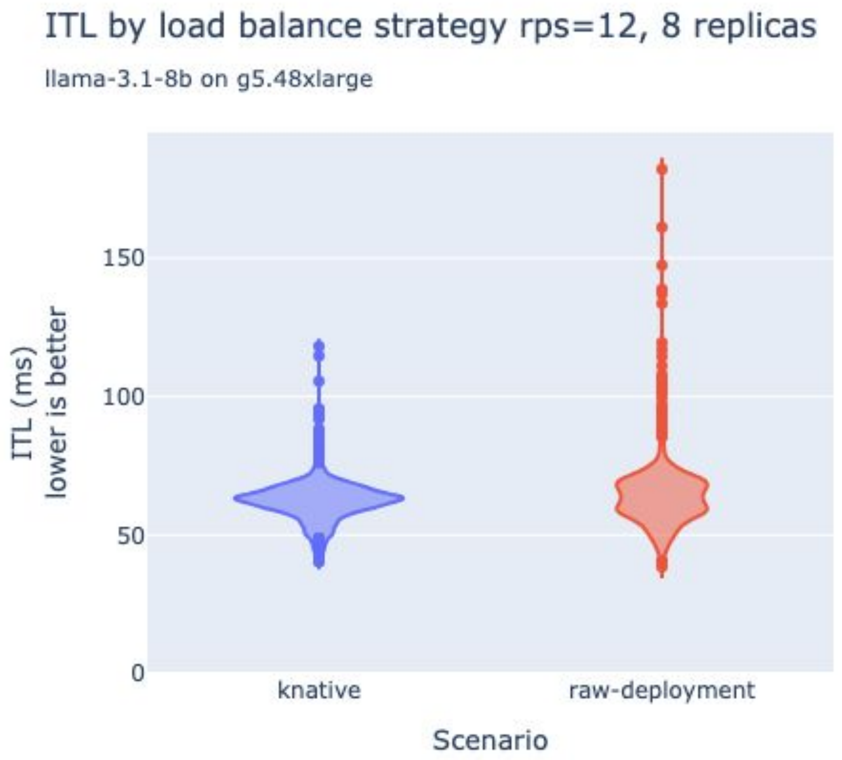
llm-load-test parameters

- Dataset: subset of OpenOrca
- Input tokens:
  - 16 - 1600 tokens
  - mean: ~540 tokens
- Output tokens:
  - 16 - 1600 tokens
  - mean: ~415
- RPS=12



Load balancing: Knative vs RawDeployment

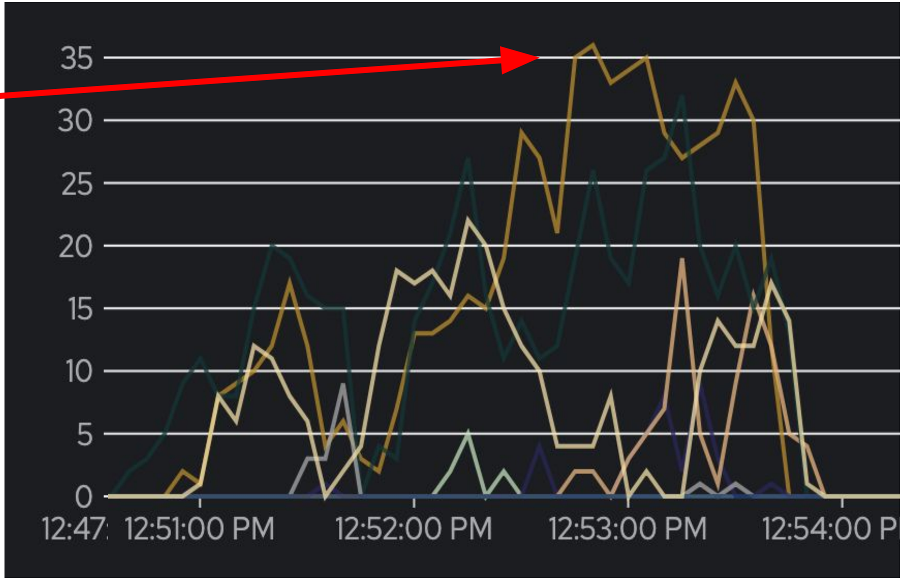




Load balancing: Knative vs RawDeployment

Large request queues on some replicas

- Max queue size (num_requests_waiting):
  - RawDeployment: **36**
  - Knative: **5**



vLLM metric: num_requests_waiting (raw-deployment)

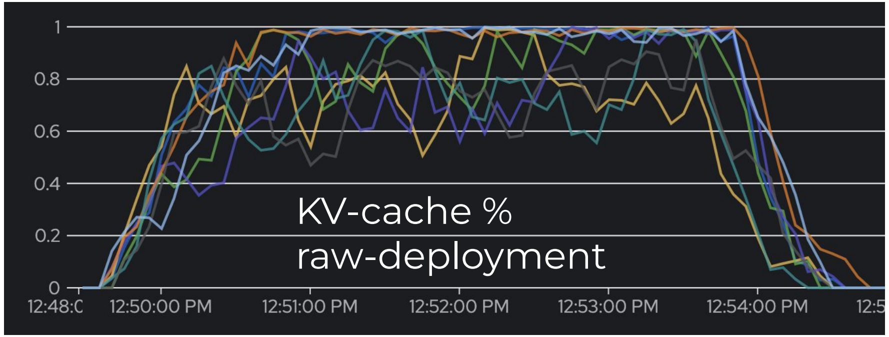
Custom load balancing

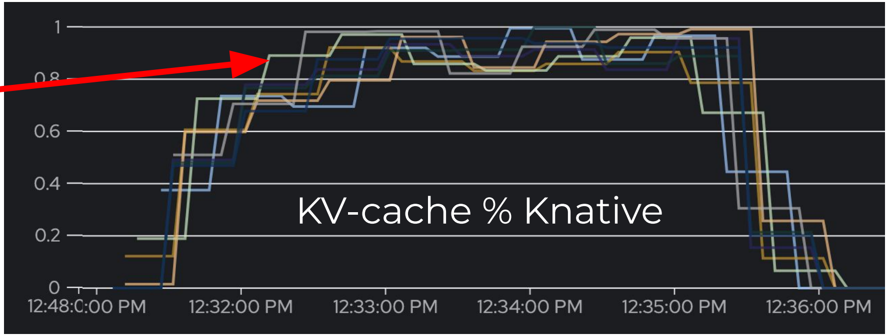
Issues with requests-based load balancing

- Maximum batch size is dynamic
- Batch size is limited by KV-cache
- KV-cache usage depends on:
  - Number of requests
  - Sequence lengths of requests
  - Model size

Some replicas near 100% KV-cache utilization while others < 80%

vLLM metric: GPU KV-cache usage %:





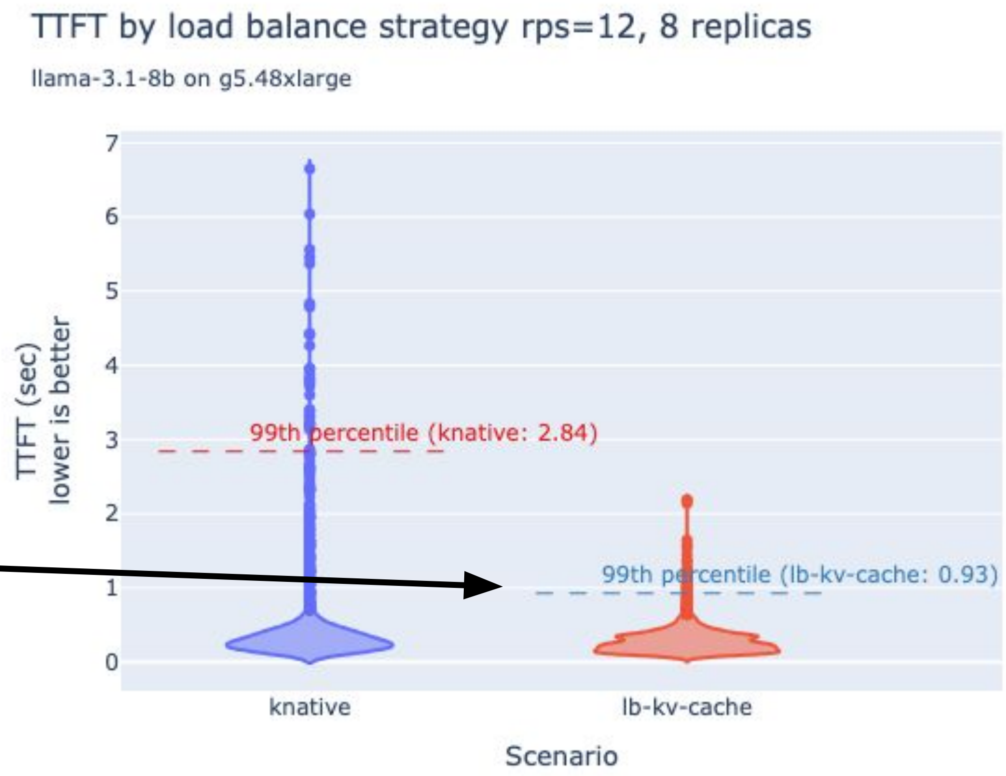
Custom load balancing results

Custom strategy

- Scrape KV-cache usage %
- Pick 2 replicas randomly
- Send to lower of the 2

Implemented client-side (PoC)

99%ile TTFT under 1 second!



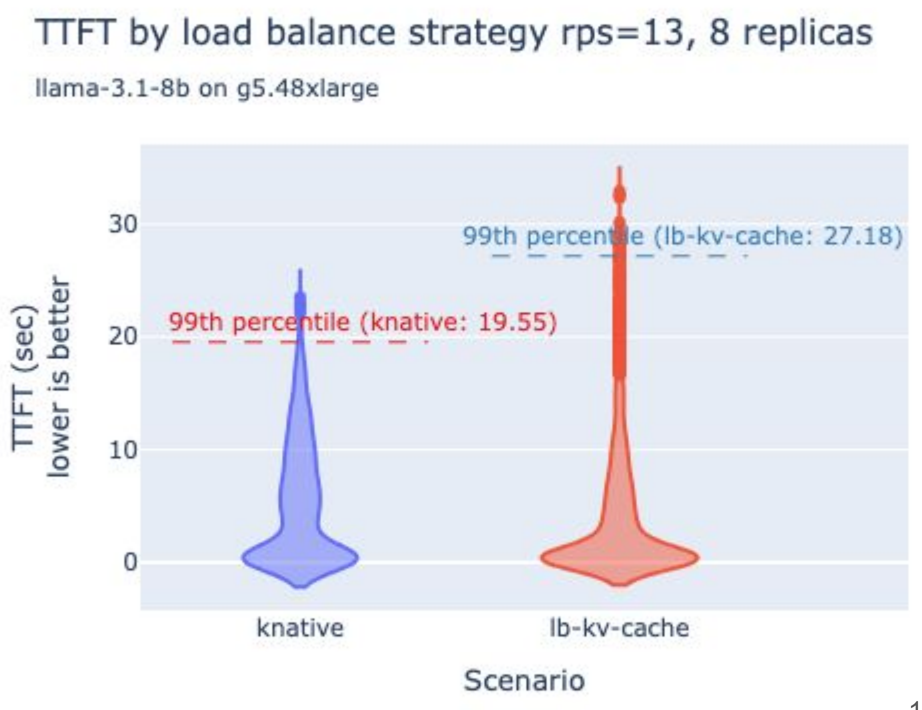
Custom load balancing: higher load

At RPS=13, Knative beats the custom strategy

The problem under high load

- Maximum num_requests_waiting:
  - Knative: **26**
  - Custom: **36**
- KV-cache usage >98% for all replicas

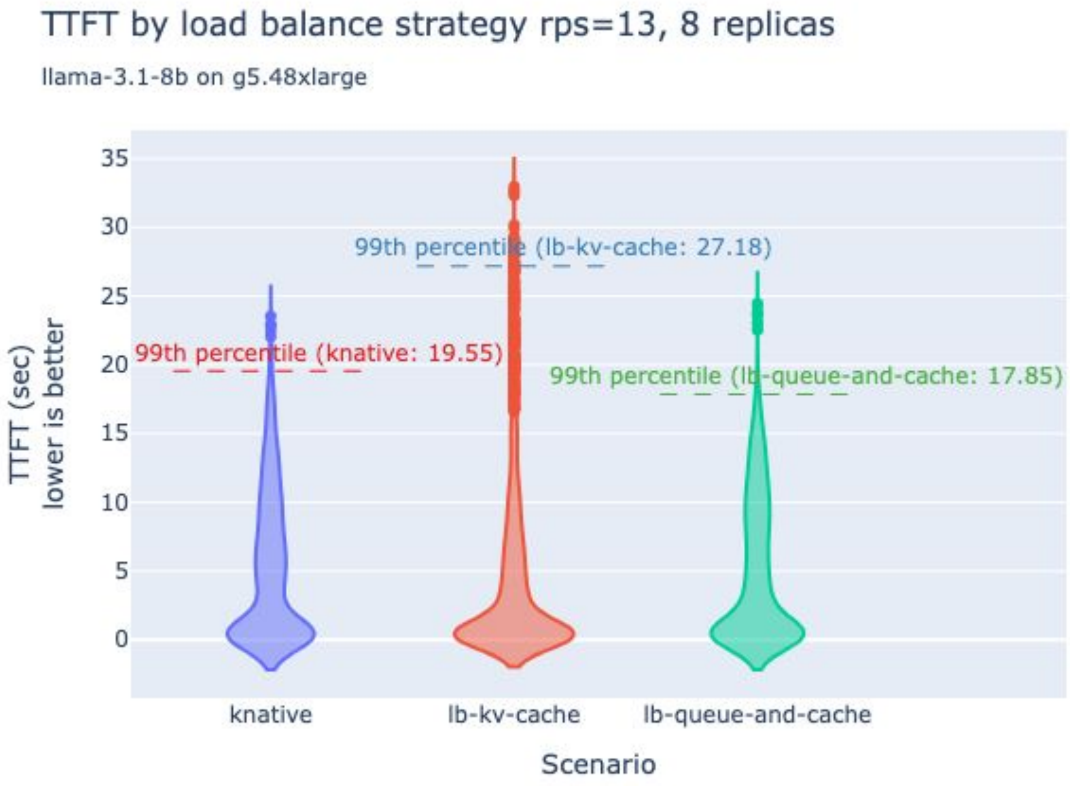
A better solution should prioritize queue size before KV-cache usage %



Custom load balancing: better strategy

Better strategy

- Scrape num_requests_waiting and KV-cache usage %
- Pick 2 replicas randomly
- Send to the one with lower num_requests_waiting
  - if tied, send to lower KV-cache usage %



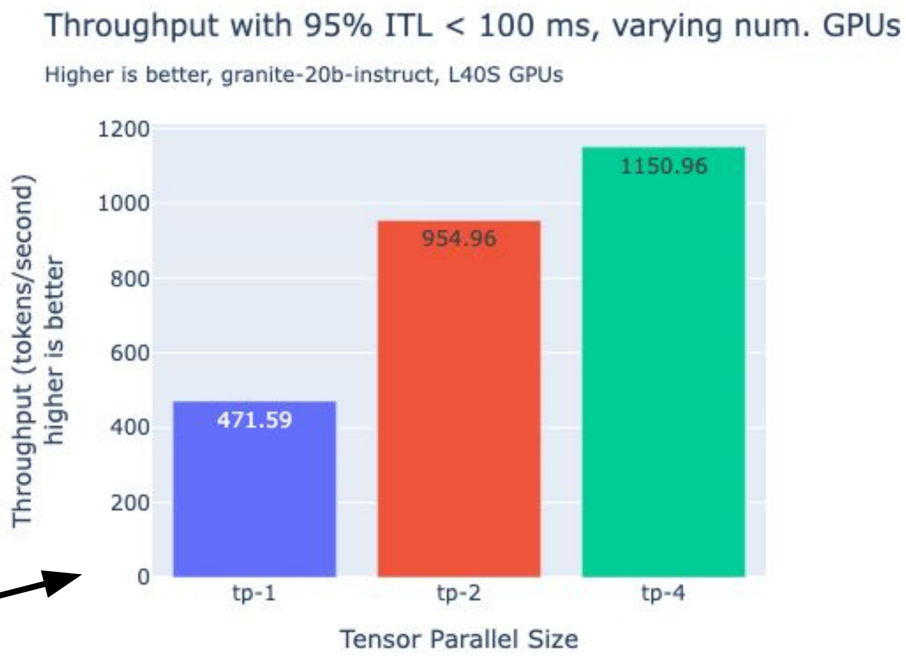
An aside: Replicas vs tensor parallelism

Options to get more throughput

- More replicas
- tensor-parallelism (TP)
- pipeline-parallelism

When to use tensor-parallelism

- Model doesn’t fit in GPU memory
- To improve per-token latency*
- Need more GPU memory for KV-cache
  - To enable higher batch sizes or longer sequences
  - Example: 20B model in 48GB GPU



* TP performance depends on GPU interconnect

Autoscaling for LLM Inference

Pod autoscaling for LLM inference

How to know when to scale up

- KServe / Knative:
  - Concurrency
  - RPS
  - CPU/Memory

- Inference-engine metrics:
  - KV-cache utilization %
  - Queue depth (num_requests_waiting)
  - ITL, TTFT

KServe feature requests for KEDA integration: #3561, #4007

```yaml
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    # Soft limit to trigger scaleup
    autoscaling.knative.dev/target: "30"
    ...
  name: llama-3-1-8b-isvc
  namespace: models
spec:
  predictor:
    # Hard limit enforced by queue-proxy
    containerConcurrency: 50
    ...
```

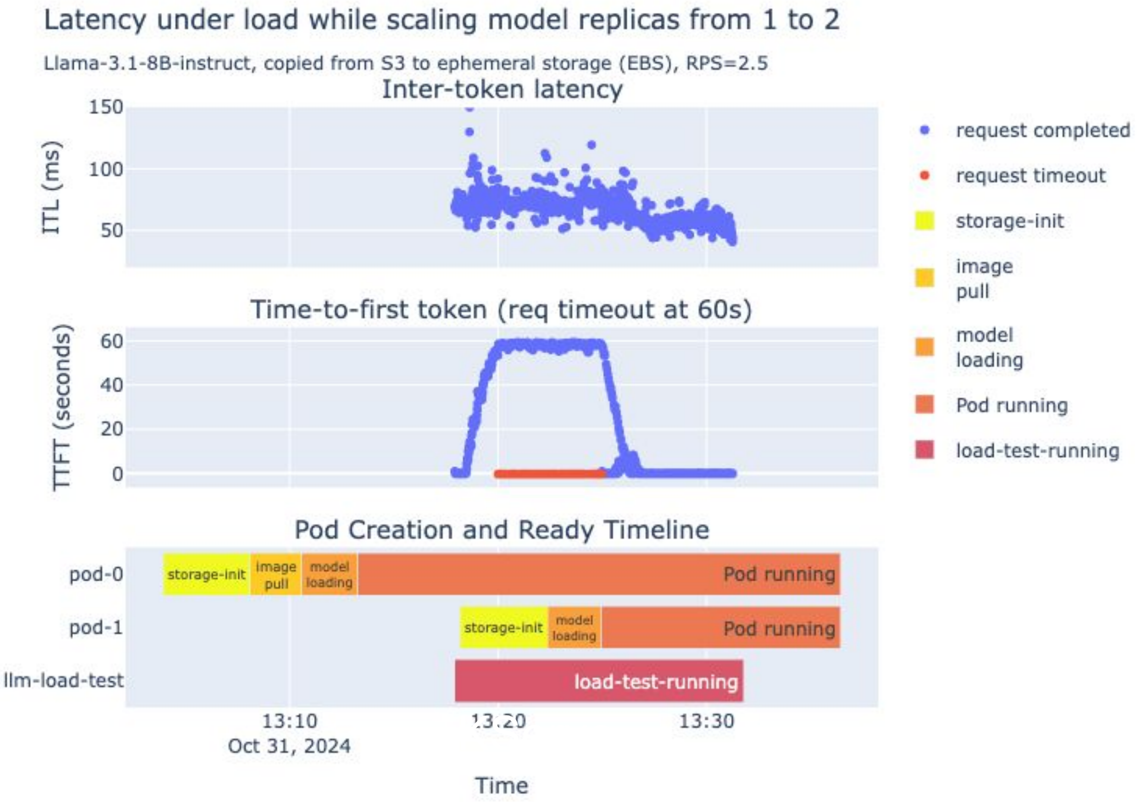
Pod autoscaling

Delays before model is ready

- Pull runtime image
- Download model files
- Load model from storage into GPU memory
- Graph warm-up

Solutions

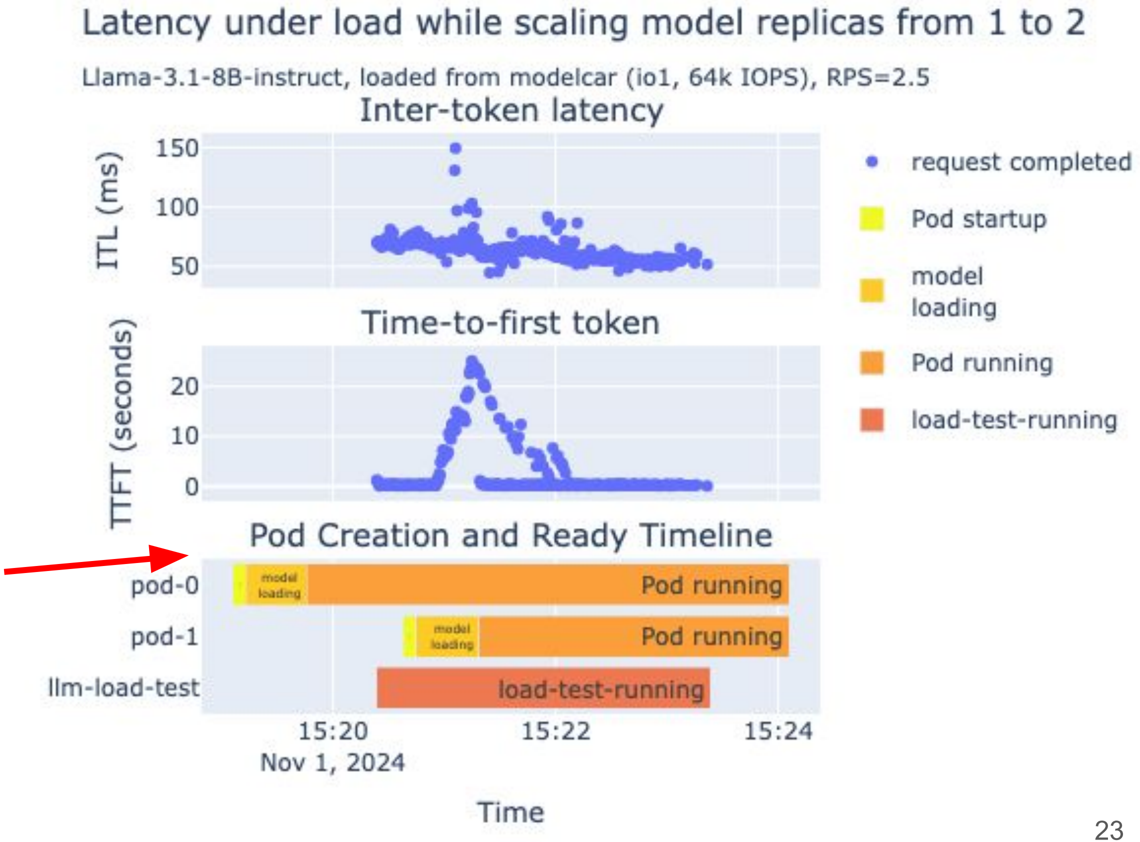
- Cache models locally
- Use faster local storage
- Store image + model files somewhere with fast upload speeds



Autoscaling optimization: modelcars

KServe Modelcars

- Load model files from containers
- Leverages k8s image caching
  - Simplifies orchestration compared to local storage
- Issues with large models (>20GB)
- Storage speed (r/w) matters
  - Ex: gp3 to io1 EBS volume



Conclusion

Summary and future ideas

Summary

- Requests-based load balancing is well suited for LLM inference.
  - Custom load balancing can further improve balancing, reducing 99% TTFT
- Autoscaling performance is dependent on fast storage
- Modelcars simplifies orchestration for caching model files locally

Future ideas

- Autoscaling + load-balancing experiments with other metrics
- KServe support for autoscaling on custom metrics (KServe issues #3561, #4007)
- fastsafetensors to load models with GPU Direct Storage
- More enhancements to load testing tools: join WG-Serving for more on this!

KubeCon | CloudNativeCon

North America 2024

Thank you!

Don’t hesitate to reach out

dagray@redhat.com

dagrayvid on GitHub

David Whyte-Gray on LinkedIn