Descarregamento de cache KV com NetApp

NetApp artificial intelligence solutions

NetApp
August 20, 2026

Índice

Descarregamento de cache KV com NetApp

A primeira onda de investimentos em IA empresarial focou no treinamento e ajuste fino de modelos para desenvolver capacidades, não em sua implementação em larga escala. À medida que as organizações passam da experimentação para a produção, o centro de gravidade se desloca para a inferência em tempo real: assistentes de busca, chatbots, copilotos e fluxos de trabalho de agentes com os quais os usuários interagem continuamente. Nesses ambientes, a utilização da GPU aumenta rapidamente não porque cada requisição execute um treinamento completo, mas porque cada pergunta subsequente, cada nova interação em uma conversa e cada usuário simultâneo adiciona carga à pilha de inferência.

Introdução

Um padrão logo se torna visível: a GPU está realizando uma quantidade surpreendente de trabalho repetido, recalculando a atenção sobre tokens que já processou. Essa repetição não é apenas um problema de ajuste; é um problema de arquitetura de memória. A resposta da indústria é uma estratégia de memória em camadas construída em torno do cache KV.

O que é o KV Cache?

Em termos simples, o armazenamento em cache KV (key-value) é uma técnica usada para otimizar a inferência de grandes modelos de linguagem (LLM) ao armazenar valores calculados anteriormente em um cache KV, para que esses valores não precisem ser calculados novamente para cada novo token gerado, o que seria necessário de outra forma.

Nota: para uma visão geral técnica mais aprofundada, consulte "Visão geral do armazenamento em cache KV do Hugging Face".

O que é o KV Cache Offloading?

A maioria dos mecanismos de inferência armazena o cache KV na memória da GPU por padrão. Isso funciona bem até que a demanda aumente. O tamanho do cache KV escala linearmente com o tamanho do lote (o número de prompts processados ao mesmo tempo) e o comprimento da sequência (o comprimento de cada conversa ou fluxo de geração). À medida que as janelas de contexto se expandem, os chats com várias interações se tornam comuns e mais usuários e agentes consomem serviços de inferência, os requisitos do cache KV podem rapidamente exceder a memória disponível da GPU. Quando isso acontece, entradas úteis do cache são frequentemente removidas e o mecanismo precisa recalcular a atenção para os tokens que já processou.

O descarregamento do cache KV resolve essa limitação movendo o cache KV de um prompt processado anteriormente para um destino externo fora da memória da GPU ou acelerador, como a RAM do sistema, o disco local ou o storage compartilhado. As entradas descarregadas podem ser retidas enquanto houver capacidade disponível e referenciadas novamente quando um prefixo semelhante ou uma solicitação subsequente chegar, em vez de serem descartadas quando a memória da GPU estiver cheia. Na prática, esses destinos de descarregamento funcionam como um espaço de troca em camadas para a memória da GPU: mais lento que a VRAM, mas maior e mais durável, ampliando a quantidade de contexto conversacional e carga de trabalho simultânea que o sistema pode suportar sem a necessidade de preenchimento prévio repetido.

Importância de um storage compartilhado em camadas

O descarregamento do cache KV já ajuda quando um único mecanismo de inferência ultrapassa a memória da GPU. Mover blocos de cache para a RAM da CPU aumenta a capacidade em um servidor. Isso é útil, mas resolve apenas parte do problema em produção. Plataformas de inferência reais raramente são executadas como uma única instância de mecanismo de serviço. Elas são executadas por trás de balanceadores de carga, escalam horizontalmente e atendem a muitos usuários simultâneos e agentes. Nesse cenário, o storage compartilhado é o que transforma o cache KV de uma otimização local em uma capacidade da plataforma.

- **O armazenamento compartilhado permite a reutilização entre instâncias de serviço** Um dos principais benefícios do storage compartilhado é a capacidade de acessar os mesmos arquivos ou objetos em várias instâncias e nós. Isso é especialmente verdadeiro no caso de uma implantação escalável de duas ou mais instâncias do mecanismo de serviço por trás de um balanceador de carga, cada uma configurada para descarregar blocos de cache KV para o mesmo bucket compartilhado ou volume NAS. Por exemplo, quando uma instância processa um prefixo de prompt e descarrega os blocos de cache resultantes, outra instância pode carregar esses blocos em acerto do cache, em vez de recalcular o mesmo trabalho de preenchimento. Isso é importante porque o tráfego de produção é distribuído; a primeira pergunta de um usuário pode atingir a instância A; uma pergunta subsequente ou de outro usuário com um prompt de sistema semelhante pode atingir a instância B. Sem storage compartilhado, cada instância mantém seu próprio ambiente de cache privado.
- **A memória da CPU sozinha não é suficiente para implantações com múltiplas instâncias.** A camada de CPU é rápida, mas é local para cada processo do mecanismo de serviço. Uma instância não pode acessar entradas de acerto do cache KV que outra instância descarregou para sua RAM de CPU local, a menos que você implemente um canal ponto a ponto, o que introduz latência adicional e complexidade operacional. O storage compartilhado remove essa barreira. A RAM da CPU continua sendo valiosa como uma camada de preparação rápida em cada nó, mas o S3 ou NAS compartilhado fornece o plano de memória comum que vários mecanismos podem ler e gravar. Você não está mais escalando GPUs isoladamente, você está escalando com infraestrutura de cache compartilhada.
- **Adotar uma estratégia de design de três níveis** + Uma arquitetura preferencial combina:

  - **Memória da GPU** — cache ativo para solicitações em andamento.
  - **Memória da CPU** — preparação rápida à medida que os prompts entram na fila.
  - **Storage compartilhado** — base de storage durável, ampla e compartilhável.

  Com essa arquitetura, os blocos de cache KV podem ser transferidos da camada compartilhada para a memória da CPU à medida que novas solicitações chegam, mantendo a GPU focada no trabalho ativo e, ao mesmo tempo, preservando o cache durável no storage.

- **Economia de inferência, não apenas velocidade** Do ponto de vista da produção, a importância do storage compartilhado não se resume à latência. É o controle sobre memória, concorrência e custo. Mecanismos de serviço como o vLLM gerenciam o cache KV de forma eficiente dentro de um único nó. Estruturas de descarregamento de cache KV, como o LMCache, transformam o cache KV em um recurso portátil e compartilhável que pode ser movido entre a memória da CPU e o storage. Plataformas de storage compartilhado como NetApp ONTAP e StorageGRID fornecem a camada durável que torna esse compartilhamento viável entre várias instâncias. Com o LMCache conectado ao storage compartilhado, várias instâncias do vLLM podem acessar as mesmas entradas de cache KV descarregadas, melhorando a taxa de transferência e reduzindo a computação redundante à medida que a concorrência aumenta. Isso reduz o desperdício de ciclos de GPU em preenchimentos repetidos, o que é diretamente relevante para chatbots, assistentes de busca, agentes e qualquer carga de trabalho com threads de múltiplas interações, prompts de sistema compartilhados ou padrões de contexto recorrentes.
- **Aumente o desempenho da inferência e o investimento em GPUs** + Este benchmark compara o

desempenho da inferência à medida que o comprimento da conversa e o número de usuários simultâneos aumentam. Quando o cache KV é mantido apenas na memória da GPU, a capacidade disponível se esgota rapidamente e a taxa de transferência cai (linha laranja). Com um design de três camadas (GPU para trabalho ativo, CPU para preparação rápida e storage compartilhado para cache durável e compartilhável), a plataforma suporta mais sessões e evita a mesma queda acentuada de desempenho. Na prática, a arquitetura em camadas ajuda você a obter mais trabalho das mesmas GPUs, reduzindo a computação de preenchimento repetida.

  - **Em termos simples:**
    - **Nível de GPU** — lida com tarefas ativas em andamento.
    - **Camada de CPU** — mantém o cache usado recentemente próximo ao mecanismo de serviço.
    - **Camada de storage compartilhado** — preserva o acerto do cache entre servidores e libera a memória da GPU/CPU para novas sessões.

Figura 1 - Benchmark de Inferência Multirrodada

O benchmark indica que adicionar storage compartilhado junto com memória da CPU não reduz o desempenho; amplia a faixa operacional utilizável antes que a degradação da taxa de transferência apareça. A arquitetura de storage em camadas adiciona efetivamente camadas de memória para inferência, reduzindo a necessidade de adicionar GPUs sempre que o número de sessões, o comprimento do contexto ou a concorrência aumentarem.

- **Adequação empresarial: protocolos padrão, sem cliente proprietário** + O storage compartilhado é importante, mas nem todo storage compartilhado é igual. NetApp suporta o descarregamento de cache KV usando protocolos e ferramentas padrão do setor:
  - **o StorageGRID S3**
  - **NFS / pNFS para ONTAP NAS**

Não é necessário nenhum cliente de inferência proprietário personalizado. As equipes de operações podem usar os mesmos protocolos de storage que já utilizam para outros serviços de dados, com a proteção de dados, segurança e mobilidade multicloud já conhecidas.

Implantar o descarregamento do cache KV

O storage compartilhado amplia a capacidade do cache KV e permite a reutilização entre instâncias de serviço. Para usar essa camada em produção, você configura um mecanismo de inferência e uma estrutura de offloading do cache KV. As seções a seguir descrevem como implementar essa stack usando opções de ferramentas comuns.

vLLM

O vLLM é um mecanismo de inferência LLM de código aberto popular e de alto desempenho. Nesta solução, ele é o mecanismo que recebe as solicitações do usuário, gera as respostas e gerencia o cache KV dentro da memória da GPU durante a inferência. O vLLM é plugável – você pode escolher qual estrutura de offloading deseja usar. As seções a seguir descrevem como implementar uma pilha de serviço baseada em vLLM com estruturas de offloading populares.

LMCache (modo em processo)

O LMCache é uma estrutura open-source popular de descarregamento de cache KV. Esta seção descreve como implementar uma pilha de serviço baseada em vLLM que utiliza o LMCache para descarregar o cache KV. Nesta implementação, o LMCache trabalha com o vLLM para mover o cache KV da memória da GPU para camadas maiores, como RAM da CPU, disco local ou storage compartilhado (como o StorageGRID S3 ou uma montagem NAS do ONTAP).

Em uma configuração padrão, o vLLM mantém o cache KV apenas na memória da GPU. À medida que a carga do sistema aumenta, isso se torna o gargalo. Nesta solução, o vLLM é emparelhado com o LMCache, onde o vLLM serve o modelo, enquanto o LMCache descarrega e reutiliza o cache entre a CPU e o storage compartilhado da NetApp. O LMCache se conecta ao vLLM por meio de um conector; você o habilita na inicialização com o parâmetro --kv-transfer-config do vLLM, para que o vLLM e o LMCache salvem o cache no storage e o recarreguem quando houver um acerto do cache.

O modo em processo é o método original de execução do LMCache com o vLLM. Quando você usa o modo em processo, o LMCache é executado dentro do processo do vLLM. Versões posteriores do LMCache adicionaram o modo multiprocesso, que é a abordagem atualmente recomendada para melhor suporte a recursos e desempenho. Esta seção aborda o modo em processo, que está marcado como obsoleto na documentação atual do LMCache. Para novas implementações, considere usar o modo multiprocesso.

Para esta implementação, você irá:

- Instale o vLLM juntamente com o LMCache
- Inicie o vLLM com o conector LMCache ativado
- Implante uma configuração de três camadas (GPU + CPU + storage compartilhado) com duas instâncias vLLM (em GPUs separadas) atrás de um roteador vLLM, ambas usando o mesmo backend de storage compartilhado.

Pré-requisitos

- Servidor Linux com GPU NVIDIA, drivers NVIDIA (incluindo CUDA) e pelo menos uma GPU (duas GPUs ou vários servidores para a configuração completa de duas instâncias + roteador)

- Python e uv instalados
- Docker e NVIDIA Container Toolkit instalados (necessários para o roteador vLLM na Etapa 4)
- Acesso à rede para baixar o modelo (por exemplo, Hugging Face) e para acessar seu endpoint de storage

Backend S3 do StorageGRID

- Bucket S3 criado para uso de cache KV
- Credenciais de leitura/gravação para o bucket
- Conectividade de rede do servidor GPU ao endpoint S3
- Solicitações S3 no estilo de hospedagem virtual ativadas

Backend ONTAP pNFS/NFS do ONTAP

- Volume ONTAP exportado para o(s) servidor(es) GPU
- Caminho de montagem criado (por exemplo, /mnt/kvcache) em **cada** servidor que executa o vLLM. Exemplo de comando de montagem para o pNFS de alto desempenho sobre RDMA (RoCE):

```
mount -o
vers=4.1,proto=rdma,trunkdiscovery,write=eager,rsize=262144,wsize=262144
my_storage.my_company.net:/kvcache /mnt/kvcache
```

[[1-install-vllm-and-lmcache]]
=== 1. Instale o vLLM e o LMCache

Crie um ambiente virtual e instale o vLLM e o LMCache. Consulte a documentação do LMCache "documentação de instalação" para obter mais detalhes. É fundamental que você siga o caminho de instalação correto para a sua versão do CUDA.

```
uv venv .venv
source .venv/bin/activate
uv pip install --upgrade lmcache vllm
```

Neste ponto, você tem o mecanismo de inferência (vLLM) e a camada de cache (LMCache).

[[2-configure-lmcache]]
=== 2. Configurar o LMCache

O vLLM não descarrega o cache KV por si só. Você habilita o LMCache através do conector de transferência KV do vLLM. Crie um arquivo YAML (lmcache-config.yaml) que defina a configuração da CPU e da camada de storage compartilhado. O LMCache lê este arquivo YAML como parte da sequência de inicialização da inferência do vLLM.

Exemplo de trecho de código: camada compartilhada S3 do StorageGRID

```
local_cpu: True
max_local_cpu_size: 100
save_decode_cache: True
remote_url: "s3://<bucket>.<storagegrid_s3_endpoint_fqdn>"
remote_serde: "naive"
extra_config:
  s3_num_io_threads: 320
  s3_prefer_http2: False
  s3_enable_s3express: False
  save_chunk_meta: False
  disable_tls: <True/False>
  aws_access_key_id: "<storagegrid_s3_access_key>"
  aws_secret_access_key: "<storagegrid_s3_secret_key>"
  s3_region: "us-east-1"
```

Observação: substitua o nome do bucket, o endpoint, as credenciais e `disable_tls` para corresponder ao seu ambiente.

Exemplo de trecho de código: o ONTAP pNFS/NAS camada compartilhada

```
local_cpu: true
max_local_cpu_size: 100
save_decode_cache: true
remote_storage_plugins: ["fs"]
extra_config:
  remote_storage_plugin.fs.base_path: /mnt/kvcache
  save_chunk_meta: false
```

Monte a exportação do ONTAP usando o procedimento NFS/pNFS do seu site antes da Etapa 3.

[[3-run-two-vllm-instances-shared-cache-across-servers]]
=== 3. Execute duas instâncias do vLLM (acerto do cache compartilhado entre servidores)

Defina variáveis de ambiente comuns em ambas as instâncias:

```
export LMCACHE_CONFIG_FILE=$(pwd)/lmcache-config.yaml
export PYTHONHASHSEED=0
export VLLM_API_KEY='<shared_api_key>'
```

Instância 1 (GPU 0, port 8001):

```
export CUDA_VISIBLE_DEVICES=0
vllm serve Qwen/Qwen3-8B \
  --tensor-parallel-size 1 \
  --kv-transfer-config
'{"kv_connector":"LMCacheConnectorV1","kv_role":"kv_both"}' \
  --port 8001
```

Instância 2 (GPU 1, porta 8002 — terminal separado):

```
export CUDA_VISIBLE_DEVICES=1
vllm serve Qwen/Qwen3-8B \
  --tensor-parallel-size 1 \
  --kv-transfer-config
'{"kv_connector":"LMCacheConnectorV1","kv_role":"kv_both"}' \
  --port 8002
```

Use o mesmo arquivo de configuração e a mesma semente de hash para que ambas as instâncias concordem com a identidade do bloco de cache e possam ler os dados descarregados uma da outra do storage compartilhado. Defina `VLLM_API_KEY` em ambas as instâncias; o roteador passa esse valor como `OPENAI_API_KEY` para que as solicitações roteadas sejam aceitas pelas instâncias.

Nota: Uma única instância de GPU também pode ser usada para implementar uma arquitetura de storage em camadas. Nesse caso, ignore a etapa 4.

[[4-deploy-the-vllm-router-single-entry-point]]
=== 4. Implante o roteador vLLM (ponto de entrada único)

Após ambas as instâncias estarem íntegras, implemente um roteador para que os clientes usem uma URL compatível com OpenAI.

```
docker run -d --rm --name vllm-router \
  --network host \
  -e OPENAI_API_KEY="${VLLM_API_KEY}" \
  ghcr.io/vllm-project/production-stack/router:latest \
  --port 8000 \
  --service-discovery static \
  --static-backends "http://localhost:8001,http://localhost:8002" \
  --static-models "Qwen/Qwen3-8B,Qwen/Qwen3-8B" \
  --routing-logic roundrobin
```

Envie tráfego para http://localhost:8000/v1. O roteador distribui as solicitações; o LMCache e o storage compartilhado permitem a reutilização do cache entre instâncias, não apenas dentro de uma única GPU.

Por exemplo, você pode enviar uma solicitação ao roteador usando curl da seguinte forma (substitua <shared_api_key> pela sua respectiva chave de API):

```
curl -s http://localhost:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H 'Authorization: Bearer <shared_api_key>' \
  -d '{
    "model": "Qwen/Qwen3-8B",
    "messages": [{"role": "user", "content": "What is 2+2?"}]
  }' | jq .
```

[[5-validate-that-tiering-is-working]]
=== 5. Verifique se a tierização está funcionando

- Ambos os processos vLLM escutam nas portas 8001 e 8002; o roteador responde na porta 8000.
- Envie uma solicitação com um prefixo longo pelo roteador.
- Confirme se os objetos ou arquivos aparecem no storage compartilhado (S3 bucket ou ls -ltr /mnt/kvcache).
- Envie uma solicitação semelhante novamente: a segunda solicitação deverá mostrar um comportamento de preenchimento automático mais rápido ou de acerto do cache nos registros.
- Repita através do roteador para que o tráfego possa atingir a outra instância.

Conclusão

A implementação de uma arquitetura de cache KV em camadas move o cache KV da memória da GPU para a memória da CPU e para o storage compartilhado da NetApp, permitindo que a inferência seja escalável de acordo com o número de usuários e o tamanho do contexto, sem desperdiçar ciclos da GPU com preenchimentos repetidos. O storage compartilhado transforma o cache em infraestrutura reutilizável entre as instâncias de serviço e ajuda você a obter mais valor do seu investimento em GPU.

NetApp storage é uma excelente opção para este nível, pois oferece o que a inferência de produção precisa além da capacidade bruta: acesso padrão via S3 e NFS/pNFS, acerto do cache compartilhado que várias instâncias do mecanismo de serviço podem usar ao mesmo tempo e serviços de dados corporativos nos quais você já confia para durabilidade, proteção e governança. Você não precisa de um cliente proprietário; as estruturas de descarregamento de cache KV se conectam ao StorageGRID S3 ou a um ponto de montagem NAS ONTAP usando protocolos familiares.

NetApp oferece a você uma base de storage familiar para o descarregamento do cache KV sem alterar como você executa a inferência ou como suas equipes gerenciam os dados.

Informações sobre direitos autorais

Informações sobre marcas comerciais