Stateful Cache-Aware Routing for Distributed LLM Inference

Team: Aman Pandey Sanjeev Kumar, Aryan Khanolkar, Guna Avula
CS 59200: Machine Learning Systems

November 2025

1 Introduction

The efficiency of Large Language Model (LLM) inference is fundamentally constrained by the memory-bound prefill phase. While node-level optimizations like PagedAttention have mitigated memory fragmentation, production environments deploying multiple replicas behind stateless load balancers fail to leverage cluster-wide cache locality. We propose a **Stateful Cache-Aware Router** that decouples routing logic from heavy orchestration platforms. Our system utilizes a lightweight, eventually consistent global map of cached prefixes, maintained via a novel **asynchronous batched eviction callback** mechanism injected directly into the vLLM kernel. This ensures that requests sharing long context prefixes (e.g., RAG system prompts) are routed to replicas holding the "hot" state, effectively eliminating redundant prefill computation.

2 Problem Statement

In multi-turn conversation and Retrieval-Augmented Generation (RAG) workloads, the prompt prefill phase exhibits quadratic complexity $O(N^2)$ with respect to sequence length. Automatic Prefix Caching (APC) features in engines like vLLM [1] allow a single GPU to reuse Key-Value (KV) cache blocks for identical prefixes, significantly reducing Time-To-First-Token (TTFT).

However, in a distributed setting with N replicas, standard load balancing algorithms (Round-Robin or Least-Connection) are stateless. They route sequential requests from the same user or RAG context to arbitrary nodes. This "cache-blind" routing destroys locality, forcing replicas to recompute the KV cache for prefixes that effectively exist elsewhere in the cluster. The core research problem is maintaining a globally consistent view of the distributed cache state without incurring the high latency of centralized locking or the complexity of distributed storage systems.

3 Status Quo and Limitations

Current approaches to inference optimization focus primarily on single-node efficiency or heavy orchestration. We distinguish our work from the following:

- **Node-Local Caching (vLLM [1], SGLang [2]):** vLLM's Block Manager and SGLang's Radix Attention efficiently manage local memory but lack visibility into peer nodes. They function as isolated islands of state; a cache hit on Node A does not benefit a request routed to Node B.
- **Disaggregated Storage (LMCache [3]):** LMCache decouples the KV cache from the compute engine, allowing state transfer between nodes. While this solves availability, fetching KV states from remote storage (even over RDMA) introduces network latency that often exceeds the cost of re-computation for shorter prefixes. Our approach prioritizes *routing to the data* rather than *moving the data*.

- **Orchestration-Coupled Scheduling (llm-d [4]):** The `llm-d` project proposes cache-aware scheduling but is implemented as a custom Kubernetes scheduler. This creates a high barrier to entry and tight coupling with infrastructure. Our router is a lightweight middleware agnostic to the underlying container orchestrator.
- **Batching Optimizations (Sarathi-Serve [5], CacheBlend [6]):** These systems optimize throughput via chunked prefill and batching schedules but do not explicitly solve the distributed routing consistency problem for prefixes.

4 Implementation Plans

We plan to implement a centralized, Python-based router service that interacts with modified vLLM workers.

4.1 Architecture Components

1. **The Tokenizer-Aligned Router:** To ensure hash consistency between the router and the worker's internal PagedAttention logic, the router will maintain a CPU-side instance of the model's tokenizer. Incoming prompts will be tokenized and hashed (SHA-256 of token IDs) to query the `GlobalCacheMap`.
2. **Surgical vLLM Modification (Batched Callbacks):** We will modify the `vllm.core.block_manager` to intercept block eviction events. To prevent network saturation (DDOS) during high-churn cache replacements, we will implement a buffered, asynchronous reporting thread that batches eviction notifications and sends them to the router at fixed intervals (e.g., 100ms).
3. **Routing Policy:** The router will implement a "Sticky-Least-Loaded" policy:
   - *Hit:* Route to the least-loaded replica possessing the prefix hash.
   - *Miss:* Route to the globally least-loaded replica and speculatively update the map.

4.2 Evaluation Strategy

We will utilize the **ShareGPT** and **MTRAG** datasets to simulate multi-turn conversation workloads. We will compare our Stateful Router against a baseline Round-Robin Load Balancer across three metrics:

- **Time-To-First-Token (TTFT):** Measuring the latency reduction from skipped prefills.
- **Cache Hit Rate:** The percentage of requests successfully routed to a "hot" replica.
- **Stale Cache Rate:** A novel metric measuring how often the router sends a request to a node that has already evicted the prefix (testing the eventual consistency of our callback mechanism).

References

[1] Kwon, W., et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention." *SOSP 2023.*

[2] Zheng, L., et al. "SGLang: Efficient Execution of Structured Language Model Programs." *arXiv 2023.*

[3] Gim, I., et al. "LMCache: A High-Performance Distributed KV Cache for LLM Inference." *arXiv 2023.*

[4] llm-d Team. "llm-d: Kubernetes-native LLM Scheduler." *GitHub 2024*.

[5] Agrawal, A., et al. "Sarathi-Serve: Taming LLM Serving Latency with Chunked Prefills." *arXiv 2024*.

[6] Patel, J., et al. "CacheBlend: Fast Large Language Model Serving for RAG with KV Cache Merging." *EuroSys 2024*.