Streaming Architectures for Low-Latency LLM Serving using Apache Flink and vLLM

Yavuz Ferhatosmanoglu

vLLM: Efficient LLM Serving

- Recently emerged as the state-of-the-art open-source LLM serving engine
- Features include:
  - Paged Attention
  - Continuous batching
  - Chunked prefill
  - Quantisation
- Supports a wide range of LLMs: Transformer, MoE, Multimodal
- Operates on diverse hardware
- In this project: optimises micro-level efficiency of generating tokens

Apache Flink: Stateful Stream Processing

- Leading open-source framework for stateful processing over datastreams
- Shifts focus from batch-processing to real-time processing
- Features include:
    - Backpressure
    - Context windowing
    - Fault-tolerance
- Used in industry-level real-time data pipelines
- In this project: optimises macro-level flow of data, bridging the gap between chaotic real-time data and the LLM inference engine

Stateful Real-Time Inference Architecture

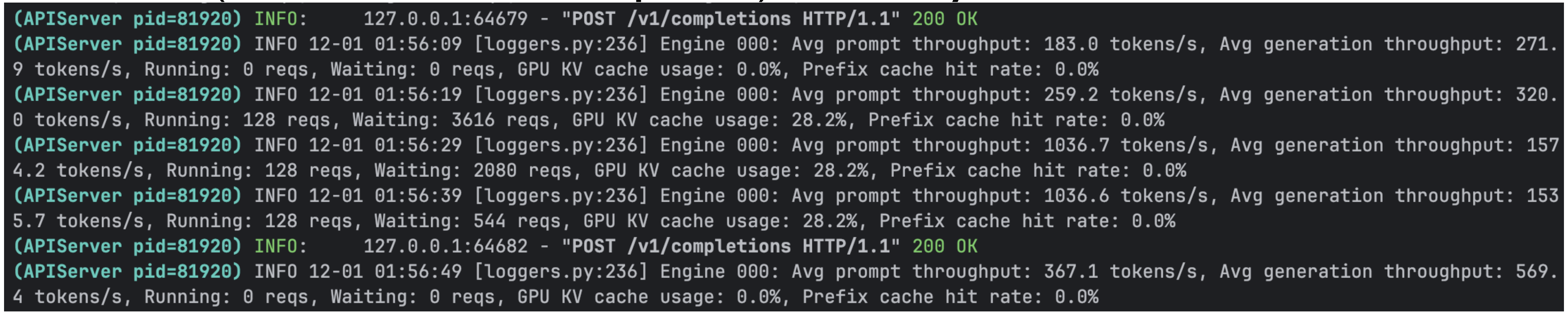
- Combined solution:
  - Stateful management (Flink) on top of stateless computations (vLLM)
  - Flink protects the inference engine from data streams
  - vLLM maximises hardware utilisation and throughput
- Many applications: identifying viral trends, emerging topic modelling, real-time summarisation pipelines
- Goals:
  - Real-time LLM serving system designed and built
  - End-to-end demonstrations with real-world data
  - Evaluation of the interaction of streaming mechanics with vLLM internals

Progress Plan

- Set up vLLM and Flink environments
- Benchmark batch processing of requests within a vLLM Server (3 Dec)
- Literature survey on LLM serving (12 Dec)
- Development and evaluation of system (19 Dec)
  - Custom stateful Flink operators to integrate with vLLM
  - Insights from application datasets to refine architecture/operators
  - Evaluation of throughput, per-event latency and resource utilisation across a range of models and parallelism settings
- Report written (9 Jan)

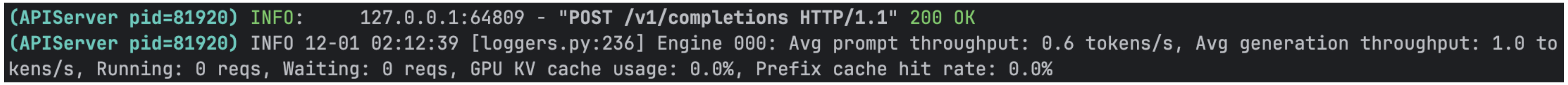
Batch Processing with vLLM Server

- Batched (5000 ~5 token requests). Latency: 26.07s



```
(APIServer pid=81920) INFO:     127.0.0.1:64679 - "POST /v1/completions HTTP/1.1" 200 OK
(APIServer pid=81920) INFO 12-01 01:56:09 [loggers.py:236] Engine 000: Avg prompt throughput: 183.0 tokens/s, Avg generation throughput: 271.
9 tokens/s, Running: 0 reqs, Waiting: 0 reqs, GPU KV cache usage: 0.0%, Prefix cache hit rate: 0.0%
(APIServer pid=81920) INFO 12-01 01:56:19 [loggers.py:236] Engine 000: Avg prompt throughput: 259.2 tokens/s, Avg generation throughput: 320.
0 tokens/s, Running: 128 reqs, Waiting: 3616 reqs, GPU KV cache usage: 28.2%, Prefix cache hit rate: 0.0%
(APIServer pid=81920) INFO 12-01 01:56:29 [loggers.py:236] Engine 000: Avg prompt throughput: 1036.7 tokens/s, Avg generation throughput: 157
4.2 tokens/s, Running: 128 reqs, Waiting: 2080 reqs, GPU KV cache usage: 28.2%, Prefix cache hit rate: 0.0%
(APIServer pid=81920) INFO 12-01 01:56:39 [loggers.py:236] Engine 000: Avg prompt throughput: 1036.6 tokens/s, Avg generation throughput: 153
5.7 tokens/s, Running: 128 reqs, Waiting: 544 reqs, GPU KV cache usage: 28.2%, Prefix cache hit rate: 0.0%
(APIServer pid=81920) INFO:     127.0.0.1:64682 - "POST /v1/completions HTTP/1.1" 200 OK
(APIServer pid=81920) INFO 12-01 01:56:49 [loggers.py:236] Engine 000: Avg prompt throughput: 367.1 tokens/s, Avg generation throughput: 569.
4 tokens/s, Running: 0 reqs, Waiting: 0 reqs, GPU KV cache usage: 0.0%, Prefix cache hit rate: 0.0%
```

- Single ~5 token request. Latency: 0.15s.



```
(APIServer pid=81920) INFO:     127.0.0.1:64809 - "POST /v1/completions HTTP/1.1" 200 OK
(APIServer pid=81920) INFO 12-01 02:12:39 [loggers.py:236] Engine 000: Avg prompt throughput: 0.6 tokens/s, Avg generation throughput: 1.0 to
kens/s, Running: 0 reqs, Waiting: 0 reqs, GPU KV cache usage: 0.0%, Prefix cache hit rate: 0.0%
```

Thank you!
Any Questions?