(optimization-and-tuning)=

Optimization and Tuning

Preemption

Due to the auto-regressive nature of transformer architecture, there are times when KV cache space is insufficient to handle all batched requests.
The vLLM can preempt requests to free up KV cache space for other requests. Preempted requests are recomputed when sufficient KV cache space becomes
available again. When this occurs, the following warning is printed:

```text
WARNING 05-09 00:49:33 scheduler.py:1057 Sequence group 0 is preempted by PreemptionMode.SWAP mode because there is not enough KV cache space. This can affect the end-to-end performance. Increase gpu_memory_utilization or tensor_parallel_size to provide more KV cache memory. total_cumulative_preemption_cnt=1
```

While this mechanism ensures system robustness, preemption and recomputation can adversely affect end-to-end latency.
If you frequently encounter preemptions from the vLLM engine, consider the following actions:

- Increase `gpu_memory_utilization`. The vLLM pre-allocates GPU cache by using gpu_memory_utilization% of memory. By increasing this utilization, you can provide more KV cache space.
- Decrease `max_num_seqs` or `max_num_batched_tokens`. This can reduce the number of

concurrent requests in a batch, thereby requiring less KV cache space.

- Increase `tensor_parallel_size`. This approach shards model weights, so each GPU has more memory available for KV cache.

You can also monitor the number of preemption requests through Prometheus metrics exposed by the vLLM. Additionally, you can log the cumulative number of preemption requests by setting disable_log_stats=False.

(chunked-prefill)=

Chunked Prefill

vLLM supports an experimental feature chunked prefill. Chunked prefill allows to chunk large prefills into smaller chunks and batch them together with decode requests.

You can enable the feature by specifying `--enable-chunked-prefill` in the command line or setting `enable_chunked_prefill=True` in the LLM constructor.

```python
llm = LLM(model="meta-llama/Llama-2-7b-hf", enable_chunked_prefill=True)
# Set max_num_batched_tokens to tune performance.
# NOTE: 2048 is the default max_num_batched_tokens for chunked prefill.
# llm = LLM(model="meta-llama/Llama-2-7b-hf", enable_chunked_prefill=True, max_num_batched_tokens=2048)
```

By default, vLLM scheduler prioritizes prefills and doesn't batch prefill and decode to the same

batch.

This policy optimizes the TTFT (time to the first token), but incurs slower ITL (inter token latency) and inefficient GPU utilization.

Once chunked prefill is enabled, the policy is changed to prioritize decode requests.

It batches all pending decode requests to the batch before scheduling any prefill.

When there are available token_budget (`max_num_batched_tokens`), it schedules pending prefills.

If a last pending prefill request cannot fit into `max_num_batched_tokens`, it chunks it.

This policy has two benefits:

- It improves ITL and generation decode because decode requests are prioritized.
- It helps achieve better GPU utilization by locating compute-bound (prefill) and memory-bound (decode) requests to the same batch.

You can tune the performance by changing `max_num_batched_tokens`. By default, it is set to 2048.

Smaller `max_num_batched_tokens` achieves better ITL because there are fewer prefills interrupting decodes.

Higher `max_num_batched_tokens` achieves better TTFT as you can put more prefill to the batch.

- If `max_num_batched_tokens` is the same as `max_model_len`, that's almost the equivalent to the default scheduling policy (except that it still prioritizes decodes).
- Note that the default value (2048) of `max_num_batched_tokens` is optimized for ITL, and it may have lower throughput than the default scheduler.

We recommend you set `max_num_batched_tokens > 2048` for throughput.

See related papers for more details (<https://arxiv.org/pdf/2401.08671> or <https://arxiv.org/pdf/2308.16369>).

Please try out this feature and let us know your feedback via GitHub issues!