Graduate Term Project: Dynamic Batching Schemes in Large-Scale Systems
Course: CENG523 Advanced Topics of Real-Time Systems
Semester: 2024-2025 Fall

Background

Dynamic batching is a key optimization technique for large-scale systems, particularly in the context of large language models (LLMs). It involves grouping multiple requests dynamically into a batch for processing, thereby improving resource utilization, reducing latency, and increasing throughput. This project focuses on studying, modeling, and verifying dynamic batching schemes using scheduling algorithms and formal methods.

Dynamic batching is a technique used during inference to group multiple requests together into a single batch for processing, maximizing the utilization of available computational resources. Unlike static batching, where the batch size is fixed beforehand, dynamic batching adjusts the batch size dynamically based on incoming requests.

How Dynamic Batching Works

1. **Request Accumulation**: As inference requests arrive, they are queued temporarily instead of being processed immediately.
2. **Batch Formation**:
   - The system waits for a short, configurable time window (batch timeout).
   - During this window, it collects incoming requests into a batch.
   - The batch size is determined by how many requests arrive within the timeout or a predefined maximum batch size.
3. **Inference Execution**:
   - The batched requests are processed together on the model.
   - The results are split and sent back to the respective clients.

Key Parameters

- **Maximum Batch Size**: The largest number of requests that can be grouped into a single batch.
- **Batch Timeout**: The time the system waits for additional requests to form a batch before processing.

Benefits of Dynamic Batching

1. **Improved Throughput**:
   Combining multiple requests into a single batch makes better use of GPU/TPU parallelism. The per-request computational cost is reduced.
2. **Reduced Latency**:

By processing several requests simultaneously, dynamic batching reduces idle time for hardware.

3. **Adaptability**:
   Works well in real-time environments where request arrival rates vary.
   Unlike static batching, it doesn't require predefining the batch size, allowing for flexibility.
4. **Cost Efficiency**:
   Maximizes resource utilization, reducing the need for overprovisioned hardware.

Objectives

1. Understand the concept of dynamic batching and its relevance to large-scale systems.
2. Conduct a short literature survey to identify key approaches and challenges.
3. Develop a scheduling algorithm to optimize the performance of dynamic batching systems.
4. Model the system using timed automata in the UPPAAL tool.
5. Verify the scheduling algorithm formally using UPPAAL.

Tasks

1. ***Literature Survey***:
- Research papers and articles on dynamic batching and related optimization techniques.
- Document key findings and summarize existing solutions.
2. ***Scheduling Algorithm***:
- Propose a scheduling algorithm for dynamic batching systems.
- The algorithm should consider parameters such as batch size, timeout, request arrival rates, and hardware constraints.

OPTIONAL: You can develop a simulator in Python to simulate your solution.

3. ***Modeling the System***:
- Use timed automata in UPPAAL to model the dynamic batching system.
- Include components like request arrival, batching, execution, and timeout handling.
4. ***Verification in UPPAAL***:
- Define properties such as throughput, latency, and resource utilization.
- Verify the correctness and performance of the scheduling algorithm against these properties.
5. ***Report and Presentation***:
- Present early findings and your approach to solve the problem in 20 minutes.
- Prepare a detailed report summarizing the work. Follow IEEE Conference format and the number of pages should be minimum 4 and maximum 8.

Key Time Components in Dynamic Batching

Inference Time ($T_{inference}$):
- Time required to run a batch through the model on the hardware.
- Increases sub-linearly with batch size due to parallel processing (e.g., doubling the batch size does not double the inference time).

Batch Accumulation Time ($T_{accumulation}$):
- Time spent waiting to accumulate enough requests to form a batch.
- Directly dependent on the arrival rate of requests and the batch timeout.

Communication Time ($T_{communication}$):
- Time to transfer input data to the hardware (e.g., GPU) and retrieve results.
- Typically constant per batch but depends on the hardware's memory bandwidth and input/output size.

Preprocessing Time ($T_{preprocessing}$):
- Time spent tokenizing text, padding inputs to uniform size, or applying other transformations.
- May vary based on batch size but is often linear.

Postprocessing Time ($T_{postprocessing}$):
- Time spent decoding model outputs or formatting results.
- Generally linear with batch size but small compared to other times.

Batch Timeout ($T_{timeout}$):
- A configured maximum time to wait for incoming requests before executing a batch.
- Directly impacts latency and batching efficiency.

Note that inference time grows sub-linearly meaning that larger batches make better use of parallelism and communication time grows slightly meaning larger batches involve more data transfer.

Other Times to Consider

Preprocessing Times

- Tokenization: Depends on the number of tokens per input and batch size.
- Example: 1 ms per input for a sequence length of 128 tokens.
- Padding: Minimal, as it's often integrated with tokenization.

Postprocessing Times

- For decoding or formatting outputs, times are small relative to inference.
- Example: 0.5 ms per output for a batch.

Batch Accumulation Times

- Depends on the request arrival rate λ and batch size:

- Higher arrival rates reduce $T_{accumulation}$.
- Example: With λ = 100 requests per second and a batch size of 8:
- $T_{accumulation}$ = 8/100 = 0.08 s.

Modeling Dynamic Batching System

When modeling a system, you can combine the above times to compute total latency per request or overall system throughput.

1. Total Latency per Request

$T_{latency} = T_{accumulation} + T_{preprocessing} + T_{communication} + T_{inference} + T_{postprocessing}$

Example (Batch Size = 8):

- $T_{accumulation}$ = 80 ms
- $T_{preprocessing}$ = 2 ms
- $T_{communication}$ = 4 ms
- $T_{inference}$ = 40 ms
- $T_{postprocessing}$ = 1 ms

T_{latency} = 80 + 2 + 4 + 40 + 1 = 127 ms

2. System Throughput

Throughput ($T_{throughput}$) is the number of requests processed per second:

$T_{throughput}$ = Batch Size / $T_{latency}$

Example for Batch Size = 8 and $T_{latency}$ = 127 ms:

$T_{throughput}$ = 8/0.127 = 63 requests/second

Problem Formulation

Requests arrive at random intervals (modeled as a Poisson process). Arrival rate(λ) is the number of requests per second. We can assume that the sizes of the requests are identical. Therefore, $T_{preprocessing}$, $T_{communication}$, and $T_{postprocessing}$ are fixed as follows:

- $T_{preprocessing}$ = 2 ms
- $T_{communication}$ = 4 ms
- $T_{postprocessing}$ = 1 ms

In this project, take into account the following table for inference and communication times.

Batch Size	**Inference Time (ms) (LLM-1)**	**Inference Time (ms) (LLM-2)**	**Inference Time (ms) (LLM-3)**	**Communication Time (ms)**
1	10	8	4	2
2	15	12	5	2.5
4	25	20	10	3
8	40	30	15	4
16	75	50	30	5
32	140	100	60	6
64	260	200	100	8

There are three LLM models deployed in the system. LLM-1 is the most precise and the others are distilled forms. The scheduler can dynamically choose one of them for each batch to be processed depending on the CPU/GPU utilization.

Deliverables

1. A literature survey document summarizing key insights from existing work.
2. UPPAAL model files of the dynamic batching system.
3. Source code for the scheduling algorithm (in Python).
4. Verification results demonstrating the performance and correctness of the scheduling algorithm.
5. A final report detailing the methodology, results, and conclusions.
6. A presentation to share findings with peers and the instructor.

Evaluation Criteria

1. Depth and clarity of the literature survey (20%).
2. Innovation and effectiveness of the scheduling algorithm (30%).
3. Accuracy and completeness of the UPPAAL model (20%).

4. Correctness of the verification results (20%).
5. Quality of the report and presentation (10%).
6. Optional task of simulator development (BONUS: 15%)

Suggested References

- Research papers on dynamic batching in LLMs and distributed systems.
- UPPAAL documentation: https://uppaal.org/
- Related textbooks on scheduling and optimization in distributed systems.
- Articles on formal methods and model verification.

Submission Guidelines

- Presentations will be done in the classroom on 23/12 and 30/12.
- Submit all files (code, UPPAAL models, report, and presentation) in a compressed folder.
- Deadline: 18 January 2025 23:59.
- Submit to: Teams CENG523 Term Project Assignment