QoE and Fairness in LLM Serving

Kalab Assefa, Nicholas Mellon, Rishika Kalidindi, Sitota Mersha

Andes: Defining and Enhancing Quality-of-Experience in LLM-Based Text Streaming Services

Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, Mosharaf Chowdhury

LLMs are a service just like compute is a service

- It only makes sense to optimise for user service
- Current optimisations revolve around improving the overall server throughput
- This takes no regard into how each user is served

Which Service Do You Prefer?

https://github.com/llm-qoe/llm-qoe.github.io

How do we quantify service?

Time to first token (TTFT) - The amount of time taken for the first token to show up

Token delivery speed (TDS) - The rate at which tokens are delivered

Token delivery timeline (TDT) - TTFT and TDS together give the token delivery timeline.

We try to match expected TDT of the application

Observations

As LLM servers become more powerful, token generation becomes faster.

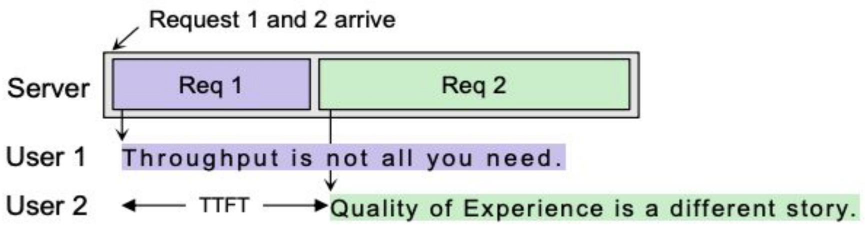
- This is pointless if the TDS of server exceeds expected TDS.

While some requests are served at higher speeds later requests are starved.

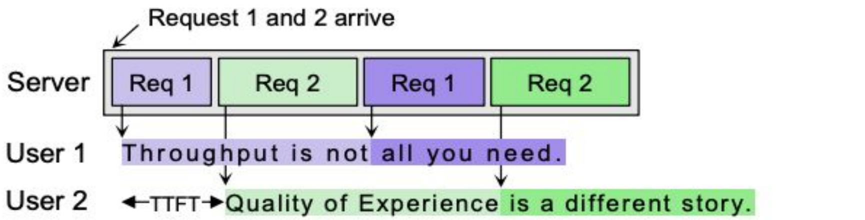
- Does this sound familiar? Yes! It is the same problem as compute resources and processes

Thus we use the same solution i.e scheduling and pre emption of requests.

Toy Example for QoE Improvement

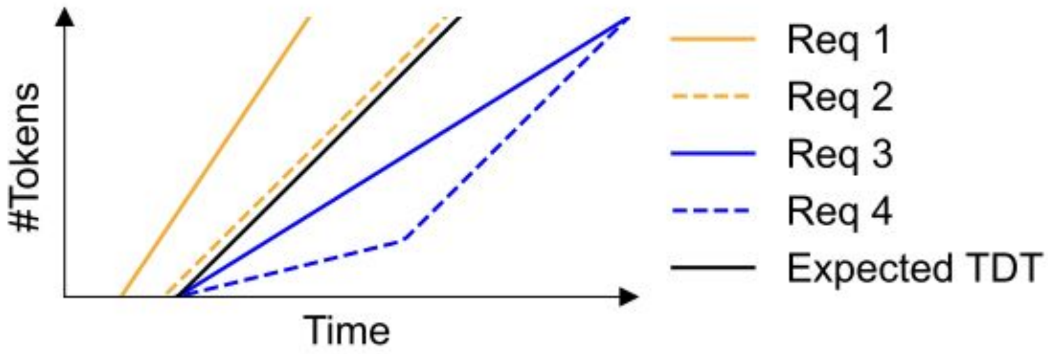


(a) Existing LLM serving systems are oblivious of QoE. User 2 experiences a long wait time (TTFT) and therefore lower QoE.



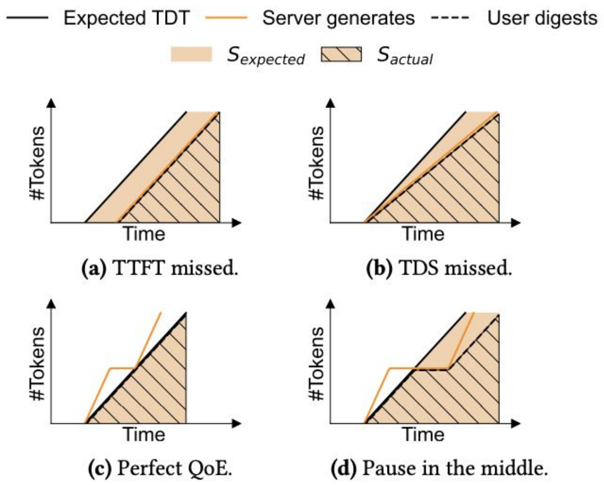
(b) A QoE-aware LLM serving system can schedule token generation over time to enhance QoE. User 2's TTFT is drastically improved without affecting User 1's token delivery timeline.

Token Delivery Visualization



- Token delivery Timeline could be plotted as a line.
- This line divides the graph space into parts with admissible QoE
- Any TDT plot consistently above the expected TDT plot has sufficient service
- The gap between expected TDT plot and actual TDT plot represents the lack or excess of service

Defining QoE



$$QoE = \frac{S_{\text{actual}}}{S_{\text{expected}}}$$

- Considering the graph space, service is representative of area under the plot
- The slope of the actual token delivery curve on the user side is capped by the expected TDS.
- Thus QoE is a ratio of area under expected TDT and actual TDT plots

QoE Aware System Solution

- We incorporate a server side queue and a client side buffer

- Request sent by the client is added to the queue.

- The server schedules the requests in the queue on hardware based on priority.
    - Scheduling policy needed

- When the request is scheduled, client receives tokens into the buffer

- The buffer outputs tokens to application at a uniform rate to prevent choppy service.

Server-side Request Scheduling in QoE-Aware System

Considerations

- How often to make scheduling decisions (time quantum)
  - Too long -> request are starved
  - Too quick -> pre emption overhead

- Which requests to serve (scheduling objective)
  - Need to get the most QoE
  - **Get difference incurred in QoE if request is scheduled vs if it is not**

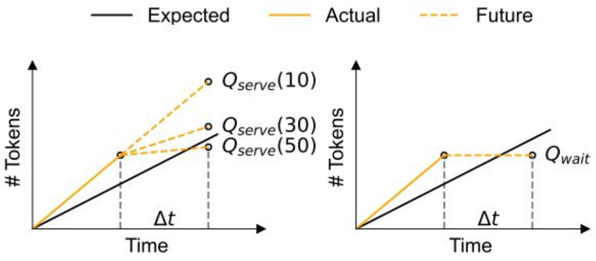
- How many requests to serve at a time (batch size)
  - Depends on size of each request and total memory available to fit

Key Policy Ideas

- We need to optimise maximum QoE achievable

- At the start of each time quantum, we need to select requests that adds the highest possible gain in total QoE

- This is like the K-item Knapsack problem
    - Slight variation of the same since value of each request also depends on number of requests in a batch (instead of being fixed)



(a) $Q_{\text{serve, i}}(B)$ **(b)** $Q_{\text{wait},i}$

Algorithm design decisions

- This is an NP-Hard problem

- Dynamic programming is needed to arrive at the optimal solution
  - Too slow and large overhead at every time quantum start

- Greedy approach is used by packing high priority request
  - High QoE gain and and low resource utilisation
  - Served requests need to be deprioritized
  - Reducing preemption

- Works well despite not being global optima with far less overhead

Additional Optimisations

- This solution is **only** triggered when limited by memory capacity or computation time

- Pre emption per request is also capped to prevent thrashing

Evaluation

Evaluation - Experiment Setup

Configurations: OPT models, A100 GPU

Workloads: ShareGPT, Multi-Round ShareGPT datasets

Baselines: vLLM (FCFS scheduling), Round Robin (fair, cyclic preemption)

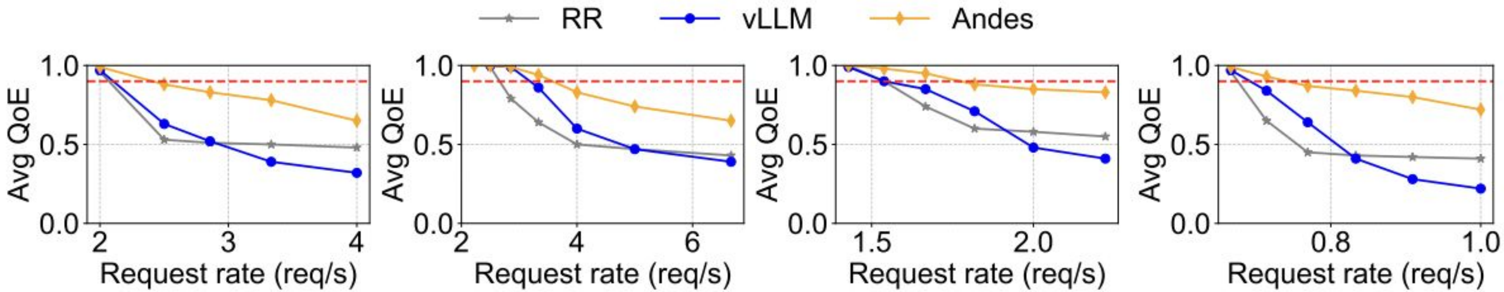
Metrics:

1. Average QoE
2. Max Request Rate while QoE above threshold (System Capacity)
3. Token Throughput

Evaluation - Results

Average QoE: up to 3.2x improvement under high loads

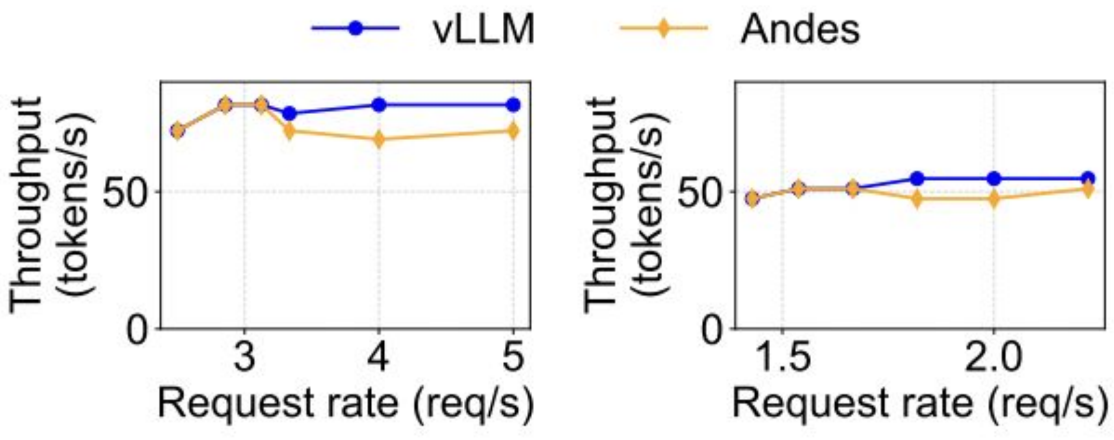
System Capacity: handles up to 1.6x higher request rate while preserving QoE



(a) OPT-13B. (b) OPT-30B. (c) OPT-66B. (d) OPT-175B.

Evaluation - Results

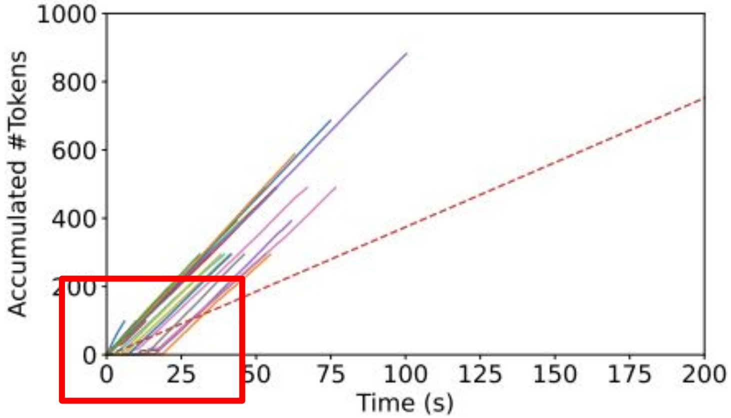
Throughput: similar to baseline, minor (<10%) drops on higher request rates



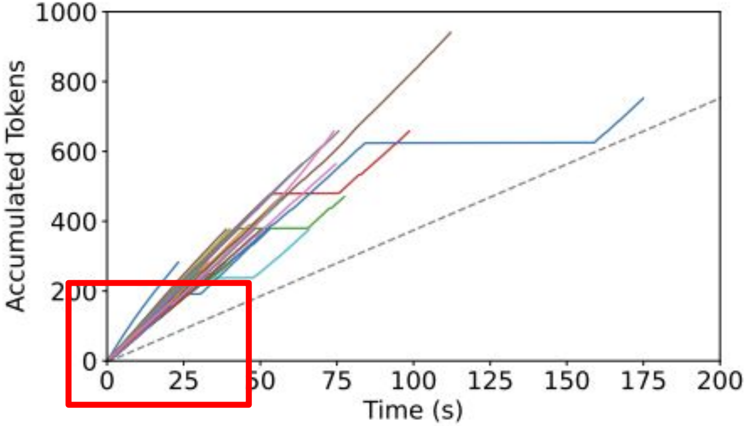
(a) ShareGPT. **(b)** Multi-Round ShareGPT.

Evaluation - Visualization

Andes delivers quicker TTFTs



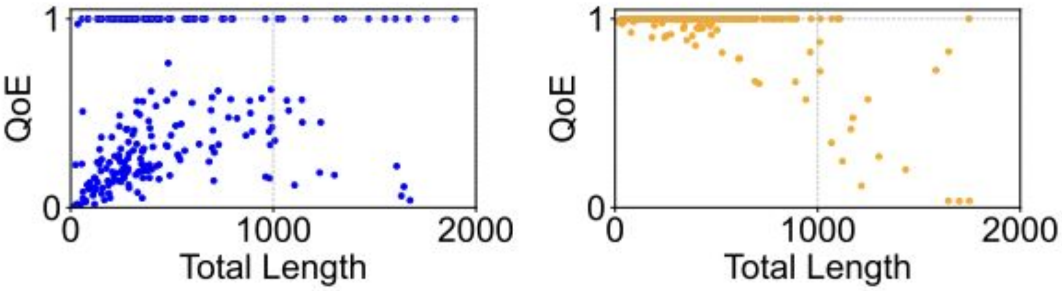
(a) FCFS.



(b) Andes.

Evaluation - Visualization

Andes slightly starves longer, more resource hungry requests

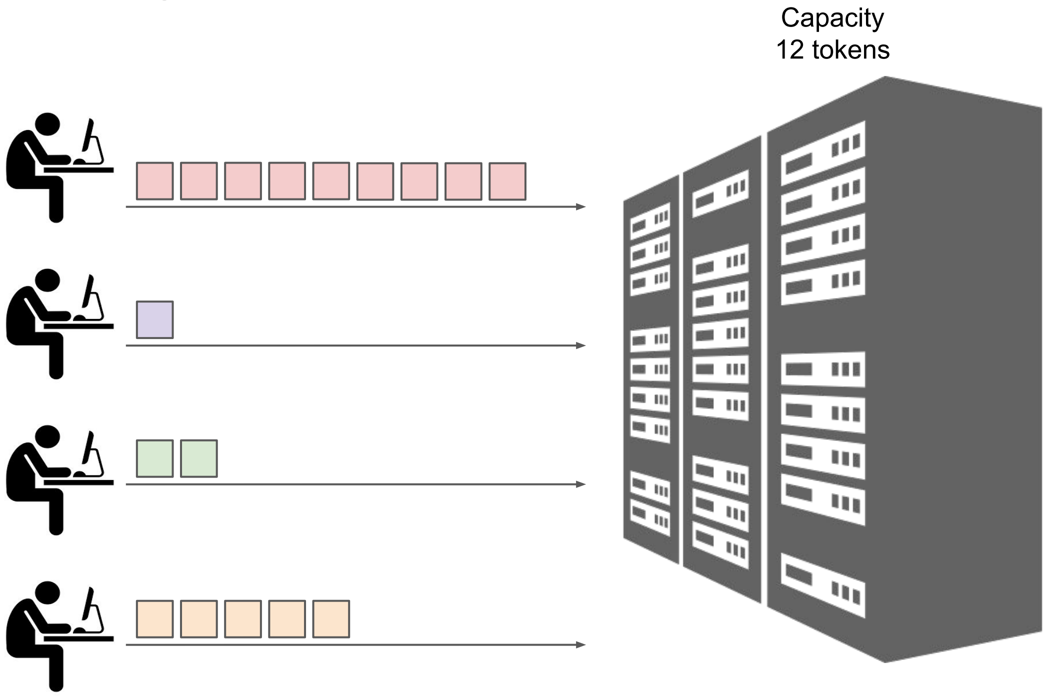


(a) vLLM. **(b)** Andes.

Fairness in Serving Large Language Models

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, Ion Stoica

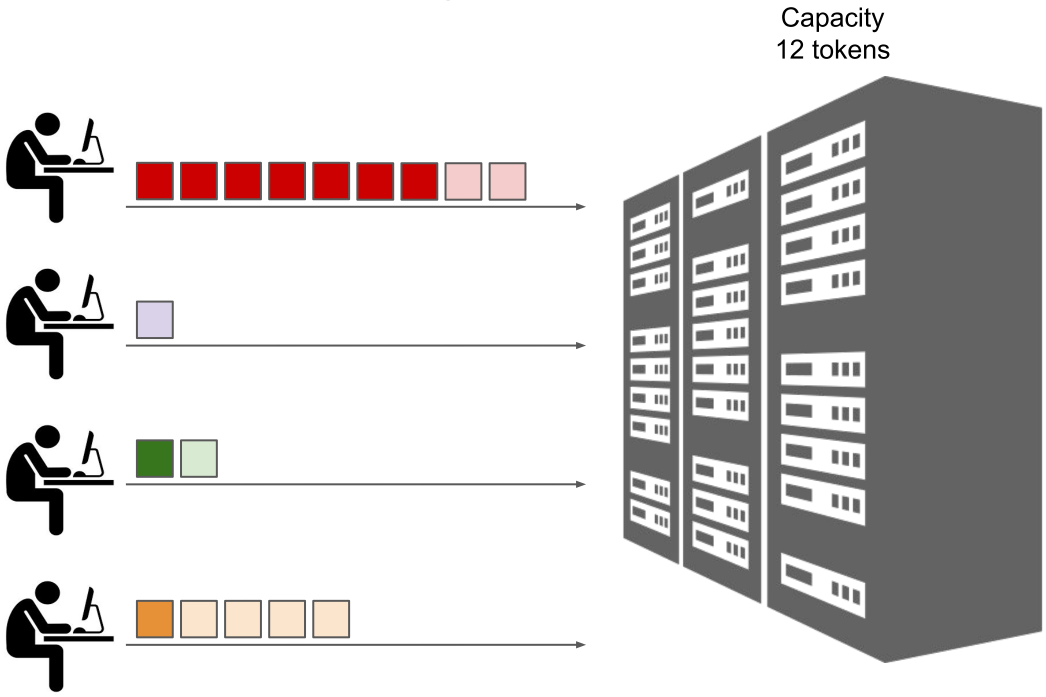
Serving LLMs



Goals

- Accuracy
- Response Time
- Resource Utilization

Issues in Serving LLMs



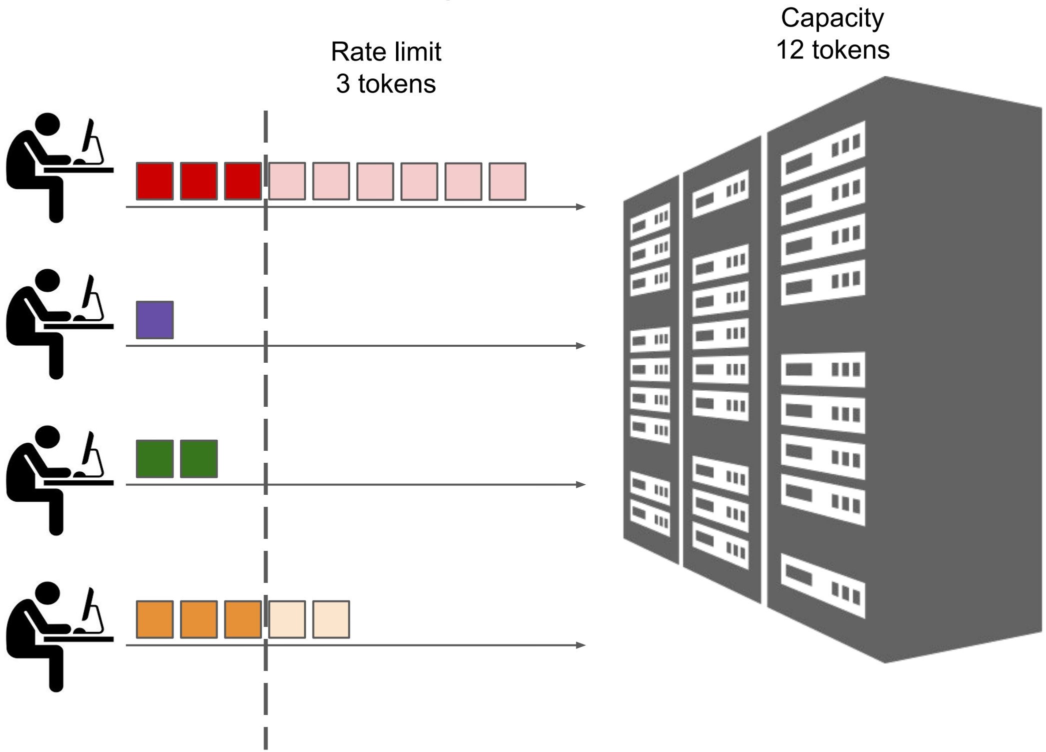
No fairness control

⚠️ Lack of isolation

User 1 is **aggressively** overloading requests

Users 2,3,4 are **starving**

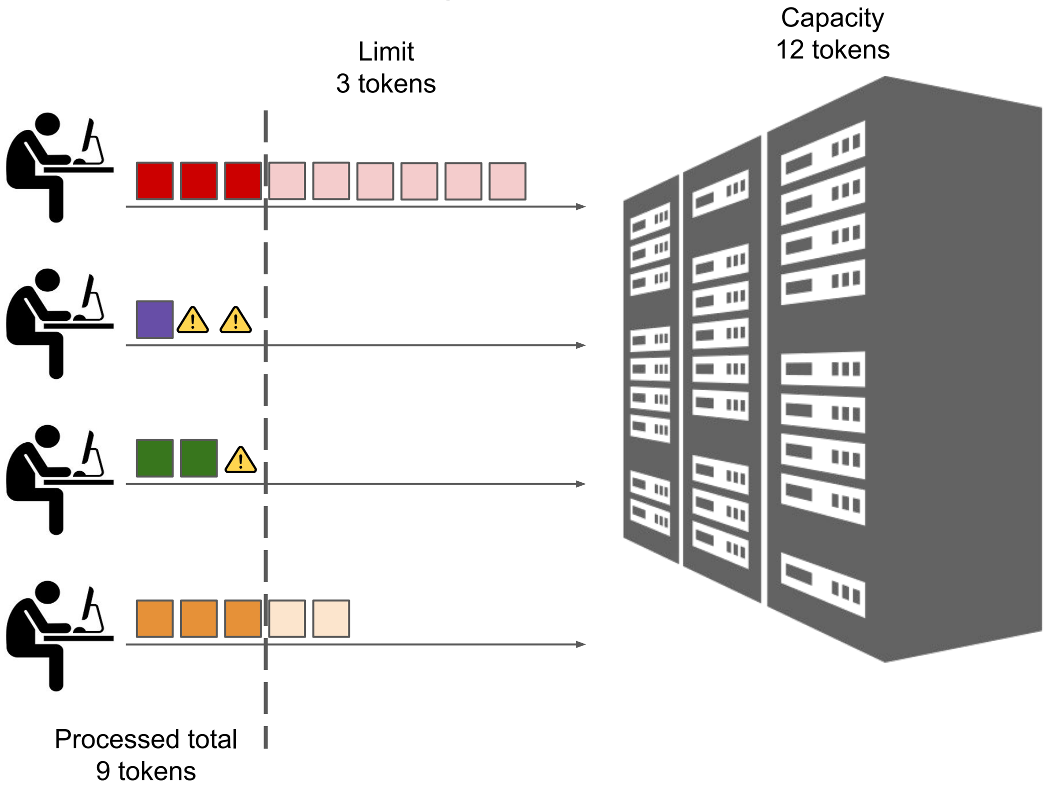
Issues in Serving LLMs



Request-per-minute limit

- ✅ Isolation

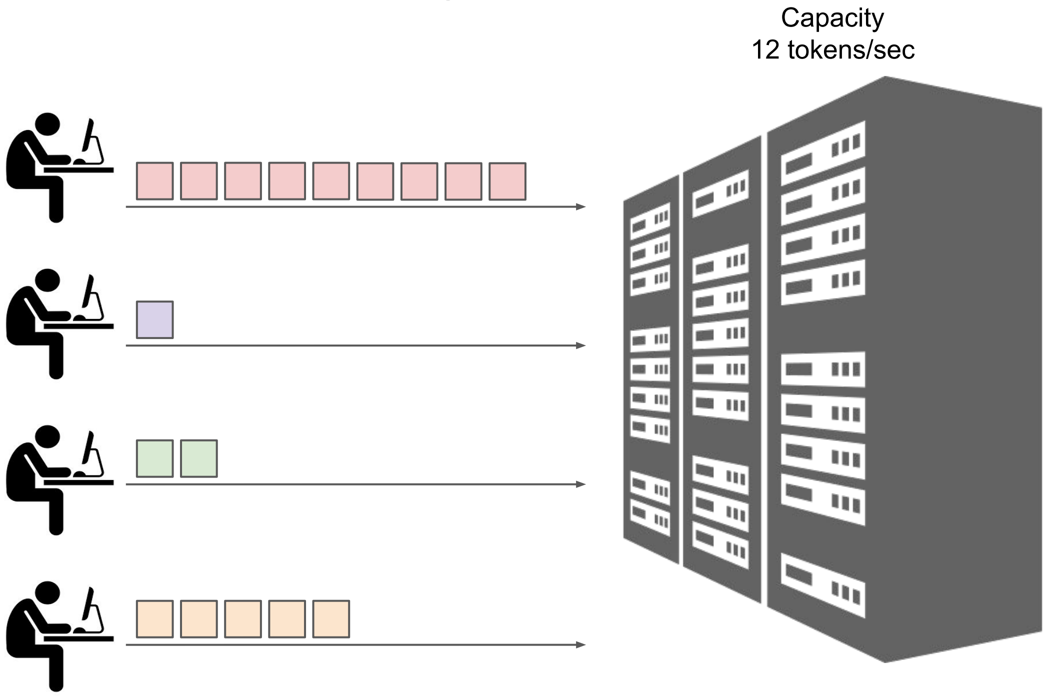
Issues in Serving LLMs



Request-per-minute limit

- ✅ Isolation
- ⚠️ Low resource utilization

Issues in Serving LLMs

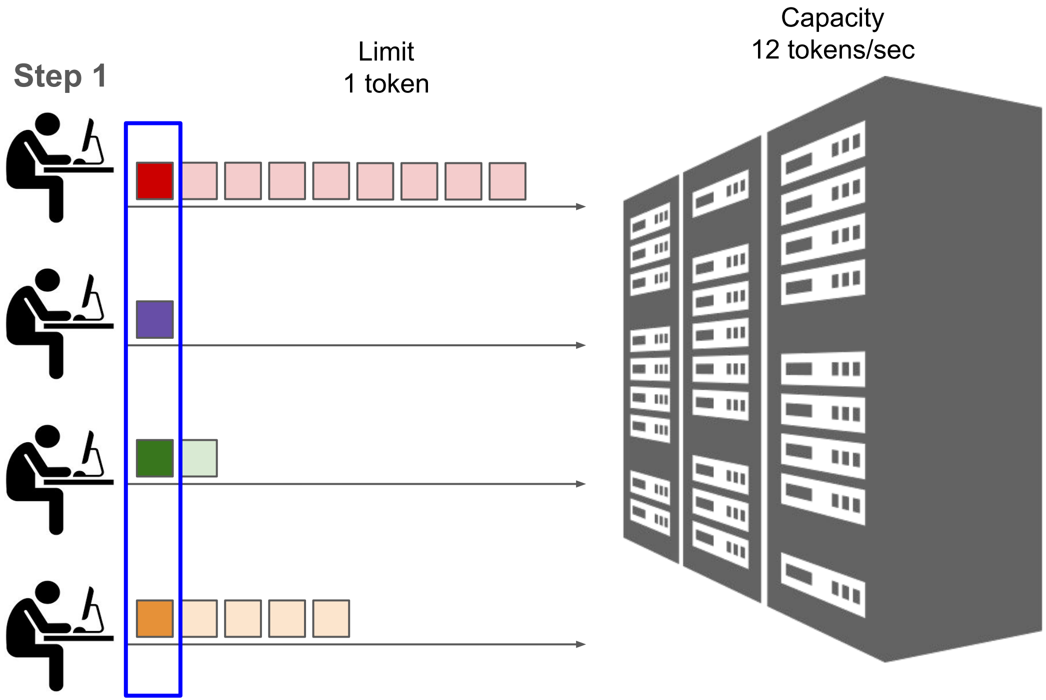


Solution needs to balance

- Isolation
- Resource utilization

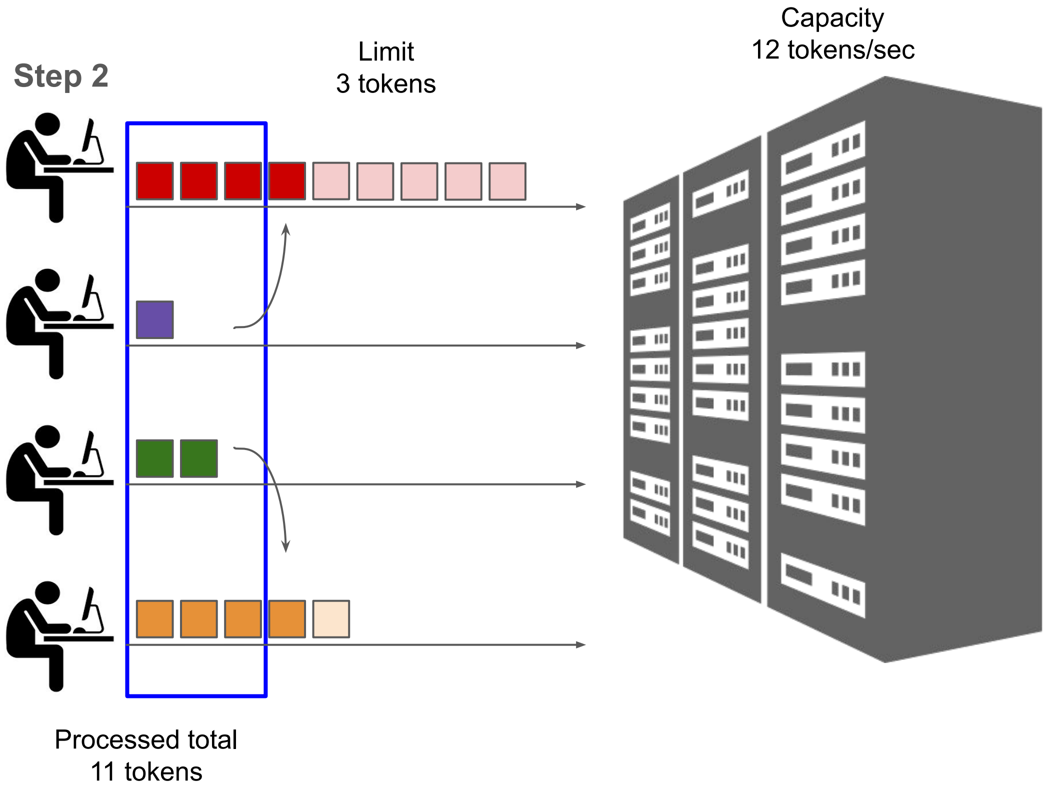
Max-min Fairness

Max-min Fairness



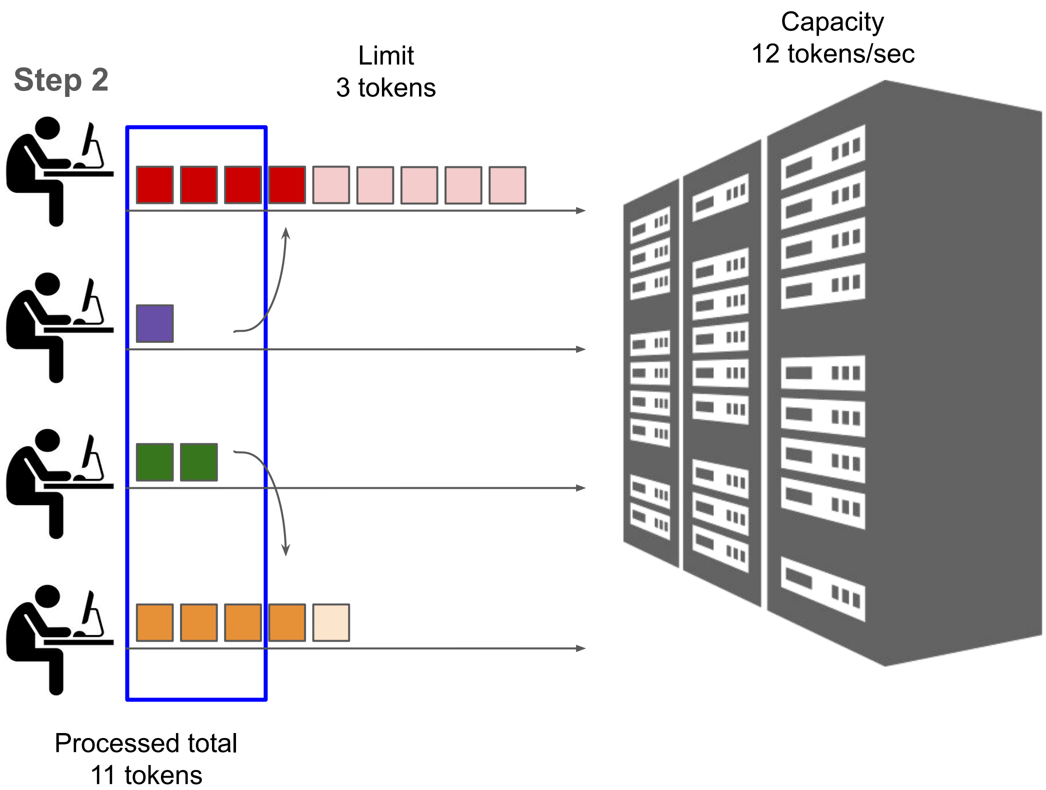
- n clients
- each client gets 1/n of resource

Max-min Fairness



- n clients
- each client gets 1/n of resource
- Surplus resource re-distributed evenly

Max-min Fairness



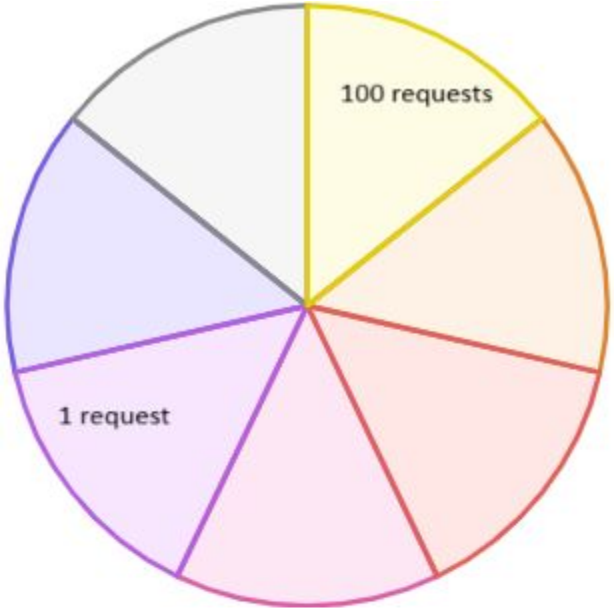
- ➔ Backlogged Clients
- ➔ Non- backlogged Clients
- ➔ Work conservation

Challenges to Measuring Fairness in LLM Serving

Factors	**Networking & CPU scheduling**	**LLM Serving**
Request output length	Known packet size in advance	Unknown in advance
Cost per token	Pre-defined bit or CPU slice	Variable (differs for input and output tokens)
Effective Capacity	Fixed	Varies over time

How do we measure service in LLM Serving?

Achieving Fairness with Virtual Token Counter (VTC)



Measuring Service in LLM Serving

Number of tokens

$$\underbrace{W(t_1, t_2)}_{\text{Service}} = \underbrace{w_p . n_p(t_1, t_2)}_{\text{Prefill phase (input)}} + \underbrace{w_q . n_q(t_1, t_2)}_{\text{Decode phase (output)}}$$

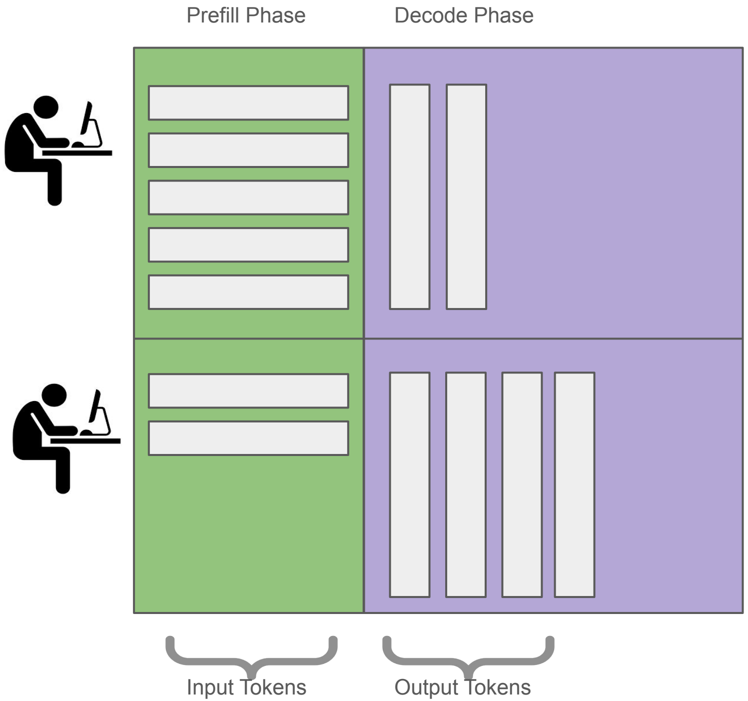
About the algorithm

- VTC tracks service received by each client and gives priority to clients with least number of services received.
- **Counter lift** :- new clients do not start from zero services received, instead they start equal to the minimum service received among the clients already running in the system.

Main points of the algorithm

- Prefill phase tokens and decode phase tokens have different impact on the machine, because prefill tokens can be parallelized.
- Thus services received by a client should be quantified by weighted combination of number of tokens in each phase.

Measuring service



Number of tokens

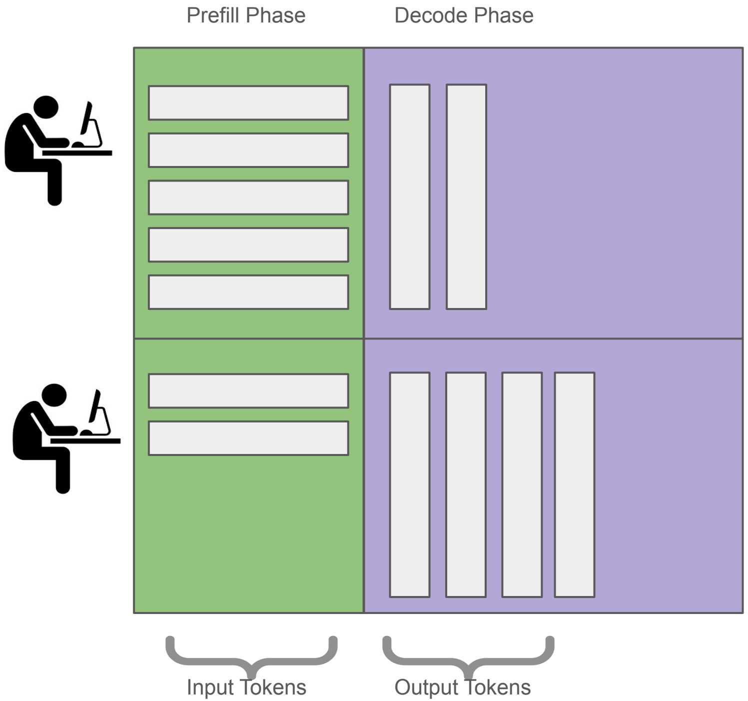
$$\underbrace{W(t_1, t_2)}_{\text{Service}} = \underbrace{n_p(t_1, t_2)}_{\substack{\text{Prefill phase} \\ \text{(input)}}} + \underbrace{n_q(t_1, t_2)}_{\substack{\text{Decode phase} \\ \text{(output)}}}$$

$W_1 = 5 + 2 = 7$

$W_2 = 2 + 4 = 6$

Client 2 is chosen next

Measuring service



Number of tokens

$$W(t_1, t_2) = w_p.\underbrace{n_p(t_1, t_2)}_{\text{Prefill phase (input)}} + w_q.\underbrace{n_q(t_1, t_2)}_{\text{Decode phase (output)}}$$

Service

$W_1 = 5 + 2 = 7$

$W_2 = 2 + 4 = 6$

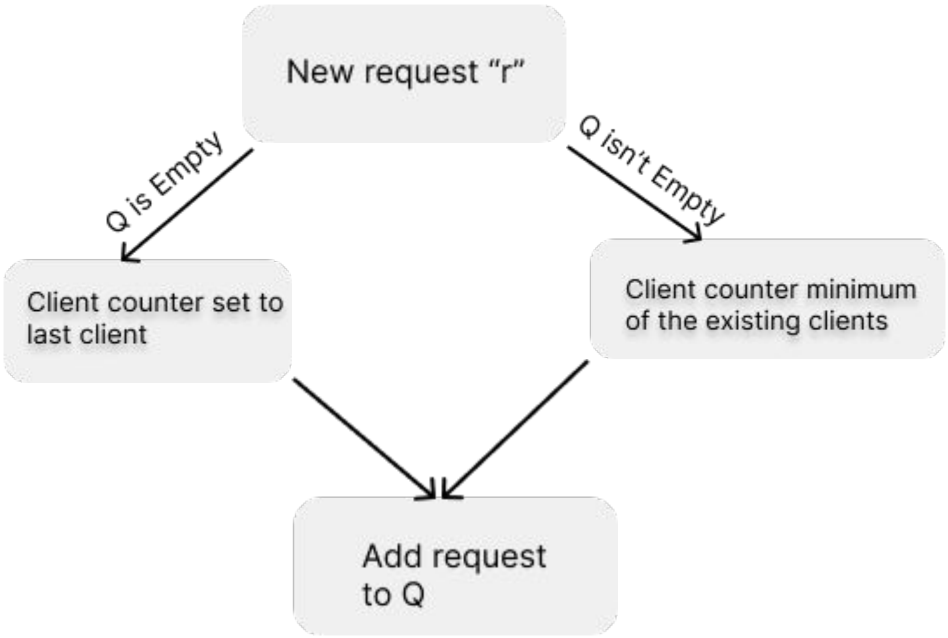
Client 2 is chosen next

$W_1 = 5 * 1 + 2 * 3 = 11$

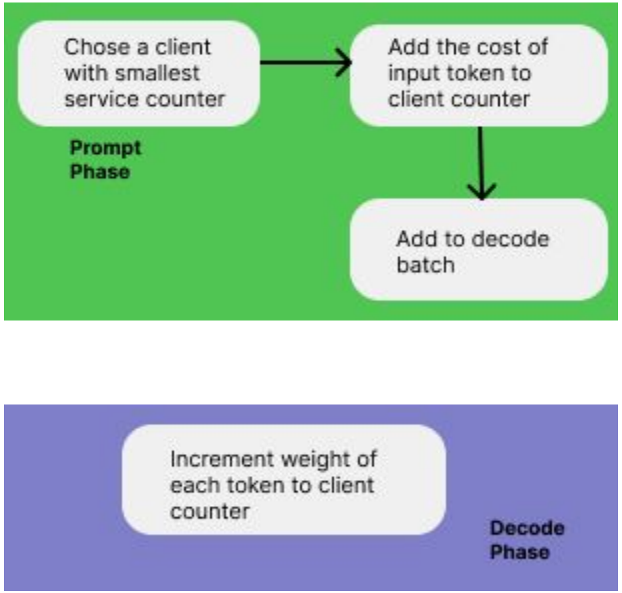
$W_2 = 2 * 1 + 4 * 3 = 14$

Client 1 is chosen next

Monitoring stream



Execution stream



Fairness for non-backlogged clients in VTC

- VTC ensures a client with request rate less than its fair share should get instant service independent of other clients request rate.
- It is mathematically proven the latency of non-backlogged clients is bounded.

Fairness for backlogged clients

- Other fairness algorithms like LCF ensure fair usage but backlogged clients are sometime punished.
- However in VTC for two backlogged clients f and g, The difference in service received by the two clients is always bounded(constant).
- The service counters for backlogged clients are always chasing each other to narrow their difference.

Fairness for backlogged clients

$$|\underbrace{W_f(t_1,t_2)}_{\text{Service received by f}} - \underbrace{W_g(t_1,t_2)}_{\text{Service received by g}}| \leq 2\overbrace{\max(w_p \cdot L_{input}, w_q \cdot M)}^{\text{System parameters}}.$$

Adapt to Different Fairness Criteria

If the service measuring function is changed, VTC can easily accommodate this change.

We just need to swap in the new cost function with their corresponding formulas in vanilla VTC.

Evaluation

Setup

- Implemented in S-Lora: includes implementations of vLLM and continuous batching.
- Fixed parameters so that output token is twice as important as input tokens.
- The service received by client at certain time is calculated in a 1 min window.
- Only about 100 lines of new code.

Metrics

- Difference in service received between two clients.
- Response time (latency).

Base line

- First Come First Serve (FCFS)
- Request per minute (RPM) Limit
- Least Counter First (LCF): Variant of VTC where new clients start with zero service received.

Scenarios

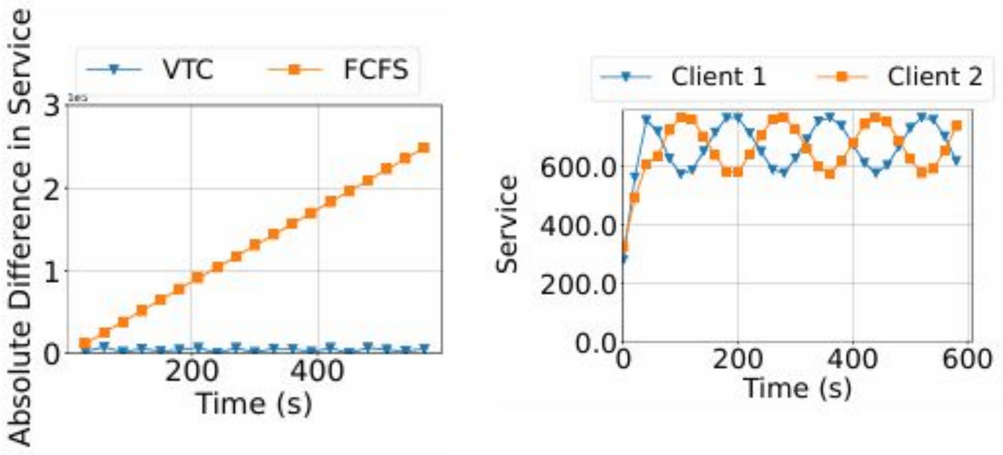
- They run multiple scenarios where each scenarios is different Combinations of overloading and non-overloading clients.
- Combinations are chosen to expose pros of VTC.

Results on Synthetic Workloads

Scenario 1

Client 1 sends its fair share

Client 2 is overloaded



(a) Absolute difference for accumulated service

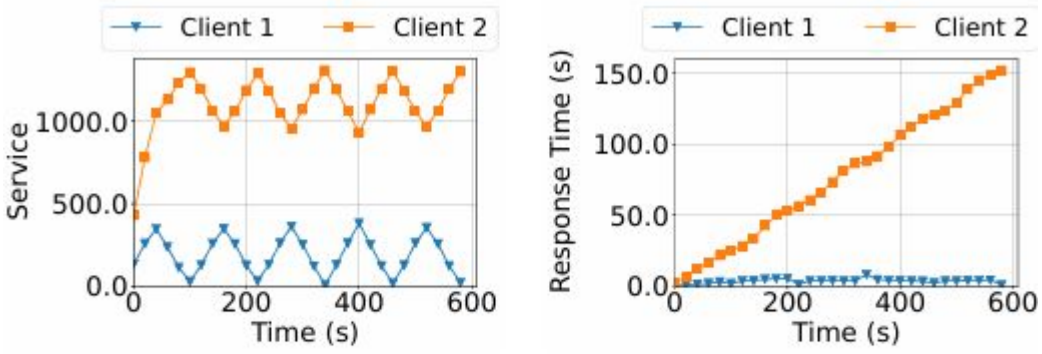
(b) Received service

Results on Synthetic Workloads

Scenario 2

Client 1 is ON/OFF operating with less capacity during ON period.

Client 2 is overloaded always.



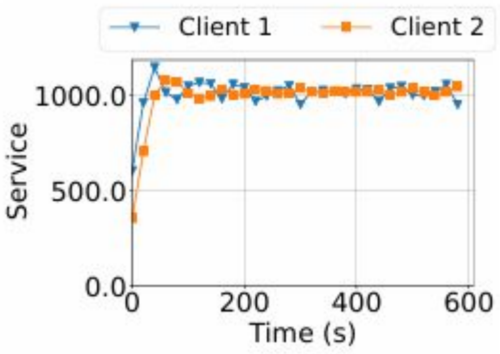
(a) Received service rate (VTC). (b) Response time (VTC).

Results on Synthetic Workloads

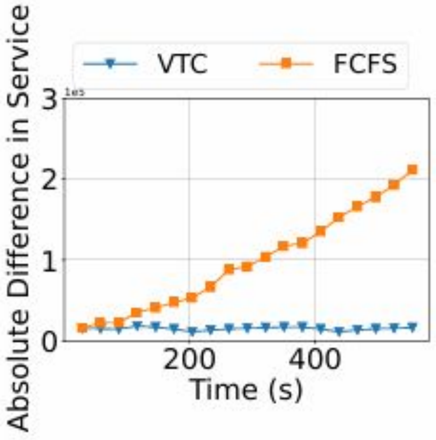
Scenario 3

Client 1, More requests
but less tokens per request.

Client 2 is Less requests
but more tokens per request.



(a) Received service rate (VTC).



(b) Absolute difference for accumulated service.

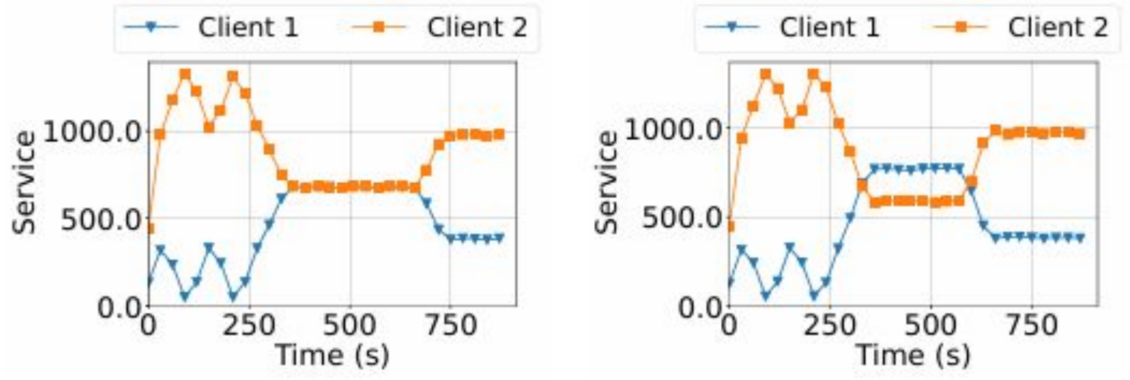
Results on Synthetic Workloads

Scenario 4

Client 1
Low-High-Low request Rate

Client 2
High-Low-High request Rate

In LCF Client 2 is punished.

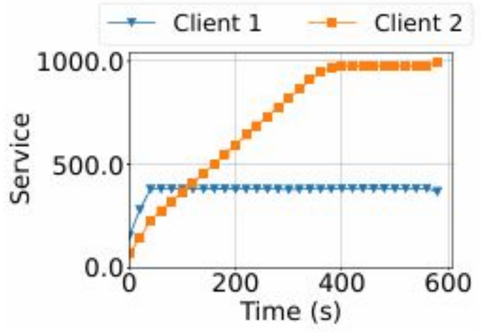


(a) Received service rate (VTC). (b) Received service rate (LCF).

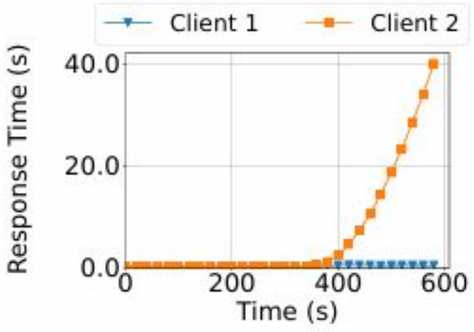
Results on Synthetic Workloads

Scenario 5

Client 2 tries to linearly increase its request rate but it is capped.



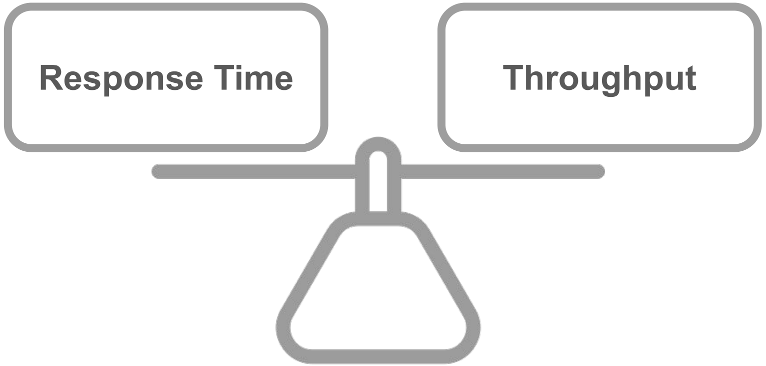
(a) Received service rate (VTC).

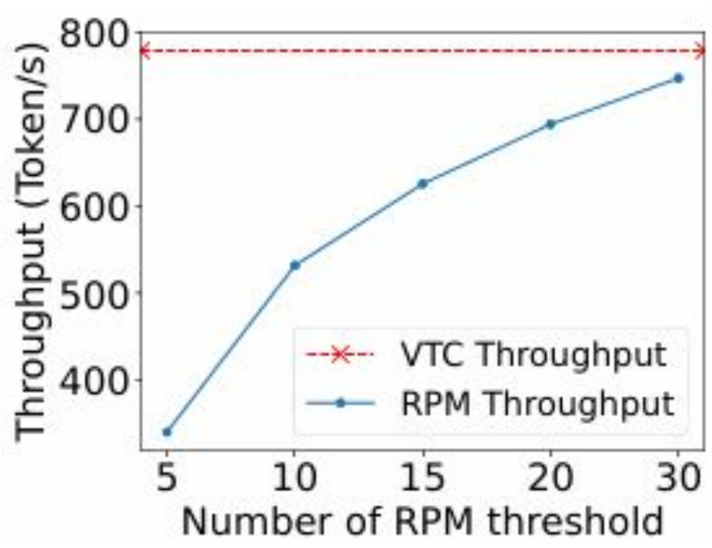


(b) Response time (VTC).

Results on Real Workloads

- A Chatbot Arena was used
- Each LLM is considered a client
- VTC VS RPM

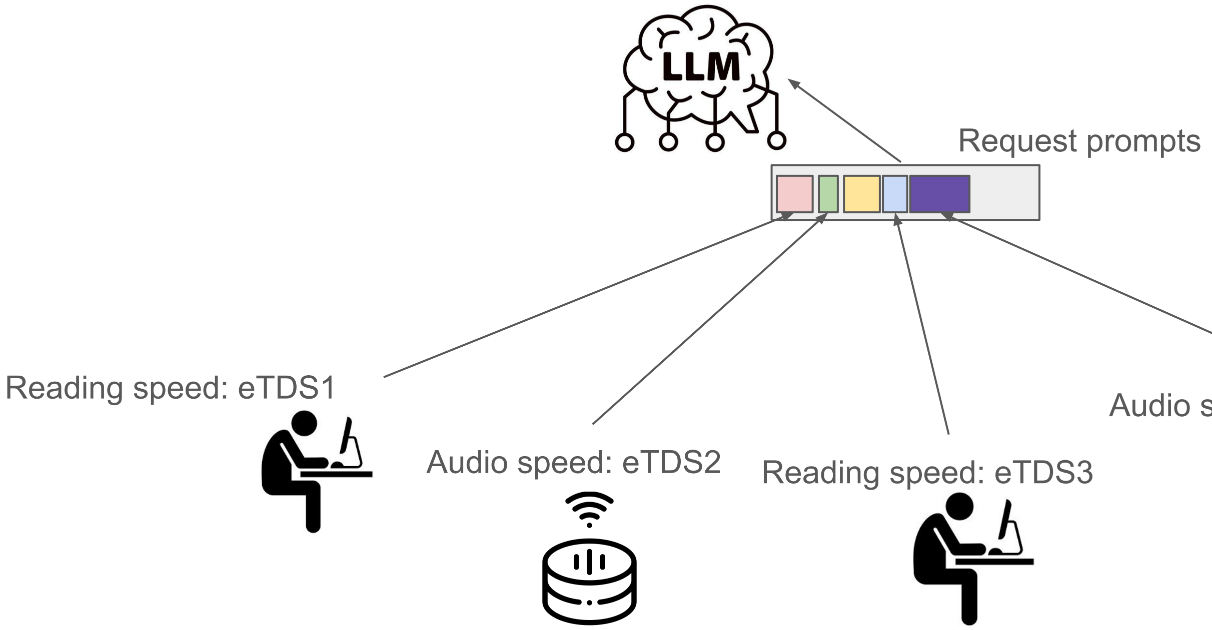




Overall comparison

Scheduler	Avg Diff	Throu
FCFS	433.53	777
LCF	323.82	778
VTC	251.66	779
VTC(predict)	240.33	773
VTC(oracle)	**227.51**	**781**
RPM(5)	83.58	340
RPM(20)	195.71	694
RPM(30)	309.45	747

Thank You!!!



TDS3>TDS2
But that request is
served later

Optimisations

Optimization #1: Selective Triggering

- Triggered only when limited by memory capacity or computation time

Optimization #2: Batch Size Search Space Pruning

- Batch size not too small or too big.

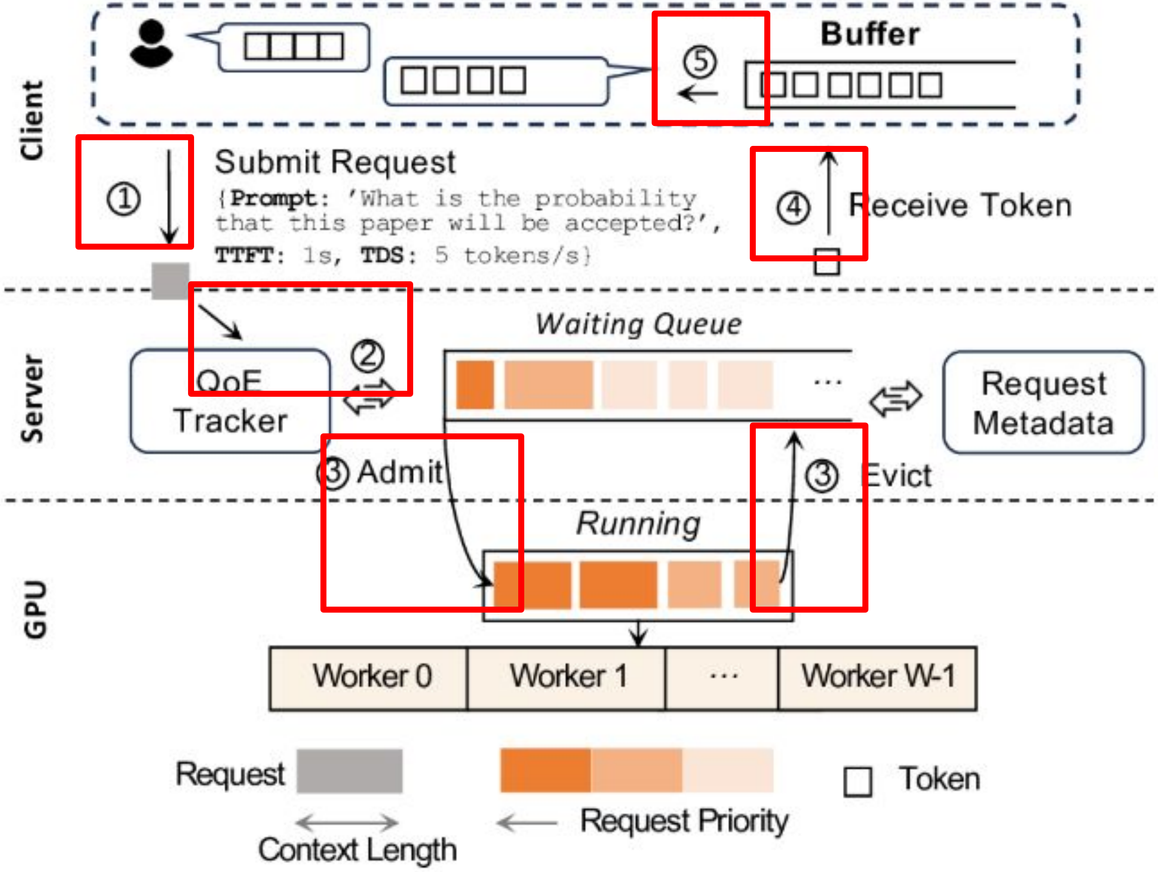
Optimization #3: Greedy Packing for Knapsack

- Most QoE gain

Optimization #4: Preemption Cap

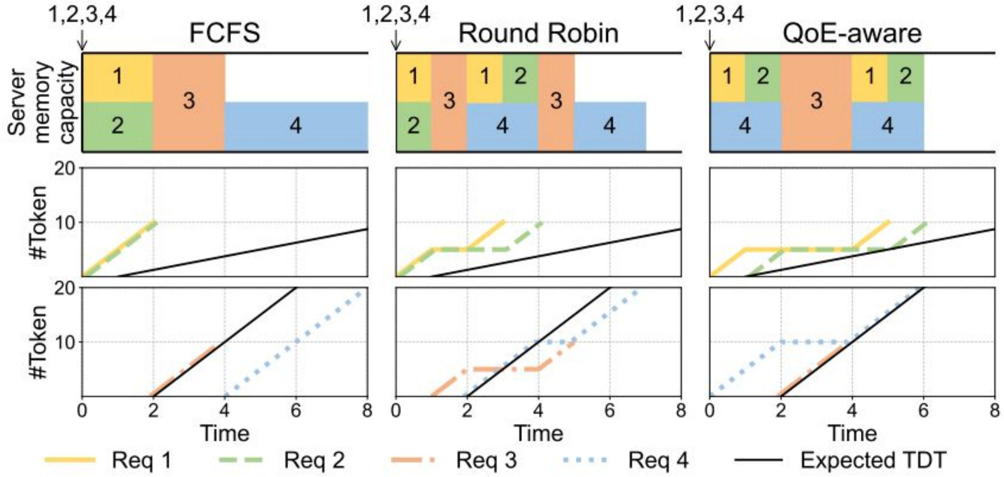
- Prevent thrashing for a request

Solution



Client
Buffer
⑤
Submit Request
{Prompt: 'What is the probability that this paper will be accepted?',
TTFT: 1s, TDS: 5 tokens/s}
①
④ Receive Token
Server
QoE Tracker
②
Waiting Queue
...
Request Metadata
③ Admit
③ Evict
GPU
Running
Worker 0
Worker 1
...
Worker W-1
Request
Context Length
Request Priority
Token

Comparing service for different policies



FCFS causes large delays in TTFT

Round robin is better but choppy QoE

Only QoE aware consistently gives above expected TDT performance for all requests

For a specific batch size B, we would like to solve:

$$
\begin{aligned}
\max_{x} \quad & \sum_{i=1}^{N} \left(Q_{\mathrm{serve},i}(B) - Q_{\mathrm{wait},i}\right) \cdot x_i \\
\text{s.t.} \quad & x_i \in \{0,1\},\ i \in 1, \ldots, N \\
& \sum_{i=1}^{N} x_i = B \\
& \sum_{i=1}^{N} l_i x_i \leq M
\end{aligned}
$$

Monitoring stream

while True **do**
  if new request r from client u arrived **then**
    if not $\exists r' \in Q, client(r') = u$ **then**
      if $Q = \emptyset$ **then**
        let $l \leftarrow$ the last client left Q
        $c_u \leftarrow \max\{c_u, c_l\}$
      else
        $P \leftarrow \{i \mid \exists r' \in Q, client(r') = i\}$
        $c_u \leftarrow \max\{c_u, \min\{c_i \mid i \in P\}\}$
    $Q \leftarrow Q + r$

Execution stream

while True **do**
    if can_add_new_request() **then**
        $B_{new} \leftarrow \emptyset$
        while True **do**
            let $k \leftarrow \arg\min_{i \in \{client(r) | r \in Q\}} c_i$
            let r be the earliest request in Q from k.
            if r cannot fit in the memory **then**
                Break
            $c_k \leftarrow c_k + w_p \cdot input\_length(r)$
            $B_{new} \leftarrow B_{new} + r$
            $Q \leftarrow Q - r$
        forward_prefill(B_{new})
        $B \leftarrow B + B_{new}$
    forward_decode(B)
    $c_i \leftarrow c_i + w_q \cdot |\{r \mid client(r) = i, r \in B\}|$
    $B \leftarrow$ filter_finished_requests(B)

Fairness for non-backlogged clients in VTC

- A client with request rate less than fair share should get instant service independent of other clients request rate.

$$\boxed{D(r_f) - A(r_f) \leq 2 \cdot (n-1) \cdot \frac{\max(w_p \cdot L_{input}, w_q \cdot M)}{a}}$$

Latency is bounded , server capacity

Adapt to Different Fairness Criteria

If the service function W(t1,t2) is changed to $h(n_p^r, n_q^r)$ VTC can easily accommodate this change by.

Just swap in this two formulas with their corresponding formulas in vanilla VTC.

$$\boxed{c_k \leftarrow c_k + h(n_p^r, 0)}$$

$$\boxed{c_i \leftarrow c_i + \sum_{r|client(r)=i, r \in B} \left(h(n_p^r, n_q^r) - h(n_p^r, n_q^r - 1)\right)}$$

Weighted Clients

Clients with different priorities.

$$\left|\frac{W_f(t_1,t_2)}{w_1} - \frac{W_g(t_1,t_2)}{w_2}\right|$$ Is bounded instead of $$|W_f(t_1,t_2) - W_g(t_1,t_2)|$$

Then we just substitute this formula in the plain VTC

$$\boxed{c_i \leftarrow c_i + \frac{\sum_{r|client(r)=i}\left(h(n_p^r, n_q^r) - h(n_p^{r(old)}, n_q^{r(old)})\right)}{w_i}}$$

VTC with Length Prediction

- When a request r is selected, the predicted number of output tokens is added to cost of the request. This is done in the monitoring stream.
- Then During the actual decoding process, adjustments are made.
- This opens new set of problems about the prediction process.

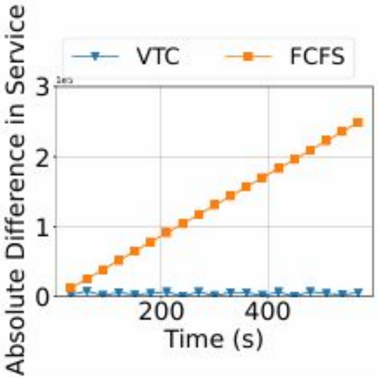
Metrics

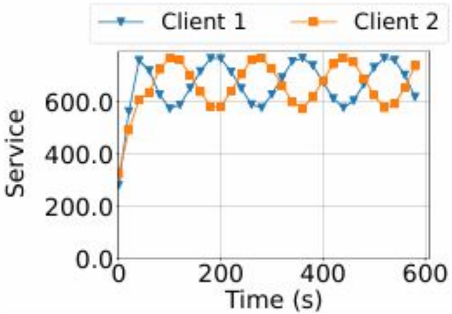
- $w_p = 1$ and $w_q = 2$.
- The service received by client i at time t = $W_i(t - T, t + T)$, T = 30s
- The absolute difference in service between clients is $|W_i(0,t) - W_j(0,t)|$.
- Latency at time t is measured in the bound t-T and t+T, T=30s

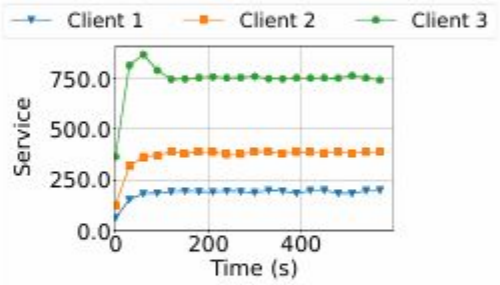
Results on Synthetic Workloads

Client 1 sends its fair share

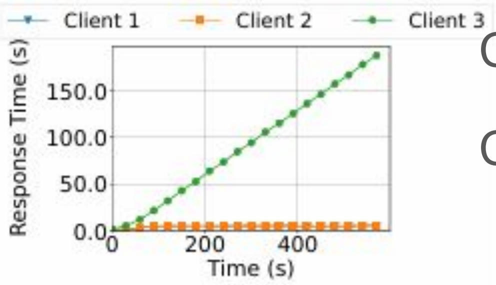
Client 2 is overloaded







(a) Received service rate (VTC).



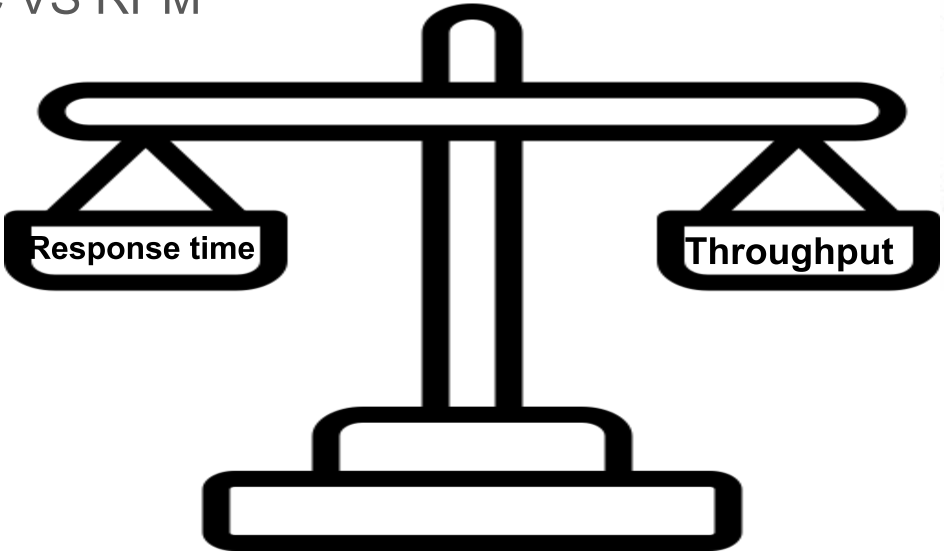
(b) Response time (VTC).

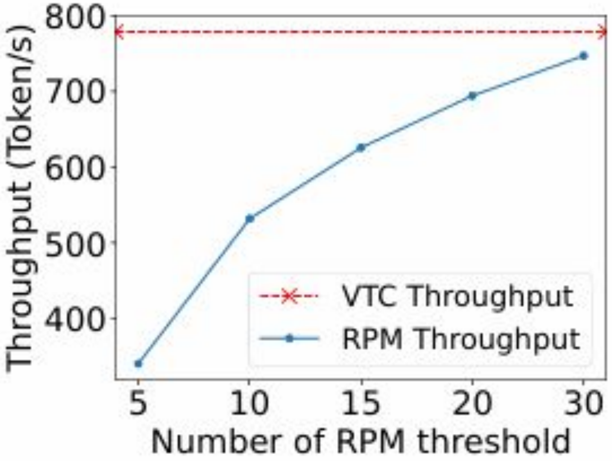
Client 3 is overloaded

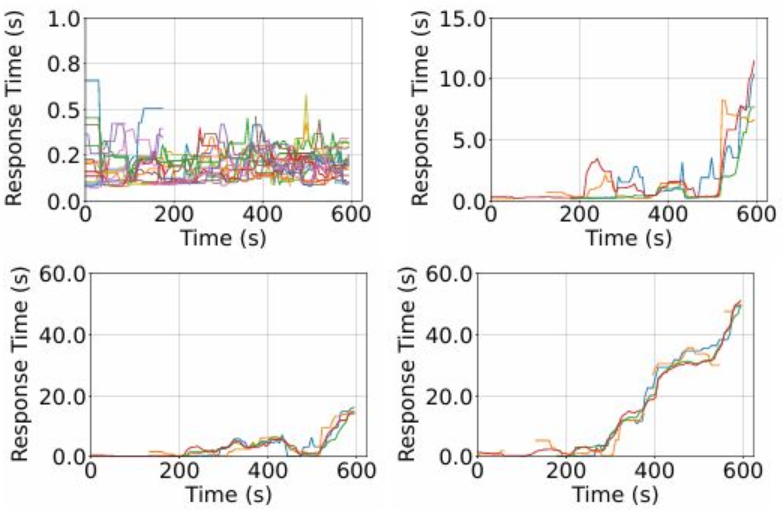
Clients 1 and 2 are underloaded

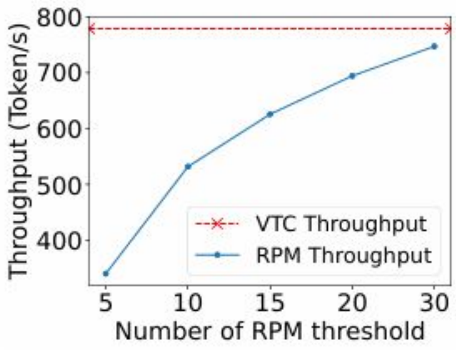
Results on Real Workloads

- A Chatbot Arena was used
- Each LLM is considered a client
- 27 clients are chosen and 4 of which are studied
- VTC VS RPM









(5-20 rpm) The greater the rpm the greater the response time but the greater the throughput.

Ablation Study

- Absolute difference in service is affected by
  - Memory size (KV size)
  - Request lengths

Discussion Topics

- What are the cons of VTC

- Is the overhead of VTC reasonable

- How can VTC be improved.