SGLang FLPM

Yichuan Wang<yichuan_wang@berkeley.edu> & LMSYS
Arxiv :https://arxiv.org/pdf/2501.14312

Locality-aware Fair Scheduling in LLM Serving

Shiyi Cao*[1 2], **Yichuan Wang*[1 2]**, Ziming Mao[1], Pin-Lun Hsu[2], Liangsheng Yin[1 2], Tian Xia[1], Dacheng Li[1 2], Shu Liu[1], Yineng Zhang[2], Yang Zhou[1], Ying Sheng[2], Joseph E. Gonzalez[1], Ion Stoica[1]

[1]UC Berkeley [2]LMSYS *indicates equal contribution.

Outlines

1. Background and motivation

2. DLPM and its distribute version D2LPM

3. Evaluation

4. Future Work

➔ 1. Background and motivation

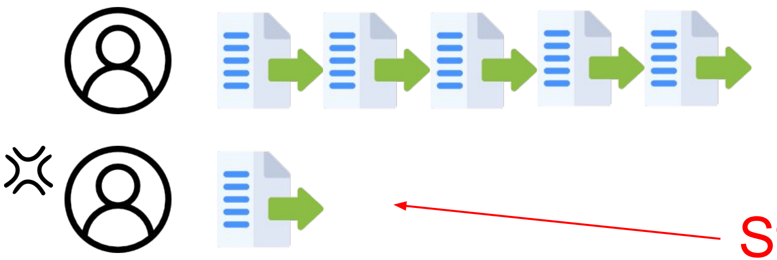
Background:

1. What is fairness in LLM serving?

- One server serves multiple users concurrently.
- The server's capacity is limited.

Overload User

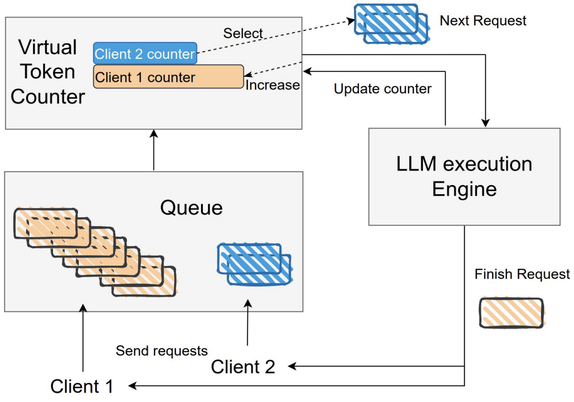
Underload User



Starvation

*Fairness in Serving Large Language Models

VTC scheduler(Done by Ying)



Simple algorithm:

- ❑ Virtual counter for each user
- ❑ Update and schedule at token granularity
- ❑ Update according to the cost per token

Virtual counters:

100

80 ⟵ New request

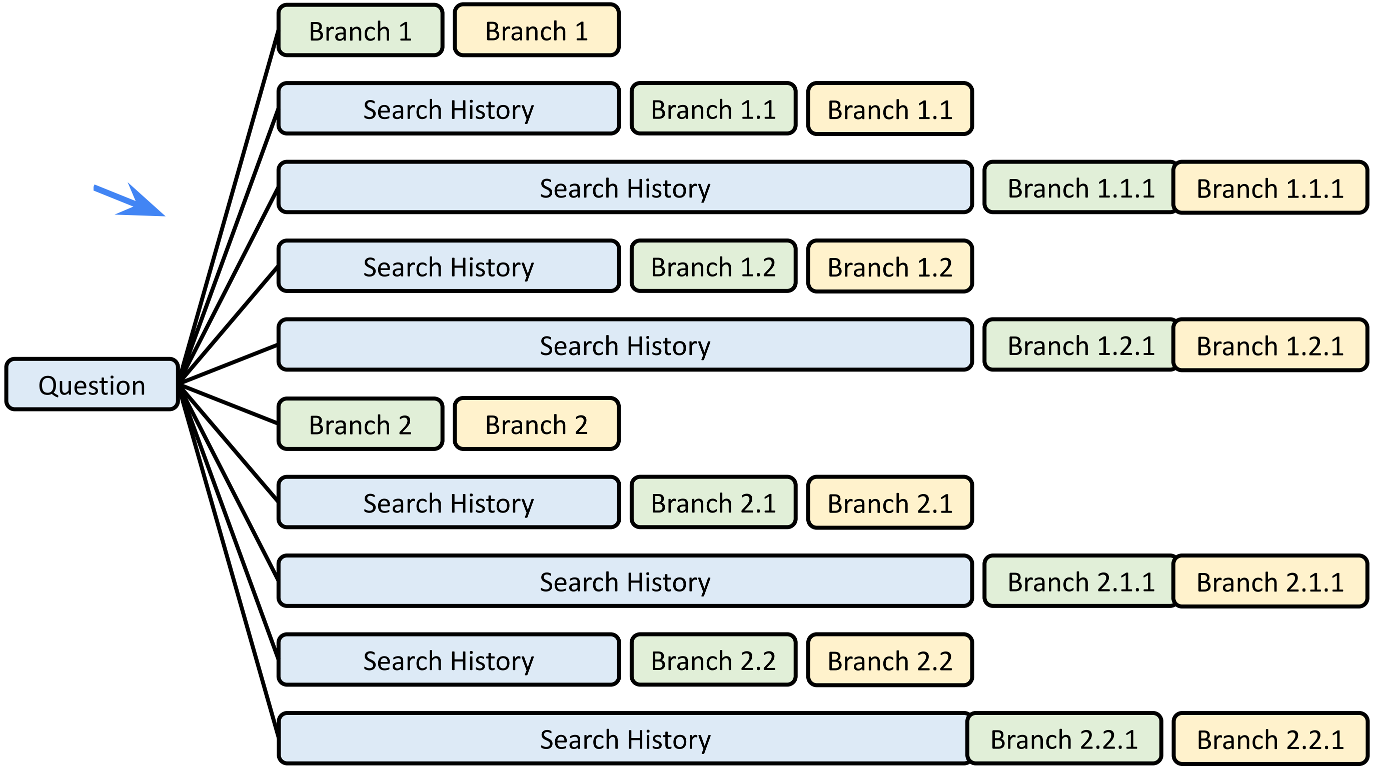
120

*Fairness in Serving Large Language Models

Background:
2. Structure in LLM Calls and Prefix Cache

Prefix Cache

Tree of Thought



*SGLang: Efficient Execution of Structured Language Model Programs

Background:

Cache-aware schedule: LPM (Longest Prefix Match)

Even more severe fairness issue

*SGLang: Efficient Execution of Structured Language Model Programs

Motivation:

LPM: Prioritize locality but ignore fairness

VTC: Ensure fairness but ignore efficiency

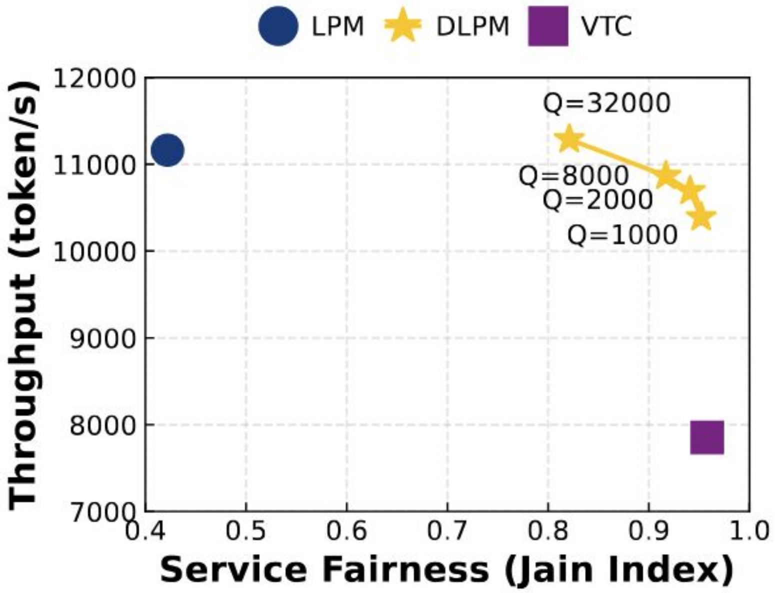
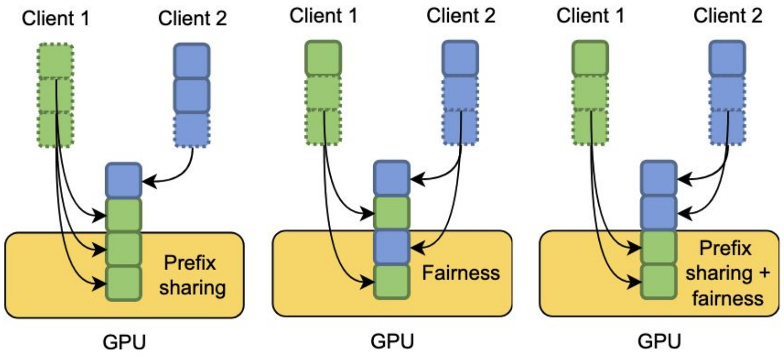


Figure 1: DLPM achieves a *new Pareto frontier* considering locality and fairness in LLM serving. Q is a hyper-parameter in DLPM, indicating how much we relax the fairness bound of DLPM. Results are obtained in a single A10 GPU.

* Jain fairness index: $J(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$

➔ 2. DLPM and its distribute version D2LPM

➔ DLPM

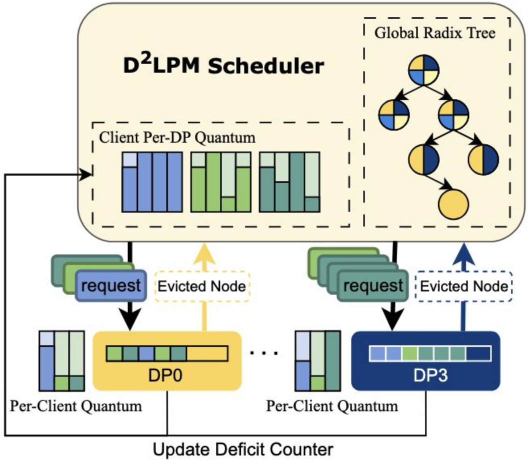


(a) LPM. (b) VTC. (c) DLPM (ours).

Algorithm 1 Deficit Longest Prefix Match (DLPM)

let l denotes the client list
let B denotes current running batch
function CHECKREFILL(l, $Queue$)
    for all $i \in \{client(r) \mid r \in Queue\}$ **do**
        if $q_i > 0$ **then return**
    for all $i \in l$ **do**
        if $q_i \leq 0$ **then** $q_i \leftarrow q_i + Q^u$
end function
▷ `with monitoring stream:`
while True **do**
    if new request r from client i arrived **then**
        if $i \notin l$ **then** $q_i \leftarrow 0, \ l \leftarrow l + u$
        $Queue \leftarrow Queue + r$
▷ `with execution stream 1:`
while True **do**
    $Queue \leftarrow$ SORTBYPREFIX($Queue$)
    while not $Queue.empty()$ **do**
        for each $r \in Queue$ **do**
            $i \leftarrow client(r)$
            if $q_i \leq 0$ **then** CHECKREFILL(l, $Queue$)
            if $q_i > 0$ and CANADD(r) **then**
                $B \leftarrow B + r$
                $q_i \leftarrow q_i - w_e \cdot extend\_length(r)$
                $Queue \leftarrow Queue - r$
    FORWARDSTEP(B)
    $q_i \leftarrow q_i - w_q \cdot |\{r|client(r) = i, r \in B\}|$
    $B \leftarrow$ filter_finished_requests(B)

➔ D2LPM



Algorithm 2 D^{2}LPM Scheduling

let s_w denotes the current queue size of worker w.
$W \leftarrow$ GetWorkers(), $R \leftarrow$ InitRadixTree($|W|$)
function SelectWorker(G, i)
  $G_{avail} \leftarrow \{w \mid q_{i,w} > 0\}$
  while $G_{avail} == \emptyset$ **do**
    for all $w \in W$ **do** $q_{i,w} \leftarrow q_{i,w} + Q^w$
  $G_{cand} \leftarrow G \cap G_{avail}$
  if $G_{cand} == \emptyset$ **then return** $\arg\min_{w \in G_{avail}} s_w$
  return $\arg\min_{w \in G_{cand}} s_w$
end function
▷ `with concurrent stream 1:`
while True **do**
  if new request r from client i arrived **then**
    $G \leftarrow$ R.LongestMatchWorkers(r)
    $w \leftarrow$ SelectWorker(G, $client(r)$)
    Dispatch(w, r)
    $q_{i,w} \leftarrow q_{i,w} - w_e \cdot r$.input_tokens
    $s_w \leftarrow s_w + 1$
    R.Insert(r.input_tokens, w)
▷ `with concurrent stream 2:`
while True **do**
  if request r from client i has finished at worker w **then**
    $q_{i,w} \leftarrow q_{i,w}$ - $w_q \cdot r$.output_tokens
    $s_w \leftarrow s_w - 1$
▷ `with concurrent stream 3:`
while True **do**
  if prefix P has been evicted at worker w **then**
    R.Evict(P, w)

➔ Theoretical Fairness Bound

- Single GPU:

  $$|W_f(t_1, t_2) - W_g(t_1, t_2)| \le 2 \cdot (U + Q^u), \text{ where } U = w_e \cdot L_{input} + w_q$$

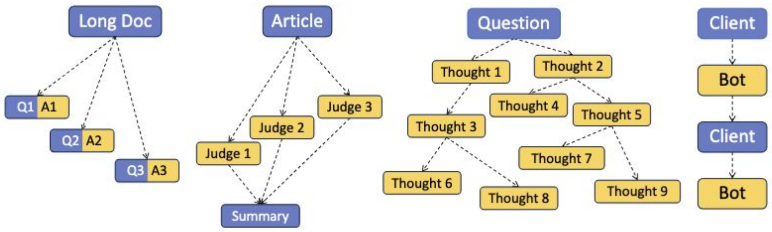
- Multiple GPU

  $$|W_f(t_1, t_2) - W_g(t_1, t_2)| \le 2 \cdot |W| \cdot (U + Q^u)$$

➔ 3. Evaluation

➔ Evaluation Setup:

◆ Workload



(a) Long-Context QA (b) LLM-as-a-Judge (c) Tree-of-Thoughts (d) Multi-Turn Chat

◆ Model

- Llama-3.2-3B on A100 80G
- Llama-3.1-8B on A10G

◆ Eval Config

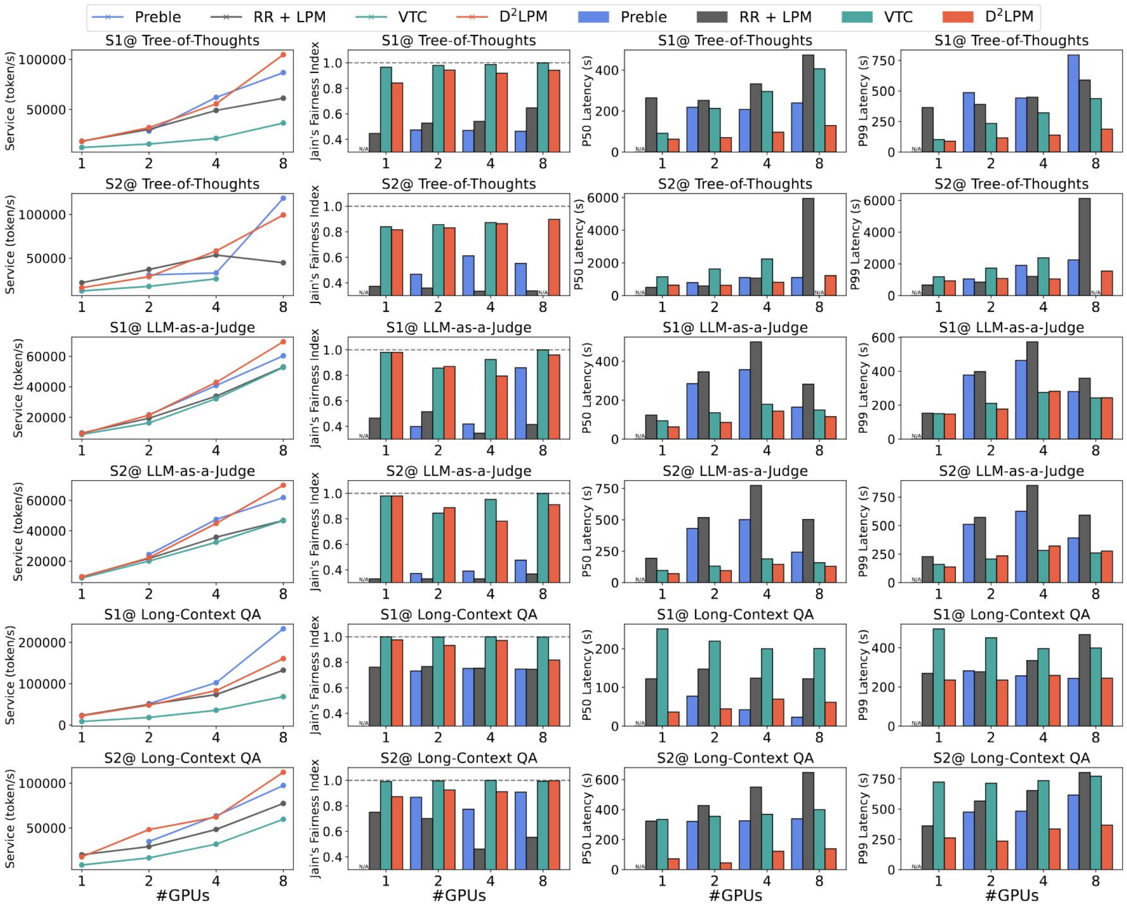
Workload	Detailed Behavior
S1: More Requests	
Long-Context QA	👎: Higher req rate
Tree-of-Thoughts	👎: Trees of 4 branches (340 req per tree) 👍: Trees of 2 branches (30 req per tree)
LLM-as-Judge	👎: Evaluation with 16 dimensions 👍: Evaluation with 2 dimensions
S2: Longer Prefix	
Long-Context QA	👎: 2× longer input documents
Tree-of-Thoughts	👎: 10× longer input questions
LLM-as-Judge	👎: Extra 600 tokens before each article

◆ Existing Scheduler

- VTC
- LPM (Round Robin+LPM in DP setting)
- Preble (locality aware router)

➔ **Main evaluation:**

- **Throughput**
- **Fairness**
- **Victim User Latency**



➔ **Case Study:**

◆ **4 GPU data parallelism**

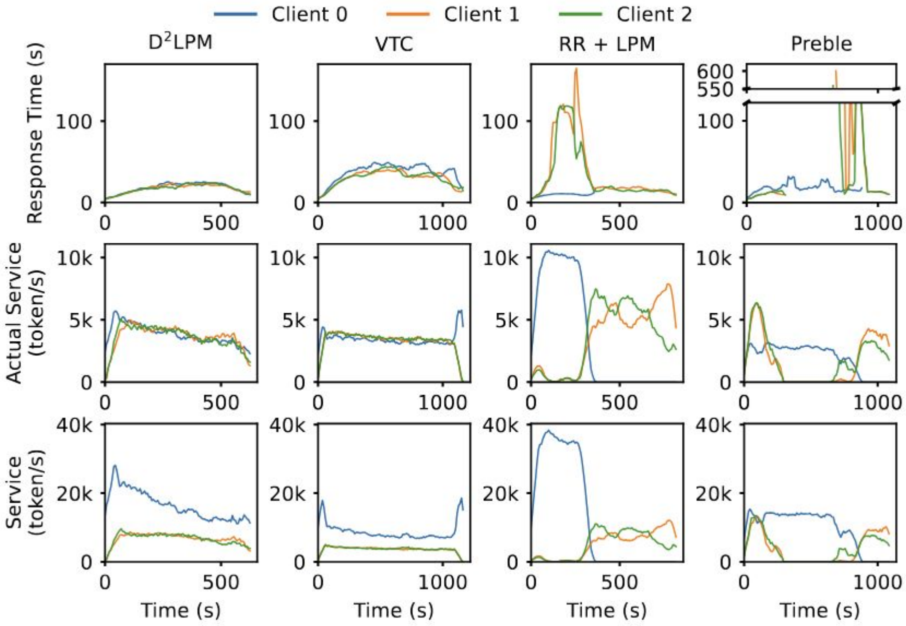
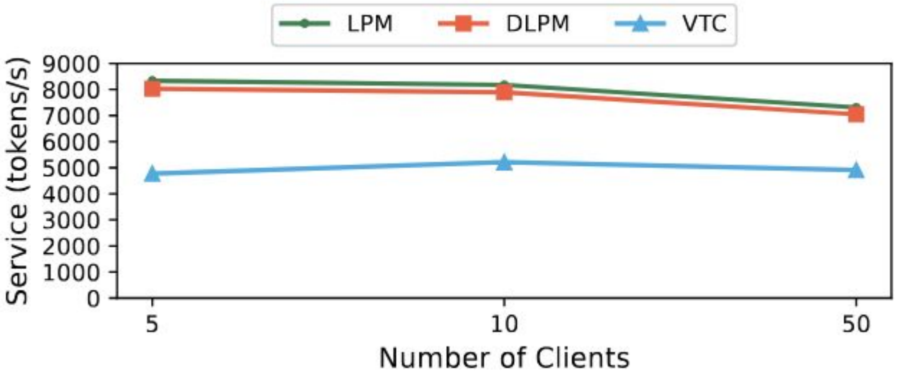
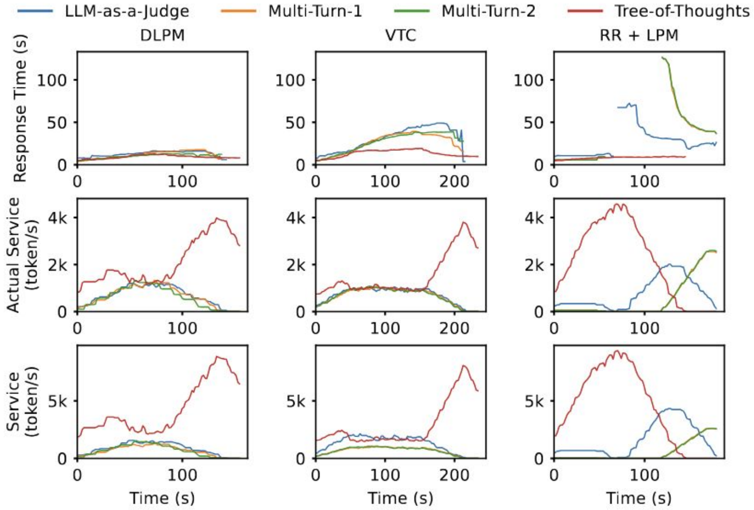


Figure 9: Fairness and performance visualization of different schedulers on Tree-of-Thoughts workloads with $D = 4$ (3B model + 4 A10G GPUs). The maximum value on the X-axis represents the end-to-end completion time for each scheduler. The actual service is calculated using the cost function defined in §3, which is a weighted sum of the number of extend tokens and the number of output tokens.

➔ **Scalability:**



➔ **Mix of Workload:**



➔ 4. Conclusion and Future Work

➔ Conclusion

- We introduces the first locality-aware fair scheduling algorithm, Deficit Longest Prefix Match (DLPM), which can maintain a high degree of prefix locality with a fairness guarantee. And we can easily extend it to distribute version.

➔ Future Work

- Gradually upstream to SGLang codebase
- More follow up directions:
  - Share Prefix among users
  - Within user request level fairness guarantee
  - Fairness issue when considering data dependency in LLM workload

Thank U

Welcome to join our Slack and use SGLang!