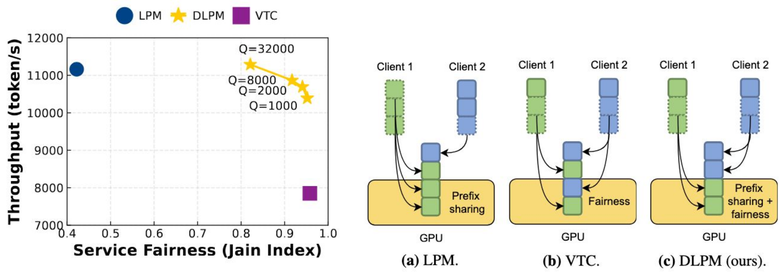
Locality-aware Fair Scheduling in LLM Serving

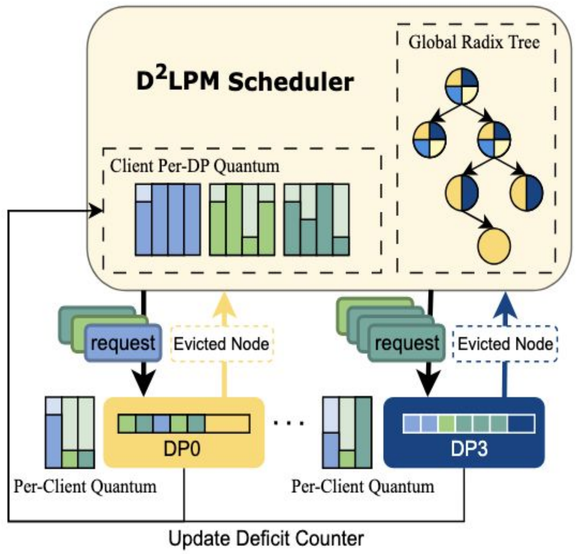
Shiyi Cao*[1 2], Yichuan Wang*[1 2], Ziming Mao[1], Pin-Lun Hsu[2], Liangsheng Yin[1 2], Tian Xia[1], Dacheng Li[1 2], Shu Liu[1], Yineng Zhang[2], Yang Zhou[1], Ying Sheng[2], Joseph E. Gonzalez[1], Ion Stoica[1]
[1]UC Berkeley [2]LMSYS *indicates equal contribution.

Background & Motivation

- **Background**: Achieving efficient online LLM inference with SLO guarantees necessitates **isolation** among **different clients** (introduced in VTC OSDI24)
- However, existing LLM serving scheduler does not carefully consider both **fairness** and **locality** (in addition, **load balance** in distributed scenario).
- LPM(longest prefix match) prioritizes locality but will result in severe fairness issue, VTC (virtual token counter) ensures fairness but leads to poor performance



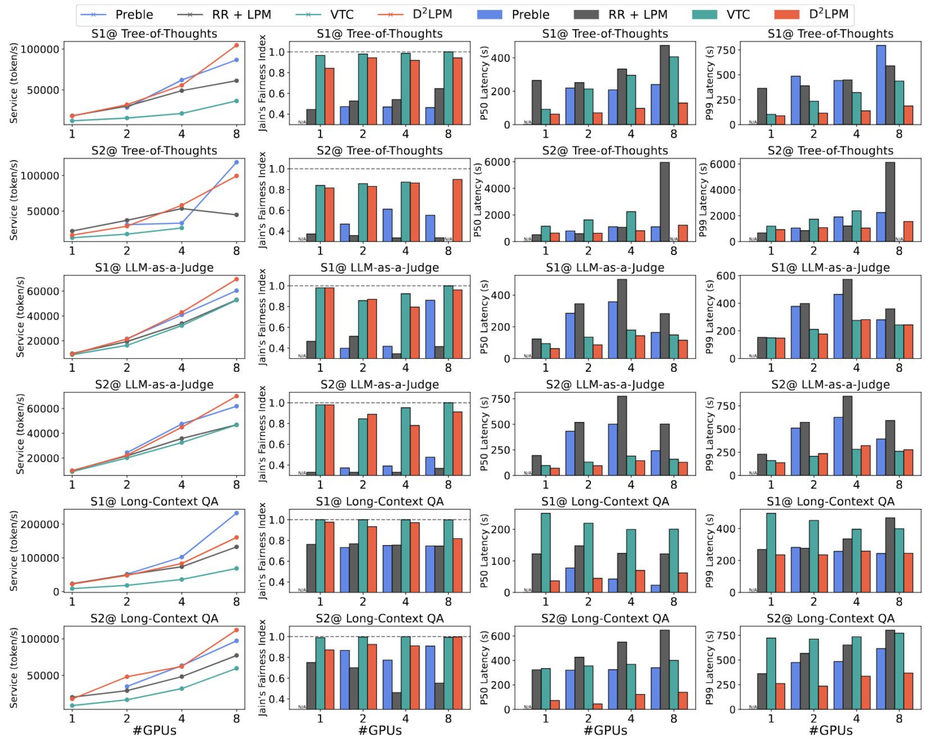
2. D^{2}LPM for Distributed scenario:



3. Strict fairness bound

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

2. Results:



* S1 means malicious client using high request rate, S2 means malicious client using longer prefix.

Main Algorithms

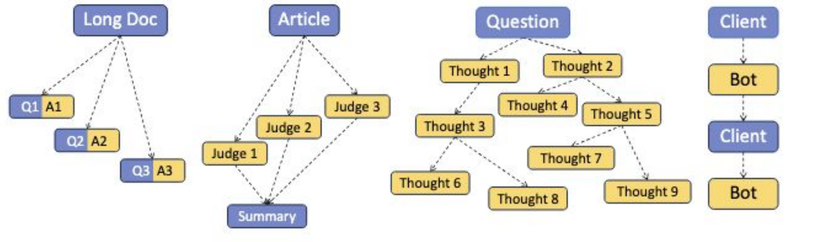
1. DLPM for Single GPU:

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 ∈ {client(r) | r ∈ Queue} do
        if q_i > 0 then return
    for all i ∈ l do
        if q_i ≤ 0 then q_i ← q_i + Q^u
end function
▷ with monitoring stream:
while True do
    if new request r from client i arrived then
        if i ∉ l then q_i ← 0, l ← l + u
        Queue ← Queue + r
▷ with execution stream 1:
while True do
    Queue ← SortByPrefix(Queue)
    while not Queue.empty() do
        for each r ∈ Queue do
            i ← client(r)
            if q_i ≤ 0 then CheckRefill(l, Queue)
            if q_i > 0 and CanAdd(r) then
                B ← B + r
                q_i ← q_i − w_e · extend_length(r)
                Queue ← Queue − r
    ForwardStep(B)
    q_i ← q_i − w_q · |{r|client(r) = i, r ∈ B}|
    B ← filter_finished_requests(B)
```

Evaluation

1. Workload:



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

3. Case study: