Appendix for the Greedy Scheduling

1 Greedy-based Solution

To address such an NP-hard problem, the request manager in Apt-Serve uses a greedy-based approximate solution. Its core idea is to allocate memory occupation iteratively to candidate requests based on their marginal gain. Specifically, this involves continuously assigning additional memory usage to the request that provides the highest marginal gain per unit of memory consumption, thereby progressively approaching the optimal solution. For simplicity, we omit the superscript e denoting the iteration in the following illustration.

For a given request i, denote its current memory usage as $\overline{\mathrm{m}}_i$, if $\overline{\mathrm{m}}_i = 0$, a marginal memory usage increase $\Delta \mathrm{m}_i$ to it involves assigning half of its maximum memory requirement $\frac{\mathrm{m}_i}{2}$, indicating that request i is scheduled using hidden cache. Similarly, if $\overline{\mathrm{m}}_i = \frac{\mathrm{m}_i}{2}$, the marginal memory usage increase $\Delta \mathrm{m}_i$ involves assigning the other half of its maximum memory requirement, indicating that request i is now scheduled using KV cache, given that it was previously scheduled with hidden cache. If $\overline{\mathrm{m}}_i = \mathrm{m}_i$, assigning further memory does not provide any additional gain in value. Therefore, for a given request i, its marginal gain in value θ_i based on its current memory usage $\overline{\mathrm{m}}_i$, is as follows:

$$\theta_i = \begin{cases} \frac{(\mathrm{p}_i - (|\mathrm{W}|+|\mathrm{R}|)\alpha \mathrm{m}_i) - 0}{\mathrm{m}_i/2 - 0} = \frac{2\mathrm{p}_i}{\mathrm{m}_i} - 2(|\mathrm{W}|+|\mathrm{R}|)\alpha, & if\ \ \overline{\mathrm{m}}_i = 0; \\ \frac{\mathrm{p}_i - (\mathrm{p}_i - (|\mathrm{W}|+|\mathrm{R}|)\alpha \mathrm{m}_i)}{m_i - m_i/2} = 2(|\mathrm{W}|+|\mathrm{R}|)\alpha, & if\ \ \overline{\mathrm{m}}_i = \frac{\mathrm{m}_i}{2}; \\ 0, & if\ \ \overline{\mathrm{m}}_i = \mathrm{m}_i. \end{cases}$$

Note that for a given request i to be scheduled, there may be cases where its hidden cache usage imposes too much of a negative impact on other requests. This is evident when the marginal gain to the overall scheduling value from allocating just enough memory to hold its hidden cache (from $\overline{\mathrm{m}}_i = 0$ to $\overline{\mathrm{m}}_i = \frac{\mathrm{m}_i}{2}$) is smaller than which by directly assigning the exact memory space to hold its KV cache (from $\overline{m}_i = 0$ to $\overline{m}_i = m_i$). In such cases, its hidden cache usage is avoided, thus its marginal gain is refined as follows:

$$\theta_i = \begin{cases} \frac{\mathrm{p}_i - 0}{\mathrm{m}_i - 0} = \frac{\mathrm{p}_i}{\mathrm{m}_i}, & if\ \ \overline{\mathrm{m}}_i = 0; \\ 0, & if\ \ \overline{\mathrm{m}}_i = \mathrm{m}_i. \end{cases}$$

Assume the total number of candidate schedules (how much marginal memory usage increase to which requests) is n, all the possible marginal gain $\theta_1, \theta_2, \ldots, \theta_n$

Algorithm 1 Greedy-based Scheduling by Apt-Serve.

Input: The candidate request set U; memory constraint M; the candidate schedule set Υ (Eq. 1).

Output: Final scheduling decisions S (Eq. 2).

Initialize scheduling decisions $S \leftarrow \{(x_i, y_i) = (0, 0) | \forall i \in U\}$;
Initialize the current memory usage $\overline{\mathrm{M}} \leftarrow \{\overline{\mathrm{m}}_i = 0 | \forall i \in \mathrm{U}\}$;
Initialize the overall scheduling values $\mathrm{V} \leftarrow 0$;
Sort Υ based on the marginal gain in a decreasing order and derive $\hat{\Upsilon} \leftarrow \{(\hat{\theta}_j, \hat{\mathrm{r}}_j, \Delta\hat{\mathrm{m}}_j, \hat{\mathrm{m}}_{\hat{\mathrm{r}}_j}) | j = 1, 2, ..., n\}$;
for $j = 1, 2, ..., n$ **do**
  if $sum(\overline{\mathrm{M}}) + \Delta\hat{\mathrm{m}}_j \leq \mathrm{M}$ **then**
    $\overline{\mathrm{m}}_{\hat{\mathrm{r}}_j} \leftarrow \overline{\mathrm{m}}_{\hat{\mathrm{r}}_j} + \Delta\hat{\mathrm{m}}_j$ ▷ Update memory usage for $\hat{\mathrm{r}}_j$.
    $\mathrm{V} \leftarrow \mathrm{V} + \hat{\theta}_j \cdot \Delta\hat{\mathrm{m}}_j$ ▷ Update the overall values.
    if $x_{\hat{\mathrm{r}}_j}$=1 **then** ▷ $\hat{\mathrm{r}}_j$ scheduled with hidden cache before.
      $y_{\hat{\mathrm{r}}_j} \leftarrow 0$ ▷ Schedule $\hat{\mathrm{r}}_j$ with KV cache now.
    else ▷ First-time schedule for $\hat{\mathrm{r}}_j$.
      if $\Delta\hat{\mathrm{m}}_j \neq \mathrm{m}_{\hat{\mathrm{r}}_j}$ **then** ▷ Hidden cache allowed for $\hat{\mathrm{r}}_j$.
        $y_{\hat{\mathrm{r}}_j} \leftarrow 1$ ▷ Schedule $\hat{\mathrm{r}}_j$ with hidden cache.
      end if
    end if
    $x_{\hat{\mathrm{r}}_j} \leftarrow 1$ ▷ Schedule $\hat{\mathrm{r}}_j$.
  else ▷ Keep adding would violate memory constraint M.
    $\mathrm{V}' \leftarrow \hat{\theta}_j \cdot \Delta\hat{\mathrm{m}}_j$
    $S' \leftarrow \{(x_i, y_i) | \forall i \in U, (x_i, y_i) = (0, 0)$ except for $i = \hat{\mathrm{r}}_j$ where $(x_{\hat{\mathrm{r}}_j}, y_{\hat{\mathrm{r}}_j}) = (1, 0)\}$
    Break
  end if
end for
if $\mathrm{V} < \mathrm{V}'$ **then** ▷ Double check to prevent the extreme case.
  $S \leftarrow S'$
end if

associated with each candidate request can be pre-recomputed. Then, we can derive the candidate schedule set Υ described as follow:

$$\Upsilon = \{(\theta_j, \mathrm{r}_j, \Delta\mathrm{m}_j, \mathrm{m}_{\mathrm{r}_j}) \mid j = 1, 2, ..., n\}. \tag{1}$$

For the j-th candidate schedule in the set Υ, θ_j represents its marginal gain in value. r_j corresponds to the request associated with the j-th candidate schedule. $\Delta\mathrm{m}_j$ indicates the marginal increase in memory usage required by this schedule, while $\mathrm{m}_{\mathrm{r}_j}$ denotes the maximum memory required by request r_j for the j-th candidate schedule. With the candidate schedule set Υ derived, together with the candidate request set U and the memory constraint M, it is fed to the greedy-based scheduling process outlined in Algorithm 1 as input, to derive the final scheduling decisions S as follow:

$$S = \{(x_i, y_i) \mid i \in U\}. \tag{2}$$

The complexity of Algorithm 1 is $\mathcal{O}(n \log n)$, resulting from sorting the candidate schedule set Υ at the beginning (Line 4). When increasing memory usage for a certain request $\hat{\mathrm{r}}_j$ causes a memory constraint violation, the resultant gain towards the overall scheduling value V' is still recorded (Lines 18-19) and compared with the existing overall scheduling value V to derive the final scheduling decisions (Lines 23-25). This double-check procedure is designed to prevent the extreme case where a request with a large absolute value but a small marginal gain in value is omitted during scheduling, resulting in an overall scheduling value far from optimal.

2 Approximation Guarantee

We further develop the approximation guarantee for the proposed greedy-based solution (Algorithm 1), which is summarized in Theorem 1.

Theorem 1. *For the hybrid-cache-based scheduling problem, let Φ_{ALG} denote the overall scheduling value produced by the greedy solution (Algorithm 1), and Φ_{OPT} denote the overall scheduling value produced by the optimal solution. We have:*

$$\frac{\Phi_{OPT}}{\Phi_{ALG}} \leq 2. \tag{3}$$

Proof. Consider an LP-relaxed variation of the original hybrid-cache-based scheduling problem, where $0 \leq x_i \leq 1$. In Algorithm 1, combining the first solution S with a simple tweak to the alternative solution S' — by setting the scheduling decision variable x'_{r_l} of the only scheduled request r_l to $\frac{M-\mathrm{sum}(\overline{M})}{\Delta\overline{m}_{\mathrm{r}_l}}$ — can result in the optimal solution for the LP-relaxed variation of the original hybrid-cache-based scheduling problem. This is because increasing the memory usage for the request with the highest marginal gain in value continuously until all the available memory is used up (achievable due to $0 \leq x_i \leq 1$), ensures that every unit of additional memory usage contributes the maximum possible value gain, thus resulting in the optimal solution $\Phi_{OPT^{LP}}$ for the LP-relaxed variant of the problem (which can be proved by contradiction). Consequently, $\mathrm{V} + \mathrm{V}' = \mathrm{V} + \hat{\theta}_{\mathrm{r}_l} \cdot \Delta\hat{\mathrm{m}}_{\mathrm{r}_l} > \mathrm{V} + \hat{\theta}_{\mathrm{r}_l} \cdot \Delta\hat{\mathrm{m}}_{\mathrm{r}_l} \cdot \frac{\mathrm{M}-\mathrm{sum}(\overline{\mathrm{M}})}{\Delta\overline{\mathrm{m}}_{\mathrm{r}_l}} = \Phi_{OPT^{LP}}$.

Then, since Algorithm 1 essentially chooses the solution corresponding to the larger value between V and V', it can be deduced that the final overall scheduling value Φ_{ALG} obtained by Algorithm 1 satisfies $\Phi_{ALG} = \max(\mathrm{V}, \mathrm{V}') \geq \frac{\mathrm{V}+\mathrm{V}'}{2} \geq \frac{\Phi_{OPT^{LP}}}{2}$, which can be easily proved by contradiction. Finally, since the solution space of the hybrid-cache-based scheduling problem is a subset of that for its LP-relaxed variant with $0 \leq x_i \leq 1$, we have $\Phi_{OPT^{LP}} \geq \Phi_{OPT}$, thus $\Phi_{ALG} \geq \frac{\Phi_{OPT}}{2}$, which ends the proof. □