EchelonFlow Case Studies and Proofs

This report provides supplementary proofs for the EchelonFlow workshop paper [1].

1. ECHELONFLOW AND ITS TARDINESS

Definition 1 (EchelonFlow). *An EchelonFlow is a set of flows whose ideal finish times are related, where the relation is represented by an arrangement function of the reference time.*

Let $H = \{f_0, f_1, ..., f_{|H|-1}\}$ be an EchelonFlow with reference time r, where $|H|$ is the cardinality (i.e., the number of flows) in H, and the flows follow the ascending order of the start time. The set $D = \{d_0, d_1, ..., d_{|H|-1}\}$ contains the ideal finish time d_j of each flow $f_j \in H$, and s_0 is the start time of f_0. Then $d_0 = r = s_0$, and we have an arrangement function $g(D, r)$.

Definition 2 (Flow Tardiness). *The tardiness of a flow is its actual finish time exceeding its ideal finish time.*

Let d be the ideal finish time of a flow f and e be the flow's actual finish time, the tardiness t_f of f is:

$$t_f = e - d \tag{S1}$$

We define tardiness to differentiate from most flow scheduling work that minimizes flow completion time. Tardiness regulates flows regarding their ideal finish times, rather than their flow start times. This definition allows computation units to realign with the arrangement per EchelonFlow.

Definition 3 (EchelonFlow Tardiness). *The tardiness of an EchelonFlow is the maximum tardiness of all its flows.*

For EchelonFlow H, following the above notations, let e_j be the actual finish time of flow f_j, corresponding to the ideal finish time d_j. The tardiness t_H of H is:

$$t_H = max(e_j - d_j), \quad 0 \leq j < |H| \tag{S2}$$

The tardiness of all the flows in an EchelonFlow should remain the same if the EchelonFlow constantly maintains the computation arrangement. The definition of maximum tardiness helps to reduce the difference in tardiness among individual flows.

Optimization Objective

Naturally, based on our earlier definitions, the optimization objective of EchelonFlow scheduling is tardiness minimization. For an individual EchelonFlow H, particularly, the objective is to minimize its tardiness t_H:

$$\textbf{Minimize:} \quad z = t_H \tag{S3}$$

For multiple EchelonFlows, the objective is to minimize the sum of their tardiness. For a set EchelonFlows $\mathcal{H} = \{H_0, H_1, ..., H_{|\mathcal{H}|-1}\}$, where $|\mathcal{H}|$ is the cardinality and t_{H_i} is the tardiness of EchelonFlow $H_i \in \mathcal{H}$, the objective is:

$$\textbf{Minimize:} \quad \hat{z} = \sum_{0 \leq i < |\mathcal{H}|} t_{H_i} \tag{S4}$$

The objective can be easily adjusted to the weighted sum of individual EchelonFlows' tardiness, should there be a proper way to assign weights to different DDLT jobs.

2. CASE STUDIES

Case I: Coflow

Coflow is a special EchelonFlow whose flows share the same ideal finish time. For a Coflow in the form of an EchelonFlow H, the ideal finish time d_j of the j_{th} flow f_j (from $|H|$ flows in total) should follow the arrangement function below, where r is the reference time, i.e., the start time of the head flow.

$$d_j = r, \quad 0 \leq j < |H| \tag{S5}$$

We prove that minimizing the tardiness of EchelonFlow H equals to minimizing the Coflow completion time.

Proof. The real finish time of the j_{th} flow f_j is e_j. z is H's tardiness.

$$\begin{aligned} Minimize \quad z &= max(e_j - d_j) \\ &= max(e_j - r) \\ &= max(e_j) - r \\ &=> z = max(e_j) \end{aligned} \tag{S6}$$

The Coflow completion time is decided by the last-finish flow. $\hat{z}$ is H's completion time.

$$Minimize \quad \hat{z} = max(e_j) \tag{S7}$$

Eqn. S6 and Eqn. S7 are the same.

Case II: Pipeline Parallelism (PP)

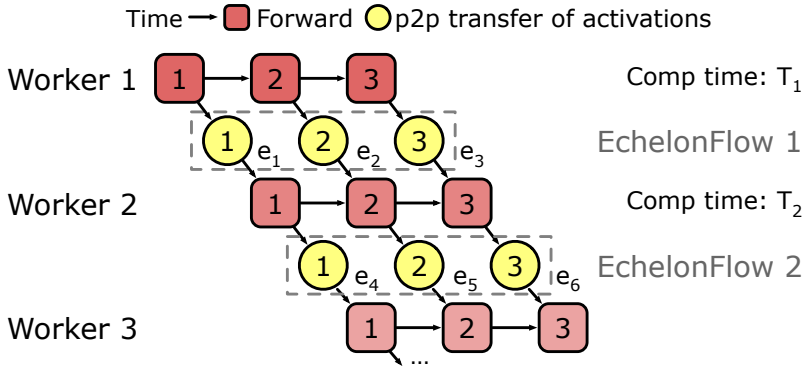

Fig. 1. EchelonFlow model for Pipeline

In Fig. 1 we have three workers and two EchelonFlows, H_1 and H_2. T_i represents the computation time of a mini batch on the i_{th} worker.

r_1 is the head flow start time in H_1. For H_1, we have:

$$d_j = \begin{cases} r_1, & j = 0 \\ d_{j-1} + T_2, & 1 \leqslant j < |H_1| \end{cases} \tag{S8}$$

r_2 is the head flow start time in H_2. For H_2, we have:

$$d_j = \begin{cases} r_2, & j = 0 \\ d_{j-1} + T_3, & 1 \leqslant j < |H_2| \end{cases} \tag{S9}$$

We prove that minimizing the tardiness of EchelonFlow H_1 equals to minimizing the task completion time.

Proof. Take Fig. 1 as an example. For simplicity, we only prove the case with three mini batches. But the proof holds for any number of mini batches.

Let $\triangle t_j = e_j - d_j, 0 \leqslant j < |H_1|$. Minimizing H_1's tardiness z_1:

$$\textbf{Minimize:} \quad z_1 = max(\triangle t_1, \triangle t_2, \triangle t_3) \tag{S10}$$

The completion time $\hat{z}$ could be expressed as:

$$\begin{aligned}
\textbf{Minimize:} \quad \hat{z} &= max(max(e_1 + T_2, e_2) + T_2, e_3) + T_2 \\
&= max(max(\triangle t_1 + d_1 + T_2, \triangle t_2 + d_2) + T_2, e_3) + T_2 \\
&= max(max(\triangle t_1 + d_2, \triangle t_2 + d_2) + T_2, e_3) + T_2 \\
&= max(max(\triangle t_1, \triangle t_2) + d_2 + T_2, e_3) + T_2 \\
&= max(max(\triangle t_1, \triangle t_2) + d_3, \triangle t_3 + d_3) + T_2 \\
&= max(max(\triangle t_1, \triangle t_2), \triangle t_3) + d_3 + T_2 \\
&= max(\triangle t_1, \triangle t_2, \triangle t_3) + d_3 + T_2 \\
&=> \hat{z} = max(\triangle t_1, \triangle t_2, \triangle t_3)
\end{aligned} \tag{S11}$$

Eqn. S10 and Eqn. S11 are the same.

Case III: Fully-Sharded Data Parallelism (FSDP)

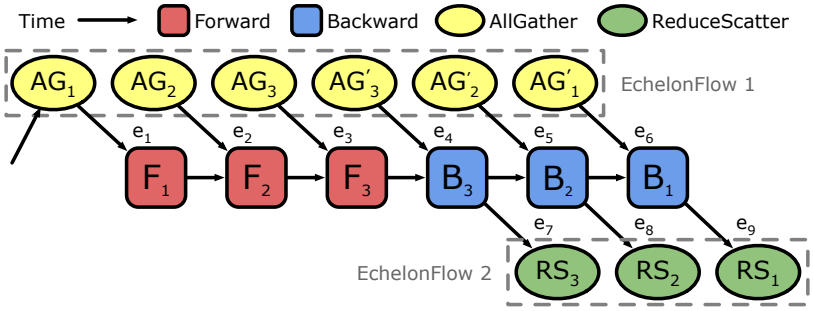

Fig. 2. EchelonFlow model for FSDP

ZeRO/FSDP [2, 3] (Fig. 2) uses the `all-gather` collective primitive to gather weights from all nodes before each layer's forward and backward computations. Flows in each `all-gather` collective form a Coflow. These Coflows along the computation timeline further form an EchelonFlow, following a pipeline-like pattern as in GPipe (Fig. 1). Every AG or RS is a Coflow, which is also a special EchelonFlow. Furthermore, all AGs form a bigger EchelonFlow H_1, and all RSes form a bigger EchelonFlow H_2. We can use the same method in Case II to prove that minimizing the tardiness of EchelonFlow H1 and H2 equals to minimizing the task completion time.

The arrangement function of EchelonFlow H_1 is similar to Eq. S8 and Eq. S9.

$$d_{c_i} = \begin{cases} r_{c_0}, & i = 0 \\ d_{c_{i-1}} + T_{fwd}, & 1 \leqslant i \leqslant n-1 \\ d_{c_{i-1}} + T_{bwd}, & n \leqslant i \leqslant 2n-1 \end{cases} \tag{S12}$$

For an n-layer neural network, let C_i be the i_{th} Coflow, then $C_0 - C_{n-1}$ and $C_n - C_{2n-1}$ belong to the forward and backward phase, respectively. The reference time r_{c_0} is the reference time of the first Coflow C_0, which is the start time of its first flow. Since all flows in a Coflow share the same ideal finish time, the (single) ideal finish time of each Coflow d_{c_i} is time T_{fwd} or T_{bwd} later than the previous Coflow $d_{c_{i-1}}$ depending on whether it lies in the forward or backward phase. T_{fwd} and T_{bwd} can both be profiled.

3. PROPERTIES

Here we list important properties of EchelonFlow.

Property 1: EchelonFlow scheduling minimizes completion times of popular DDLT paradigms.

Tardiness minimization aims to advance computation units while maintaining the desirable computation arrangement, which ultimately speeds up training. We prove it case-by-case for the popular DDLT paradigms in §2.

Property 2: EchelonFlow is a superset of Coflow.

Coflow can be presented as a special EchelonFlow where all the flows share a common ideal finish time. In this case, by definition, the tardiness of every flow is its finish time minus the start time of the first flow. Our EchelonFlow optimization objective of minimizing the maximum tardiness among all the flows (Eq. S3) becomes minimizing Coflow completion time. This is proved in Case I of §2.

Property 3: EchelonFlow scheduling is NP-hard.

Coflow scheduling is NP-hard [4], so the superset problem EchelonFlow scheduling is also NP-hard.

Property 4: Coflow scheduling algorithms can be adapted to EchelonFlow scheduling at the same complexity.

There exists a one-to-one mapping between EchelonFlow and Coflow metrics. By the flow tardiness definition in Eq. S1, the finish time of each flow could be calculated as:

$$e = t_f + d \tag{S13}$$

For each EchelonFlow $H = \{f_0, f_1, ..., f_{|H|-1}\}$, according to its arrangement function $D = \{d_0, d_1, ..., d_{|H|-1}\}$, the flow completion time could be presented as $F = \{f_0 + d_0, f_1 + d_1, ..., f_{|H|-1} + d_{|H|-1}\}$. By substituting the flow completion time with the tardiness metric, we change the optimization goal from minimizing the flow completion time to minimizing the flow tardiness. In this sense, we can adapt Coflow scheduling algorithms to EchelonFlow scheduling, with a different metric for evaluating flows. In MADD [4], for example, in intra-EchelonFlow scheduling, we estimate the latest flow that has the largest tardiness, rather than the longest flow completion time as for Coflow; in inter-EchelonFlow scheduling, we rank EchelonFlows by each EchelonFlow's tardiness (Eq. S4), instead of the Coflow completion time. This mapping does not change the algorithm complexity.

REFERENCES

1. R. Pan, Y. Lei, J. Li, Z. Xie, B. Yuan, and Y. Xia, "Efficient flow scheduling in distributed deep learning training with echelon formation," in *The 21st ACM Workshop on Hot Topics in Networks (HotNets '22), November 14– 15, 2022, Austin, TX, USA,* (2022).
2. Facebook, "Fairscale," in *https://github.com/facebookresearch/fairscale*, (2022).
3. S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis,* (IEEE, 2020), pp. 1–16.
4. M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM,* (2014), pp. 443–454.