Difference between IMPALA and other MF-OnPolicy Algorithms

- Goal is to be as Data and Resource efficient as possible
- IMPALA enables a fast parallel architecture in contrast to other OnPolicy Methods
- It achieves that by sacrificing On-Policy-ness -> Introduces an OffPolicy Algorithm (V-Trace)
- The Learner updates the Parameters of the main policy π with trajectories from actors using potentially older policies π^{μ}

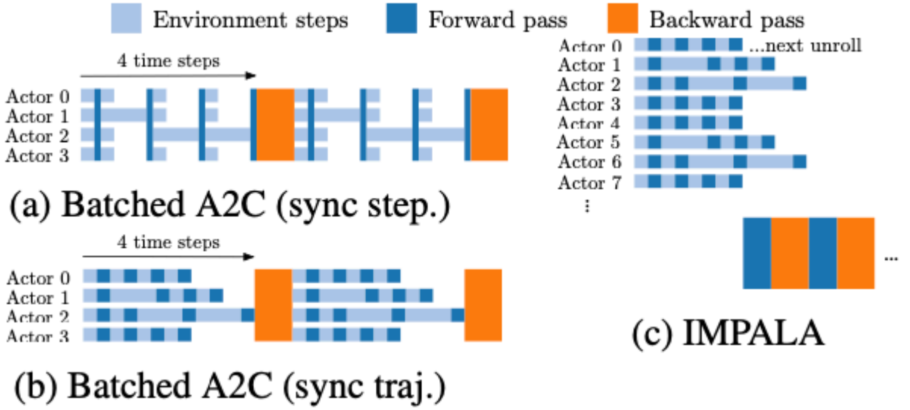

Figure 2. Timeline for one unroll with 4 steps using different architectures. Strategies shown in **(a)** and **(b)** can lead to low GPU utilisation due to rendering time variance within a batch. In **(a)**, the actors are synchronised after every step. In **(b)** after every n steps. IMPALA **(c)** decouples acting from learning.

Source: https://arxiv.org/abs/1802.01561

Difference between IMPALA and other MF-OnPolicy Algorithms

- Results in a very high throughput compared to A2C/A3C

Architecture	CPUs	GPUs[1]	FPS[2]	
Single-Machine			Task 1	Task 2
A3C 32 workers	64	0	6.5K	9K
Batched A2C (sync step)	48	0	9K	5K
Batched A2C (sync step)	48	1	13K	5.5K
Batched A2C (sync traj.)	48	0	16K	17.5K
Batched A2C (dyn. batch)	48	1	16K	13K
IMPALA 48 actors	48	0	17K	20.5K
IMPALA (dyn. batch) 48 actors[3]	48	1	21K	24K
Distributed				
A3C	200	0	46K	50K
IMPALA	150	1	80K	
IMPALA (optimised)	375	1	200K	
IMPALA (optimised) batch 128	500	1	250K	

[1] Nvidia P100 [2] In frames/sec (4 times the agent steps due to action repeat). [3] Limited by amount of rendering possible on a single machine.

Table 1. Throughput on `seekavoid_arena_01` (task 1) and `rooms_keys_doors_puzzle` (task 2) with the shallow model in Figure 3. The latter has variable length episodes and slow restarts. Batched A2C and IMPALA use batch size 32 if not otherwise mentioned.

Source: https://arxiv.org/abs/1802.01561