REVERB : A Framework For Experience Replay

Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, Manuel Kroiss

구글이 분산 강화학습을 하는 법

Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, Manuel Kroiss

서론

- 강화 학습(RL)의 핵심 구성 요소는 데이터인 경험(experience)입니다. 경험을 생성하고 소비하는 데 사용되는 메커니즘은 강화학습의 성능에 중요한 영향을 끼치죠
- Reverb에서는 최대 수천 클라이언트의 분산 강화학습에서도 작동하도록 설계되었습니다. (menger…)
- 이 논문은 Reverb가 어떻게 설계되어있는지에 관한 논문입니다.
  - 어떻게 selecting하는지 어떻게 삭제하는지 비율은 어떻게 정하는지 어떻게 insert를 하는지를 설명합니다.

- me) 현재 구글에서 사용하고 있는 TF-agent의 Distributed RL은 어떻게 되고 있는지 볼 수 있습니다 (RL 프레임워크마다 DRL은 특징이 좀 다릅니다~)

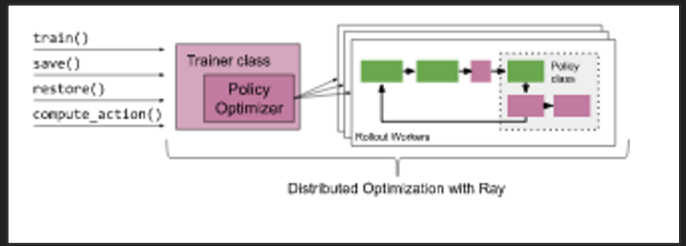
강화학습 프레임워크를 보다 그런 생각이 들으신 적 있으신지요

딥러닝에선 Pseudo code의 방식으로 코드를 짜죠.

근데 강화학습에선 Actor와 Learner 등으로 쪼갭니다.

원래 그러지 않았느냐라고 물으신다면 그렇지는 않습니다.

분산처리 알고리즘이 나오면서 Actor와 Learner의 분리가 필요해진거죠.



보통 많이 보이던 형태일겁니다.
code로는 trainer 정의하고 trainer.learn 뭐 이런식으로
간략하게 되어 있죠

```
# train
for ep in range(1000):
    s = env.reset()
    # pi.epsilon = max(0.01, pi.epsilon * 0.95)
    # env.record_metrics({'EpsilonGreedy/epsilon': pi.epsilon})

    for t in range(env.spec.max_episode_steps):
        a = pi(s)
        s_next, r, done, info = env.step(a)

        # extend last reward as asymptotic best-case return
        if t == env.spec.max_episode_steps - 1:
            assert done
            r = 1 / (1 - tracer.gamma)  # gamma + gamma^2 + gamma^3 + ... = 1 / (1 - gamma)

        # trace rewards and add transition to replay buffer
        tracer.add(s, a, r, done)
        while tracer:
            buffer.add(tracer.pop())

        # learn
        if len(buffer) >= 100:
            transition_batch = buffer.sample(batch_size=32)
            metrics = qlearning.update(transition_batch)
            env.record_metrics(metrics)

        # sync target network
        q_targ.soft_update(q, tau=0.01)

        if done:
            break

        s = s_next

    # early stopping
    if env.avg_G > env.spec.reward_threshold:
        break
```

<- 현재도 Actor Learner 형태를 따르지 않는 프레임워크도 있습니다.

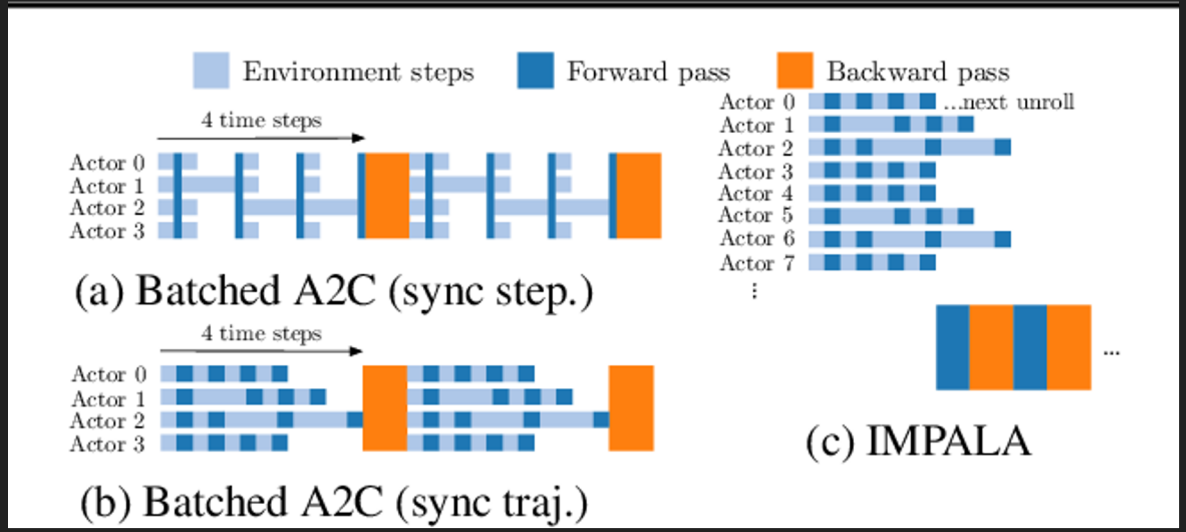
sync, async…

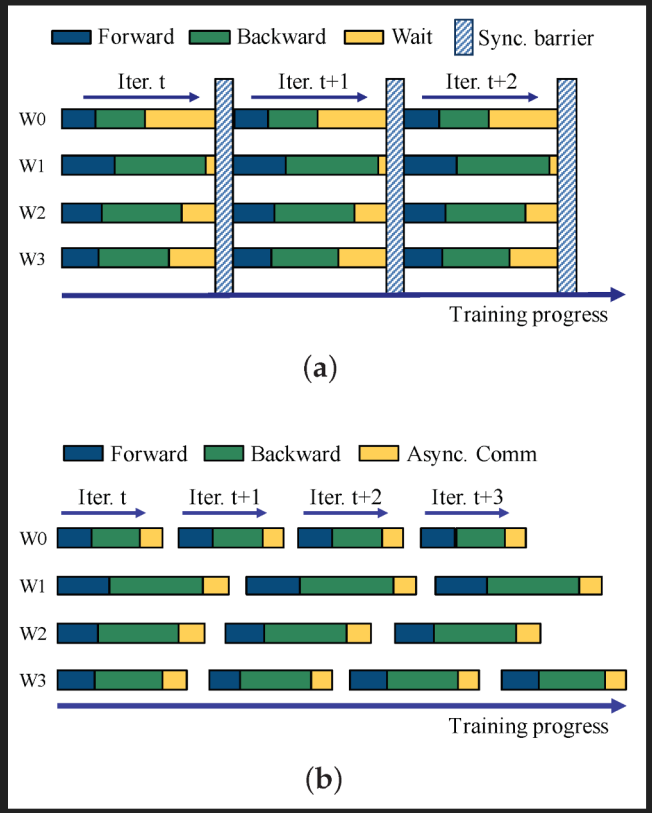
async는 아시다 싶이 policy가 자주 바뀌는 문제가 있죠.

그래서 sync를 했더니 다른 process가 끝날때까지 기다려야 합니다.

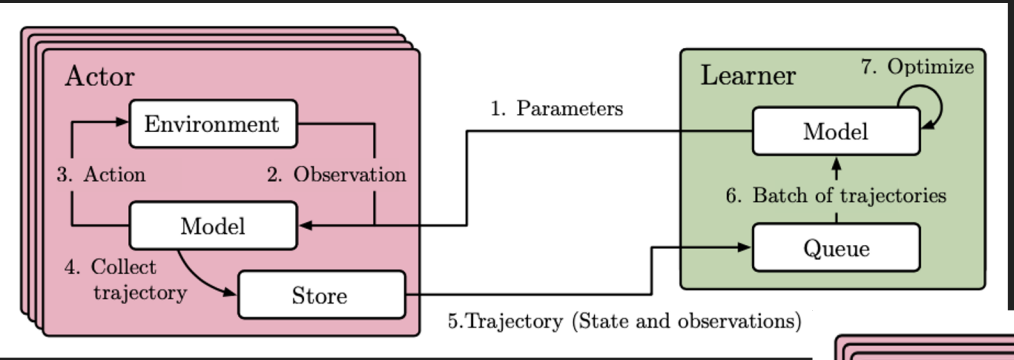
결국, 구글은 Actor와 Learner를 완전히 분리하는 방향으로 가게 됩니다.

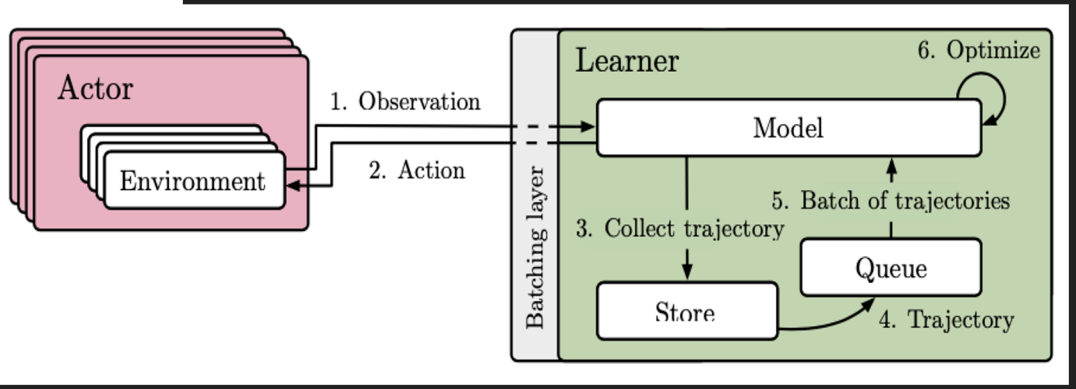
이에 따라 둘 사이에 **Replay buffer**와 parameter server가 필요해진거죠.





Related Work (SEED RL)





-> SEED RL은 Learner에게 과하게 많은 역할을 부여하고 이는 Scale을 높일 때 문제가 됩니다.

서론

- 이제 이 노트에 다 같이 적게 해볼 겁니다.
- **근데 문제가 있네요.**
  - 공책이 너무 두꺼워졌습니다
  - 분신 친구들이 적는 속도가 너무 느리군요.
  - 다 같이 한 노트에 적으니 계속 티격태격해서 속도가 느려졌습니다.
  - 분신 친구들이 규칙없이 적습니다
  - 제가 이걸 어떻게 읽어야 할지도 난감하네요.
- 이제 공부 잘하는 구땡 친구 노트는 어떻게 돼있나 봅시다.

본론 (디자인)

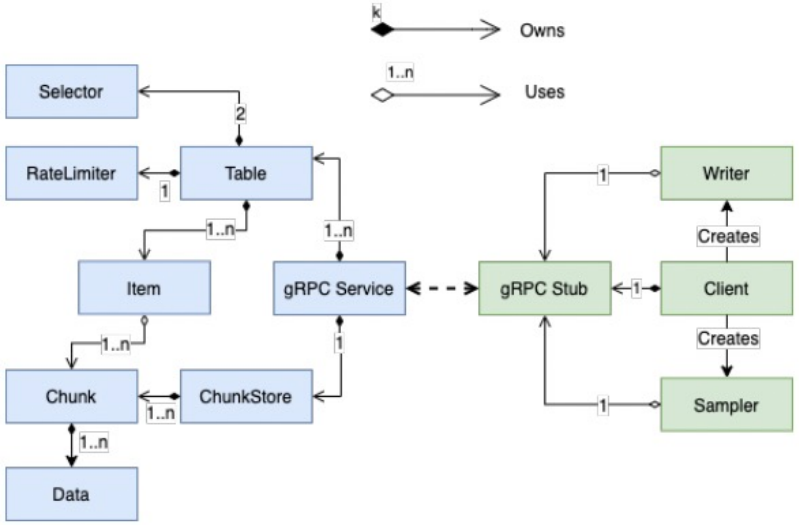


Figure 2. High level schema of how the most important classes relate to each other.

- At its core, Reverb is a data store that exposes a gRPC service with an API for clients to write raw tensor data.
- Data is written to a ChunkStore as parts of a Chunk, Chunks can be referenced by Items, and each Item is owned by a Table. The Table encapsulates Items and controls sample and insert requests, using a RateLimiter and two Selectors (one Sampler and one Remover)
- These Components are independently configured and can be mixed and matched to provide a high degree of flexibility and a wide range of different behaviors.

Chunking and Chunk store

- 비슷한 데이터 들어오는데 좀 줄여줄 수 없을까요?

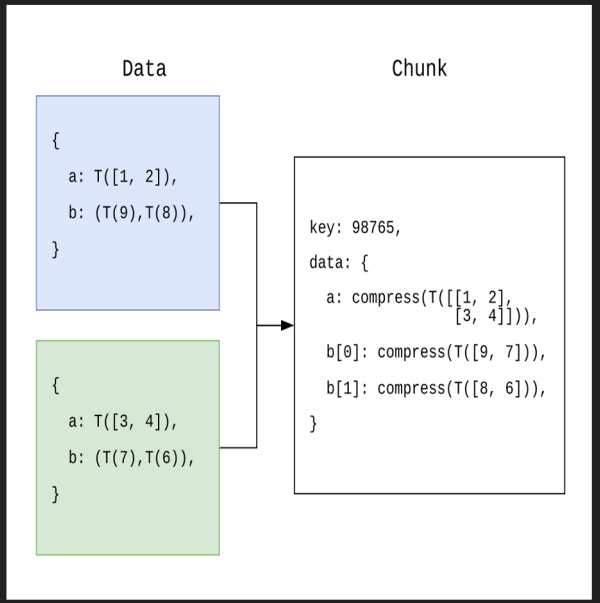
이전 이미지

차이는 고작 점의 위치

다음 이미지

Chunk

Sequential data



Data Index	Chunk Key	a	b[0]	b[1]
0	98765	[1, 2]	9	8
1	98765	[3, 4]	7	6
2	...	...	...	...
3	...	...	...	

Chunk store

Chunk optimization

- First, sequential data elements are grouped into a Chunk and compressed.
- Second, the Chunk abstraction over the raw data allows multiple Items (which can belong to different Tables) to reference the same underlying data instead of maintaining separate copies.

Item (and Table)

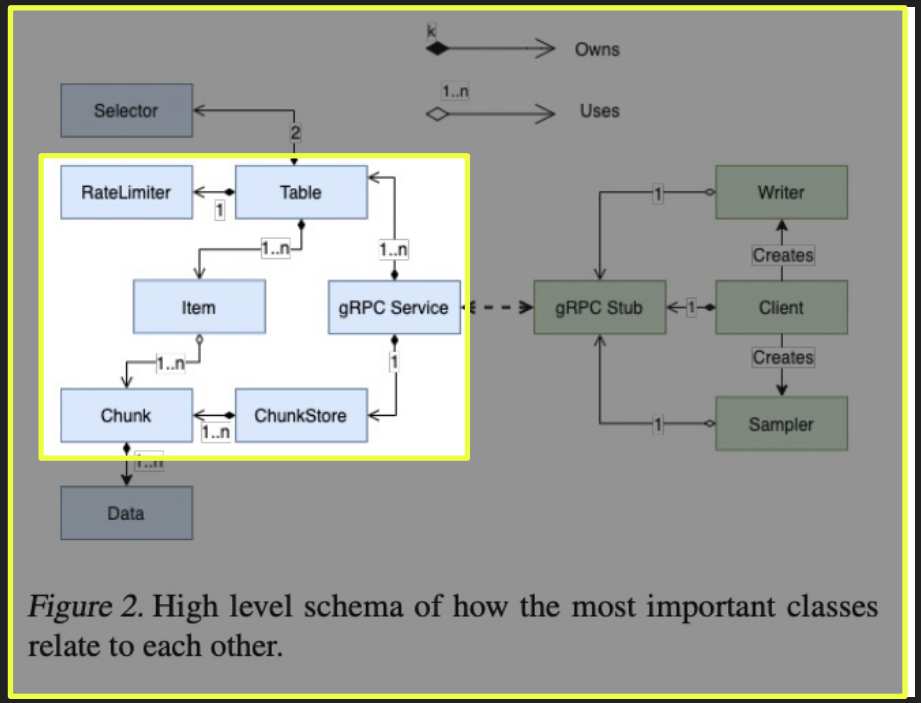


Figure 2. High level schema of how the most important classes relate to each other.

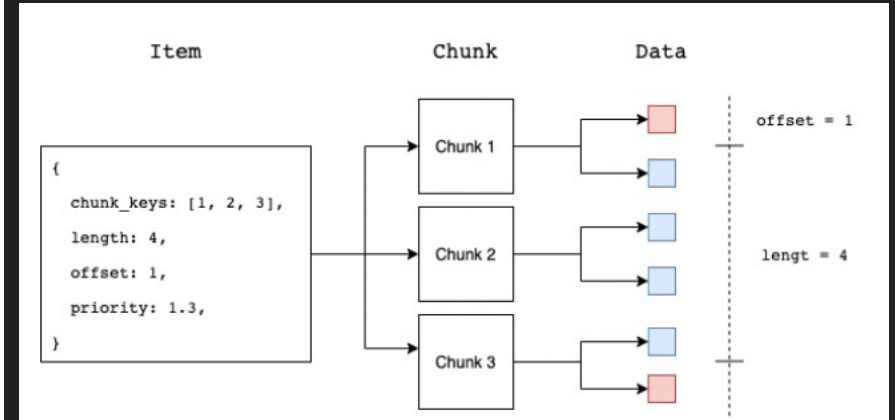


Figure 3. An `Item` references experience using one or more `Chunks`. `Offset` and `length` determine the exact steps selected.

Selector

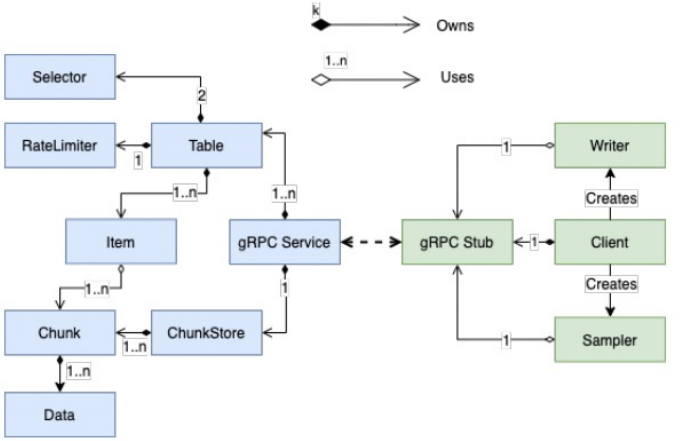


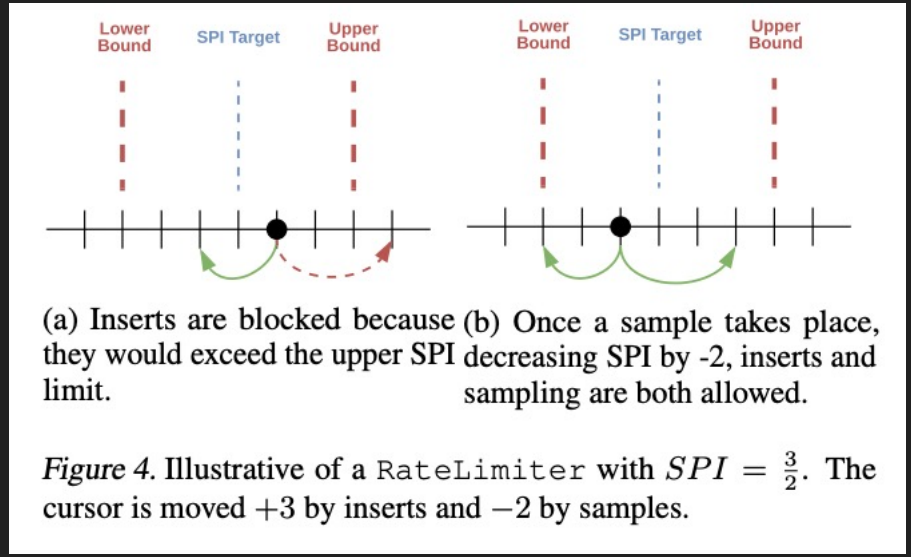
Figure 2. High level schema of how the most important classes relate to each other.

- A Selector is a strategy used to select an item in a Table. A Table uses two Selectors:
  - • The Sampler is used to select Items when the client requests them.
  - • The Remover is used to select an Item to remove when the Table is full.
- FIFO, LIFO, PER 등을 이를 이용해 구현할 수 있습니다.

Rate Limiting

- The RateLimiter controls when Items are allowed to be inserted and/or sampled from a Table. The RateLimiter monitors two aspects of the Table: 1) the number of Items in the table, and 2), the SPI defined as:

$$SPI = \frac{\texttt{num\_sampled\_items}}{\texttt{num\_inserted\_items}}$$



(a) Inserts are blocked because they would exceed the upper SPI limit.

(b) Once a sample takes place, decreasing SPI by -2, inserts and sampling are both allowed.

Figure 4. Illustrative of a `RateLimiter` with $SPI = \frac{3}{2}$. The cursor is moved $+3$ by inserts and -2 by samples.

Table Extension

- Tables support a TableExtension API which provides the ability to execute extra actions as part of the atomic operations of the parent Table

Checkpointing

- The state and content of both the ChunkStore and Tables can be serialized and stored to disk as a checkpoint. Potential data loss in the event of unexpected server failures can be limited through the use of periodic checkpointing. Stored checkpoints can be loaded by Reverb servers at construction time.

Sharding

샤딩(**sharding**)이란 하나의 거대한 데이터베이스나 네트워크 시스템을 여러 개의 작은 조각으로 나누어 분산 저장하여 관리하는 것을 말합니다.

Reverb 서버를 여러 개 둘 경우 분산되어 저장되어 관리되며,

샘플링 요청은 여러 서버에 동시에 되어 하나의 스트림으로 병합됩니다.

이러한 방식으로 Reverb는 여러 개의 노트(Replay Buffer)를 운영합니다.

Reverb Client

- The Reverb Client wraps the gRPC client to provide a higher level API for writing, modifying, and reading data from the server.
- A Sampler manages a pool of long lived gRPC streams
- A Writer is used to stream sequential data to the server and insert Items into one or more Tables.

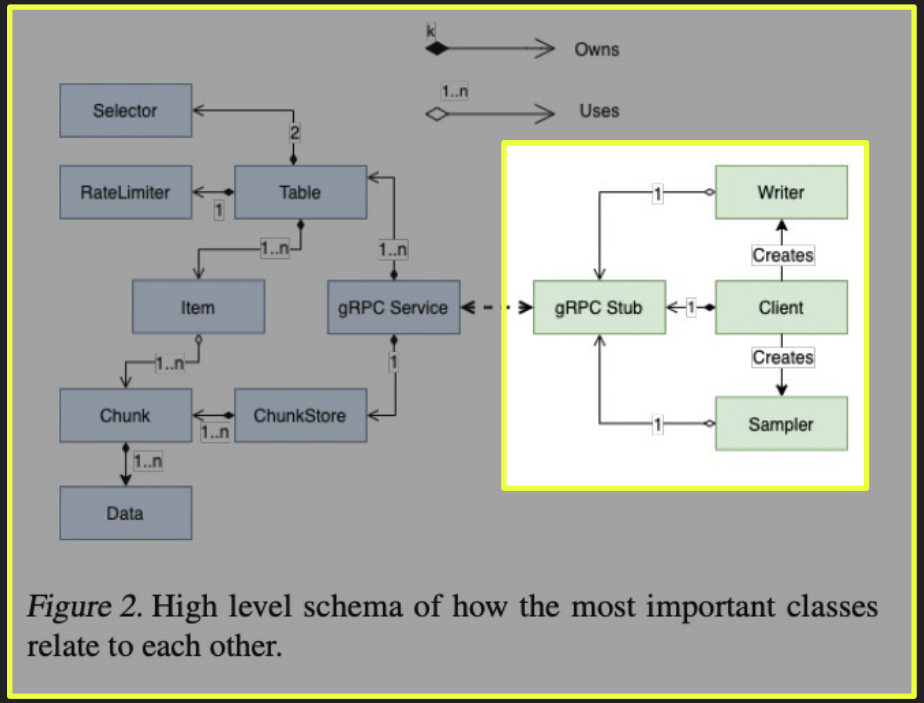
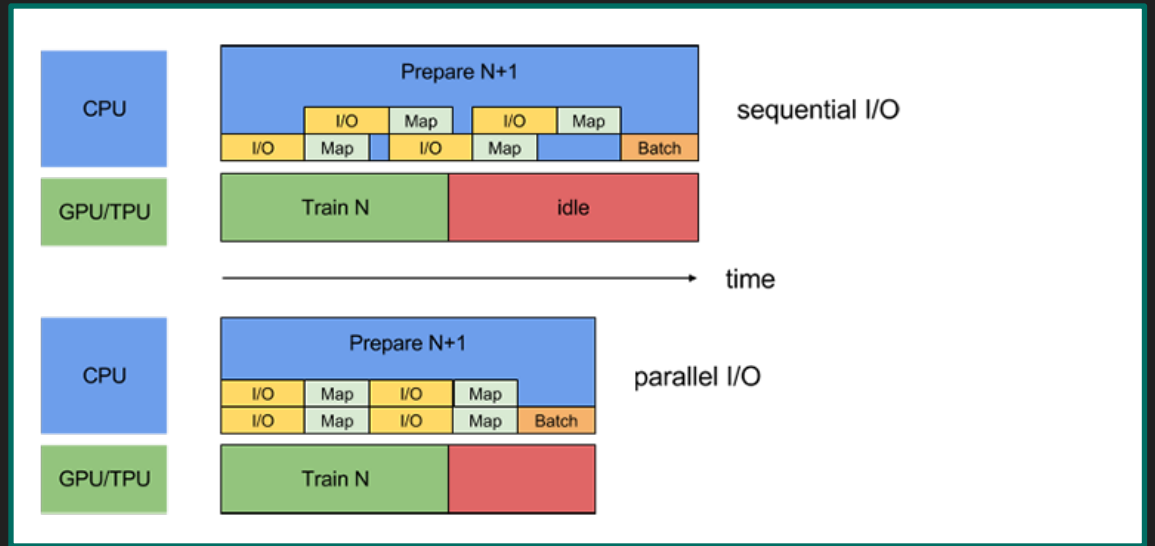


Figure 2. High level schema of how the most important classes relate to each other.

Efficient Sampling with tf.data API

Sampler

Reverb utilizes the tf.data framework (Abadi et al., 2015) to provide pipelined high throughput data sampling directly to training modules.
Tf.data is a proven high performance and scalable input pipeline solution for transforming and feeding data.

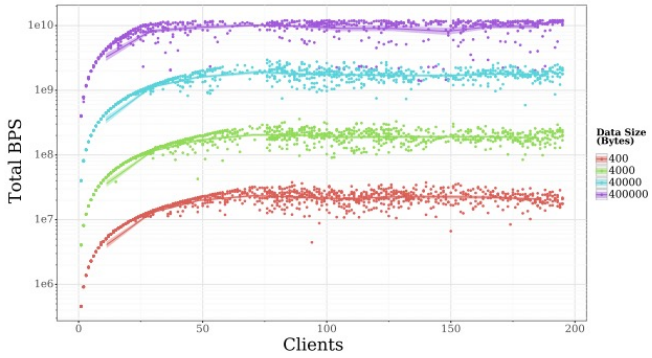


Experiment

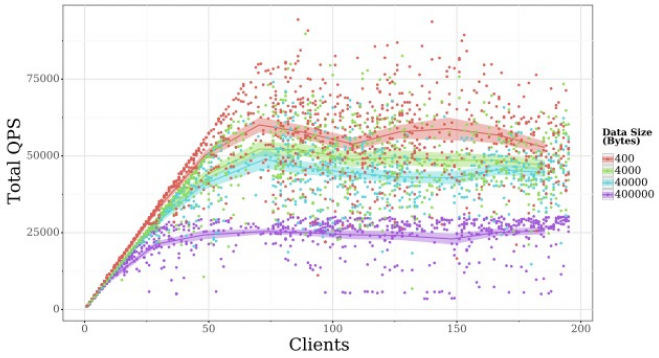
1. Each data element is a single float32 tensor whose values have been randomly sampled from a uniform distribution over the half-open interval [0, 1). Random tensors were intentionally used to negate variability introduced by compression. Reported Bytes/s are much lower than what would be observed in the exact same setup modulo the source of data. For example, in Atari we observe compression rates of up 90% in sequences of 40 frames. The effective throughput would therefore be up to 10x higher in that scenario compared with these benchmarks utilizing random synthetic data.
2. Chunk and sequence length is 1 resulting in Items not sharing data.
3. Clients solely generate load as fast as possible.

성능

- The maximum BPS(Total Byte Size) is 11 GB/s for both insertion and sampling. This limit is observed at multiple payloads sizes and strongly indicates that the observed BPS limitations cannot be attributed to Reverb. The limitation is most likely the result of network bandwidth constraints.
- To test our hypothesis that QPS(Query per size) for inserts is currently limited by **mutex contention**. The load was sharded across multiple Tables on the same server. This improved the maximum insert QPS by approximately 200%, and convinced us that the insert QPS limitations are due to mutex contention.

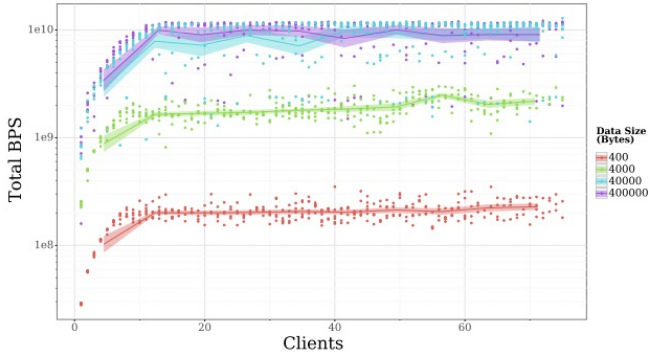


(a) Bytes Per Second (BPS) that the server handles plotted against the number of connected clients.

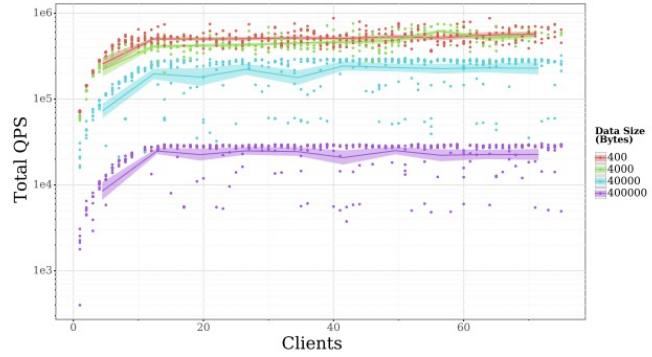


(b) Queries Per Second (QPS) that the server handles plotted against the number of connected clients.

Figure 5. Server performance benchmarks for inserting data. Both 5a and 5b plots the results for tensor payload sizes between 400 Bytes and 400 kB.



(a) Bytes Per Second (BPS) that the server handles plotted against the number of connected clients.



(b) Queries Per Second (QPS) that the server handles plotted against the number of connected clients.

Figure 6. Server performance benchmarks for sampling data. Each plot in 5a and 5b is for a different tensor payload size, from 400 Bytes to 400 kB.

결론

- Reverb은 강화 학습 연구를 위해 설계된 효율적이고 사용하기 쉬운 데이터 저장 및 전송 시스템입니다. (tf-agent)
- Reverb를 사용하면 단일 서버 당 O(100k) QPS로 불러오고 및 최소 11GB/s의 압축 데이터를 처리하면서 쉽게 한 개의 클라이언트에서 수백 개의 클라이언트로 쉽게 확장할 수 있습니다.
- 이러한 스케일링 특성과 유연함으로 연구자들이 단일 프로세스와 동일한 설정으로 수천 대의 기계를 사용하여 실험을 실행할 수 있게 한다.
  (~~사실 단일 프로세스에선 성능이 좋진 않다, tf agent에서는 기존 RB와 Reverb를 둘다 제공중이다.~~)
- 결과적으로, Reverb는 현재 많은 연구자들이 다양한 RL 실험에 사용하고 있다. (특히 구글, 딥마인드)