Ray. Распределенное обучение

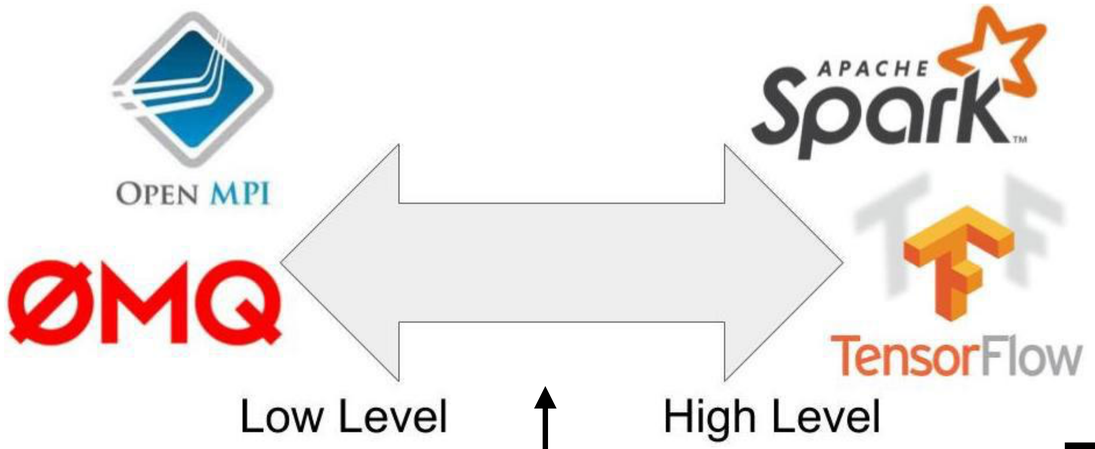
Посевин М.Э. 2022

Ray

Ray представляет собой проект с открытым исходным кодом для параллельных вычислений и распределенной разработки Python.

Экосистема Ray состоит из 3 частей:

- базовой системы Ray,
- расширяемых библиотек для машинного обучения (включая собственные библиотеки и сторонние библиотеки)
- системы для запуска кластеров в любом кластере или инструменте поставщика облачных вычислений.

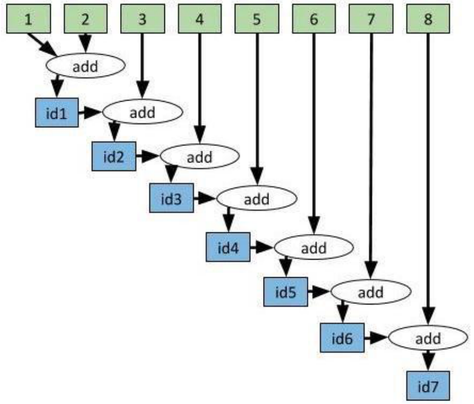


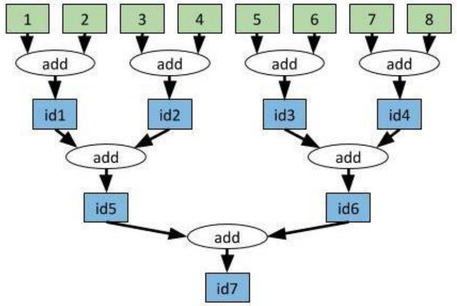
низкоуровневые примитивы для отправки и получения сообщений

инструменты для конкретной предметной области

Ray. Он не вводит новых концепций, но использует концепции функций и классов и преобразует их в распределенные задачи и акторов. Ray может распараллеливать последовательные приложения без внесения серьезных изменений.

Удаленные функции






```python
import time

@ray.remote
def add(x, y):
    time.sleep(1)
    return x + y

id1 = add.remote(1, 2)
id2 = add.remote(id1, 3)
id3 = add.remote(id2, 4)
id4 = add.remote(id3, 5)
id5 = add.remote(id4, 6)
id6 = add.remote(id5, 7)
id7 = add.remote(id6, 8)
result = ray.get(id7)
```

```python
import time

@ray.remote
def add(x, y):
    time.sleep(1)
    return x + y

id1 = add.remote(1, 2)
id2 = add.remote(3, 4)
id3 = add.remote(5, 6)
id4 = add.remote(7, 8)
id5 = add.remote(id1, id2)
id6 = add.remote(id3, id4)
id7 = add.remote(id5, id6)
result = ray.get(id7)
```

Акторы

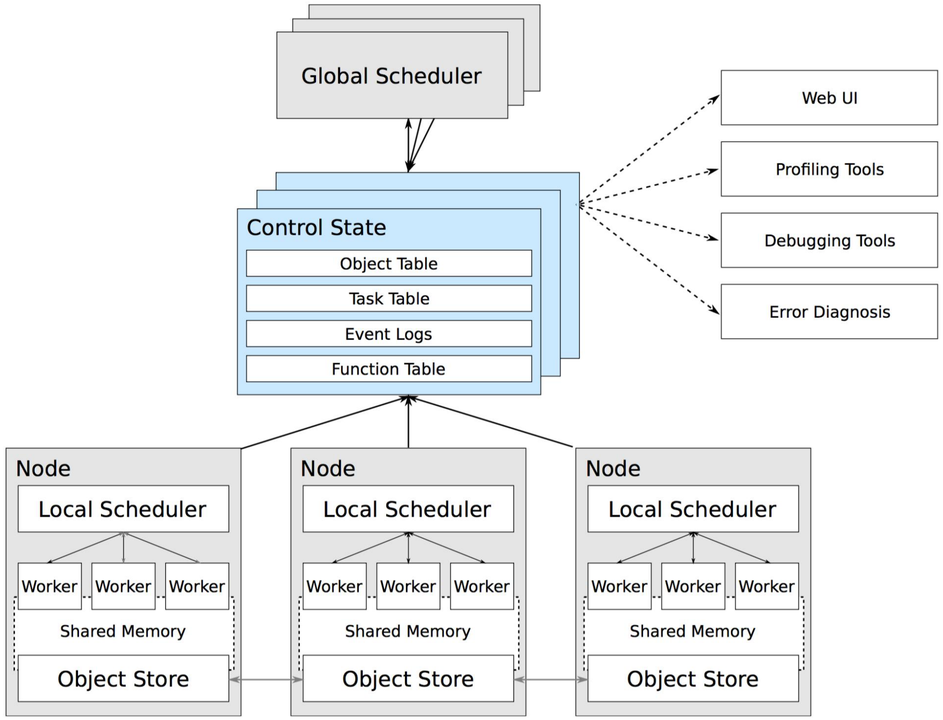
При помощи одних только удаленных функций и вышеописанного обращения с задачами невозможно добиться, чтобы несколько задач одновременно работали над одним и тем же разделяемым изменяемым состоянием. Такая проблема при машинном обучении возникает в разных контекстах, где разделяемым может быть состояние симулятора, весовые коэффициенты в нейронной сети, что-нибудь совершенно иное.

Абстракция актора используется в Ray для инкапсуляции изменяемого состояния, разделяемого между множеством задач.

Системная архитектура

В качестве распределенной вычислительной системы Ray следует типичной конструкции Master-Slave: Master отвечает за глобальную координацию и поддержание состояния, а Slave выполняет задачи распределенных вычислений. В режиме кластерного развертывания Ray запустил следующие ключевые компоненты:

- **GlobalScheduler**: На главном устройстве запускается глобальный планировщик, который принимает задачи, представленные локальным планировщиком, и распределяет задачи по соответствующему локальному планировщику задач для выполнения.
- **RedisServer**: Запустите один или несколько RedisServers на главном сервере, чтобы сохранить информацию о состоянии (ControlState) распределенных задач.
- **LocalScheduler**: На каждом подчиненном сервере запускается локальный планировщик, который используется для отправки задач глобальному планировщику и назначения задач рабочему процессу на текущей машине.
- **Worker**: Каждое ведомое устройство может запускать несколько рабочих процессов для выполнения распределенных задач и сохранения результатов вычислений в ObjectStore.
- **ObjectStore**: Каждый Slave запускает ObjectStore для хранения объектов данных, доступных только для чтения. Работники могут получить доступ к этим данным объекта через общую память, что может эффективно снизить стоимость копирования памяти и сериализации объектов. Нижний уровень ObjectStore реализован Apache Arrow.



Основные операции

ray.init() - запустить ray локально

ray.put() - сохранить объекты в локальном ObjectStore и асинхронно возвращать уникальный ObjectID.

ray.get() - получить объекты из ObjectStore через ObjectID и преобразовать их в объекты

ray.wait() - ожидание

ray.error_info() - получить информацию об ошибке

Ray Train

Ray Train - библиотека, позволяющая упросить распределенной обучение.

Frameworks: Ray Train создан для абстрагирования от настройки координации / конфигурации распределенных фреймворков глубокого обучения, таких как Pytorch Distributed и Tensorflow Distributed, позволяя пользователям сосредоточиться только на реализации логики обучения.

- Для Pytorch Ray Train автоматически обрабатывает построение распределенной группы процессов
- Для Tensorflow Ray Train автоматически обрабатывает координацию TF_CONFIG. Текущая реализация предполагает, что пользователь будет использовать MultiWorkerMirroredStrategy стратегию, но в ближайшем будущем это изменится.

Практические результаты

Ray Train использовался во время обучения 2 моделей (tensorflow):

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d_2 (MaxPooling 2D)	(None, 12, 12, 6)	0
conv2d_3 (Conv2D)	(None, 8, 8, 16)	2416
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 16)	0
flatten_1 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 120)	30840
dense_4 (Dense)	(None, 84)	10164
dense_5 (Dense)	(None, 10)	850

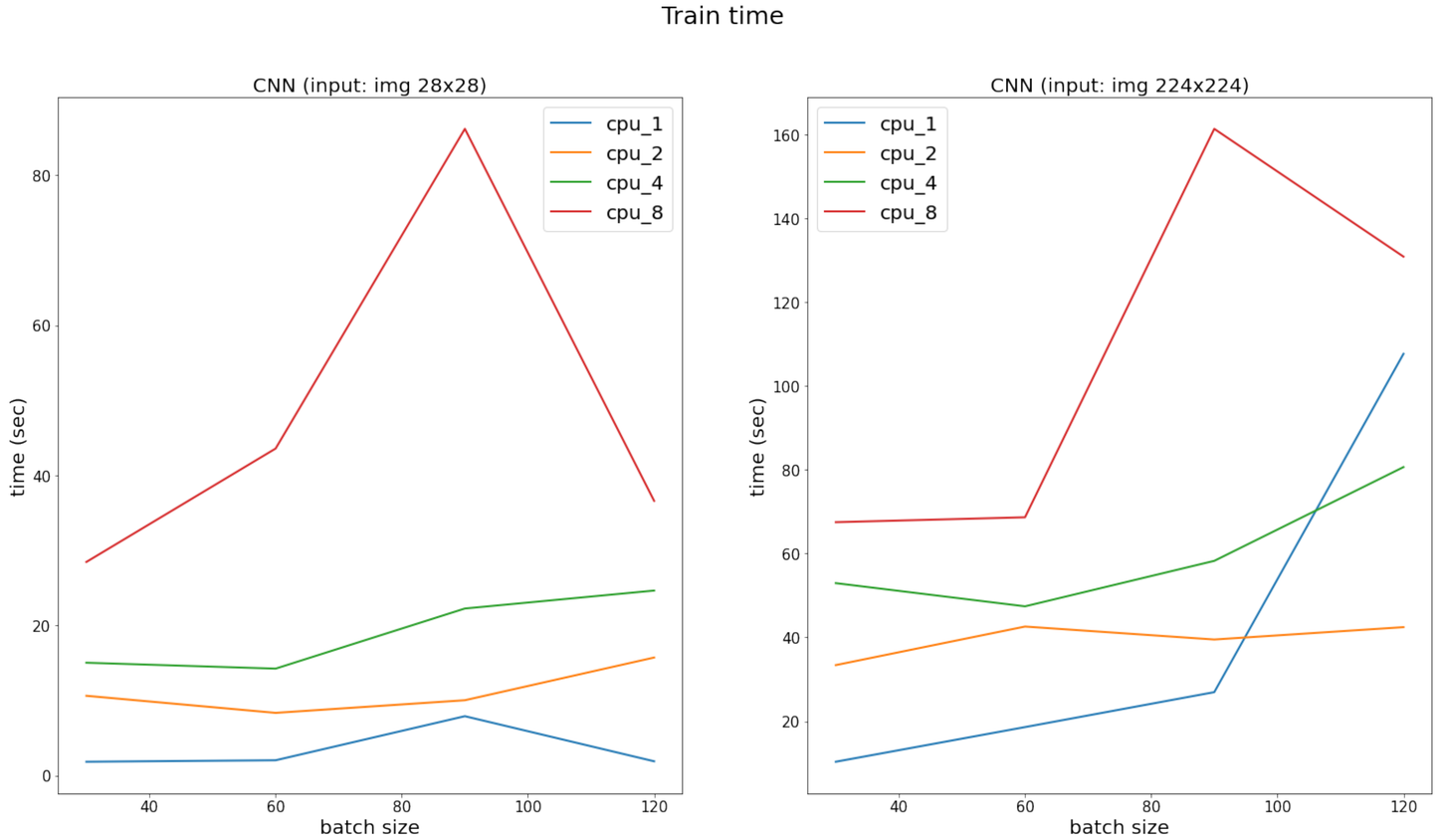
Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 112, 112, 32)	0
conv2d_5 (Conv2D)	(None, 112, 112, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 56, 56, 64)	0
flatten_2 (Flatten)	(None, 200704)	0
dense_6 (Dense)	(None, 2)	401410

Total params: 420,802
Trainable params: 420,802
Non-trainable params: 0

Практические результаты

Сравнение скорости обучения с применением Ray Train и без:



Источники

- https://docs.ray.io/en/latest/train/train.html
- https://github.com/LuckyZXL2016/Machine-Learning-Papers/blob/master/Ray/Ray%2CA%20Distributed%20Framework%20for%20Emerging%20AI%20Applications.pdf
- https://habr.com/ru/company/piter/blog/420695/
- https://www.codetd.com/ru/article/12526970
- https://www.machinelearningmastery.ru/modern-parallel-and-distributed-python-a-quick-tutorial-on-ray-99f8d70369b8/
- https://russianblogs.com/article/92751001142/
- https://russianblogs.com/article/18301057190/
- https://github.com/LuckyZXL2016/Machine-Learning-Papers/blob/master/Ray/Real-Time%20Machine%20Learning%20The%20Missing%20Pieces.pdf
- https://github.com/ray-project/ray