1. Triton Architecture

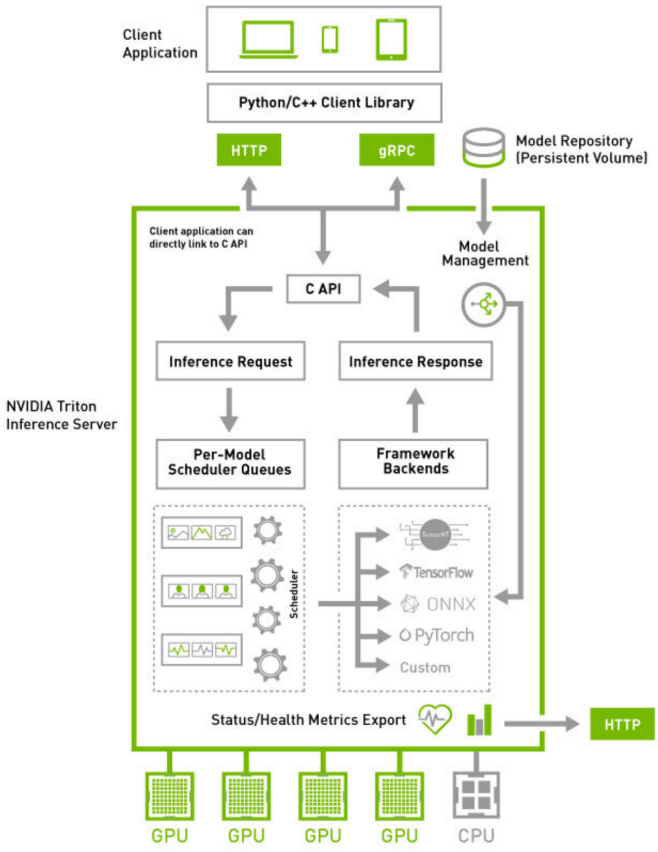


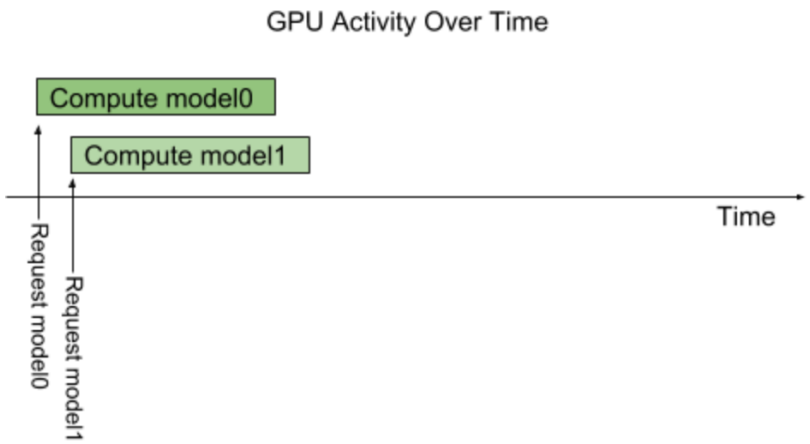
FIGURE 1.TRITON ARCHITECTURE

Model Repository 는 파일-시스템으로 구성된 모델로, Triton 이 추론할 때 사용한다. 추론 요청은 서버에 HTTP/REST 또는 GRPC, 또는 C-API 를 통해 이루어지며, 이 요청은 적절한 모델의 스케쥴러로 라우팅된다. Triton 은 모델 별로 설정할 수 있는 스케쥴링과 배칭 알고리즘을 구현한다. 각 모델의 스케쥴러는 추론을 배치 형태로 수행하고, 요청을 모델 타입에 따른 backend 로 보낸다. Backend 에서 실제로 추론을 수행하고, 요청된 output 를 반환한다.

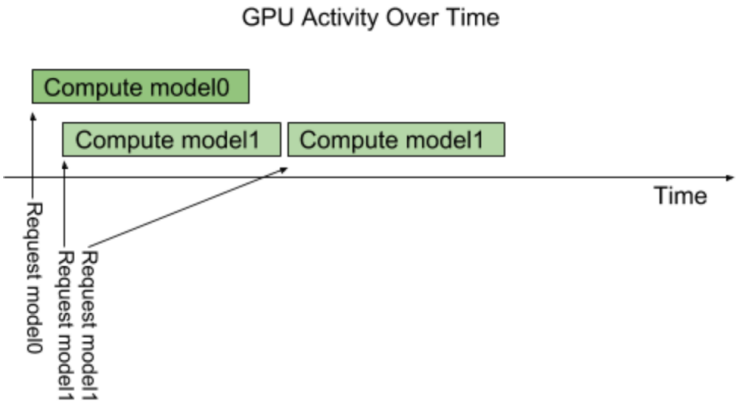
Triton 은 backend C API 를 지원하는데, 커스텀 pre- 또는 post-processing 이 가능하도록 한다. Triton 이 서빙하는 모델들은 model management API 를 통해 제어되고 쿼리될 수 있다. 여기에는 HTTP/REST 또는 GRPC 프로토콜을 포함한다.

2. Concurrent Model Execution

Triton 아키텍쳐는 여러 모델 또는 같은 모델을 가진 여러 개의 인스턴스가 병렬적으로 같은 시스템 내에서 실행되도록 한다. 시스템은 0,1,또는 여러 개의 GPU 를 가지고 있을 수 있다. 아래는 모델 0, 모델 1 이 실행되는 예시를 보여준다. Triton 이 현재 어떠한 요청도 처리하고 있지 않다고 가정할 때, 두개의 요청이 동시에 도착하면, Triton 은 두개를 GPU 에 스케쥴링하고 GPU 의 하드웨어 스케쥴러는 두개의 연산을 병렬적으로 처리하기를 시작한다. 시스템의 CPU 에서 실행하는 것도 비슷하게 처리되는데, CPU 스레드의 스케쥴링이 시스템의 OS 에 의해 처리된다는 것만 다르다.

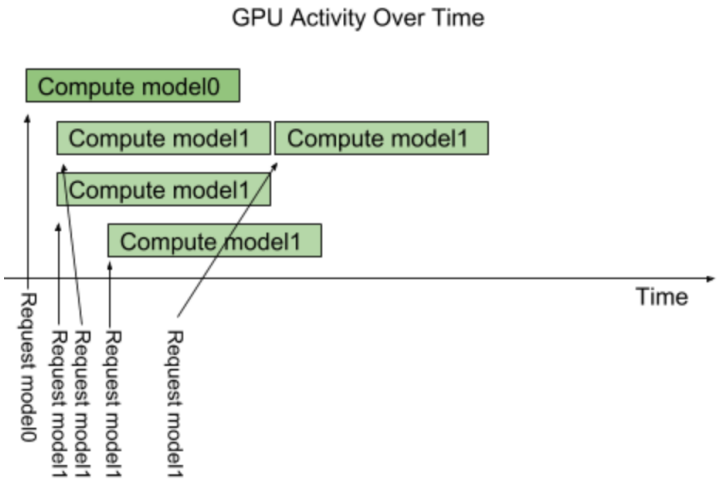


기본적으로, 같은 모델에 대한 여러 개의 요청이 동시에 도착하면, Triton 은 각 요청을 한번에 하나만 GPU 에 순차적으로 스케쥴링한다.



Triton 은 instance-group 이라는 모델 config 옵션을 제공하는데, 각 모델에 대해 얼만큼의 병렬 실행을 가능하게 할 것인지 설정할 수 있다. 이렇게 병렬적으로 실행되는 것을 instance 라고 한다. 기본적으로, Triton 은 시스템에 사용가능한 GPU 별로, 각 모델 별 single instance 를 준다. 모델 config 옵션의 instance_group 를 설정해서 이 개수를 설정할 수 있다. 아래는 model1 에 3 개의 instances 를 설정한 것이다.

Model1 의 처음 3 개 추론 요청은 병렬적으로 처리된다. 네번째 model1 추론은 3 개의 요청들이 끝나기를 기다려야 한다.



3. Models and Schedulers

Trion 은 각 모델에 따라 선택할 수 있는 여러 개의 스케쥴링과 배칭 알고리즘을 제공한다. 아래는 stateless, stateful, ensemble models 가 Triton 의 스케쥴링 알고리즘을 활용하는 방법에 대해 소개한다. 주어진 모델에 따라, 어떤 스케쥴러를 사용할지는 모델의 config 파일에서 설정할 수 있다.

3.1 Stateless Models

Stateless model 은 추론 요청 사이에 어떠한 상태도 유지하지 않는다. 각 추론은 다른 추론 요청들에게 독립적이다. Stateless model 의 예시로는 이미지 분류와 탐지를 하는 CNN 모델이 있다. Default scheduler 과 dynamic batcher 가 이러한 stateless model 을 위한 스케쥴러로 사용될 수 있다.

3.2 Stateful Models

Stateful model 은 추론 요청 사이에 상태를 유지한다. 이때는 sequence batcher 를 사용할 수 있다. Sequence batcher 는 sequence 내에 있는 모든 추론 요청들이 같은 모델 instance 로 들어가도록 보장할 수 있다. Sequence batcher 는 모델과 소통해서 시퀀스가 언제 시작하고 끝나는지 알려줄 수 있다.

3.2.1 Control Inputs

Stateful model 이 sequence batcher 과 올바르게 작동하려면, 모델은 Triton 이 모델과 소통하기 위해서 사용하는 한 개 이상의 control input tensors 를 인풋으로 받아야 한다. 모델 config 의 ModelSequenceBatching::Control 은 모델이 sequence batcher 를 위해 어떠한 텐서를 사용하는지 보여줄 수 있다. 모든 control 은 선택이다.

3.2.2 Implicit State Management

Implicit State Management 는 모델이 Triton 내부에서 상태를 유지할 수 있도록 해준다. Implicit state 를 사용할 때, 모델은 추론을 위한 상태를 모델 내부에 저장하지 않아도 된다.

아래는 model config 의 일부로, 모델이 implicit state 를 사용하는 것을 나타낸다.

```
sequence_batching {
  state [
    {
      input_name: "INPUT_STATE"
      output_name: "OUTPUT_STATE"
      data_type: TYPE_INT32
      dims: [ -1 ]
    }
  ]
}
```

3.2.3 State Initialization

기본적으로, 시퀀스 내에 있는 첫번째 요청은 input state 의 초기화되지 않은 데이터를 포함하고 있다. 모델은 start flag 을 사용해서 새로운 시퀀스의 시작을 감지할 수 있고, 이것을 이용해 model state 를 초기화할 수 있다. Triton 은 이외에도 2 가지의 초기화 방법을 제공한다.

3.2.3.1 Initializing State from Zero.

아래는 초기 상태를 0 으로 초기화하는 설정이다.

```
sequence_batching {
  state [
    {
      input_name: "INPUT_STATE"
      output_name: "OUTPUT_STATE"
      data_type: TYPE_INT32
      dims: [ -1 ]
      initial_state: {
       data_type: TYPE_INT32
       dims: [ 1 ]
       zero_data: true
       name: "initial state"
      }
    }
  ]
}
```

3.2.3.2 Initializing State from File

파일로부터 상태를 초기화할 수 도 있다. 모델 디렉토리 아래 “initial_state” 디렉토리를 생성해야 한다.

```
sequence_batching {
  state [
    {
      input_name: "INPUT_STATE"
      output_name: "OUTPUT_STATE"
      data_type: TYPE_INT32
      dims: [ -1 ]
      initial_state: {
       data_type: TYPE_INT32
       dims: [ 1 ]
       data_file: "initial_state_data"
       name: "initial state"
      }
    }
  ]
}
```

3.2.4 Scheduling Strategies

Sequence Batcher 는 같은 모델 instance 에 할당된 시퀀스들을 배치하기 위해 2 가지 스케쥴링 전략 중 하나를 사용할 수 있다. 두가지 전략은 direct 와 oldest 이다.

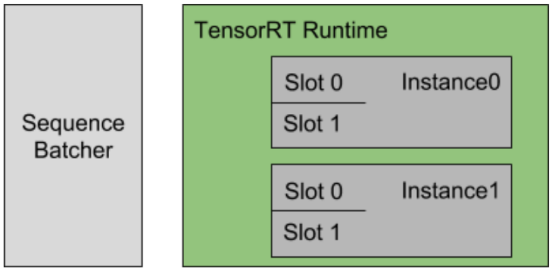
3.2.4.1 Direct

Direct 스케쥴링 전략은 sequence batcher 가 모든 추론 요청을 같은 model instance 에 라우팅되도록 할 뿐만 아니라, 각 시퀀스가 model instance 내에 지정된 batch 내에 라우팅되도록 한다. 이 전략은 모델이 각 batch slot 마다 상태를 유지할 때 사용된다.

아래 TensorRT stateful model config 의 예시를 보자.

```
name: "direct_stateful_model"
platform: "tensorrt_plan"
max_batch_size: 2
sequence_batching {
  max_sequence_idle_microseconds: 5000000
  direct { }
  control_input [
    {
      name: "START"
      control [
        {
          kind: CONTROL_SEQUENCE_START
          fp32_false_true: [ 0, 1 ]
        }
      ]
    },
    {
      name: "READY"
      control [
        {
          kind: CONTROL_SEQUENCE_READY
          fp32_false_true: [ 0, 1 ]
        }
      ]
    }
  ]
}
input [
  {
    name: "INPUT"
    data_type: TYPE_FP32
    dims: [ 100, 100 ]
  }
]
output [
  {
    name: "OUTPUT"
    data_type: TYPE_FP32
    dims: [ 10 ]
  }
]
instance_group [
  {
    count: 2
  }
]
```

여기서 sequence_batching 부분을 보면, 모델이 sequence batcher 를 사용하고 direct scheduling 전략을 사용해야 함을 알 수 있다. 이 예시에서 모델은 sequence batcher 로부터 start 와 ready control input 만 필요하므로 이것만 명시하고 있다. Instance_group 를 통해서 2 개의 모델 인스턴스가 초기화되야 하고 max_batch_size 를 통해 각 인스턴스는 batch-size 2 인 추론을 수행해야 함을 알 수 있다. 아래는 이러한 config 를 통해 실행되어야 하는 sequence batcher 과 inference resources 를 나타낸다.



각 모델 인스턴스는 각 batch slot 마다 상태를 유지하며, 상태가 올바르게 업데이트되기 위해 주어진 시퀀스에 있는 모든 추론 요청들은 같은 slot 으로 라우팅 되기를 예상한다. 이 예시에서는 Triton 은 최대 4 개의 시퀀스까지 추론을 동시에 할 수 있다는 것을 의미한다.

3.2.4.2 Oldest

Oldest scheduling strategy 는 모든 추론 요청이 같은 모델 인스턴스로 라우팅되고, dynamic batcher 를 사용해서 다른 시퀀스에서 온 여러 개의 추론 요청을 함께 배치하여 처리한다. 이 전략으로는 모델은 CONTROL_SEQUENCE_CORRID 를 사용해서 배치안에 있는 요청이 어느 시퀀스에 속하는지 알 수 있다.

3.2.5 Ensemble Models

Ensemble model 은 하나 이상의 모델과 그 모델들 사이의 입력 및 출력 tensor 의 연결을 나타내는 파이프라인을 의미한다. Ensemble model 은 "데이터 전처리->추론->데이터 후처리"와 같은 여러 모델을 포함하는 절차를 캡슐화하는데 사용된다. 이러한 목적으로 Ensemble model 을 사용하면 중간 tensor 를 전송하는 오버헤드를 피하고 Triton 에 전송해야 하는 요청 수를 최소화할 수 있다.

Ensemble model 을 위해서는 ensemble scheduler 를 사용해야 한다. 모델 config 에서 ModlelEnsembling::Step 항목으로 모델 간의 데이터 흐름을 지정한다.

4. Reference

- https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html#ensemble-models