Inference profiler

지능형 추론 서비스 프로파일링 프로그램

문서 기술의 목적

본 문서는 Inference profiler – 지능형 추론 서비스 프로파일링 프로그램의 세부 구성 및 사용 매뉴얼 등을 상세히 설명한 문서이다.

Inference profiler 는 triton inference server 를 사용한 지능형 추론 작업에서 구간별 추론 스택과 OS 커널의 전계층을 프로파일링 할 수 있는 프로파일러이다. 커널 스택 분석을 위한 ebpf 및 bpftrace 를 활용한 기법과 NVIDIA triton inference server 의 프로파일링 도구를 활용하여 통합 프로파일링 인터페이스를 제공한다.

목차

1. eBPF 및 bpftrace 를 활용한 프로파일링

엣지 환경에서 수행되는 대표적인 지능형 컴퓨팅 작업 심층신경망 추론 서비스는 다양한 소프트웨어 및 커널 계층을 거침에 따라 분석 복잡도가 높고, 전 계층에 대한 단일 분석 도구의 부재로 각 계층의 개별 분석 도구들을 활용해야 한다. 현재 추론 서비스의 소프트웨어 스택을 구성하는 GPU 컴퓨팅 라이브러리, 추론 서빙 시스템, 그리고 리눅스 커널에서는 개별적인 분석 도구들을 제공하고 있다. 지능 컴퓨팅 작업이 거치는 각 구간의 실행시간 측정 및 지연 원인 분석을 위해서는 이러한 개별적인 분석 도구들을 활용하여 통합 프로파일링 인터페이스를 개발해야 한다. 이러한 통합 프로파일러 인터페이스 개발을 위하여 먼저 리눅스 커널에서 제공하고 있는 분석 도구인 eBPF 및 bpftrace 활용하여 프로파일링을 진행하였다.

1-1. eBPF & bpftrace

■ eBPF 의 정의

BPF 는 Berkeley packet filter 의 약자로 Unix 계열 OS 의 Kernel Level 에서 Bytecode 에 따라 동작하는 경량화 된 Virtual Machine 이다. 처음에는 packet filter 라는 이름 그대로 Network Packet 을 Filtering 하는 Program 을 구동하는 용도로 사용되었다. 하지만 사용자가 원하는 Program 을 언제든지 Kernel Level 에서 구동 할 수 있다는 장점 때문에 BPF 는 꾸준히 발전하게 되었고, 현재는 다양한 기능을 수행할 수 있게 되었다.

BPF 가 다양한 기능을 수행하게 되면서 기존의 매우 제한적인 Resource 들이 큰 걸림돌이 되었다. 이러한 문제점을 해결하기 위해서 Linux 는 더 많은 Resource 와 기능을 이용할 수 있는 Extened BPF, 즉 ebpf 를 정의하였다.

그림 1. eBPF

■ **eBPF 의 동작 원리**

그림 2 는 eBPF Program 의 Compile 과정과 bpf() System Call 의 동작을 나타내고 있다. 개발자가 eBPF Source Code 를 작성하면 LLVM/clang 을 통해서 eBPF Bytecode 로 Compile 된다. 그 후 Bytecode 는 tc 나 iproute2 같은 어플리케이션을 통해 Kernel 의 eBPF 에 적재되는데, 이 과정에서 bpf 시스템콜을 이용하고 있다.

eBPF Bytecode 는 Kernel Level 에서 동작하기 때문에 system 전체에 영향을 줄 수 있는 위험한 상황을 방지하기 위하여 Verifier 로 이상이 없는지 검사한후 커널에 적재하도록 설계되어 있다. Verifier 는 Bytecode 가 허용되지 않은 Memory 영역을 참조하거나 무한 Loop 가 발생하는 경우가 있는지도 검사한다. 또한 허용되지 않은 Kernel Helper Function 을 호출했는지도 검사한다. 검사를 통과한 Bytecode 는 eBPF 에 적재되어 동작하는데, 필요에 따라 일부는 JIT (Just-in-time) Compiler 를 통해서 Native Code 로 변환되어 Kernel 에서 동작하게 된다.

bpf() System Call 은 Bytecode 를 적재하는 것뿐 아니라 어플리케이션이 Map 에 접근할 수 있게 만들어준다. 따라서 어플리케이션과 eBPF 는 Map 을 이용하여 통신을 할 수 있고, 이를 활용하여 더욱 다양한 기능을 수행할 수 있다.

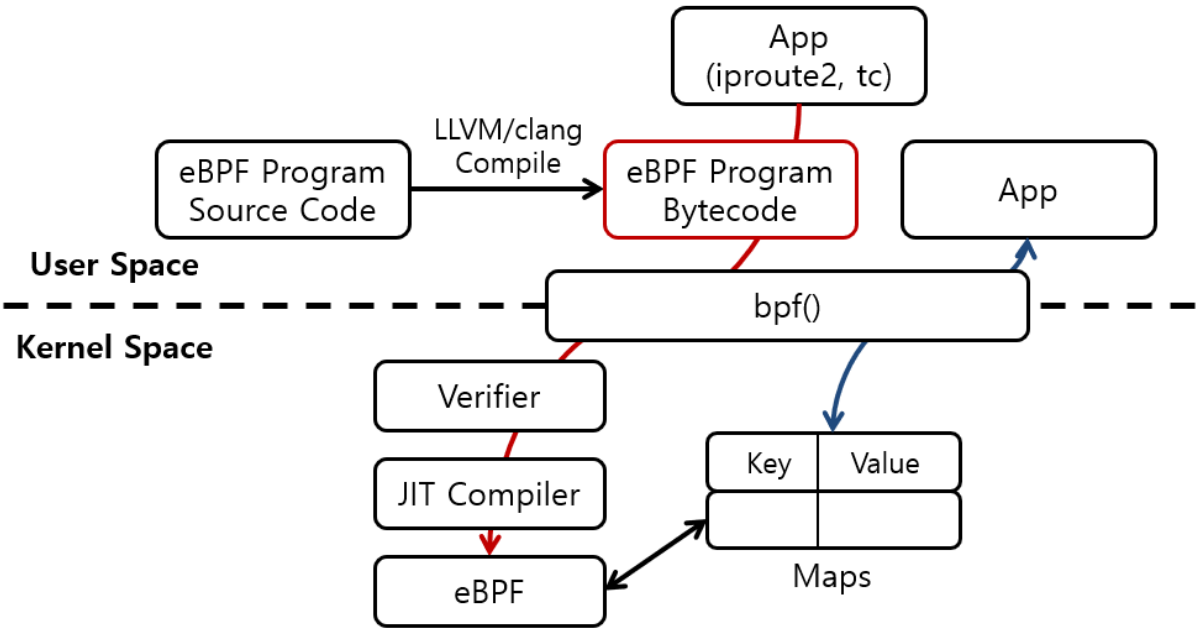

그림 2. eBPF 동작 원리

■ **bpftrace 의 정의**

bpftrace 는 커널 트레이스를 할 때 사용하는 ebpf 를 활용한 프레임워크로 eBPF 에서 커널 트레이싱을 할 수 있도록 별로도 정의된 고수준의 언어이다. bpftrace 는 LLVM 을 백엔드로 사용하여 BPF-bytecode 로 스크립트를 컴파일하고, BCC 를 활용하여 bpf 시스템과 상호작용할 수 있다.

Bpftrace 는 probe 라는 지점을 추적하는 방법으로 커널 영역을 분석한다. Probe 는 소프트웨어나 하드웨어에서 bpf 프로그램을 실행할 수 있는 이벤트를 생성하는 지점을 말한다.

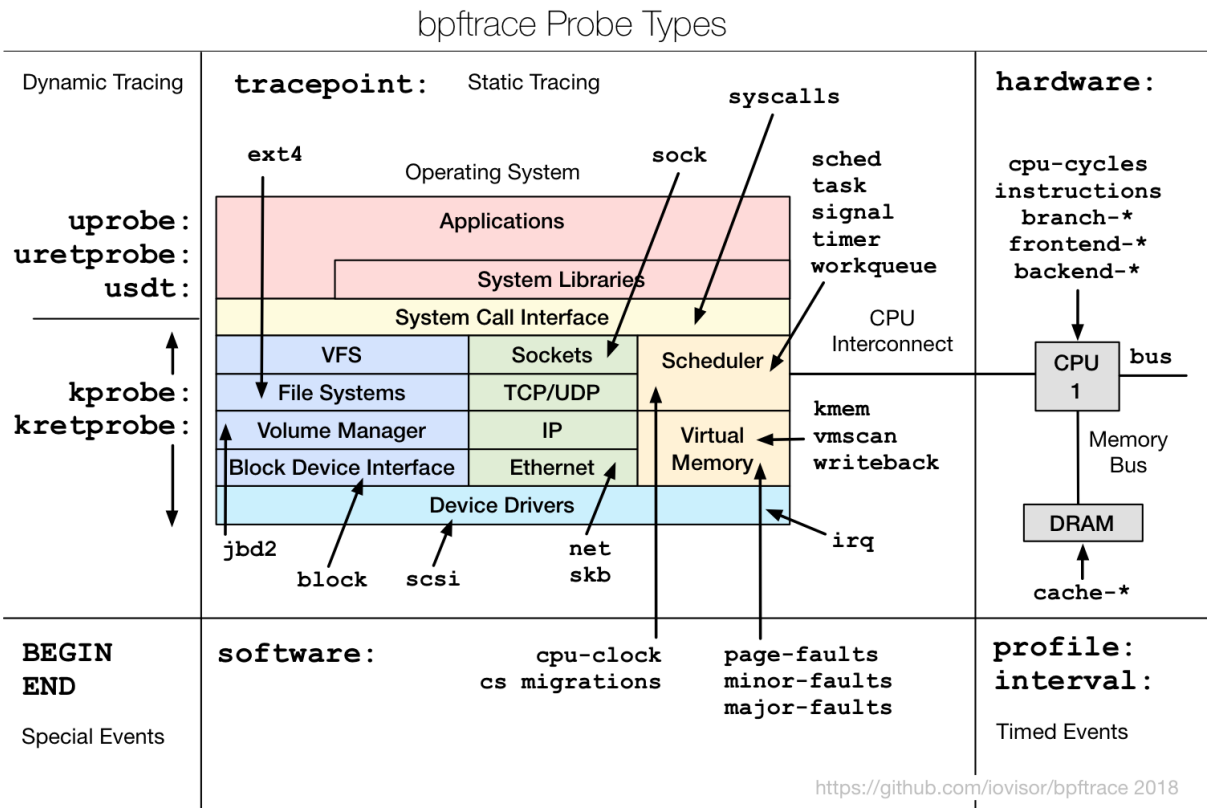

그림 3. Bpftrace probe types

그림 3 은 Bpftrace 에서 제공되는 probe 이다. 유저 영역을 트레이싱할 수 있는 uprobe, 커널영역을 트레이싱할 수 있는 kprobe, 하드웨어, 소프트웨어 등의 probe 를 통해 여러가지 이벤트를 추적할 수 있다. Bpftrace 코드를 작성할 때는

추적하고자 하는 probe 를 지정하고, 해당 이벤트가 실행될 경우 수행할 코드를 작성한다.

■ **bpftrace 설치**

ebpf_trace.bt 프로그램을 실행하기 위해서는 bpftrace 를 설치해야 한다. Ubuntu OS 에서는 다음의 명령어를 통해 bpftrace 를 설치할 수 있다.

```
$ sudo apt-get install -y bpftrace
```

1-2. 코드 설명 (ebpf_trace.bt)

```
#!/usr/bin/env bpftrace
#include <linux/socket.h>
#include <net/sock.h>
BEGIN
{
  printf("Tracing tcp send/recv. Hit Ctrl-C to end.\n");
  printf("%-8s %-8s %-16s %35s %48s %20s\n", "TIME", "PID", "COMM",
"SADDR:SPORT", "DADDR:DPORT", "TIME(ns)");
}
kprobe:tcp_sendmsg
{
  $sk = ((struct sock *) arg0);
  $inet_family = $sk->__sk_common.skc_family;
  if ($inet_family == AF_INET || $inet_family == AF_INET6) {
    if ($inet_family == AF_INET) {
      $daddr = ntop($sk->__sk_common.skc_daddr);
      $saddr = ntop($sk->__sk_common.skc_rcv_saddr);
    } else {
      $daddr = ntop($sk->__sk_common.skc_v6_daddr.in6_u.u6_addr8);
      $saddr = ntop($sk->__sk_common.skc_v6_rcv_saddr.in6_u.u6_addr8);
    }
    $lport = $sk->__sk_common.skc_num;
    $dport = $sk->__sk_common.skc_dport;
    // Destination port is big endian, it must be flipped
    $dport = ($dport >> 8) | (($dport << 8) & 0x00FF00);
    $state = $sk->__sk_common.skc_state;
    time("%H:%M:%S ");
    printf("%-8d %-16s ", pid, comm);
    printf("%39s:%-6d %39s:%-6d", $saddr, $lport, $daddr, $dport);
    printf("\t%-10u",elapsed);
    printf("    *SEND\n")
  }
}
kprobe:tcp_recvmsg
{
```

```
  $sk = ((struct sock *) arg0);
  $inet_family = $sk->__sk_common.skc_family;
  if ($inet_family == AF_INET || $inet_family == AF_INET6) {
    if ($inet_family == AF_INET) {
      $daddr = ntop($sk->__sk_common.skc_daddr);
      $saddr = ntop($sk->__sk_common.skc_rcv_saddr);
    } else {
      $daddr = ntop($sk->__sk_common.skc_v6_daddr.in6_u.u6_addr8);
      $saddr = ntop($sk->__sk_common.skc_v6_rcv_saddr.in6_u.u6_addr8);
    }
    $lport = $sk->__sk_common.skc_num;
    $dport = $sk->__sk_common.skc_dport;
    // Destination port is big endian, it must be flipped
    $dport = ($dport >> 8) | (($dport << 8) & 0x00FF00);
    $state = $sk->__sk_common.skc_state;
    time("%H:%M:%S ");
    printf("%-8d %-16s ", pid, comm);
    printf("%39s:%-6d %39s:%-6d", $saddr, $lport, $daddr, $dport);
    printf("\t%-10u",elapsed);
    printf("    *RECV\n")
  }
}
```

NVIDIA 의 triton inference server 는 client-server 구조로 되어있으며, 네트워크 통신을 통해 작동하기 때문에 추론 작업에서의 구간별 분석을 위해서는 네트워크 분석이 필수적이다. ebpf_trace.bt 는 커널 함수인 tcp_sendmsg, tcp_recvmsg 를 추적하여 네트워크 통신 상황을 확인할 수 있도록 한다. tcp 통신이 진행될 때의 각 시간을 분석하여 clinent-server 간 통신 시간 등을 확인할 수 있다.

먼저 begin 을 활용해 프로그램 시작 시 한번 안내 메세지를 출력한다. 그 후로는 kprobe:tcp_sendmsg 를 사용하여 메시지를 전송하는 순간의 시간, 프로세스 아이디, 커멘드 이름, 아이피 주소, 프로그램 시작으로부터 흐른 시간, send 표시를 출력한다. 다음으로는 kprobe:tcp_recvmsg 를 사용하여 메시지를 수신하는 순간에 동일한 정보를 출력한다. 마지막으로는 recv 표시를 출력해 메시지를 수신한 상황이라는 것을 확인할 수 있도록 한다.

2. NVIDIA triton inference server 를 활용한 프로파일링

추론 서비스의 종단간 지연 시간을 분석하기 위해서는 추론 서빙 시스템 뿐만 아니라 클라이언트와 서버 간의 지연을 분석해야 한다. 따라서 본 연구에서 개발한 통합 프로파일링 인터페이스는 Triton Inference Server 의 자체 trace 기능과 Performance Analyzer 를 기반으로 구간별 지연 시간을 추출한다.

2-1. NVIDIA triton inference server

- **NVIDIA triton inference server 의 정의**

  최근 IoT 단말기 수의 증가에 따라, 데이터 센터의 네트워크 대역폭이 한계에 다다르게 되었고, 요청 작업의 SLO(Service Level Objective)를 준수하기 어려워졌다. 이에 대한 해결책으로 엣지 컴퓨팅 기술이 주목받고 있는데, 이에 따라 지능형 서비스의 추론 작업 또한 클라우드 컴퓨팅에서 엣지 컴퓨팅으로 이동하게 되었다. 대표적인 GPU 공급 업체 Nvidia 는 AI 엣지 컴퓨팅 인프라를 구축하기 위한 Nvidia EGX AI 플랫폼을 출시하였으며 Triton Inference Server 를 개발하여 다양한 프레임워크들에 대한 통합적인 추론 서빙을 지원하도록 하였다.

  Nvidia Triton Inference Server 에서 사용자의 추론 요청 및 결과는 그림 4 와 같이 Linux, Docker, Triton 의 네트워크 스택 경로를 따라 사용자의 NIC(Network Interface Card)부터 엣지 서버의 DRAM, GPU 메모리를 거치게 되며, 추론 요청에 따라 적합한 모델이 원격 저장소로부터 NIC 로 이동하여 엣지 서버의 DRAM, 그리고 GPU 메모리로 로드되어 추론을 수행한다.

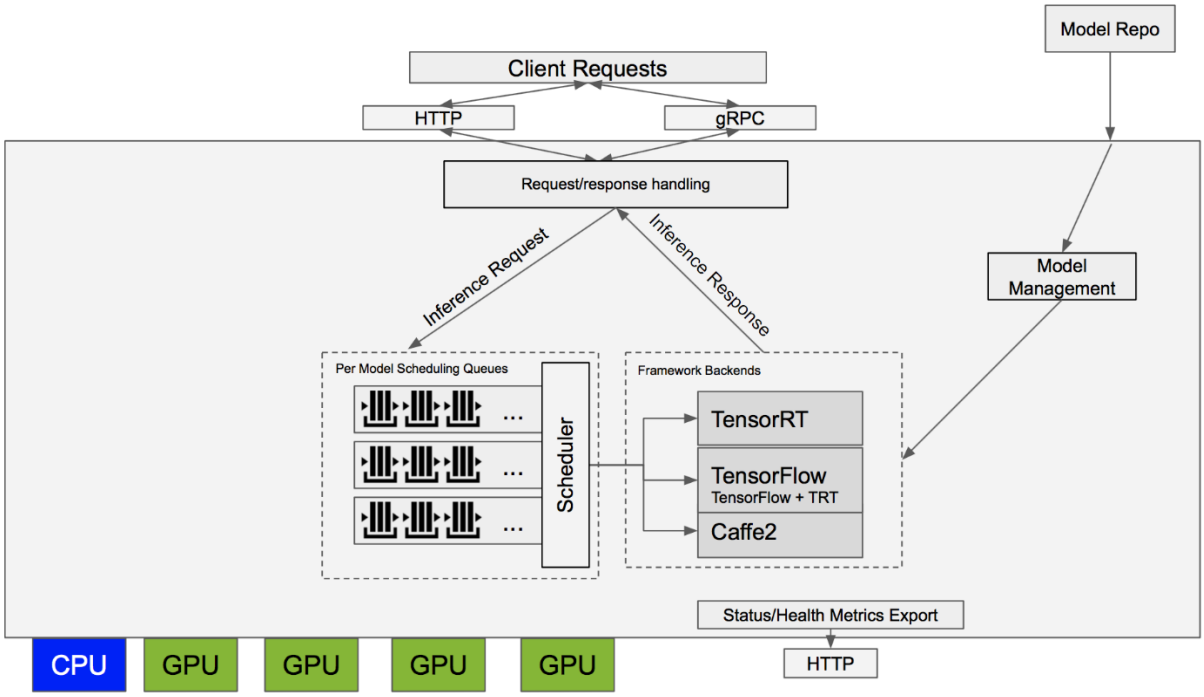

그림 4. Triton inference server 구조

■ **NVIDIA triton inference server 의 trace 기능**

NVIDIA triton inference server 에서는 추론 과정에서의 구간 분석을 위한 프로파일링 기능을 제공한다. Triton server 실행 시 다음의 옵션을 추가하여 프로파일링 기능을 활성화할 수 있다. --trace-file 옵션에서 지정해준 경로에 json 형식의 파일으로 프로파일링 결과가 저장된다.

```
$ tritonserver --trace-file=/tmp/trace.json --trace-rate=100 --trace-level=MAX ...
```

■ **Performance Analyzer를 통한 사용자 관점 지연시간 측정**

Performance Analyzer는 특정 모델에 대한 추론 요청을 생성하고 요청의 처리량과 지연 시간을 측정할 수 있는 도구이다. 추론 요청을 다양한 확률 분포로 생성 가능하며, 동시 요청, 요청 비율 등을 조절하여 실제 서비스 환경을 시뮬레이션 가능하다. 또한 실제 원격 Triton Inference Server와 통신을 통해 사용자 관점에서 HTTP/gRPC 통신을 포함하는 추론 요청 및 반환에 대한 지연 시간을 관측할 수 있다.

2-2. 코드 설명 (triton_trace_parser.py)

```
#!/usr/bin/python3

import json
import operator

def trace(json_data):

    time = {}
    timesum = {}
    num = 1
    count = 0
    starttime = 0
    for data in json_data:
        if data["id"]==num+1:
            if "GRPC_WAITREAD_START" in time:
                count += 1
                time.clear()
            num += 1
            if num == 2+count:
                timesum = time.copy()
            else:
                for k,v in time.items():
                    timesum[k] += v
            time.clear()

        if data["id"]==num and "timestamps" in data:
            timestamps = data["timestamps"]
            for timestamp in timestamps:
                if timestamp["name"]=="HTTP_RECV_START":
                    starttime = timestamp["ns"]
                time[timestamp["name"]] = timestamp["ns"]- starttime

    if num == 1+count:
        timesum = time.copy()
    else:
        for k,v in time.items():
            timesum[k] += v

    stime= sorted(timesum.items(), key=operator.itemgetter(1))
    print_func(stime,num-count)



def print_func(stime,num):

    value = []
    for i in range(10):
        value.append(stime[i+1][1]-stime[i][1])

    ### print ###
```

```
    flag=True

    print()
    for t in stime:
        print('%-21s'%(t[0]), end="")
    print()

    for t in stime:
        print('%-21s'%(str(t[1]//num)+" ns"), end="")
    print()
    for i in stime:
        print("----+----------------",end="")
    print(">")
    for v in value:
        if flag:
            print('%17s ns'%(v//num), end="")
            flag=False
            continue
        print('%18s ns'%(v//num), end="")
    print("\n")

    print("time")
    for t in stime:
        print('%s'%(str(t[1]//num)))
    print("\n")
    print("interval")
    for v in value:
        print('%s'%(v//num))
    print("\n")



if __name__ == '__main__':

    with open("/tmp/trace.json", "r") as f:
        contents = f.read()
        json_data = json.loads(contents)

    trace(json_data)
```

triton_trace_parser.py 를 통해 triton inference server 에서 제공하는 자체 trace 기능을 효율적으로 활용할 수 있다. triton inference server 의 trace 기능을 사용하여 구간별 지연 시간을 json 형식의 파일로 저장한 후 위의 프로그램을 실행하면 위의 json 파일을 파싱 하여 정보를 쉽게 확인할 수 있도록 분석해 준다.

Trace 함수를 통해 json 파일의 데이터를 활용하여 구간별 지연시간을 계산한다. 이후 print function 을 통해 정보를 시각화 하여 출력한다.

3. Inference profiler 의 시각화

본 연구에서 개발한 통합 프로파일러 인터페이스는 여러가지 개별적인 분석 도구들을 활용하였기 때문에, 통합적으로 시각화 하는 프로그램이 필요하다. 따라서 matplotlib 라이브러리를 활용하여 위에서 분석된 자료를 시각화 한다.

3-1. Matplotlib

■ **Matplotlib 의 정의**

Matplotlib 는 Python 프로그래밍 언어 및 수학적 확장 NumPy 라이브러리를 활용하여 데이터를 차트나 플롯(plot)으로 그려주는 라이브러리 패키지로서 가장 많이 사용되는 데이터 시각화(Data Visualization) 패키지로 알려져 있다. Matplotlib 는 라인 플롯, 바 차트, 파이차트, 히스토그램, Box Plot, Scatter Plot 등을 비롯하여 다양한 차트와 플롯 스타일을 지원한다.

■ **Matplotlib 의 사용**

Matplotlib 의 pyplot 모듈을 사용하여 간편하게 그래프를 만들고 변화를 줄 수 있다. 예를 들어, 그래프 영역을 만들고, 몇 개의 선을 표현하고, 레이블로 꾸미는 등의 일을 할 수 있다.

3-2. 코드 설명 (inference_profiler_visualizer.ipynb)

```
import matplotlib.pyplot as plt
from datetime import date
import numpy as np

%matplotlib inline

dates = [0,1,2,3,4,5]
labels = ['Start','Network+Server Send/Recv','Server Queue','Server Compute
Input','Server Compute Infer','Server Compute Output']
# Labels with associated dates

data_f = open("/content/test2.csv")

temp =[]

flag = 0
for line in data_f:
```

```
    if flag == 0:
        flag += 1
        continue
    temp = line.split(',')

date =[]
sum = 0
temp[2] = '0'
for i in range(6):
    date.append(sum + int(temp[i+2]))
    sum = date[i]
temp = temp[3:8]
print(temp)
data_f.close()

for i in range(len(date)):
    labels[i] = labels[i] + "\n" + str(date[i]) + " usec"

font1 = {'family': 'serif','color':  'dimgrey',
        'weight': 'normal','size': 11}

fig, ax = plt.subplots(figsize=(16, 4), constrained_layout=True)
ax.set_ylim(-2, 1.75)
ax.axhline(0, xmin=0.05, xmax=0.95, c='grey', zorder=1)
ax.scatter(dates, np.zeros(len(dates)), s=120, c='darkgrey', zorder=2)
ax.scatter(dates, np.zeros(len(dates)), s=30, c='black', zorder=3)

label_offsets = np.zeros(len(dates))
label_offsets[::2] = 0.7
label_offsets[1::2] = -1
for i, (l, d) in enumerate(zip(labels, dates)):
    ax.text(d, label_offsets[i], l, ha='center', fontfamily='serif',
fontweight='bold', color='black',fontsize=12)

stems = np.zeros(len(dates))
stems[::2] = 0.5
stems[1::2] = -0.5
markerline, stemline, baseline = ax.stem(dates, stems,
use_line_collection=True)
plt.setp(markerline, marker=',', color='grey')
plt.setp(stemline, color='grey')

for spine in ["left", "top", "right", "bottom"]:
    _ = ax.spines[spine].set_visible(False)

ax.set_xticks([])
ax.set_yticks([])

ax.set_title('Inference profiler visualizer', fontweight="bold",
fontfamily='serif', fontsize=18, color='black')

i = 0.4
```

```
for time in temp:
    plt.text(i, 0.1, str(time)+" usec", fontdict=font1)
    i += 1
```

inference_profiler_visualizer.ipynb 를 통해 추론 스택 분석 결과를 시각화 할 수 있다. Triton inference server 의 클라이언트 프로그램으로 perf_analzer 를 사용하였을 경우, 구간별 지연 시간 분석 결과를 csv 형식의 파일로 저장할 수 있다. 위의 프로그램을 실행하면 해당 csv 파일을 시각화 할 수 있다.

위의 프로그램은 matplotlib 의 stem plot 을 활용하여 타임라인 형태로 정보를 출력할 수 있도록 했다. 시간 정보를 계산하여 출력하였기 때문에 각 구간별 소요시간 또한 확인할 수 있다.

4. 프로그램 작동 방법

- **ebpf_trace.bt**

ebpf_trace.bt 는 bpftrace 를 사용하여 작성된 프로그램이다. 실행을 위해서는 먼저 bpftrace 를 설치해야 한다. bpftrace 의 설치가 완료되었다면 다음의 명령어를 통해 프로그램을 실행할 수 있다.

```
$ sudo ./ebpf_trace.bt
```

- **triton_trace_parser.py**

NVIDIA tritonserver 실행 시 다음의 옵션을 추가하여 프로파일링 기능을 활성화한다. --trace-file 옵션을 통해 결과가 저장될 경로를 지정한다.

```
$ tritonserver --trace-file=/tmp/trace.json --trace-rate=100 --trace-level=MAX
```

tritonserver 를 실행한 후 클라이언트에서 요청한 추론 작업을 완료하면 서버를 종료시킨다. 그 후 다음의 명령어를 실행하면 json 파일로 저장되어 있는 정보를 분석하여 최종 프로파일링 결과가 출력된다.

```
$ python3 triton_trace_parser.py
```

- **inference_profiler_visualizer.ipynb**

주피터 노트북 또는 구글의 colab 을 활용하여 위의 프로그램을 실행한다.

5. 결과 예시

■ ebpf_trace.bt 실행 결과

```
Attaching 4 probes...
Tracing tcp send/recv. Hit Ctrl-C to end.
TIME     PID    COMM                    SADDR:SPORT              DADDR:DPORT        TIME(ms)
17:59:47 54643  image_client            127.0.0.1:33782          127.0.0.1:8000     2033   *SEND
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58736   2033   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58736   2033   *SEND
17:59:47 54643  image_client            127.0.0.1:33782          127.0.0.1:8000     2033   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58736   2033   *RECV
17:59:47 54643  image_client            127.0.0.1:33786          127.0.0.1:8000     2033   *SEND
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58740   2033   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58740   2034   *SEND
17:59:47 54643  image_client            127.0.0.1:33786          127.0.0.1:8000     2034   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58740   2034   *RECV
17:59:47 54643  image_client            127.0.0.1:33790          127.0.0.1:8000     2080   *SEND
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
17:59:47 54643  image_client            127.0.0.1:33790          127.0.0.1:8000     2081   *SEND
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
17:59:47 54498  tritonserver            172.17.0.2:8000          172.17.0.1:58744   2081   *RECV
```

■ Triton_trace_parser.py 실행 결과

```
<vgg19> batch:1

HTTP_RECV_START  HTTP_RECV_END  REQUEST_START  QUEUE_START  COMPUTE_START  COMPUTE_INPUT_END  COMPUTE_OUTPUT_START COMPUTE_END  REQUEST_END  HTTP_SEND_START  HTTP_SEND_END
0 ns             534205 ns      554729 ns      557110 ns    565549 ns      629058 ns          5744489 ns           5748390 ns   5761488 ns   5763348 ns       5767295 ns
----+----------------+---------------+-------------+--------------+------------------+--------------------+------------+------------+----------------+-------------+-------------->
       534205 ns        20523 ns        2381 ns       8438 ns       63508 ns        5115431 ns          3900 ns      13098 ns      1859 ns        3947 ns


<vgg19> batch:64

HTTP_RECV_START  HTTP_RECV_END  REQUEST_START  QUEUE_START  COMPUTE_START  COMPUTE_INPUT_END  COMPUTE_OUTPUT_START COMPUTE_END  REQUEST_END  HTTP_SEND_START  HTTP_SEND_END
0 ns             43285441 ns    43322490 ns    43326847 ns  43349445 ns    67479399 ns        317569163 ns         317630032 ns 317661006 ns 317664383 ns     317672001 ns
----+----------------+---------------+-------------+--------------+------------------+--------------------+------------+------------+----------------+-------------+-------------->
       43285441 ns      37049 ns        4356 ns       22598 ns      24129953 ns     250089763 ns        60869 ns     30974 ns      3377 ns        7617 ns


<vgg19> batch:128

HTTP_RECV_START  HTTP_RECV_END  REQUEST_START  QUEUE_START  COMPUTE_START  COMPUTE_INPUT_END  COMPUTE_OUTPUT_START COMPUTE_END  REQUEST_END  HTTP_SEND_START  HTTP_SEND_END
0 ns             87095784 ns    87131091 ns    87135077 ns  87156883 ns    135543859 ns       855408896 ns         855498421 ns 855530140 ns 855534192 ns     855542920 ns
----+----------------+---------------+-------------+--------------+------------------+--------------------+------------+------------+----------------+-------------+-------------->
       87095784 ns      35307 ns        3985 ns       21806 ns      48386975 ns     719865037 ns        89525 ns     31718 ns      4051 ns        8728 ns
```

■ Inference_profiler_visulalizer.ipynb 실행 결과