Step-by-Step Guide to *gpulet*

Seungbeom Choi*
sb1024.choi@samsung.com
Samsung SDS
(This work was done during Ph.D in KAIST)

Sunho Lee
myshlee417@casys.kaist.ac.kr
School of Computing, KAIST

December 26th, 2024

Contents

1 Introduction

To maximize the resource efficiency of inference servers, we proposed a key mechanism to exploit hardware support for spatial partitioning of GPU resources. With the partitioning mechanism, a new abstraction layer of GPU resources is created with configurable GPU resources. The scheduler assigns requests to virtual GPUs, called ***gpulet***, with the most effective amount of resources. The prototype framework auto-scales the required number of GPUs for a given workloads, minimizing the cost for cloud-based inference servers. The prototype framework also deploys a remedy for potential interference effects when two ML tasks are running concurrently in a GPU.

2 Evaluated Environment

The prototype was evaluated with multi-GPU server with the following OS/software and hardware:

2.1 OS/Software

- Ubuntu 18.04
- Linux Kernel 4.15
- CUDA 10.2
- cuDNN 7.6
- PyTorch 1.10

2.2 Hardware

- RTX 2080ti (11GB global memory)
- Intel Xeon E5-2630 v4
- Servers connected with 10 GHz Ethernet

3 Getting Started

3.1 Prerequisites

Install the following libraries and drivers to build the prototype.

- LibTorch (PyTorch library for C++) = 1.10
- CUDA $\geq$ 10.2
- CUDNN $\geq$ 7.6
- Boost $\geq$ 1.6 (script provided)
- opencv $\geq$ 4.0 (script provided)
- cmake $\geq$ 3.19, use cmake to build binaries (script provided)
- pandas (optional, only required for analyzing results)

3.2 Prerequisites for Running Docker Image

- Docker version $\geq$ 20
- nvidia-docker, for utilizing NVIDIA GPUs.

3.3 Step-by-step building instructions

1. Download LibTorch (https://pytorch.org/cppdocs/installing.html) and extract all the content as LibTorch under the root directory of this repository. (example: *glet/libtorch*)
2. Go to *scripts/*.
```
$ cd scripts
```
3. Execute *build_all.sh*. The script will use cmake to auto-configure build environments and build binaries.
```
$ ./build_all.sh
```

4 Running Examples

Below are step-by-step examples for running the server and standalone components.

4.1 Executing Inference on a Single Local GPU

Use *execLocal.sh* to execute ML inference on local GPU. Useful for testing whether you have installed compatible SW stack and profiling latency.

1. Make sure you have downloaded models you want to execute and store them under *resource/models/*.

2. Go to *scripts/*.
   ```
   $ cd scripts
   ```

3. Execute *execLocal.sh* with parameters: 1) name of model you want to execute, 2) number of executions 3) batch size 4) interval between executions 5) (optional) percentage of computing resource.
   ```
   $ (example) ./execLocal.sh resnet50 1000 1 0.1 50
   ```

4.2 Executing Offline Scheduler

Highly recommended that you replace example profile files with profile info on the platform you wish to execute before you use the offline scheduler.

The offline scheduler is used for generating static scheduling results (default file is *ModelList.txt* but the name of the file can be customized with the *−output* argument). The static results are used as a guideline for the frontend server's *static* scheduler and used for debugging the scheduler.

The following is step-by-step explanation on how to setup the scheduler. Don't worry! We have also prepared example files for each step. Please refer to *execScheduler_MultiModelApp.sh* and *execScheduler_MultiModelScen.sh*, if you just want an example of how to execute the scheduler (and please refer to *src/standalone_scheduler.cpp* for further details).

1. Make task configuration (as csv) files. Task configuration file should contain the number of models, N, on the first line of file. For the next N lines, each line must specify (in the exact order) the ID of the model, incoming request rate (RPS) and SLO (in ms). The following command is an example of automating the process of creating task configuration files for multi model scenarios. Please refer the *createMultiModelAppConfigs.py* for multi model applications.
   ```
   $ python createMultiModelScenConfigs.py (output_dir)
   ```

2. Prepare scheduler configuration file (default is *resource/sim-config.json*). This JSON file specifies 1) the type and number of GPU that should be used for scheduling (each type of device should be provided with device configuration files and more details are specified in the next step), 2) various flags regarding interference and incremental scheduling and 3) the amount of latency buffer to use when scheduling.

3. Prepare device configuration files and directories for each type of GPU you want to use for scheduling. Please refer to *resource/device-config.json* and make sure that you have all the related files specified in *device-config.json*.

4.3 Example Script for Experimenting on Multiple Servers

Below are example scripts that will help you get started when experimenting with multiple servers.

1. *setupServer.sh*: Boots servers that are listed within the script. Make sure each server can be accessed by ssh without passwords.

2. *shutdownServer.sh*: Shutdowns servers that are listed within the script. The script shutdowns the backend servers and frontend servers in order.

3. *experMultiModelApp.sh*: Boots servers (with *setupServer.sh*), generates requests for given app, shutdowns servers (with *shutdownServer.sh*) and analyzes the results. All scripts required for analyzing are provided in the repository.

4. *experMultiModelScen.sh*: Works similarly to *experMultiModelApp.sh* but experiments with a given configuration (JSON) file which specifies the request rate of each model.

5. *experiment_ps.sh*: An example script for showing how to use *experMultiModelApp.sh* and *experMultiModelScens.h*.

4.4 Steps for Executing Toy Example (with Docker)

1. Navigate to *glet/scripts* directory.
```
$ cd glet/scripts
```
2. Start MPS with `sudo` (or else MPS will not be available to docker).
```
$ sudo ./start_MPS.sh
```
3. Create an overlay network with dockers. The script will create an attachable network for subnet range *10.10.0.0/24*.
```
$ cd ../docker_files
$ ./create_docker_overlay_network.sh
```
4. On separate terminals, run backend servers on each terminal (private IP address should match those listed in *glet/resource/Backend_docker.json*). Each script will setup a backend server for GPU 0 and GPU 1 respectively.
```
$ ./run_interactive_backend_docker.sh 0 10.10.0.3
$ ./run_interactive_backend_docker.sh 1 10.10.0.4
```
5. On another terminal, run the frontend server.
```
$ ./run_interactive_frontend_docker.sh oracle 10.10.0.20 1
```
6. On another terminal, run the clients.
```
$ ./run_interactive_client_docker.sh 10.10.0.21 test-run
```
7. As an result *log.txt* will be created under *glet/scripts*, which is a logging file that has various results when each request was executed in the server.
8. Terminate process on each terminal to end the example.
9. (Optional) in order to shutdown MPS, execute *shutdown_MPS.sh* under *glet/scripts*.
```
$ sudo ../shutdown_MPS.sh
```

4.5 Steps for Producing Custom Docker Server Image

We also provide the *base* image we have used. This will build an image with *Dockerfile* and the image will be tagged with *glet-server:latest*.

1. Pull the base image.
```
$ docker pull sbchoi/glet-base:latest
```
2. Execute building script stored under *glet/docker_files*.
```
$ cd glet/docker_files && ./build_dockers.sh
```

4.6 Additional Features

- Source code of all SW components used in experimentation: standalone inference binary, standalone scheduler, request generator, backend server, frontend server and proxy server.
- Scripts used for executing and analyzing experiments.
- Docker related files. e.g.) Dockerfile and scripts.

5 Release Note

- 2022.07.11. "Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing" was presented in USENIX ATC 2022.
- 2022.08.24. *gpulet* was released in public (`https://github.com/casys-kaist/glet`).

- 2022.09.29. (Done) Source code of all SW components used in experimentation: standalone inference binary, standalone scheduler, request generator, backend server, frontend server and proxy server.
- 2022.12.12. Added instructions for executing inference on a single local GPU.
- 2023.06.20. Added instructions for executing offline scheduler.
- 2023.07.18. Added instructions for experimenting on Multiple Servers.
- 2023.07.19. (Done) Scripts used for executing and analyzing experiments. Added instructions for producing custom docker server image.

6 Acknowledgments

This document was written under the support of the Institute of Information & Communications Technology Planning & Evaluation (IITP) (IITP-2017-0-00466).