Artifact Claims

The version of the paper we submitted has been revised based on the suggestions from the shepherd. In particular, we have added experiments to validate the efficacy of MinFlow under prime function parallelism (refer to Fig 18 and Fig 19 in our paper). It is essential to highlight that our artifacts align with the version of the paper that was submitted.

Introduction

MinFlow is a serverless workflow engine that achieves high-performance and cost-efficient data passing for I/O-intensive stateful serverless analytics.

Hardware Dependencies and Private IP Address

1. In our experiment setup, we use 10 AWS EC2 instances installed with Ubuntu 22.04 LTS (m6i.24xlarge , cores: 96, DRAM: 384GB, network bandwidth: 37.5 gigabits/s) for each node.
2. Please select a node as the master node to generate execution plans and save the private IP address of the master node as the **<master_ip>**, and save the private IP address of the other worker nodes as the **<worker_ip>**.
3. We recommend placing the source code on the master node and using `sshfs` to mount the source code to other worker nodes in order to update the code synchronously.

About Config Setting

There are 3 places for config setting, and we provide an introduction to the parameters that require updating.

1. `src/container/container_config.py` specifies the following configs:
   - Account information of AWS S3
     - S3_ACCESS_KEY
     - S3_SECRET_KEY
     - S3_REGION_NAME
     - S3_INPUT_BUCKET (Store input data)
     - S3_INTER_BUCKET (Store intermediate data and output data)
   - The number of nodes
     - NODE_NUM = 10
2. `src/grouping/node_info.ymal` specifies the node ip address.
3. All other configurations are in `config/config.py`.
   - Ip configs
     - GATEWAY_ADDR = '172.31.34.109:7001' (**need to be updated as your master private_ip**)
   - method configs
     - SHUFFLE_MODE = 'min' # single, min
     - DATA_MODE = 'optimized' # raw, optimized
     - MODELER = False (default)

        - LAMBADA_OPT = False (default)
        - BALANCE_STATISTICS = False (default)
    - AWS S3 configs
        - S3_ACCESS_KEY
        - S3_SECRET_KEY
        - S3_REGION_NAME
        - S3_INPUT_BUCKET (Store input data)
        - S3_INTER_BUCKET (Store intermediate data and output data)
    - Benchmark configs
        - NODE_NUM = 10
        - WORKFLOW_NAME = 'mapreduce-sort'
        - INPUT_DATA_SIZE = 200000 # MB
        - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}

Installation and Software Dependencies

Clone our code `https://github.com/lt2000/MinFlow.git` and Perform the following four steps in sequence on each node:

1. Python
    - The Python version is 3.9, and we recommend using Anaconda to manage Python packages

      ```
      conda create -n minflow python=3.9
      conda activate minfow
      ```

    - Run `scripts/python_install.bash` to install packages.
2. Docker and database
    - We utilize Docker for container management while employing CouchDB and Redis as metadata and log databases, respectively.
    - To install docker and those databases, you need to run `scripts/docker_install.bash`.

      ```
      conda activate minfow
      ./docker_install.bash
      ```

    - In order to omit sudo when using docker, execute the following command and restart the instance.

      ```
      sudo usermod -aG docker ubuntu<your_user_name>
      sudo newgrp docker
      ```

3. Build docker images
    - Run `scripts/create_image.bash` to build base images (`minflow_base` and `workflow_async_base`)

        - If you make modifications to the files in the `src/container` folder, it is necessary to rebuild the `workflow_async_base` image
        - Please be aware that we utilized the Tsinghua and Shanghai Jiaotong University sources for building the image. In case of a timeout error (e.g., over ten minutes), consider commenting out the following code in the `src/base/Dockerfile` file.

          ```
          COPY pip.conf /etc/pip.conf
          COPY sources.list /etc/apt/sources.list
          RUN rm -r  /etc/apt/sources.list.d
          ```

    - Run `benchmark/mapreduce-sort/create_image.sh`, `benchmark/tpcds-16/create_image.sh`, `benchmark/wordcount-shuffle/create_image.sh` to build image for Terasort, TPC-DS-Q16, WordCount, respectively.
4. Mount `tmpfs` as local storage

    -
      ```
      sudo mkdir ~/ramdisk/
      sudo mount -t tmpfs -o size=30G tmpfs ~/ramdisk/
      ulimit -n 524288
      ```

Generate execution plans for different methods on the master node and keep all nodes alive:

1. Baseline
    - Change the configurations in `src/config/config.py`
        - SHUFFLE_MODE = 'single' # single, min
        - DATA_MODE = 'raw' # raw, optimized
        - MODELER = False
        - LAMBADA_OPT = False
        - BALANCE_STATISTICS = False
    - Run `src/grouping/metadata.py`

      ```
      python metadata.py <function_number> <workflow_name>
      e.g.,
      python metadata.py 600 mapreduce-sort
      python metadata.py 400 tpcds-16
      python metadata.py 200 wordcount-shuffle
      ```

2. FaaSFlow
    - Change the configuration by in `src/config/config.py`
        - SHUFFLE_MODE = 'single' # single, min
        - DATA_MODE = 'optimized' # raw, optimized
        - MODELER = False
        - LAMBADA_OPT = False
        - BALANCE_STATISTICS = False
    - Run `src/grouping/metadata.py`
3. Lambada
    - Change the configuration by in `src/config/config.py`

        - SHUFFLE_MODE = 'min' # single, min
        - DATA_MODE = 'raw' # raw, optimized
        - MODELER = False
        - LAMBADA_OPT = True
        - BALANCE_STATISTICS = False
    - Run `src/grouping/metadata.py`
4. MinFlow
    - Change the configuration by in `src/config/config.py`
        - SHUFFLE_MODE = 'single' # single, min
        - DATA_MODE = 'optimized' # raw, optimized
        - MODELER = Ture
        - LAMBADA_OPT = False
        - BALANCE_STATISTICS = False
    - Run `src/grouping/metadata.py`

Generate Input Data and Upload to S3

Note: As the Terasort benchmark is simpler to generate input data compared to TPC-DS-Q16 and WordCount benchmarks, we recommend that AEC initially employ the Terasort benchmark to reproduce our results.

1. Terasort
    - Modify the configuration with your AWS S3 account information in `benchmark/gendata/mapreduce-sort/container_config.py`
    - Run `benchmark/gendata/mapreduce-sort/gendata.py`

      ```
      python gendata.py <function_number> <data_size(MB)>
      e.g.,
      python gendata.py 400 200000
      ```

2. TPC-DS-Q16
    - Modify the configuration with your AWS S3 account information in `benchmark/gendata/tpcds-16/container_config.py`
    - We need to generate the five tables involved in TPC-DS-Q16
        - data_dim

          ```
          python gen_date_dim.py <function_number> <data_size(GB)>
          e.g.,
          python gen_date_dim.py 400 328
          ```

          It should be noted that 328GB is the total size of all tables in TPC-DS, and the corresponding TPC-DS-Q16 involves five tables with a size of 100 G, similarly, 643GB=200G

        - call_center

```
python gen_call_center.py <function_number> <data_size(GB)>
e.g.,
python gen_call_center.py 400 328
```

- customer_address

```
python gen_customer_address.py <function_number> <data_size(GB)>
e.g.,
python gen_call_center.py 400 328
```

- catalog_sales and catalog_returns

```
python gen_catalog.py <function_number> <data_size(GB)>
e.g.,
python gen_call_center.py 400 328
```

3. WordCount
   - Modify the configuration with your AWS S3 account information in `benchmark/gendata/wordcount-shuffle/container_config.py`
   - Download the Wiki dataset from https://engineering.purdue.edu/~puma/datasets.htm

```
wget ftp://ftp.ecn.purdue.edu/puma/wikipedia_50GB.tar.bz2
# we used file9 as our input
```

   - Run `benchmark/gendata/wordcount-shuffle/clean.py` to replace non-word characters with Spaces
     - Output cleaned_data
   - Run `benchmark/gendata/wordcount-shuffle/gendata.py`
     - Use cleaned_data to generate input data

```
python gendata.py <function_number> <data_size(GB)>
e.g.,
python gendata.py 400 100
```

"Hello world"-sized example

We will demonstrate how to execute MinFlow with a "Hello world"-sized example, i.e., 8 mapper x 8 reducer under 8MB Terasort benchmark.

1. Generate input data

```
cd benchmark/gendata/mapreduce-sort/
python gendata.py 8 8
```

2. Change the configurations
   - `config/config.py`:
     - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
     - SHUFFLE_MODE = 'min' # single, min
     - DATA_MODE = 'optimized' # raw, optimized

        - MODELER = True
        - LAMBADA_OPT = Fasle
        - BALANCE_STATISTICS = False
        - WORKFLOW_NAME = 'mapreduce-sort'
        - NODE_NUM = 10
    - `/test/asplos/data_overhead/run.py`
        - workflow_pool = ['mapreduce-sort']

3. Generate execution plans
    - Keep all nodes alive and run `src/grouping/metadata.py` on the master node

      ```
      python metadata.py 8 mapreduce-sort
      ```

4. Start the engine proxy and gateway
    - Enter `src/workflow_manager` and start the engine proxy with the local <worker_ip> on each node by the following command:

      ```
      python proxy.py <worker_ip> 8001             (proxy start)
      ```

    - Then enter `src/workflow_manager` and start the gateway on the master node by the following command:

      ```
      python gateway.py <master_ip> 7001           (gateway start)
      ```

5. Run the workflow
    - Enter `test/fast/data_overhead` and run the following command:

      ```
      python run.py --num=8 --method=3
      #num: The number of functions
      #method: Baseline=0,FaaSFlow=1,Lambada=2,MinFlow=3
      ```

    - The job completion time is written to `mapreduce-sort_request.json`
        - The meaning of each field in `mapreduce-sort_request.json` is shown below

          ```
          {
              "method": {
                  "request_id": {
                      "function_num-run_number": job_completion_time
                  }
              }
          }
          ```

          e.g.,

```
{
    "MinFlow": {
        "0330dfc3-640f-4484-b81f-02b63dba1233": {
            "8-1": 0.5357868671417236
        }
    }
}
```

- It's important to note that the initial run represents the cold start of the container, and the second run are considered.

6. Rerun
    - **Note:** We recommend restarting the `proxy.py` on each node and the `gateway.py` on the master node whenever you start the `run.py` script, to avoid any potential bug.

Run Experiment

We will use 200G Terasort under 600 function parallelism running on Minflow as an example to show how to reproduce our main experiments

Shuffle Time (Section 4.2 Figure7，8，9)

1. Generate input data

    ```
    cd benchmark/gendata/mapreduce-sort/
    python gendata.py 600 200000
    ```

2. Change the configurations
    - `config/config.py`:
        - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
        - SHUFFLE_MODE = 'min' # single, min
        - DATA_MODE = 'optimized' # raw, optimized
        - MODELER = True
        - LAMBADA_OPT = False
        - BALANCE_STATISTICS = False
        - WORKFLOW_NAME = 'mapreduce-sort'
        - NODE_NUM = 10
    - `/test/asplos/data_overhead/run.py`
        - workflow_pool = ['mapreduce-sort']
3. Generate execution plans
    - Keep all nodes alive and run `src/grouping/metadata.py` on the master node

        ```
        python metadata.py 600 mapreduce-sort
        ```

4. Start the engine proxy and gateway
    - Enter `src/workflow_manager` and start the engine proxy with the local <worker_ip> on each node by the following command:

```
    python proxy.py <worker_ip> 8001            (proxy start)
    ```

  - Then enter `src/workflow_manager` and start the gateway on the master node by the following command:

    ```
    python gateway.py <master_ip> 7001          (gateway start)
    ```

5. Run the workflow
   - Enter `test/fast/data_overhead` and run the following command:

     ```
     python run.py --num=600 --method=3
     #method number: Baseline=0,FaaSFlow=1,Lambada=2,MinFlow=3
     ```

   - The job completion time is written to `mapreduce-sort_request.json`
6. Calculate the shuffle time
   - We break down the job completion time into three parts: in/output time, computing time, and shuffle time. Each function autonomously records its respective breakdown of execution time to the Redis instance residing on the node where it is executed. We aggregate breakdown results from all nodes onto the master node for the purpose of calculating the shuffle time for the job. The detailed operations are as follows:
   - Copy breakdown results
     - Change the configurations by `hostlist=[nodes ip]` in `/test/breakdown/copydata.py`

     -
       ```
       python copydata.py
       ```
     - Copy the file `mapreduce-sort_request.json` obtained in step 5 to `/test/breakdown`
   - Calculate the in/output time, computing time, and shuffle time
     - Enter `/test/breakdown`

     -
       ```
       python overall_break_critical.py mapreduce-sort 600
       ```
     - The results will be printed on the terminal

Load Balance (Section 4.2 Figure 10)

At intervals of 50 milliseconds, we collect statistics on each node's CPU utilization, memory utilization, and network send and receive throughput. To obtain statistical information, the details are as follows:

1. Change configurations in `config/config.py`
   - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
   - SHUFFLE_MODE = 'min' # single, min
   - DATA_MODE = 'optimized' # raw, optimized
   - MODELER = True
   - LAMBADA_OPT = False
   - BALANCE_STATISTICS = True

    - WORKFLOW_NAME = 'mapreduce-sort'
    - NODE_NUM = 10
2. - Change configurations by `node_dict={nodes ip}` in `/src/workflow_manager/resource.py`
   - Change configurations by `path=/path/to/statistics` in `/src/workflow_manager/resource.py`
3. Run the workflow
4. Copy the statistics of all nodes to `/test/load_balance/dataset` of the master node
   - Enter `/test/load_balance` and run `/test/load_balance/balance.py`

   -
     ```
     python balance.py <request_id> <node_num>
     e.g.,
     python balance.py ce943bb1-25a4-45cf-978d-c5b97eafe16d 10
     ```
   - Output four figures:`cpu.png`, `mem.png`, `sent.png`, `resv.png`

Overall Time (Section 4.3 Figure11，12，13)

- Similar to Shuffle Time (Section 4.2), the overall time is stored in the file `mapreduce-sort_request.json`.

Technique Breakdown (Section 4.4 Figure14)

We progressively integrate the three components (i.e., Topology optimizer, Function scheduler, Configuration modeler) to show their respective contribution to MinFlow's shuffle time reduction.

1. Topology optimizer
   - Change the configurations in `config/config.py`
     - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
     - SHUFFLE_MODE = 'min'
     - DATA_MODE = 'raw'
     - MODELER = False
     - LAMBADA_OPT = False
     - BALANCE_STATISTICS = False
     - WORKFLOW_NAME = 'mapreduce-sort'
     - NODE_NUM = 10
   - Run the workflow as Shuffle Time (Section 4.2)
2. Topology optimizer + Function scheduler
   - Change the configurations in `config/config.py`
     - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
     - SHUFFLE_MODE = 'min'
     - DATA_MODE = 'optimized'
     - MODELER = False
     - LAMBADA_OPT = False

        - BALANCE_STATISTICS = False
        - WORKFLOW_NAME = 'mapreduce-sort'
        - NODE_NUM = 10
    - Run the workflow as Shuffle Time (Section 4.2)
3. Topology optimizer + Function scheduler + Configuration modeler
    - Change the configurations in `config/config.py`
        - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
        - SHUFFLE_MODE = 'min'
        - DATA_MODE = 'optimized'
        - MODELER = True
        - LAMBADA_OPT = False
        - BALANCE_STATISTICS = False
        - WORKFLOW_NAME = 'mapreduce-sort'
        - NODE_NUM = 10
    - Run the workflow as Shuffle Time (Section 4.2)

Scalability (Section 4.4 Figure15)

Evaluate the impact on the overhead of the Topology optimizer and Function scheduler with increasing function parallelism (from 1 to 1000).

1. Change the configurations in `config/config.py`
    - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
    - SHUFFLE_MODE = 'min'
    - DATA_MODE = 'optimized'
    - MODELER = False
    - LAMBADA_OPT = False
    - BALANCE_STATISTICS = False
    - WORKFLOW_NAME = 'mapreduce-sort'
    - NODE_NUM = 10
2. Enter `/test/scalability` and run `scalability.py`

    ```
    python scalability.py
    ```

    Output two files: ay.pickle, by.pickle
3. Run `/test/scalability/plot.py`

    ```
    python plot.py
    ```

    Output a figure: scalability.png

Various input size and Tunable parallelism (Section 4.5 Figure16， 17)

- Similar to Shuffle Time (Section 4.2)

Prime function parallelism (Section 4.5 Figure18， 19)

- Impact of α (Figure18)
  - Change the configurations in `config/config.py`
    - FUNCTION_INFO_ADDRS = {'mapreduce-sort': '../../benchmark/mapreduce-sort'}
    - SHUFFLE_MODE = 'min'
    - DATA_MODE = 'optimized'
    - MODELER = False
    - LAMBADA_OPT = False
    - BALANCE_STATISTICS = False
    - WORKFLOW_NAME = 'mapreduce-sort'
    - NODE_NUM = 10
  - Enter `/test/alpha` and run `prime_alpha.py`

    ```
    python prime_alpha.py
    ```

  - Run `/test/alpha/plot.py`

    ```
    python plot.py
    ```

    Output a figure: alpha.png
- Shuffle time under prime function parallelism (Figure 19)

  We will use 200G Terasort under 587 function parallelism as an example
  - Run `/test/prime/copy.bash`
    - replace `src/grouping/parse_yaml_min.py` with `/test/prime/parse_yaml_min.py`
    - replace `src/grouping/network.py` with `/test/prime/network.py`
    - replace `src/workflow_manager/workersp.py` with `/test/prime/workersp.py`
    - Please back up the original file before replacing it
  - Generate input data

    -
      ```
      cd benchmark/gendata/mapreduce-sort/
      python gendata.py 587 200000
      # For Baseline,FaaSFlow,Lambada
      ```

    -
      ```
      cd benchmark/gendata/mapreduce-sort/
      python gendata.py 588 200000
      # For Minflow
      ```

- Run the workflow
    - Similar to Shuffle Time (Section 4.2)