Readme for reproducibility submission of SIGMOD'21 paper ID 68

1 Source code info

Repository: https://github.com/Yanqing-UTAH/ATTPCode
Contact: Zhuoyue (zzhao35@buffalo.edu)
Programming Langauge: mainly C/C++, some of the scripts are written in Python
Software/library:

- (Required for our code) gcc/g++ >= 8 (requires support for -std=gnu++17), make, lapack and lapacke, blas and cblas, fftw3
- (Required for Vertica baseline in Figure 1) unixodbc, Vertica
- (Required for creating datasets) python3, numpy, scipy, sklearn and the common shell programs (bash, grep, sed, and etc.).
- (Required for plotting figures): jupyter notebook, python3, matplotlib, pandas, numpy
- (Optional, only if you need to regenerate the configure script): m4, autoconf, autoheader

Hardware requirement: we assume the architecture implements <= 48-bit virtual address space and always uses x86-64 canonical addresses. That's the case for any combination of Intel/AMD processor and Linux kernel, except for the kernels that are configured to allocate memory beyond 47-bit user address space on the processors that support 5-level paging (e.g., Intel Ice Lake).

2 Test environment

Software environment:

- Ubuntu 18.04.6 LTS
- GNU make 4.1
- GCC/G++ 8.3.0
- liblapack-dev 3.7.1
- liblapacke-dev 3.7.1
- libblas-dev 3.7.1
- libatlas-base-dev 3.10.3
- libfftw3-dev 3.3.7
- python3 3.6.9
- sklearn 0.22.2
- scipy 1.4.1
- numpy 1.19.0
- unixodbc-dev 2.3.4

- Vertica 10.0.1 Community Edition

Note that these are the ones installed at the time we performed the experiments but it should be ok if you use newer versions. For Vertica 10.0.1 Community Edition, you may find the free community version at `https://www.vertica.com/`.

Hardware environment:

For all experiments in Section 6: we have 10 type-1 nodes and 6 type-2 nodes and we launched any of the experiments on any one of them depending on the availability. They have comparable performance and we made sure that there were no computation or I/O heavy programs running concurrently.

Node type	1	2
CPU	Intel Core i7-3820	Intel Xeon E5-1650 v3
CPU Frequency	3.6 GHz	3.50 GHz
L1 Cache	32KB + 32 KB	32KB + 32 KB
L2 Cache	256 KB	256 KB
L3 Cache	10 MB (shared)	15 MB (shared)
Memory	64GB (DDR3-1600 x 8)	128GB (DDR4-2133 x 4)
Secondary storage	WD HDD 7200RPM 2TB	Seagate HDD 7200RPM 1TB
Network	not used	not used

For Figure 1 scalability test against Vertica, we used the following server to make it run faster. Note that this figure is only used for introduction and is not used for comparison against any of the existing persistent sketches.

Node type	3
CPU	
CPU Frequency	3.6 GHz
L1 Cache	32KB + 32 KB
L2 Cache	256 KB
L3 Cache	10 MB (shared)
Memory	64GB (DDR3-1600 x 8)
Secondary storage	WD HDD 7200RPM 2TB
Network	not used

3 Preparing datasets

We have two scripts for creating the datasets: 1) the datasets based on the FIFA World Cup 98 website access logs for the ATTP/BITP heavy hitter experiments; 2) the synthetic datasets for the ATTP frequent direction experiments.

For the world-cup datasets (for the ATTP/BITP heavy hitters), run:

```
$ ./data_proc/world-cup/prepare_data.sh
```

It takes about 4 hours to download and generate all the datasets. In case the website hosting the raw logs is unreachable, please contact Zhuoyue (zzhao35@buffalo.edu) for our own copy.

For the matrix datasets (for the ATTP frequent directions): run:

```
$ ./data_proc/gen_mat_data.sh
```

It takes less than 12 minutes to generate all the three datasets (small, medium and large). If you only want one or some of the three datasets, specify its name as a command line argument to the script (e.g., `./data_proc/gen_mat_data.sh small`)
All the generated data are put into the data/ directory, see data/README.md for descriptions.

4 Prepare the system and building the code

If you're using Ubuntu, you should be able to use the following to prepare all the required prerequisites except Vertica:

```
$ sudo apt install gcc g++ make liblapacke-dev libatlas-base-dev \
> libfftw3-dev python3 python3-pip unixodbc unixodbc-dev
$ pip3 install sklearn scipy numpy
```

To build our code:

```
$ ./configure
$ make
```

To setup a single-node Vertica for the baseline in Figure 1, first request and download the free vertica community version from `https://www.vertica.com/` and install it in your system. In case it's no longer available, we still have a copy on our server. We provided a script `vertica/setup_vertica.sh` to help you set it up on Ubuntu. Please take a look at the first a few lines of the script (change them if needed) and do the following:

```
$ dpkg -i <path-to-vertica-deb-package>
$ sudo ./vertica/setup_vertica.sh # you might have to tweak it, see comments
$ sudo su verticadba
$ admintools # follow prompts to finish setup and create a DB
$ exit # back to your own account
$ /opt/vertica/bin/vsql <your-db-name> verticadba # use this to check if the DB is running
$ export VERTICAINI=${HOME}/.vertica.ini
```

You'll also need to create two files $\sim$ /`.odbc.ini` and $\sim$ /`.vertica.ini` for Vertica ODBC connection. We provided a sample for them in `vertica/` and you might want to tweak them according to your own system setup.

5 Running the experiments

We provide the config files in `configs` for running the experiments in Section 5 and Figure 1. To run the experiments specified in a config file, say `config/xxx.conf`, run:

```
$ ./driver configs/xxx.conf
```

Note that this must be run from the repository root, as the data file location are specified as relative path from the working directory in the configuration files. Most of these experiments are both computation and memory intensive. The largest one would be `configs/test-client-id-bitp.conf`, which should run for a few days and with around 100GB of memory. So we actually split them into smaller batches of experiments when we ran them. To do so, just copy the config file to a

different machine, edit the file to disable some of the sketch and/or remove some of the experiment parameters from the settings, and run that subset of experiments there.
To run vertica baseline in Figure 1, use the `vertica/run_vertica.sh` script.
We have the original logs saved in `plot/raw_logs` for Section 5, and in `plot/scalability_logs` for Figure 1. The experiments took us a couple of days to run with all the 17 servers, so you might find the raw logs to be useful for validating the results. Also note that the log files contain additional data points that were not shown in the experiment figures mainly because their x-axis values (memory usage) are too small or too large to be fit into the figures – these files need to be processed by `plot/filtered_logs/filter.sh` before they may be used for plotting the figures.

6 Plotting the figures

You'll need jupyter notebook, matplotlib, numpy and pandas to plot the figures. The scripts provided in plot/ work on my local WSL 2.0 installation with Ubuntu 20.04 LTS, python 3.8.10, matplotlib 3.4.3, numpy 1.21.3, pandas 1.3.4 and jupter notebook 6.0.3.

```
$ sudo apt install python3 python3-pip jupyter-notebook
$ pip3 install matplotlib numpy pandas
```

The following table lists which notebook you should run to generate specific figures as well as the expected input file. These scripts also generates pdf files for the figures in the plot/ directory.

Notebook	Figures	Input
`HH_ATTP_clientid.ipynb`	2, 3(left), 4	`filtered_logs/client_id_attp_filtered_combined.txt`
`HH_ATTP_objectid.ipynb`	3(right), 5, 6	`filtered_logs/object_id_attp_filtered_combined.txt`
`HH_BITP_clientid.ipynb`	7, 8(left), 9	`filtered_logs/client_id_bitp_filtered_combined.txt`
`HH_BITP_objectid.ipynb`	8(right), 10, 11	`filtered_logs/object_id_bitp_new_filtered_combined.txt`
`MAT_ATTP_small.ipynb`	12(a), 13(left), 14	`filtered_logs/ms_small_attp_filtered_combined.txt`
`MAT_ATTP_medium.ipynb`	12(b), 13(right), 15	`filtered_logs/ms_medium_attp_filtered_combined.txt`
`MAT_ATTP_big.ipynb`	12(c), 16	`filtered_logs/ms_big_attp_filtered_combined.txt`
`scalability_test_client_id_ATTP.ipynb`	1	`scalability_logs/scalability-test-client-id.log`