

### Required Installations:

- **Percy++** - <https://percy.sourceforge.net/download.php> (required for case study results only)
- **Socket++** - <https://github.com/malaterre/socketxx/tree/master> (required for case study results only)
- **NTL** - <https://libntl.org/download.html> (required for both benchmarking and case study results)
- **GMP** - <https://gmplib.org/#DOWNLOAD> (required for both benchmarking and case study results)

### Additionally, the following commands are to be run before reproducing any results:

1. `g++ -std=c++11 -o ccs create_iaq_store_ccs.cc -lnl -lpthread -lgmp`
2. `g++ -std=c++11 -o createindex create_evaluate_batch_iaqs.cc -lnl -lpthread -lgmp`
3. `make genuint`
4. `head -c 10737418240 </dev/urandom > database`

The first three are required for the benchmarking experiments, and all four are required for the case studies.

### Benchmarking (Section V.A):

The purpose of our benchmarking experiments is to evaluate the multiplication between the client query vector and the sparse index of aggregate queries (IAQ) matrix. This essentially reflects the client query throughput – how many clients (queries) can be served per second – for the protocol. There are a number of parameters whose impact on the client query vector and IAQ multiplication are evaluated through these experiments. There are three figures in this section (Figs. 2 - 4) that have been used to demonstrate query throughput variation, and each of them is produced by varying a single parameter while keeping the rest fixed. The general steps for these experiments require the following steps:

1. Generate indexes of aggregate queries (IAQ) matrices with an arbitrary number of 1s in each row. This is done using the `create_iaq_store_ccs.cc` file. The output of this step is matrices in compressed column storage (CCS) representation. The CCS formatted each simple IAQ matrix has two files, ending with `*.col` and `*.row`.
2. Batch multiple IAQ matrices – that are in CCS format produced in step 1 – by u-are Polynomial Batch Coding technique and evaluate the resultant matrix of polynomials at given x-coordinates. This is done using the `create_evaluate_batch_iaqs.cc` file. The CCS formatted each batched IAQ matrix has three files, e.g., `val.<x_coordinate>`, `out.col`, and `out.row`.
3. Run the efficient (by hand-optimized assembly coding) multiplication of arbitrary client query vector and a single evaluation of the batched IAQ matrix produced in step 2 on the GPU. The modular reduction is done by the Barrett algorithm. This step is performed using the `VspM_barrett_GPU.cu` file

The list of parameters that are varied is as follows:

- `r` - number of columns in the IAQ matrices (i.e., number of records in the database)

- p - number of search terms or keywords (i.e., number of rows in IAQ matrices)
- u - number of IAQ matrices batched together
- v - number of 1s in a row of IAQ matrix (essentially, the number of records on which the aggregation is expected to be done over the database)
- q - number of bits in the modulus

#### Reproducing Fig 2 (GPU required):

Executing the shell script 'bashFig2.sh' in the repository results in the execution of steps 1 to 3 with the relevant parameter values for generating the results for producing Fig 2. The parameter values for this process are set to:

- r - varied from  $2^{16}$  to  $2^{24}$
- p -  $2^{16}$
- u - 6
- v -  $2^2$
- q - varied from  $2^7$  to  $2^9$

#### Reproducing Fig 3 (GPU required):

Executing the shell script 'bashFig3.sh' in the repository results in the execution of steps 1 to 3 with the relevant parameter values for generating the results for producing Fig 3. The parameter values for this process are set to:

- r -  $2^{20}$
- p - varied from  $2^1$  to  $2^{17}$
- u - 6
- v -  $2^4$
- q - varied from  $2^7$  to  $2^9$

#### Reproducing Fig 4 (GPU required):

Executing the shell script 'bashFig4.sh' in the repository results in the execution of steps 1 to 3 with the relevant parameter values for generating the results for producing Fig 4. The parameter values for this process are set to:

- r -  $2^{16}$
- p -  $2^{14}$
- u - 6
- v - varied from  $2^1$  to  $2^{11}$
- q -  $2^8$

The fifth and final figure in the benchmarking section is slightly different from the other figures. It tests the scalability of the protocol by measuring the time required to batch progressively larger numbers of IAQ matrices.

#### Reproducing Fig 5 (GPU not required):

This experiment requires only steps 1 and 2, and they are executed through the execution of the shell script 'bashFig5.sh'. The parameter values for this process are set to:

- $r - 2^{16}$
- $p - 2^{14}$
- $u$  - varied from  $2^1$  to  $2^{10}$
- $v - 2^4$
- $q - 2^8$

Next, the Python script `nnz_column_count.py` is to be executed in order to obtain the percentage of non-zero columns in the batched indexes of aggregate queries.

### Case studies (Section V.B):

The objective of the case studies is to demonstrate the implementation of our protocol in realistic scenarios. The results of the case studies are collated in Table 1. The experiments are performed with three different datasets. The following are the steps to reproduce the results in Table 1 for all three case studies:

- Real-world Dataset collection (GPU not required):
  - The Twitter dataset is compiled by scraping the website and compiling tweets and tweet statistics. A subset of this dataset is provided in the repository. The filename is 'twitter\_data\_sample.csv.'
  - The MIMIC-3 dataset is publicly available upon completion of certification training. <https://physionet.org/content/mimiciii/1.4/>
  - The Yelp dataset is publicly available at <https://www.yelp.com/dataset>
- Scanning database and generating the IAQ matrices (GPU not required):

This is the step where the collected dataset is scanned for creating the IAQ matrices. This results in the generation of CCS representation files for the matrices. It is the equivalent of Step 1 for the Benchmarking experiments but with specific and real-world meaningful data. We have uploaded the output of this step, i.e., the CCS files for the matrices for all the queries in Table 1 in the repository. The filenames (in order in Table 1) are

  - `twitter_likeCount_333286_1004129.col`, `twitter_likeCount_333286_1004129.row`
  - `twitter_retweet_333286_1004129.col`, `twitter_retweet_333286_1004129.row`
  - `twitter_filtered_likeCount_2714_1004129.col`,  
`twitter_filtered_likeCount_2714_1004129.row`
  - `twitter_filtered_retweet_2714_1004129.col`,  
`twitter_filtered_retweet_2714_1004129.row`
  - `count_of_admissions.col`, `count_of_admissions.row`
  - `count_of_hispanic_admission.col`, `count_of_hispanic_admission.row`
  - `max_of_urgent_admissions.col`, `max_of_urgent_admissions.row`
  - `min_of_urgent_admissions.col`, `min_of_urgent_admissions.row`
  - `mimic_new_query_5.col`, `mimic_new_query_5.row`
  - `mimic_new_query_6.col`, `mimic_new_query_6.row`

- yelp\_categories\_query\_1.col, yelp\_categories\_query\_1.row
- max\_yelp\_categories\_query\_2.col, max\_yelp\_categories\_query\_2.row
- yelp\_categories\_augmented\_query\_1.col, yelp\_categories\_augmented\_query\_1.row
- max\_yelp\_categories\_exhaustive\_query\_2.col, max\_yelp\_categories\_exhaustive\_query\_2.row

Unzip the total\_dosage and mimic\_new\_query\_5 zip files in the archive

The above files can be produced using the scripts in the 'case\_study\_IAQ\_gen' folder when the data is available.

- **Batching the IAQ matrices (GPU not required):**  
In this step, the IAQ matrices (stored in CCS format as produced from the previous step) are batched together. This is the equivalent of Step 2 of the Benchmarking experiments but done instead with specific and real-world meaningful data. This step is performed by executing the shell scripts 'twitter\_all\_batch.sh', 'twitter\_filtered\_batch.sh', 'mimic\_all\_batch.sh', 'mimic\_filtered\_batch.sh', 'yelp\_all\_batch.sh' and 'yelp\_aug\_batch.sh'. The output of this step is used to populate the '**Multiple index matrices batching time**' column in Table 1.
- **Client query vector and IAQ sparse matrix multiplication time (GPU required):**  
This step involves multiplying the client query vectors with the IAQs on a GPU, and the parameters of these depend on the specific datasets. The number of elements in the client query (p) and the number of records (r) in the database are set, and this step is executed upon execution of the shell scripts twitter\_all\_batchGPUthroughput.sh, twitter\_filtered\_batchGPUthroughput.sh, mimic\_all\_batchGPUthroughput.sh, mimic\_filtered\_batchGPUthroughput.sh, yelp\_all\_batchGPUthroughput.sh and yelp\_aug\_batchGPUthroughput.sh. These will produce the results for the 'VspM throughput on GPU' column in Table 1.
- Next, the number of zero elements in the product (i.e., potentially a sparse vector) of the client query vector and batched IAQ matrix (which is equal to the number of zero columns in the batched IAQ matrix) is calculated using the Python scripts columnCount\_twitter\_all.py, columnCount\_twitter\_filtered.py, columnCount\_mimic\_all.py, columnCount\_mimic\_filtered.py, columnCount\_yelp\_all.py and columnCount\_yelp\_aug.py. This will help efficient server response generation for our protocol in the upcoming sections.
- **Server response generation time (GPU not required):**  
This section outlines how to reproduce the results for the three sub-columns 'All Attr.', 'Essen. Attr.' and 'Regular IT-PIR' in the 'Server response generation time per query' column. The following steps are to be followed:
  - A database file is created using the Linux command (binary file using uniform random generation). This file is made to be greater than or equal in size when compared to the largest dataset used for our case studies. Thus, different subsets of the same database file can be used to recreate each of our case study

datasets in terms of size and dimension. This file is generated during the execution of step 4 of the initial four setup commands.

- Install the Percy++ library, and locate the default itserver.cc file and replace it with our itserver.cc file.
- Open one terminal to be the client terminal and run the pirclient command.

Sample command – `./pirclient 1004129 32 -w128 "1:localhost:65441 1:localhost:65442" 1 1 -mZZ_P`

- r (number of records in the database) - 1004129
- s (number of words in a record) - 32
- server id (PID) - 1
- Ports - 65441 and 65442

- Open two more terminals and run the pirserver command in each terminal. These terminals will be the server terminals.

Sample command – `./pirserver database 1 1004129 32 -w128 -mZZ_P -p65442`

- (binary) database file path - database
- server id (PID) - 1
- r (number of records in the database) - 1004129
- s (number of words in a record) - 32
- Port - 65442

Each value in the 'All Attr.', 'Essen. Attr.' and 'RegularIT-PIR' columns are obtained by executing this command with different values of r and s. Also, note that the value of the port should be different in the two server terminals; the rest of the script should be identical in both terminals. So if it's 65442 in one terminal, try 65441 in the other, with everything else remaining the same.

- Regular IT-PIR (Baseline) sub-column:

The pirclient command to be run for each of the 6 pirclient commands is the sample pirclient command with the {r, s} values {1004129, 80}, {1004129, 80}, {58976, 128}, {4156450, 128}, {150346, 64} and {601384, 64}.

The corresponding pirserver command to be run for each of the 6 pirclient commands is the sample pirserver command (in both server terminals) with the {r, s} values {1004129, 80}, {1004129, 80}, {58976, 128}, {4156450, 128}, {150346, 64} and {601384, 64}. Plugging in these values of r and s in the pirserver and pirclient commands, and executing it 6 times (once each for every pair of values) in order will produce the values in the 'Regular IT-PIR' from the first to the last row.

- All Attr. sub-column:

- The number of zero elements in the product (i.e., potentially a sparse vector) of the client query vector and batched IAQ matrix (which is equal to the number of zero columns in the batched IAQ matrix) is calculated using the Python scripts `columnCount_twitter_all.py`, `columnCount_twitter_filtered.py`, `columnCount_mimic_all.py`, `columnCount_mimic_filtered.py`, `columnCount_yelp_all.py` and `columnCount_yelp_aug.py`, and this is used for identifying untouched records in the database. This step is common to Essen. Attr. sub-column

too. For ease of reproducing results, these values have been preloaded into our modified itserver.cc file.

- Open the itserver.cc file and uncomment line 694 if commented and save the file.
- There will be 6 runs of the client and server processes in order to produce the results of the 6 rows in Table1 under the 'All Attr.' sub-column. For each run, first open the itserver.cc file and uncomment the corresponding line of code, as indicated by the comments. So for the first run, line 39 is to be uncommented and then save the itserver.cc file. Then execute the client and server commands outlined in the following bullet points. Once the results are obtained for row 1, reopen the itserver.cc file, comment out the line of code that was uncommented for the previous run and uncomment line 40 and repeat the process of saving the file and running the client and server commands described in the following bullet points. Similarly, for the remaining 4 rows of the table, lines 41, 42, 43 and 44 in the itserver.cc are to be left uncommented while leaving the other line numbers mentioned in this paragraph commented, for each run of the process.
- The pirclient command to be run for each of the 6 pirclient commands is the sample pirclient command with the {r, s} values {1004129, 80}, {1004129, 80}, {58976, 128}, {4156450, 128}, {150346, 64} and {601384, 64}.
- The corresponding pirserver command to be run for each of the 6 pirclient commands is the sample pirserver command (in both server terminals) with the {r, s} values {1004129, 80}, {1004129, 80}, {58976, 128}, {4156450, 128}, {150346, 64} and {601384, 64}. Plugging in these values of r and s in the pirserver and pirclient commands, and executing it 6 times (once each for every pair of values) in order will produce the values in the 'All Attr.' from the first to the last row.
- Essen. Attr. sub-column:
  - (Same as All Attr. sub-column) Open the itserver.cc file and uncomment line 694 if commented and save the file.
  - (Same as All Attr. sub-column) There will be 6 runs of the client and server processes in order to produce the results of the 6 rows in Table1 under the 'All Attr.' sub-column. For each run, first open the itserver.cc file and uncomment the corresponding line of code, as indicated by the comments. So for the first run, line 39 is to be uncommented and then save the itserver.cc file. Then execute the client and server commands outlined in the following bullet points. Once the results are obtained for row 1, reopen the itserver.cc file, comment out the line of code that was uncommented for the previous run and uncomment line 40 and repeat the process of saving the file and running the client and server commands described in the following bullet points. Similarly, for the remaining 4 rows of the table, lines 41, 42, 43 and 44 in the itserver.cc are to be left

uncommented while leaving the other line numbers mentioned in this paragraph commented, for each run of the process.

- The pirclient command to be run for each of the 6 pirclient commands is the sample pirclient command with the {r, s} values {1004129, 32}, {1004129, 32}, {58976, 48}, {4156450, 48}, {150346, 16} and {601384, 16}.

The corresponding pirserver command to be run for each of the six pirclient commands is the sample pirserver command (in both server terminals) with the {r, s} values {1004129, 32}, {1004129, 32}, {58976, 48}, {4156450, 48}, {150346, 16} and {601384, 16}. Plugging in these values of r and s in the pirserver command, and executing it six times (once each for every pair of values) in order will produce the values in the 'Essen. Attr.' from the first to the last row.