



Hewlett Packard
Enterprise

HPE CTY


Network Flow Database

A Report On
The performance of PostgreSQL

Prepared by
AMSHU G M

About HPE CTY Project:

The "**Catch Them Young (CTY)**" Framework by HPE is designed with the primary objectives of fostering deeper early engagement with colleges and building industry-ready engineers equipped with the necessary competencies in HPE technologies. By targeting students in their 5th and 6th semesters, the program aims to hire and nurture the right talent through pre-placement offers, ensuring that graduates are well-prepared to meet industry demands. This initiative not only enhances the skill set of the students but also allows HPE to identify and cultivate potential future employees from an early stage.



1. POSTGRESQL DATABASE

Introduction:

PostgreSQL, often referred to as Postgres, is a powerful, open-source object-relational database system. Known for its robustness, performance, and advanced features, PostgreSQL has been in active development for over 30 years, maintaining a strong reputation for reliability, data integrity, and correctness.

Key Features:

1. Open Source: PostgreSQL is released under the PostgreSQL License, a liberal open-source license, making it free to use, modify, and distribute. This has fostered a large, active community that contributes to its continuous improvement.

2. Advanced Data Types and Indexing:

- **Data Types:** PostgreSQL supports a wide range of data types including integers, floating-point numbers, character strings, binary strings, and more advanced types like JSON, XML, and arrays.
- **Indexing:** It supports several types of indexes such as B-tree, Hash, GiST, SP-GiST, GIN, and BRIN, which enhance query performance.

3. ACID Compliance: PostgreSQL ensures atomicity, consistency, isolation, and durability (ACID) for transactions, which guarantees data integrity and reliability, even in the event of a system failure.

4. Extensibility: One of the standout features of PostgreSQL is its extensibility. Users can define their own data types, operators, and even index types. It also supports custom procedural languages through its Procedural Language handler.

5. SQL Compliance: PostgreSQL boasts a high level of SQL compliance, supporting most of the SQL standard features and often extending beyond it with additional features like table inheritance and custom aggregates.

6. Concurrency: PostgreSQL uses Multi-Version Concurrency Control (MVCC) to handle concurrent transactions. This allows for high levels of concurrency while maintaining data consistency.

7. Replication and Fault Tolerance:

- **Replication:** PostgreSQL supports both asynchronous and synchronous replication, ensuring data availability and redundancy.
- **Fault Tolerance:** Features like point-in-time recovery (PITR), tablespaces, and write-ahead logging (WAL) provide mechanisms for data recovery and fault tolerance.

8. Full-Text Search: PostgreSQL includes advanced full-text search capabilities, allowing for indexing and querying of text data efficiently.

9. Community and Support: PostgreSQL has a strong, global community offering extensive documentation, mailing lists, and various forums. There are also numerous companies providing professional support and services for PostgreSQL.

Use Cases:

1. Web Applications:

- Support for JSON data format, ideal for web applications dealing with semi-structured data.
- Efficient handling of complex queries, ensuring optimal performance for data-intensive web applications.
- Scalability and responsiveness, making it suitable for high-traffic websites and applications.

2. Data Warehousing:

- Robust support for large datasets, crucial for data warehousing environments.
- Advanced indexing capabilities optimize query execution for analytics and reporting.
- Full-text search functionality enhances data discovery and retrieval in data warehouses.

3. Geospatial Data:

- PostGIS extension adds support for geographic objects, enabling PostgreSQL to handle spatial data.
- Suitable for Geographic Information Systems (GIS), urban planning, and location-based services.
- Capable of storing and processing maps, coordinates, and geographical features efficiently.

4. Financial Systems:

- Strong emphasis on transactional integrity and ACID compliance, critical for financial applications.
- Ensures data consistency, reliability, and accuracy in financial transactions.
- Rollback mechanisms in case of failures, ensuring data security and regulatory compliance in financial systems.

2. PostgreSQL For Designing Network Flow Database

Structure of Network Flow Database:

A Network Flow database is designed to efficiently capture, store, and analyze packet information passing through the switches in the network. This allows users to efficiently store, retrieve, and modify information about different packets sent in a network.

The Network flow database contains the following attributes:

- 1) **"src_ip"**: This field is used to store the source IP address of the packet.
- 2) **"dest_ip"**: This field is used to store the destination IP address of the packet.
- 3) **"src_port"**: This field stores the source port number of the application.
- 4) **"dest_port"**: This field stores the destination port number.
- 5) **"ip_type"**: (IPv4 or IPv6) This stores the protocol used in the network layer.

To avoid redundancy, all five attributes should be unique in the database. Thus, any packet containing all five-attribute information is stored in the network flow database and used as per the client's requirements.

Considering this structure of the database, PostgreSQL is a good option to design this database for the following reasons:

1. **Primary Key and Indexing Capabilities:** PostgreSQL provides robust indexing mechanisms that can enhance the performance of queries involving primary keys. Given that "src_ip," "dest_ip," "src_port," "dest_port," and "ip_type" are all part of the primary key, PostgreSQL's indexing will ensure efficient retrieval and uniqueness checks, even with complex and large datasets.
2. **Concurrency and Performance:** With its Multi-Version Concurrency Control (MVCC), PostgreSQL ensures high performance and data consistency even under heavy concurrent load. This is crucial for a network flow database that may need to handle a high volume of simultaneous writes and reads.
3. **ACID Compliance:** PostgreSQL's strict adherence to ACID properties guarantees data integrity and reliability, which are essential for a network flow database where accurate and consistent data capture is critical.
4. **Extensibility and Customizability:** PostgreSQL allows for extensive customization through its support for custom functions, data types, and procedural languages. This can be leveraged to add specialized functionalities or optimize the handling and processing of network flow data.
5. **Scalability:** PostgreSQL can scale effectively both vertically (on a single server) and horizontally (across multiple servers using replication). This ensures that your database can grow as your network flow data increases.
6. **Community and Support:** PostgreSQL has a strong, active community and a wealth of resources for support and development. This ensures that any challenges faced during the implementation and maintenance of the network flow database can be promptly addressed.

Disadvantages of PostgreSQL:

- 1. Complexity of Configuration:** Setting up PostgreSQL for optimal performance can be complex and may require detailed configuration settings, especially when dealing with large-scale network flow data.
- 2. Storage Overhead:** PostgreSQL's storage mechanisms, such as MVCC and transaction logs, can introduce overhead in terms of disk space usage. This may become a concern when storing and managing large volumes of network flow data over time.
- 3. Scaling Challenges:** While PostgreSQL offers scalability options, such as vertical scaling and replication, horizontally scaling PostgreSQL for massive datasets or high throughput can be challenging and may require additional expertise and infrastructure.
- 4. Limited Built-in Network Analysis Features:** PostgreSQL is primarily a relational database management system (RDBMS) and may lack built-in features specifically tailored for network flow analysis, such as advanced packet inspection capabilities or real-time analytics functionalities.
- 5. Community and Support for Niche Use Cases:** While PostgreSQL has a strong community and support ecosystem, finding specialized assistance or resources for niche network flow analysis requirements within the PostgreSQL community may be more challenging compared to dedicated network analysis tools or platforms.
- 6. Latency in High-Volume Write Scenarios:** In scenarios where there are high volumes of concurrent writes to the database, PostgreSQL may experience latency issues, especially when ensuring data consistency and integrity through ACID compliance.

3. Key features required for network flow database

Relational/Non-Relational?	In memory/Disk based (RAM v/s SSD)	Publisher-Subscriber notification support for row/column updates (Y/N)	Partial key search support (Y/N)	Opensource License used	Client Library language support	Multi-threaded publisher/subscriber support
Relational	Disk-based	Yes	Yes	PostgreSQL is released under the PostgreSQL License, a liberal Open-Source license, similar to the BSD or MIT licenses.	C, C++, Delphi, Java, JavaScript (Node.js), Perl, PHP, Python and Tcl.	Yes, Multiprocessor architecture is used in PostgreSQL

PostgreSQL is a relational database, and it is disk-based, meaning it stores data on disks instead of in RAM. Being a relational database, PostgreSQL supports partial key search. It is open-source, which means it is free to use from anywhere and by anyone. PostgreSQL supports most of the general programming languages frequently used in data analytics. Publisher-subscriber notifications can be provided using methods like LISTEN and NOTIFY, which signal the listening client whenever another client changes the state of the database or table.

4. Additional Features of PostgreSQL

1. Caching in PostgreSQL for improved performance :

PostgreSQL employs caching to accelerate data access by reducing costly disk I/O operations. Caching focuses on three main areas:

- Table Data: Storing frequently accessed table data in memory.
- Indexes: Keeping index data in memory to speed up query execution.
- Query Execution Plans: Storing execution plans to save CPU cycles.
- Key Caching Mechanism:

shared_buffers: This setting is in the `postgresql.conf` configuration file determines the amount of memory allocated for caching table and index data. The value of `shared_buffers` specifies how many pages can be cached at any time.

Page Fetching and Caching:

- When a query is executed, PostgreSQL fetches the required page from disk and stores it in the `shared_buffers` cache.
- Subsequent accesses to the same page or any tuple within the page are served from the cache, avoiding disk I/O and speeding up the query.
- PostgreSQL uses the LRU algorithm to manage the cache.
- Pages that are least recently used are more likely to be evicted from the cache when new pages need to be loaded.
- Frequently accessed pages remain in the cache, ensuring faster access times.

Resizing the Cache for Performance Improvement:

- The default size of `shared_buffers` might not be optimal for all workloads. Increasing its size can significantly improve performance for read-heavy operations.
- However, setting it too high can lead to inefficient memory usage and potential performance degradation, especially if it exceeds the available system memory.

Modifying shared_buffers:

- Locate the PostgreSQL configuration file (named `postgresql.conf`), typically located in the `/etc/postgresql/<version_number>/main/` directory on Ubuntu.
- To edit this file, administrative access is required. Open a terminal, type `sudo nano postgresql.conf`, and press Enter.

The following changes were made to the configuration file:

1. shared_buffers = 8GB:

- **Function:** This setting determines the amount of memory the database server uses for shared memory buffers.
- **Impact:** Increasing `shared_buffers` can improve performance by allowing more data to be cached in memory, reducing disk I/O. However, it should be set to a fraction of the total system memory, typically 25% of the available RAM.

2. temp_buffers = 2GB:

- **Function:** This setting determines the amount of memory allocated for temporary buffers used by each database session.
- **Impact:** Setting a high value for `temp_buffers` can be beneficial if your queries create a lot of temporary tables or do large sorts, but it increases the memory footprint per session.

3. work_mem = 2GB:

- **Function:** This setting determines the amount of memory allocated for internal sort operations and hash tables before spilling to disk.
- **Impact:** Increasing `work_mem` can significantly improve performance of complex queries that require sorting or hashing.

4. maintenance_work_mem = 1GB:

- **Function:** This setting specifies the maximum amount of memory to be used by maintenance operations such as `CREATE INDEX`, and `ALTER TABLE ADD FOREIGN KEY`.
- **Impact:** Higher `maintenance_work_mem` speeds up maintenance operations. This setting doesn't affect regular queries.

5. wal_buffers = 16MB:

- **Function:** This setting determines the amount of memory allocated for the Write-Ahead Logging (WAL) buffers.
- **Impact:** Increasing `wal_buffers` can improve write performance, especially for workloads with a high rate of write operations.

6. checkpoint_completion_target = 0.9

- **Function:** This setting is a fraction that determines the target of the interval between checkpoints for writing out dirty buffers. A value of 1.0 means spreading the checkpoint writes evenly over the entire checkpoint interval.
- **Impact:** Higher values (e.g., closer to 1.0) spread the checkpoint I/O more evenly, which can lead to more stable performance and fewer spikes in disk I/O. Lower values can cause bursts of I/O, potentially leading to performance degradation.

Summary of Impacts

- **Memory Utilization:** A significant increase in memory allocations for various buffers and operations is done. This can lead to improved performance, especially for workloads that benefit from large memory allocations for caching, sorting, and maintenance operations.
- **Performance Optimization:** The changes are geared towards optimizing the performance for read-heavy and write-heavy operations, with considerations for both query execution and database maintenance tasks.

2. Usage of Temporary Tables in PostgreSQL to Improve Performance

Definition and Scope:

Temporary tables in PostgreSQL are special table structures used to store intermediate results temporarily.

They exist only within the session that created them and are automatically dropped at the end of the session.

1. Intermediate Result Storage:

- Temporary tables store intermediate results of complex queries or calculations.
- By breaking down complex operations into simpler steps and storing intermediate results in temporary tables, you can reduce the computational load and improve query performance.

2. Reducing Disk I/O:

- Temporary tables hold data in memory for the duration of a session, which reduces the need for repeated disk reads and writes. This is particularly beneficial for operations that require frequent access to the same data, as fetching from memory is much faster than disk I/O.

3. Efficient Resource Usage:

- Reduced Disk Space: Temporary tables are dropped automatically at the end of a session, ensuring that they do not consume disk space permanently.
- Memory Management: PostgreSQL can efficiently manage memory usage for temporary tables, which helps in optimizing resource utilization for queries and transactions involving these tables.

4. Reduced Transaction Overhead:

- No Logging: Changes to temporary tables are not logged to the Write-Ahead Log (WAL), reducing the overhead associated with ensuring data durability. This leads to faster data modifications.

5. Performance testing of PostgreSQL using Python modules

We have written code to test the performance of PostgreSQL by performing bulk insertions, deletions, and updates on a large dataset. Here's a brief explanation:

The dataset used in performance measurement, is prepared by a Python script which generates random unique IP address and port number pairs, simulating real network packet exchanges. It includes functions to generate random IPv4 and IPv6 addresses, random ports, and to randomly select the IP type. The script initializes parameters, tracks unique pairs with a dictionary, and uses a counter to ensure a large number of unique request-response pairs. Each pair and its reverse (representing the response packet) is written to a CSV file, ensuring unique and realistic network packet data.

Code Overview (batch operations):

1. Setup and Libraries:

- **Import:** Import necessary libraries such as ``psycopg2`` for PostgreSQL interaction, ``datetime`` for time calculations, and ``matplotlib.pyplot`` for plotting results.
- **Data Reading Function:** The ``read_data_from_file`` function reads data from a CSV file and processes it into a list of tuples.

2. Main Functionality:

- **Database Connection:** Establishes a connection to the PostgreSQL database using ``psycopg2``.
- **Temporary Tables:** Creates temporary tables (``temp_delete_table`` and ``temp_update_table``) to store intermediate data for deletion and update operations. (This is used only in the shared memory configuration. In the below graphs temporary tables are not used to provide a grasp on the raw performance of PostgreSQL)

3. Batch Operations Menu:

- A menu-driven interface allows the user to select operations:
- **Delete Tuples:** Reads data, copies it to a temporary table, and performs deletions in batches.
- **Insert Tuples:** Reads data and inserts it into the main table (``flow_table``) in batches using the ``COPY`` command.
- **Update Tuples:** Reads data, copies it to a temporary table, and performs updates in batches.

4. Time Measurement and Display:

- **Print Table:** A function (``print_table``) is defined to print the timing results in a tabular format.
- **Plot Graph:** Another function (``print_graph``) is provided to plot the timing results using ``matplotlib``.

5. Error Handling and Cleanup:

- **Exception Handling:** Catches and prints any errors that occur during database operations.
- **Cleanup:** Ensures that database cursors and connections are properly closed in the ``finally`` block.

Usage Flow:

1. The user runs the script, which connects to the PostgreSQL database.
2. A menu is displayed for the user to choose between deleting, inserting, or updating tuples.
3. The user inputs the batch size and number of batches.
4. The selected operation is performed in batches, with the time taken for each batch recorded and displayed.
5. After completing the operations, the script prints the total time taken and optionally plots a graph of the timings.

This structure ensures efficient handling of large datasets, provides clear performance metrics, and maintains the integrity of the database operations through proper error handling and resource management.

GitHub:

https://github.com/AmshuRao/HPE_CTY_PROJECT/blob/master/WEEK_7/batch_crud_operations.py

Code Overview (normal operations):

Here each tuple is inserted, deleted, and updated individually.

Code Overview:

1. Setup and Libraries:

- Import: The code imports ``psycopg2`` for PostgreSQL database interaction, ``datetime`` for measuring time intervals, and ``matplotlib.pyplot`` (though ``matplotlib`` is not used in the code).

2. Main Functionality:

- Database Connection: The code establishes a connection to a PostgreSQL database using ``psycopg2`` and creates a cursor object to execute SQL queries.
- Time Tracking Lists: Lists are initialized to store the time taken for each deletion, insertion, and update operation.

3. Menu-Driven Interface:

- User Menu: A menu is displayed to allow the user to choose between three operations: deleting, inserting, or updating tuples, or exiting the script.

4. Operations:

- Insertion operation is performed by inserting the csv file data (``data_100_tuples.csv``, ``data_1000_tuples.csv``, etc.) into the database serially.
- Delete operation is performed by reading the files (``data_100_tuples.csv``, ``data_1000_tuples.csv``, etc.) and deleting serially.
- Update operation is performed by reading the files (``data_100_tuples.csv``, ``data_1000_tuples.csv``, etc.) and updating the `src_port` to 10 0 in the respective tuple.

5. Time Measurement:

- The code records the time before and after each operation to measure the duration.
- It calculates and prints the average time taken for deletions, insertions, and updates for each file.

6. Error Handling:

- The code includes a try-except block to catch and print any exceptions that occur during database operations.

Usage Flow:

- 1. Run Script:** The user runs the script, which connects to the PostgreSQL database.
- 2. Display Menu:** The script displays a menu for the user to select an operation.
- 3. Perform Operation:** Based on user input, the script performs the selected operation (delete, insert, update) on the database.
- 4. Time Calculation:** The script measures the time taken for each operation and calculates the average time per record for each file.
- 5. Display Results:** The script prints the average time taken for each file.

Conclusion:

This structure provides a way to measure the performance of individual deletions, insertions, and updates on a PostgreSQL database. It records the time taken for each operation and calculates average times, helping users understand the performance characteristics of their database operations.

GitHub:

https://github.com/AmshuRao/HPE_CTY_PROJECT/blob/master/WEEK_7/normal_opertaions.py

6. RESULTS

System Overview: we have used a System with 8GB RAM, and 1TB storage. All the data is recorded using the UBUNTU operating system (version 22.04).

1. Performance results for the automation code:

The table below shows performance metrics for different numbers of batch sizes and shared memory sizes (keeping 10 lakh tuples in the database):

Number of Tuples in each batch	Shared Memory Size	Insert (ms)	Delete (ms)	Update (ms)
100	128 MB	172.18	488.66	4589.2
	8 GB	105.52	22.88	2206.82
1000	128 MB	136.57	422.75	482.004
	8 GB	17.25	10.66	380.79
10000	128 MB	13.56	132.05	220.68
	8 GB	6.43	3.29	46.71
1 lakh	128 MB	18.17	10.35	140.08
	8 GB	4.87	3.92	14.27

Fig: reading of the performance of PostgreSQL

- **Observation:** From the table, we can see that the time required for insertion, deletion, and updating in Postgres is less compared to other databases. And shared memory can increase the efficiency of these operations which is evident by the data.

	100		1000		10,000		1,00,000	
	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory
Insert	6.7	0.5	37.2	1.2	47.4	1.1	58.3	1.7
Delete	94.7	1.4	96.7	1.3	97.7	1.5	55.4	2
Update	97.3	2.5	97	1.6	92.4	2.1	90.4	2.3

Fig: CPU and Memory Usage

- The CPU and memory consumption for optimized setting consumes more memory and CPU.

Data size of 10 lakh and batch operations were executed in batches of 100,1000, 10000, and 100000

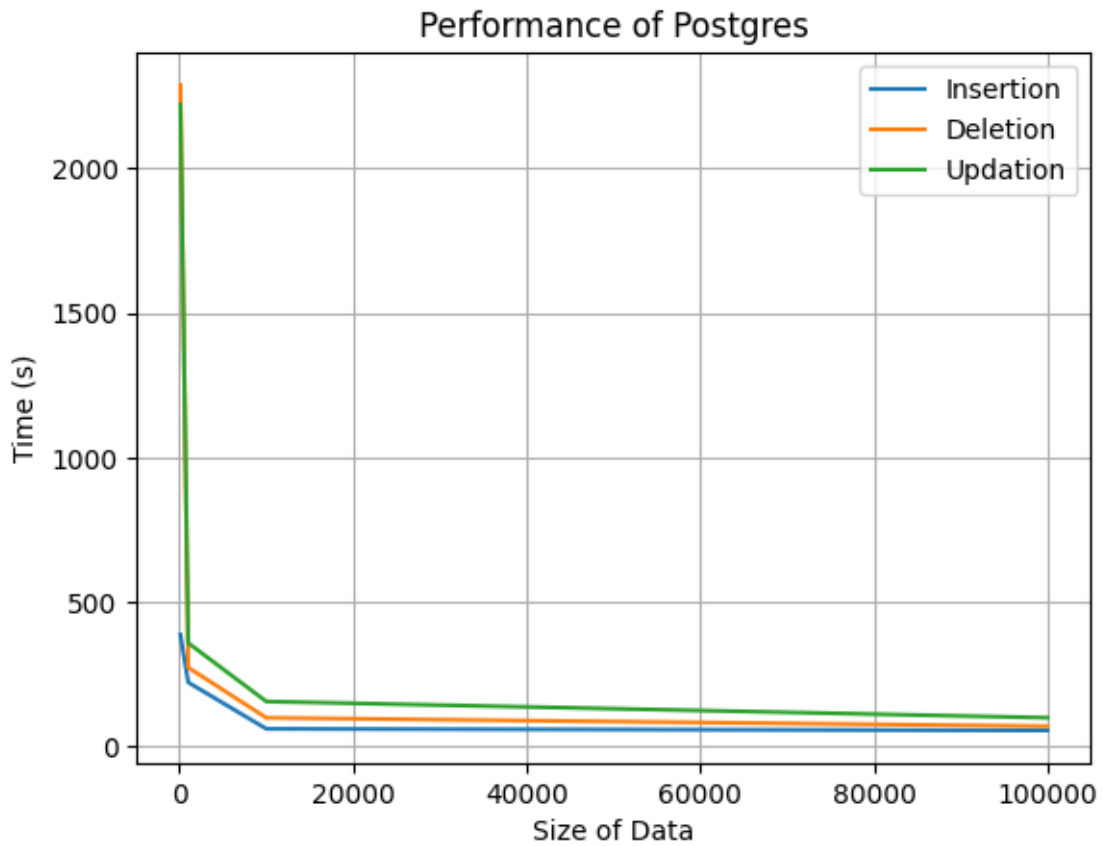
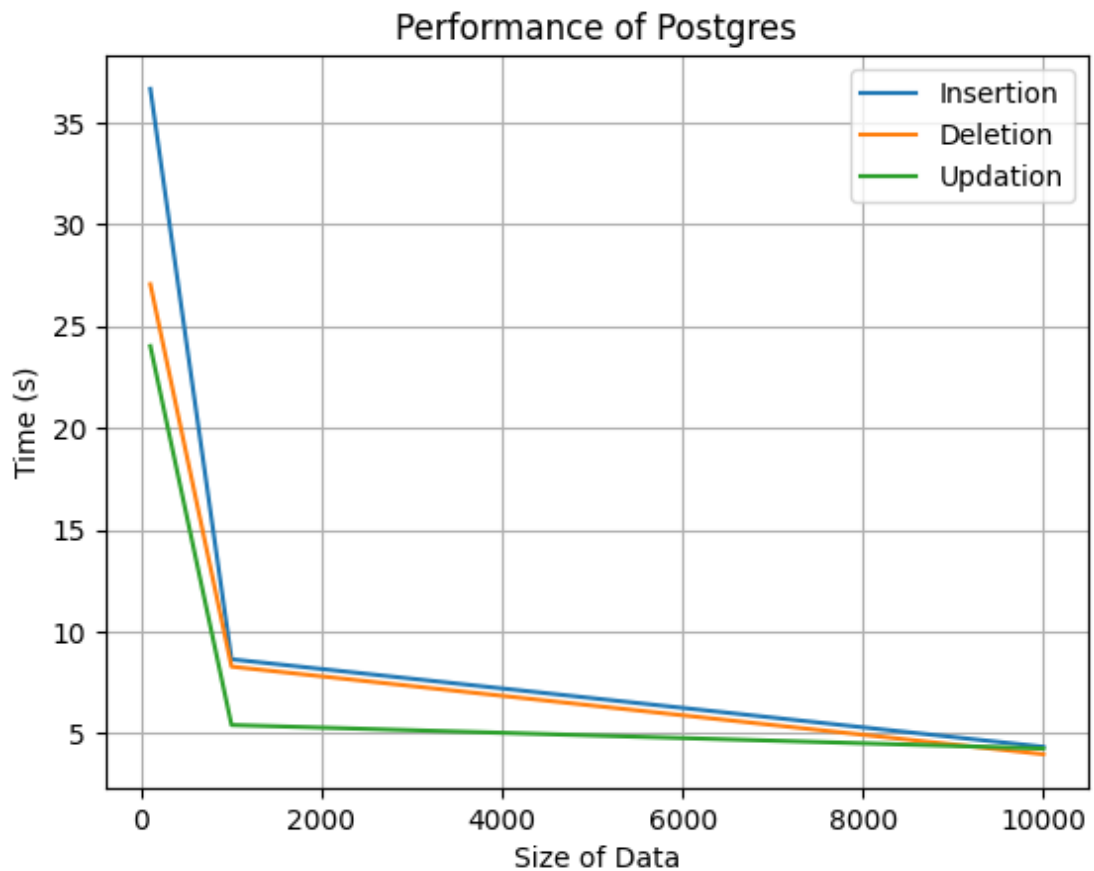


Fig: Performance of PostgreSQL

Data size of 1 lakh and batch operations were executed in batches of 100,1000, and 10000



Data size of 10 thousand and batch operations were executed in batches of 10,1000, and 1000

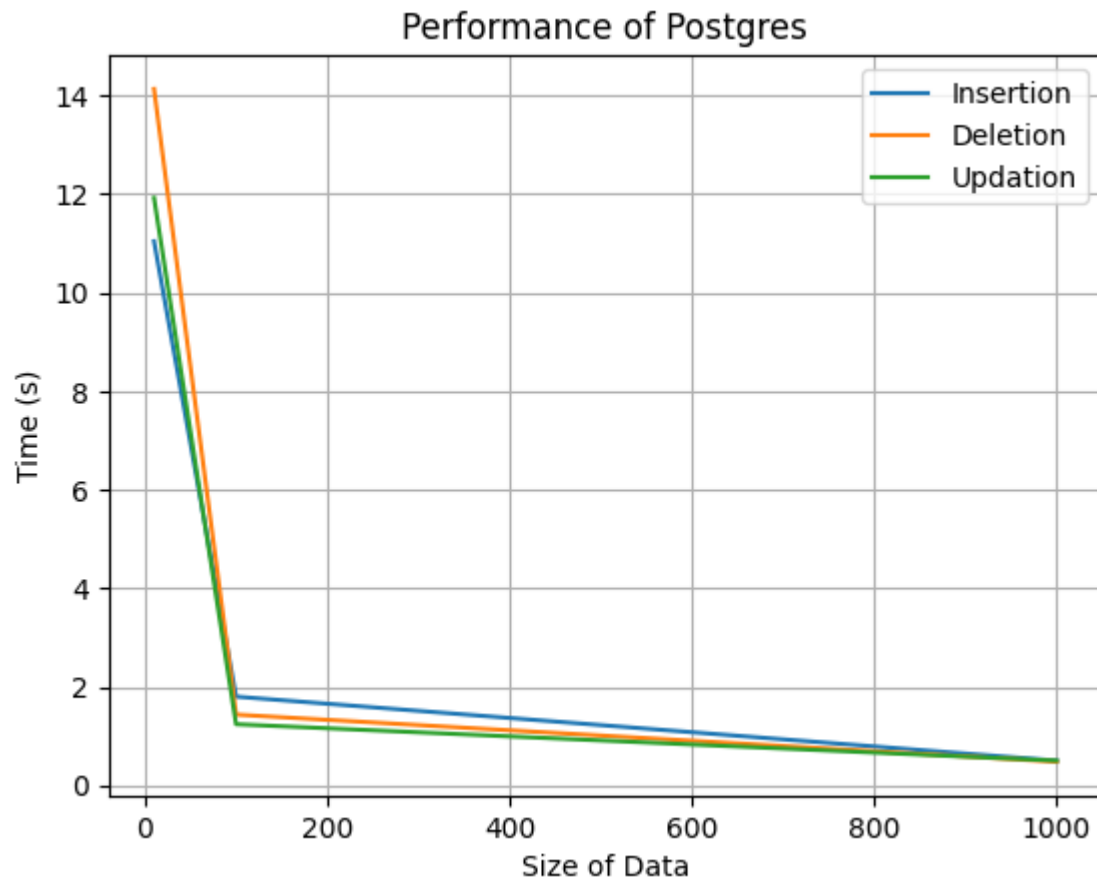


Fig: Performance of PostgreSQL

File Name	Process name	CPU%	Mem%
Normal Operations	Insert	11.5	0.5
	Delete	36.9	0.5
	Update	41.3	2.1
Batch Operations	Insert	24.3	0.5
	Delete	58.3	0.5
	Update	24.3	0.7

Fig: CPU and Memory Utilization of PostgreSQL (without optimization settings)

2. Performance comparison of Normal operations with the YCSB tool:

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source database benchmarking suite and a critical analytical component of cloud-based database management system (DBMS) evaluation. It allows users to comparatively measure how various modern SQL and NoSQL DBMS perform simple database operations on generated datasets.

We used the JDBC binding in Yahoo Cloud serving benchmark to execute the queries in PostgreSQL.

The structure of the default table in PostgreSQL jdbc – binding is as follows:

```
CREATE TABLE usertable (  
    YCSB_KEY VARCHAR (255) PRIMARY KEY,  
    FIELD0 TEXT, FIELD1 TEXT,  
    FIELD2 TEXT, FIELD3 TEXT,  
    FIELD4 TEXT, FIELD5 TEXT,  
    FIELD6 TEXT, FIELD7 TEXT,  
    FIELD8 TEXT, FIELD9 TEXT  
);
```

These are the types of workloads in YCSB

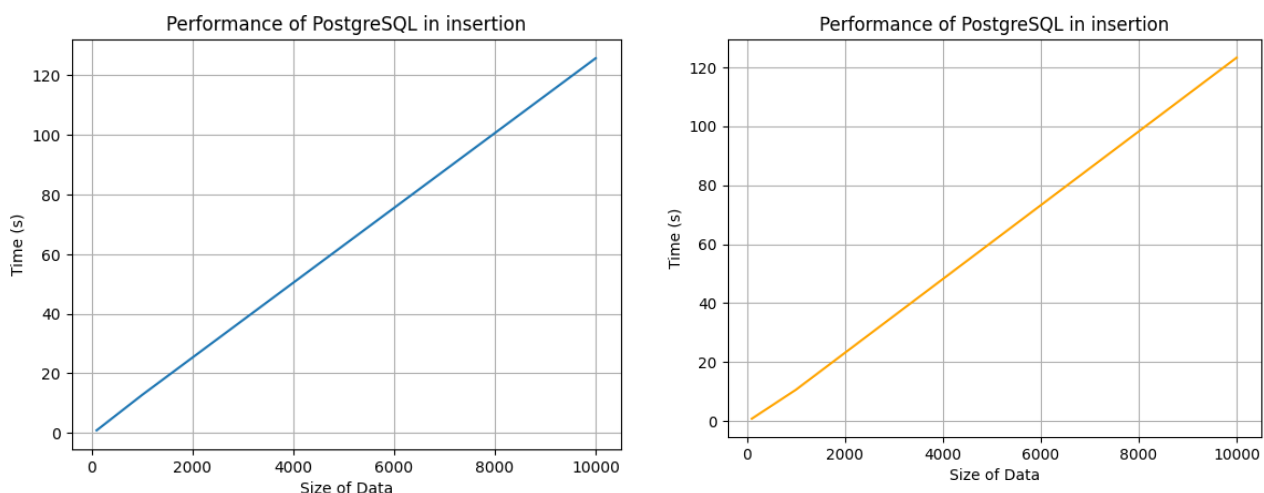
- Workload A: Update heavy workload: 50/50% Mix of Reads/Writes
- Workload B: Read mostly workload: 95/5% Mix of Reads/Writes
- Workload C: Read-only: 100% reads
- Workload D: Read the latest workload: More traffic on recent inserts
- Workload E: Short ranges: Short range-based queries
- Workload F: Read-modify-write: Read, modify, and update existing records

The results of all these workloads are given in the following link: ([GitHub](#))

1. Insertion of tuples of size 100, 1000 and 10000

Count	Python Script Values	YCSB Values	Ratio
100	0.84	0.83	1.01
1000	12.7	10.64	1.19
10000	125.76	123.23	1.02

Fig: Comparison between YCSB and PostgreSQL scripts

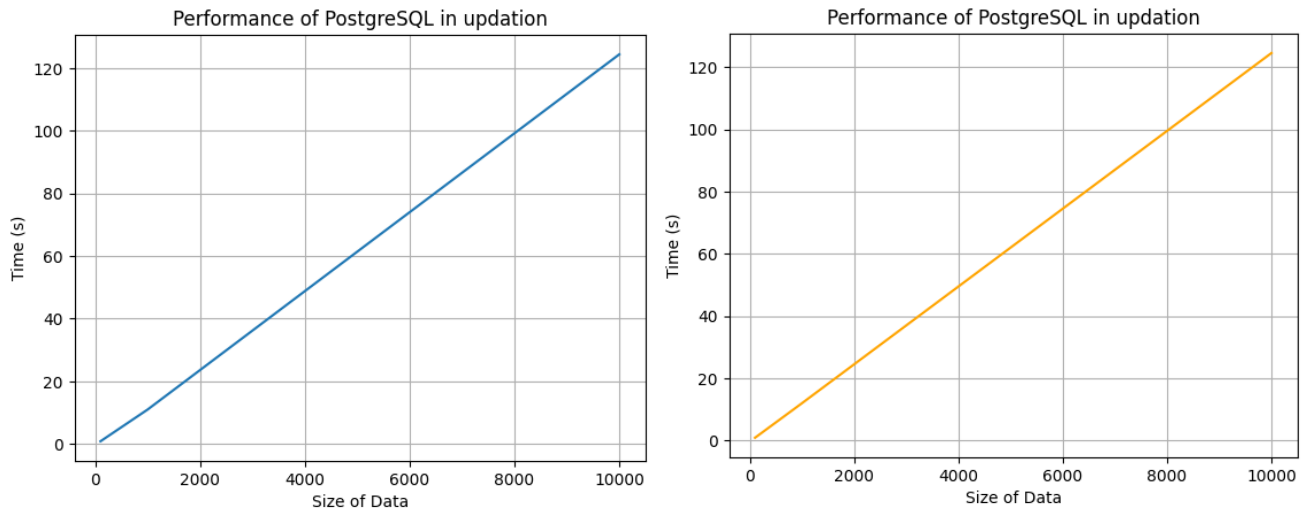


The trends above clearly indicate that our implemented code performs on par with the YCSB tool, confirming the accuracy of our work.

1. Updating the tuples of size 100, 1000 and 10000

Count	Python Script Values	YCSB Values	Ratio
100	0.83	0.84	0.98
1000	10.98	11.959	0.91
10000	124.46	124.63	0.99

Fig: Comparison between YCSB and PostgreSQL scripts



The trends above clearly indicate that our implemented code performs on par with the YCSB tool, confirming the accuracy of our work.

5. Conclusion

PostgreSQL emerges as a highly desirable database for managing a network flow database due to its robust feature set and reliable performance. Its disk-based storage mechanism ensures data persistence and integrity, essential for maintaining comprehensive records of network traffic. PostgreSQL's support for partial key searches and extensive indexing capabilities facilitates efficient querying and retrieval of network flow data. The open-source nature of PostgreSQL makes it a cost-effective solution, accessible to organizations of all sizes without licensing constraints.

The database's ability to handle concurrent operations through its multi-process architecture ensures high performance and scalability, crucial for the dynamic and high-volume nature of network flow data. PostgreSQL's support for various programming languages commonly used in data analytics enhances its versatility and ease of integration into existing systems.

Furthermore, PostgreSQL's logical replication feature supports a multi-threaded publisher-subscriber model, enabling real-time data distribution and synchronization across multiple instances. This is particularly beneficial for network monitoring and analysis, where timely data availability is critical.

In summary, PostgreSQL offers a reliable, scalable, and cost-effective solution for a network flow database, backed by a strong community and extensive support resources. Its comprehensive feature set and high performance make it well-suited to meet the demanding requirements of network flow data management and analysis.

7. References

- GitHub Repository: https://github.com/AmshuRao/HPE_CTY_PROJECT/tree/master
- An Overview of Caching for PostgreSQL: <https://severalnines.com/blog/overview-caching-postgresql/>
- PostgreSQL documentation: <https://www.postgresql.org/docs/>
- Using PostgreSQL in Python: <https://www.datacamp.com/tutorial/tutorial-postgresql-python>
- YCSB tool: <https://github.com/brianfrankcooper/YCSB>