Hewlett Packard Enterprise

HPE CTY

Network Flow Database

A Report On

ARANGO DB

Documented By:
DEEKSHA.M.ACHARYA

ARANGO DB:

Arango DB is an open source database management system that supports data models : document , graph and key-value. Arango DB is written in C++ and built to work at scale ,in the cloud or on-premises.

Some of the features of Arango DB include:

- **Multi-model:** ArangoDB supports multiple data models including key-value pairs, documents (using JSON-like documents), and graphs (with nodes, edges, and properties). This allows developers to work with different types of data within a single database system.

- **No-SQL:** ArangoDB falls under the category of NoSQL databases, which means it provides flexible schema design and is well-suited for applications requiring high scalability, flexibility, and performance

- **ACID transactions:** ArangoDB supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity and consistency even in distributed environments.

- **Query language:** ArangoDB uses AQL (ArangoDB Query Language) for querying data. AQL is SQL-like and allows for complex queries across different data models including documents and graphs.

USE CASES:

1. **Arango Db as graph database:**

   1. **Fraud detection:** Uncover illegal activities by discovering difficult-to-detect patterns. ArangoDB lets you look beyond individual data points in disparate data sources, allowing you to integrate and harmonize data to analyze activities and relationships all together, for a broader view of connection patterns, to detect complex fraudulent behavior such as fraud rings.
   2. **Recommendation engine:** Suggest products, services, and information to users based on data relationships. For example, you can use ArangoDB together with PyTorch Geometric to build a movie recommendation system , by analyzing the movies users watched and then predicting links between the two with a graph neural network (GNN).

2. **Arango DB as document database:**

   1. **E-commerce systems**: ArangoDB combines data modeling freedom with strong consistency and resilience features to power online shops and ordering systems. Handle product catalog data with ease using any combination of free text and structured data, and process checkouts with the necessary transactional guarantees.

   2. **Internet of things**: Collect sensor readings and other IoT data in ArangoDB for a single view of everything. Store all data points in the same system that also lets you run aggregation queries using sliding windows for efficient data analysis.

3. **Arango DB as key value document**:

   1. **Session management in web applications:**
   In web applications, session management is essential for maintaining user state across multiple interactions. ArangoDB's key-value pair functionality offers an efficient solution for storing session data. Each session can be represented as a key-value pair, where the session ID serves as the key, and relevant session data, such as user preferences, authentication tokens, and shopping cart items, serves as the corresponding value. ArangoDB's distributed architecture ensures rapid access to session data, even under heavy loads. Moreover, developers can leverage ArangoDB's flexibility to implement TTL-based expiration for sessions, automatically removing stale session data from the database. This makes ArangoDB a reliable choice for seamless and efficient session management in web applications.

Why ArangoDB may suit a network flow database:

- **Schema Flexibility**: ArangoDB's flexible data model allows you to represent network flow data with varying structures, such as source IP, destination IP, source port, destination port, protocol, etc. This flexibility can accommodate diverse data types and relationships commonly found in network flow data.

- **Multi-model Database**: ArangoDB supports multiple data models including document, key-value, and graph. This versatility enables you to model network flow data according to its characteristics. For example, you can represent network nodes and connections as vertices and edges in a graph model.

- **Query Language**: ArangoDB provides AQL (ArangoDB Query Language), which is powerful and expressive. With AQL, you can perform complex queries to analyze network flow data, such as filtering, aggregation, and graph traversals.

- **Scalability**: ArangoDB is designed to scale horizontally, allowing you to distribute network flow data across multiple servers to handle large volumes of traffic. This scalability is essential for growing network infrastructures and high-throughput environments.

- **Built-in Indexing:** ArangoDB supports indexing on document attributes and graph properties, which can improve query performance when searching for specific network flow records or patterns.

Why ArangoDB may not suit a network flow database:

- **Performance Concerns:** While ArangoDB is capable of handling large datasets, its performance may not match that of specialized databases optimized for specific use cases, such as network flow analysis tools that are finely tuned for high-speed packet processing.

- **Complexity:** Managing and querying data in a multi-model database like ArangoDB may introduce additional complexity compared to using a specialized network flow database that offers streamlined functionality tailored specifically for network traffic analysis.

- **Resource Requirements:** ArangoDB requires sufficient computational resources (CPU, memory, disk space) to operate efficiently, especially in distributed setups. Deploying and maintaining such infrastructure can be resource-intensive, particularly for smaller organizations or projects with limited budgets.

- **Data Consistency:** In distributed environments, ensuring data consistency and integrity across multiple nodes can be challenging, especially during network partitions or failures. Although ArangoDB provides mechanisms like replication and sharding for data redundancy and fault tolerance, achieving strong consistency guarantees may require additional configuration and monitoring.

ARANGODB DEFAULT CONFIGURATION FILE:

```
[server]
endpoint = tcp://127.0.0.1:8529
authentication = true
uid = arangodb
gid = arangodb

# Storage engine settings
[storage]
directory = /var/lib/arangodb3

# Logging settings
[logging]
file = /var/log/arangodb3/arangodb.log
level = info

# Database settings
[database]
wait-for-sync = false

[rocksdb]
```

ARANGODB PROPERTIES:

Relational/Non Relational	**In-memory /on-disc**	**Publisher-Subscriber notification support for row or column updates:**	**Partial key search support**	**Open source license used**	**Client Library language support**	**Multi-threaded publisher/subscriber support**
Non relational	**Both**	**No**	**Yes**	**Apache2.0 license**	**Javadriver/go driver/C#/.NET driver/Node.js driver/Python driver**	**Yes**

Partial key search code: https://github.com/deekshamacharya/HPE-CTY-PROJECT/blob/OS-LAB/ARANGODB/partial_search.py

ArangoDB's MMFiles storage engine allows you to store data primarily in memory. ArangoDB also supports disk-based storage through the RocksDB storage engine.

Through its Change Data Capture (CDC) feature, ArangoDB can notify subscribers about changes in the data, enabling real-time data synchronization and event-driven architectures.

ADDITIONAL INVESTIGATION DETAILS ABOUT ARANGODB:

1. PARTIAL SEARCH:

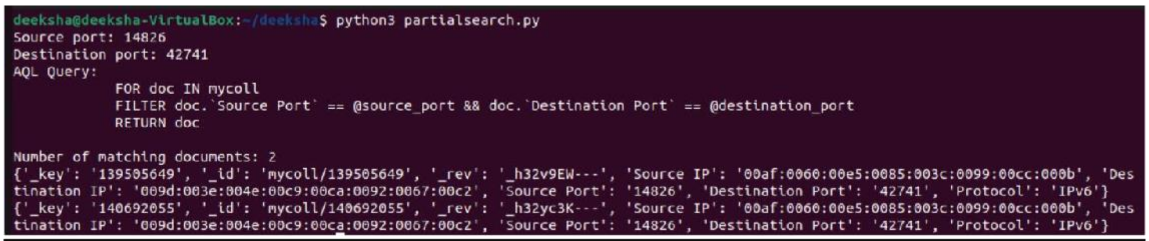
LINK: https://github.com/deekshamacharya/HPE-CTY-PROJECT/blob/OS-LAB/ARANGODB/partial_search.py

Partial search involves finding matches based on incomplete or partial information provided by the user, allowing for flexible and adaptable querying of datasets. In the context of network data, partial search by source and destination ports enables users to identify connections or flows based on partially specified port values, enhancing search flexibility and query precision.

The script facilitates a partial search functionality within an ArangoDB collection, specifically targeting documents based on partial matches of their source and destination ports. The query utilizes partial string matching to find documents with source and destination ports that contain the specified partial port values.

The AQL query is executed using the aql.execute method with the partial port values as bind variables.

OUTPUT:



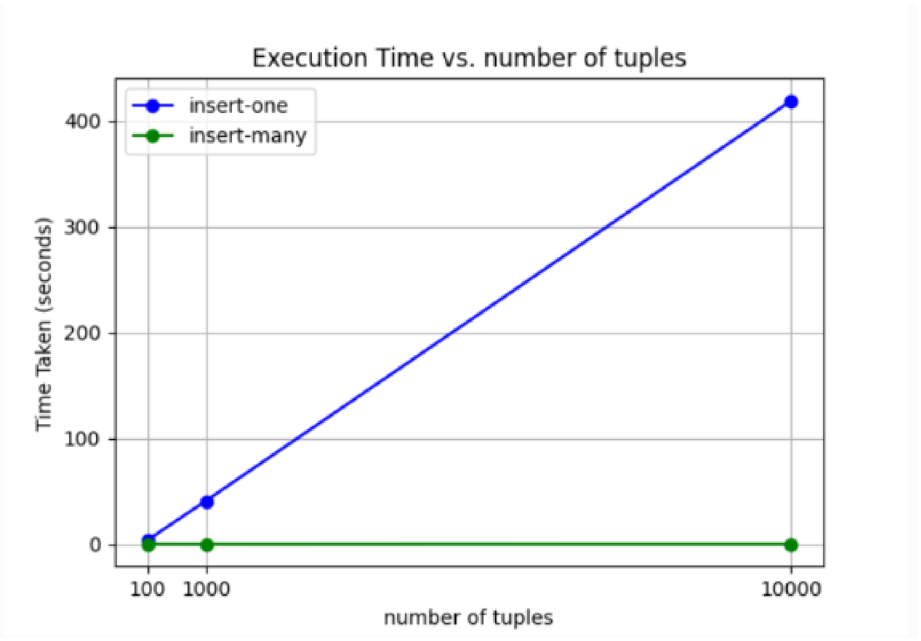
```
deeksha@deeksha-VirtualBox:~/deeksha$ python3 partialsearch.py
Source port: 14826
Destination port: 42741
AQL Query:
          FOR doc IN mycoll
          FILTER doc.`Source Port` == @source_port && doc.`Destination Port` == @destination_port
          RETURN doc

Number of matching documents: 2
{'_key': '139505649', '_id': 'mycoll/139505649', '_rev': '_h32v9EW---', 'Source IP': '00af:0060:00e5:0085:003c:0099:00cc:000b', 'Des
tination IP': '009d:003e:004e:00c9:00ca:0092:0067:00c2', 'Source Port': '14826', 'Destination Port': '42741', 'Protocol': 'IPv6'}
{'_key': '140692055', '_id': 'mycoll/140692055', '_rev': '_h32yc3K---', 'Source IP': '00af:0060:00e5:0085:003c:0099:00cc:000b', 'Des
tination IP': '009d:003e:004e:00c9:00ca:0092:0067:00c2', 'Source Port': '14826', 'Destination Port': '42741', 'Protocol': 'IPv6'}
```

2. INSERT MANY FUNCTIONALITY:

LINK: https://github.com/deekshamacharya/HPE-CTY-PROJECT/blob/OS-LAB/ARANGODB/insert_many.py

ArangoDB's insert_many() functionality provides a streamlined approach for bulk insertion of documents into collections, offering efficiency and atomicity in the process. This feature is particularly beneficial when dealing with large datasets, as it minimizes network overhead by allowing multiple documents to be sent in a single request. Ensuring data integrity, insertMany() guarantees that either all documents are successfully inserted or none are, enabling robust error handling and ensuring consistency in the database. By leveraging insertMany(), developers can optimize performance and streamline operations, especially in scenarios where individual document insertion would be cumbersome or inefficient.

OPERATIONS	**NUMBER OF TUPLES**		
	100	**1000**	**10000**
Insert-One	**3.5166**	**40.8**	**419.06**
Insert-Many	**0.0717**	**0.0407**	**0.1213**



BRIEF DESCRIPTION ABOUT THE CODE:

NORMAL OPERATIONS:

The script is designed to handle a dataset consisting of tuples with five attributes: source IP, destination IP, source port, destination port, and protocol. It inserts these tuples into the database individually, starting with 100 entries and increasing in size with subsequent batches (e.g., 1000 entries, 10000 entries, etc.). After insertion, it performs deletion operations, removing data from the database that matches the entries in CSV files. The deletions are conducted individually for each tuple in the CSV files with counts of 100, 1000, 10000, etc., and the time taken for each deletion operation is recorded. Additionally, the script updates the source port of the tuples to 0 individually for each tuple in the CSV files with counts of 100, 1000, 10000, etc., noting the time taken for each update operation. This approach ensures accurate measurement of time for each individual operation, providing insights into performance at varying dataset sizes.This Python script interacts with ArangoDB, a NoSQL database, to perform insertion, deletion, and update operations based on CSV data

LINK: https://github.com/deekshamacharya/HPE-CTY-PROJECT/blob/OS-LAB/ARANGODB/normal_operations.py

Libraries Used:

- **time**: For measuring time durations.
- **matplotlib.pyplot**: For plotting graphs.
- **csv**: For reading CSV files.
- **arango.ArangoClient**: For connecting to ArangoDB and performing operations.

Script Structure:

- **Database Connection:** Connects to the ArangoDB server using the ArangoClient.

Function Definitions:

- **insert_documents_from_csv**: Inserts documents from a CSV file into a collection.
- **delete_documents**: Deletes documents from the collection based on given attributes.
- **delete_documents_from_csv**: Deletes documents from the collection based on entries in a CSV file.
- **update_documents**: Updates documents in the collection based on given attributes.
- **update_documents_from_csv**: Updates documents in the collection from a CSV file.
- **plot_graph:** Plots a graph showing time taken vs. number of entries for an operation.

Main Function:

- Allows the user to choose an operation (insert, delete, update).
- Calls corresponding functions based on the user's choice.
- Plots a graph showing the time taken for each operation.

Functionality:

- **Insertion:** Inserts documents from CSV files into the collection, measuring the time taken.
- **Deletion:** Deletes documents from the collection either based on specific attributes or by reading entries from CSV files.
- **Update:** Updates documents in the collection either based on specific attributes or by reading entries from CSV files.

Data Source:

The script expects CSV files with different numbers of entries for testing insertion, deletion, and update operations.

BATCH OPERATIONS:

This script provides functionality for batch insertion, deletion, and update operations in ArangoDB, along with logging capabilities and error handling.

LINK: https://github.com/deekshamacharya/HPE-CTY-PROJECT/blob/OS-LAB/ARANGODB/batch_operations.py

ArangoDB API Version: The script interacts with ArangoDB using the arango.ArangoClient library. The specific version of the ArangoDB API depends on the version supported by the installed arango.ArangoClient library.

Libraries Used:

- **arango.ArangoClient**: Used for connecting to the ArangoDB server and performing database operations.
- **csv:** Utilized for reading data from CSV files.
- **time:** Employed for measuring time durations during various database operations.
- **logging:** Configured to log errors encountered during database operations.
- **matplotlib.pyplot:** Used for plotting graphs to visualize the time taken for different database operations.

It consists of several functions:

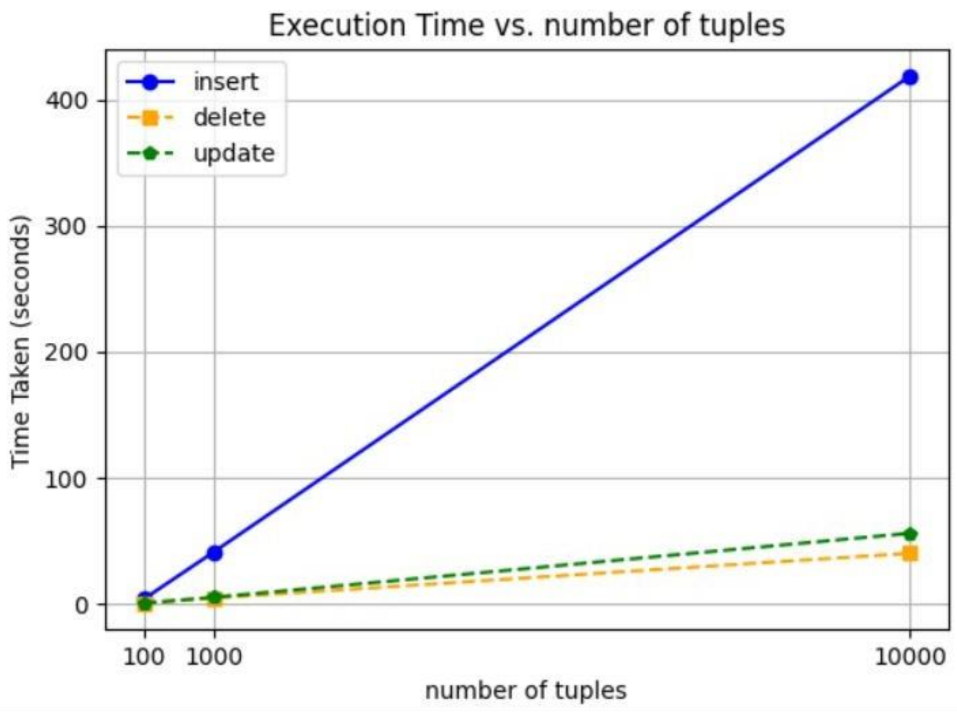
- **connect_to_database**: Establishes a connection to the ArangoDB server.
- **batch_insert_documents**: Inserts documents into the database in batches.
- **plot_graph**: Plots a graph showing time taken vs. number of batches for insertion.
- **batch_delete_documents_from_csv**: Deletes documents from the database based on entries in a CSV file in batches.
- **update_documents:** Updates documents in the database based on specified attributes.
- **update_documents_from_csv**: Updates documents in the database from a CSV file in batches.
- **Main function:** Allows the user to select an operation (insertion, deletion, or update) and performs the corresponding action.

MEASUREMENT OF SCALE AND PERFORMANCE:

NORMAL OPERATIONS: (default configurations):

OPERATIONS	NUMBER OF TUPLES		
	100	1000	10000
INSERT	3.5166	40.8	419.06
DELETE	0.459	4.512	40.0858
UPDATE	0.499	5.0716	55.9545

(Measured in seconds)



CPU AND MEMORY UTILIZATION:

The CPU and memory utilization has been recorded with the help of top command by getting the process ID of ArangoDB server running on the system.

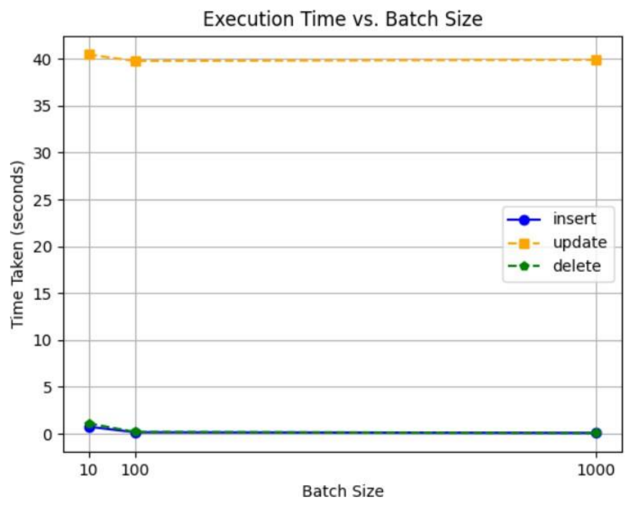
	CPU	**MEMORY**
INSERT	**2**	**0.8**
DELETE	**85**	**0.8**
UPDATE	**88**	**0.8**

It can be inferred from the above results that the memory consumption is least for the operations performed but CPU utilization is high for update and delete compared to insert operation.

BATCH OPERATIONS:

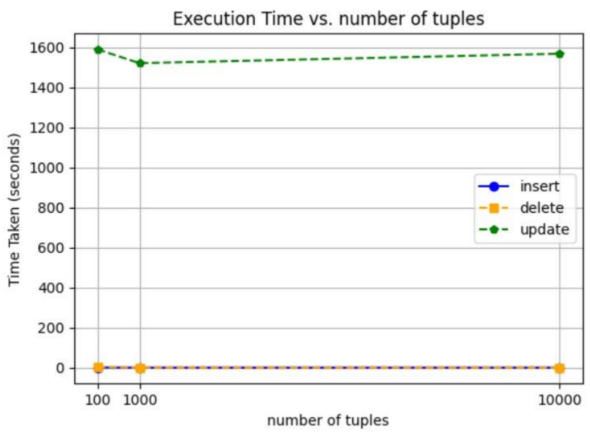
TOTAL NUMBER OF TUPLES(10000):

OPERATIONS	NUMBER OF TUPLES		
	10	100	1000
INSERT	0.7492	0.1573	0.0863
UPDATE	40.4613	39.7757	39.9091
DELETE	1.1098	0.1902	0.0737



TOTAL NUMBER OF TUPLES(100000):

OPERATIONS	NUMBER OF TUPLES		
	100	1000	10000
INSERT	1.8439	0.8817	0.9407
UPDATE	1589	1520	1567
DELETE	4.309	0.76	0.71



MEASUREMENT OF SCALE AND PERFORMANCE USING YCSB TOOL:

YCSB: Measures a database performance.

How it works:

2 phases:

- **Load**: loads data into database.
- **Run**: performs operations like read,scan etc.

What it measures:

- **Throughput:** number of operations per second
- **Latency**: time taken to perform operation on single record
- **Runtine**

Workload provided by YCSB:

WORKLOAD A:
- READ : 50%
- UPDATE:50%

WORKLOAD B:
- READ:95%
- UPDATE 5%

WORKLOAD C:
- READ:100%

WORKLOAD D :
- READ:95%
- INSERT:5%

WORKLOAD E:
- SCAN:95%
- INSERT:5%

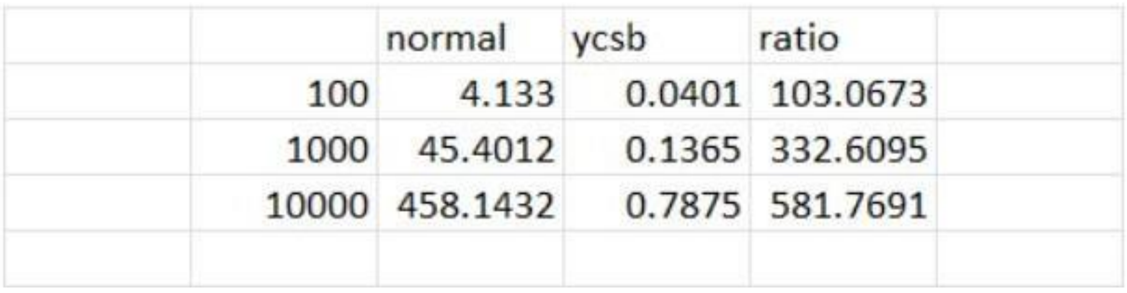
WORKLOAD F:
- READ:50:
- READ-MODIFY-WRITE:50%

	LOAD	RUN
WORKLOAD A	Insert: 0.849273 s [1000 tuples]	Read:0.287802 s [495 tuples] Update:0.443819 s [505 tuples]
WORKLOAD B	Insert:0.141619 s [1000 tuples]	Read:0.094733 s [964 tuples] Update:0.012280 s [36 tuples]
WORKLOAD C	Insert:0.142223 s [1000 tuples]	Read: 0.097763 s [1000 tuples]
WORKLOAD D	Insert:0.14108 s [1000 tuples]	Read:0.105140 s [957 tuples] Insert:0.01487 s [43 tuples]
WORKLOAD E	Insert:0.136702 s [1000 tuples]	Scan:0.626897 s [959 tuples] Insert:0.052098 s [41 tuples]
WORKLOAD F	Insert:0.150345 s [1000 tuples]	Read-Modify-Write:0.130824 s [516 tuples] Read:0.87701 [1000 tuples]

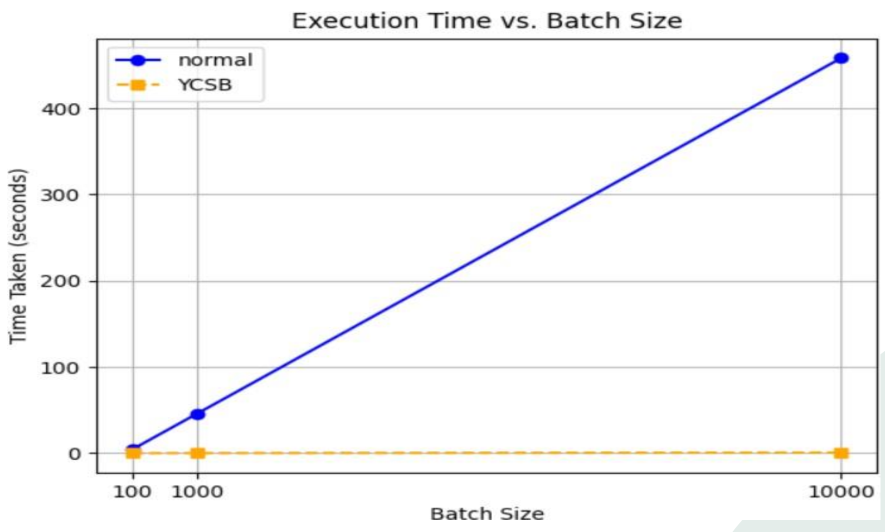
WORKLOAD G:

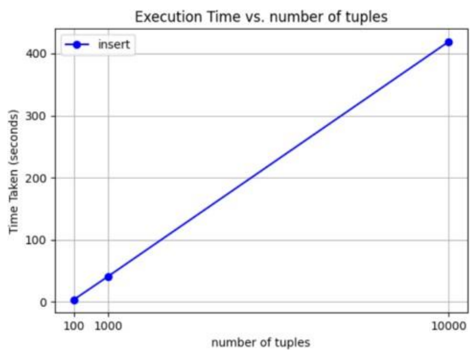
```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload C: Read only
#    Application example: user profile cache, where profiles are constructed elsewhere
(e.g., Hadoop)
#
#  Read/update ratio: 100/0
#  Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
# Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=site.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0
readmodifywriteproportion=1

requestdistribution=zipfian
```

INSERTION:

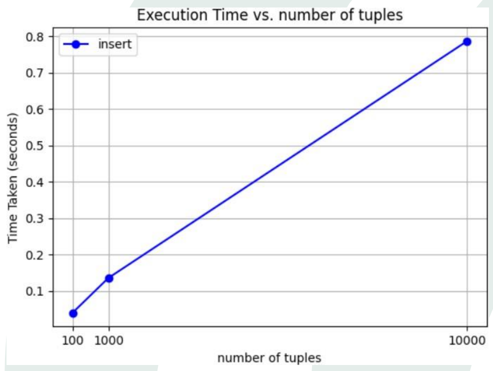


		normal	ycsb	ratio	
	100	4.133	0.0401	103.0673	
	1000	45.4012	0.1365	332.6095	
	10000	458.1432	0.7875	581.7691	



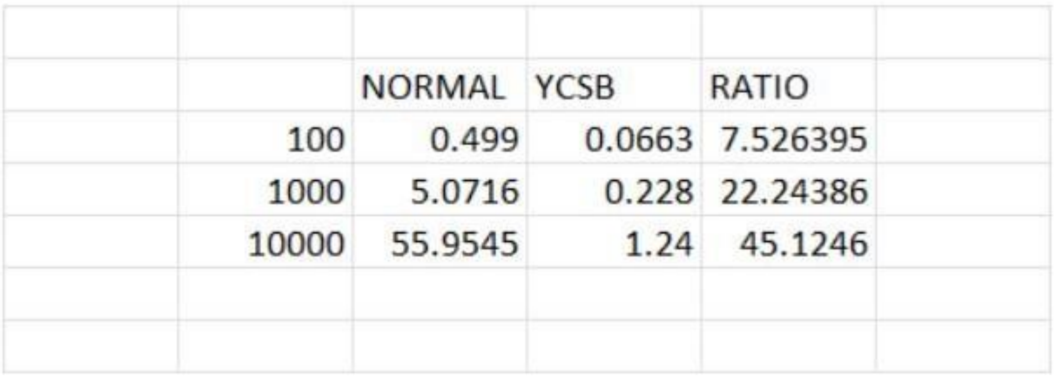


Python script

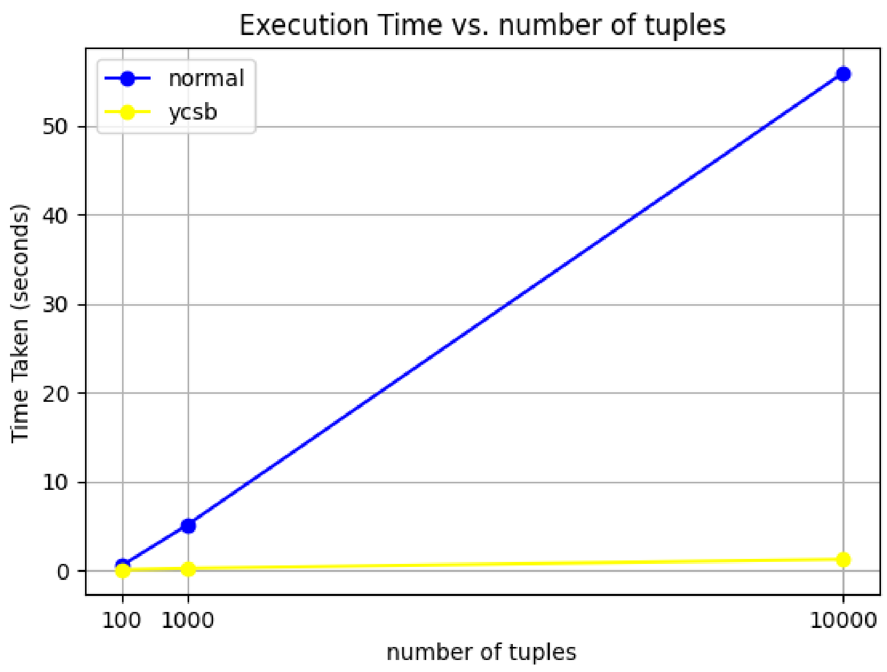


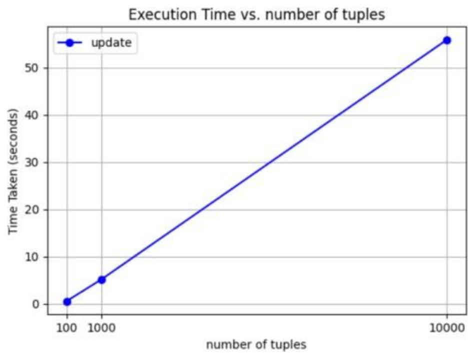
YCSB

UPDATION:

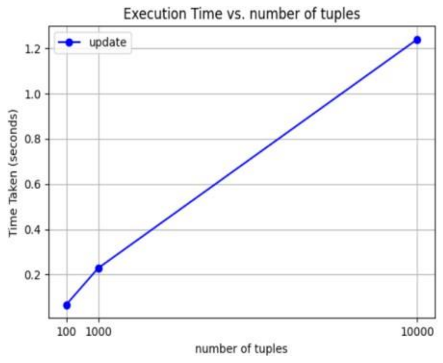


		NORMAL	YCSB	RATIO	
	100	0.499	0.0663	7.526395	
	1000	5.0716	0.228	22.24386	
	10000	55.9545	1.24	45.1246	





Python script



YCSB

CONCLUSION:

In conclusion, ArangoDB presents itself as a compelling option for managing diverse datasets, including network flow data. Its multi-model architecture, combining document, key-value, and graph storage, offers flexibility in modeling complex network structures and relationships. ArangoDB's query language, AQL, provides powerful capabilities for analyzing network flow data, enabling users to perform complex queries and aggregations efficiently. Additionally, ArangoDB's scalability features make it well-suited for handling large volumes of network traffic, allowing for horizontal scaling to accommodate growing data needs. However, while ArangoDB offers numerous benefits for network flow databases, it's essential to carefully evaluate factors such as performance, complexity, and resource requirements to ensure it meets the specific requirements and constraints of the network monitoring and analysis use case. Overall, ArangoDB stands out as a versatile and capable solution for managing network flow data and supporting advanced analytics in network security and monitoring applications.

GITHUB LINK: https://github.com/deekshamacharya/HPE-CTY-PROJECT/tree/OS-LAB