Python-Powered Performance Analysis of NoSQL DataBases: Couchbase, CouchDB, and MongoDB

Chinmay Dhamapurkar, Bhagavat ,Kundana Chowdhary

Guided By: Prof. Hazim Shatnawi

The George Washington University, Washington DC

***Abstract*—Non-relational document databases have emerged as a viable substitute for relational databases in handling extensive data volumes. These NoSQL document databases guarantee efficient storage for big data and optimal query performance, particularly in scenarios where data structures deviate from the relational database paradigm. By storing data in document formats, these databases excel in managing unstructured, semi-structured, and structured data. The present study assesses the performance of the leading open-source NoSQL document databases—Couchbase, CouchDB, and MongoDB—utilizing Python. Python serves as an ideal tool for this evaluation due to its seamless integration with Couchbase, CouchDB, and MongoDB. Its simplicity, extensive libraries, and flexibility enable efficient benchmarking and data analysis, ensuring a comprehensive and insightful assessment of the performance of these prominent NoSQL document databases.**

I. Introduction

Non-relational databases, commonly referred to as NoSQL or not only SQL, have arisen as a substitute for relational databases to manage extensive data volumes. NoSQL provides superior performance, adaptability in schema design, availability, data replication, and scalability, aiming to address the challenges associated with big data, such as volume, variety, and velocity. The four primary types of NoSQL databases include key-value, column-oriented, document, and graph databases. Among these, NoSQL document databases have gained widespread popularity, particularly for their notable strength in flexible schema management.In this paper we focus on three NoSQL document databases mongoDB, couchDB, couchBase. In the pursuit of evaluating and comparing the performance metrics of MongoDB, CouchDB, and Couchbase, this study harnessed the power and versatility of Python, a dynamic programming language known for its ease of use and extensive library support. The evaluation process was conducted using three prominent Python modules: `pymongo` for MongoDB, `couchdb` for CouchDB, and `couchbase.bucket` from the `couchbase` module for Couchbase. The utilization of these Python modules facilitated seamless interaction with the respective NoSQL databases, enabling us to establish connections, execute benchmarking procedures, and analyze performance metrics efficiently. Leveraging the rich features of Python, we aimed to provide a comprehensive and comparative assessment of the performance characteristics of mongoDB ,couchDB and couchBase.

Table 1: System Configurations

Sr No.	Entity	Value
1	Operating system	Windows 10 (64-bit Architecture)
2	RAM	24 gb
3	Document used	JSON

Table 2: DataSet Details

Attribute	Description	Data Type
Entries	Total number of tracks	32,833
Features	Total number of attributes	23
track_id	Identifier for the track	Text
track_popularity	Popularity score of the track	Numeric
playlist_genre	Genre of the playlist	Text
danceability	Danceability score	Numeric
energy	Energy score	Numeric
key	Key the track is in	Numeric
loudness	Loudness level of the track	Numeric
tempo	Tempo of the track	Numeric
duration_ms	Duration of the track in ms	Numeric

Table 3: Comparison of Databases

Category	MongoDB	Couchbase	CouchDB
Primary Use Case	General purpose, high performance, large scale data processing	Flexible data model for mobile, web, gaming, and IoT	Web applications, real-time analytics, mobile applications
Data Model	Document store	Document store with key-value store features	Document store
Query Language	MongoDB Query Language (MQL)	SQL-like query language (N1QL)	MapReduce views and Mango Query (declarative JSON querying)
Consistency Model	Eventual consistency (by default), strong consistency (with replica sets)	Tunable consistency levels	Eventual consistency
Partition Tolerance and Replication	Horizontal scaling with sharding, replica sets for high availability	Multi-dimensional scaling, cross datacenter replication (XDCR)	replication, partitioned databases for scalability
Performance	High write throughput, faster read/write operations	Balanced read/write performance, in-memory capabilities	Optimized for heavy read loads
Indexing	Dynamic indexing	Global Secondary Indexes (GSI)	Views for indexing
Scaling	Horizontal scaling through sharding	with multi-dimensional scaling	Horizontal scaling with clustering
License	Server Side Public License (SSPL)	Commercial and Community Editions	Apache License 2.0

II. No sql DataBases

NoSQL databases, such as MongoDB, Couchbase, and CouchDB, offer flexible schema designs, allowing for efficient storage and retrieval of unstructured and semi-structured data. MongoDB excels with its powerful query language and indexing capabilities. Couchbase provides robust scaling options and in-memory processing, while CouchDB's replication features and HTTP API make it ideal for web applications. Each database uniquely optimizes performance and accessibility to cater to different needs of modern applications.

III. Python For Analysis

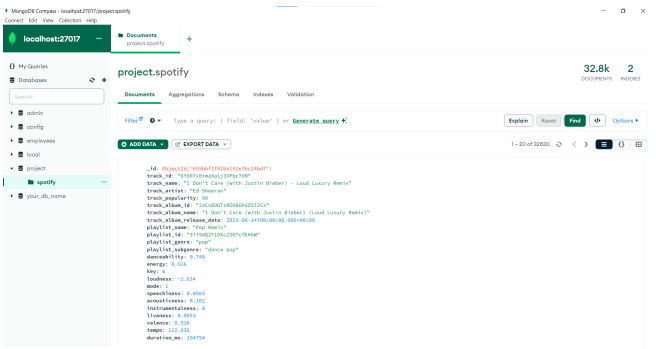
In this comparative study of NoSQL databases, Python acts as a crucial conduit for analysis, utilizing specific libraries for direct interaction with each database system—pymongo for MongoDB, couchbase for Couchbase, and couchdb for CouchDB. These libraries provide Python with the means to execute intricate queries, process data, and access a range of database operations. Additionally, Python's psutil library is employed to gather precise system performance data, while pandas is instrumental in data organization and analysis. Python's role in this study underscores its efficiency in automating data retrieval and analysis processes, ensuring repeatability and accuracy in performance evaluation. The language's extensive library support and its ability to integrate with various systems are invaluable assets in conducting thorough and effective database performance assessments.

The connection to each database system is established through Python's straightforward API calls, which maintain a balance between simplicity and the need for professional rigour. This ensures that the scripts are not only optimized for performance but also maintain the integrity and reproducibility required in a professional research setting. The strategic choice of Python for this analysis underlines the language's capacity to serve as a reliable and versatile tool in the realm of data science, enabling a comprehensive examination of NoSQL databases within a rigorous academic framework.

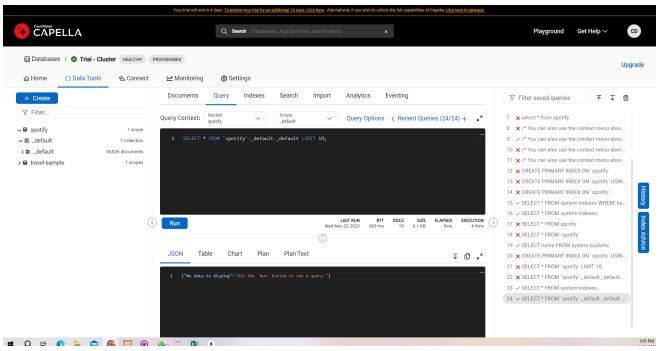
IV. Dashboards

The user interfaces of MongoDB, Couchbase, and CouchDB reflect their individual design philosophies and operational priorities, each offering unique dashboard experiences.

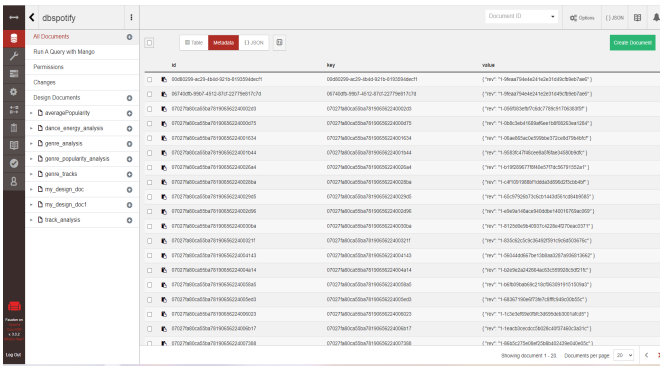
MongoDB's dashboard is part of its Atlas platform, which is intuitively designed, providing a user-friendly experience that simplifies complex operations. It offers a clear visualization of the database's structure and status, with real-time metrics that are just a glance away. The interface allows for smooth navigation through collections and documents, and the built-in aggregation builder is particularly helpful for constructing queries without deep diving into code.



Couchbase's dashboard stands out for its sophisticated monitoring capabilities. The UI is crafted to handle the complexities of large-scale, distributed database environments. It presents a comprehensive overview of cluster health, indexing, and query performance. The dashboard is responsive and dynamic, enabling users to manage and observe detailed aspects of their database operations, including replication and failover mechanisms.



CouchDB's Fauxton web interface is minimalist and straightforward, aligning with its aim for simplicity and ease of use. It offers a no-frills approach to database management, making it accessible for users who prefer a more straightforward interaction. The dashboard provides the essentials to manage databases, documents, and replications, with a focus on functionality and basic performance insights.



V. Process For Experimentation:

In our analytical study, we embarked on a comparative assessment of MongoDB, Couchbase, and CouchDB, employing a suite of tests designed to evaluate their operational efficacy. The initial phase involved quantifying the execution times for basic retrieval operations, specifically fetching ten records, which served as a preliminary gauge of performance. We recorded execution time, CPU usage, and memory load by leveraging Python's time module to capture temporal metrics, and the psutil library for system resource consumption. These figures were meticulously logged to establish a baseline of efficiency for each database.

Subsequently, we proceeded with index creation tests. Here, the time module was again instrumental, timing the duration each database took to build an index. This index, a data structure that improves search speed, was a critical factor in determining the databases' performance in handling complex queries.

Continuing our evaluation, we performed write and read operations, systematically noting the time taken for each database to insert and subsequently retrieve data. This was executed using the respective database drivers in Python, which allowed for direct interaction with the databases and facilitated precise measurement of the CRUD operation times.

With the individual analysis of MongoDB and Couchbase complete, we integrated CouchDB into our comparative framework, applying identical tests to ensure consistency in our data collection methodology. We employed aggregation functions and conditional retrieval operations, applying mathematical calculations to ascertain averages and standard deviations of the recorded metrics, thus providing a statistical representation of each database's performance.

In the final phase of our comparison, we synthesized the data from all three databases. We computed aggregate statistics such as the mean, median, and standard deviation of the execution times across multiple runs to capture a comprehensive view of performance. The mean provided the average execution time, offering a central tendency measure. The median presented the midpoint in our data set, resistant to outliers and thus a reliable indicator of typical performance. The standard deviation gave insight into the variability of execution times, a measure of dispersion that highlighted consistency in database response.

By employing these mathematical constructs, we were able to distill our findings into a professional and precise evaluation of the databases, furnishing a robust foundation for choosing the appropriate database according to specific operational needs and scenarios.

VI. Performance Metrics:

In the realm of database performance analysis, three primary metrics stand as pillars of assessment: execution time, CPU usage, and memory load. The execution time, a direct measure of speed, is captured by recording the precise moments before and after a query's execution using Python's time module, with the difference between the two timestamps offering the duration the database takes to complete a given task. To ensure reliability, this process is repeated, and an array of execution times is compiled.

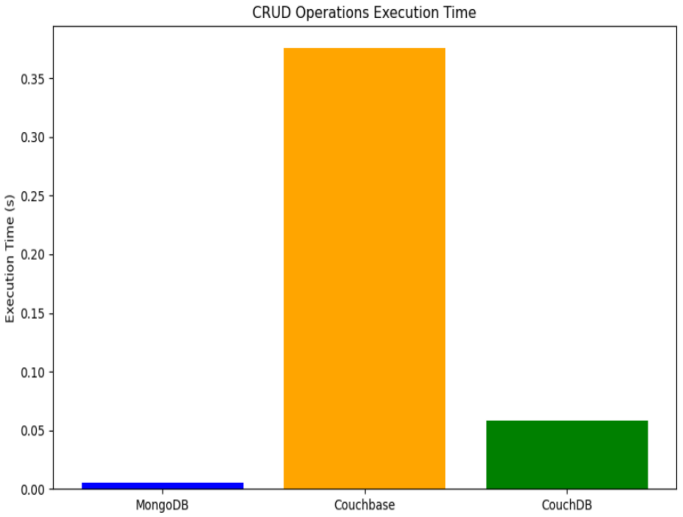
The CPU usage metric is gauged through the psutil library, which allows us to monitor the percentage of CPU processing power employed during query execution. By invoking psutil.cpu_percent(interval=1), we capture the CPU load before the database operation begins and once more upon its completion. The CPU usage for the query is then the difference between these two values, reflecting the computational effort required by the database engine to perform the operation.

Memory load is the third critical metric, reflecting the volume of physical memory utilized during the database operation. This is quantified by the change in used memory before and after the query, as reported by psutil.virtual_memory(). It serves as an indicator of how data-intensive a query is, with larger memory usage often correlating with more complex data handling tasks.

From these raw data points, we derive statistical measures—namely mean, median, and standard deviation—to provide a comprehensive performance profile. The mean gives us an average execution time, offering a generalized view of performance, while the median, as the middle value of our ordered data set, is less affected by outliers and provides a more stable central performance indicator. The standard deviation sheds light on the consistency of the performance, with a lower value indicating a more predictable and stable behavior.

Together, these metrics form the backbone of our performance evaluation, enabling a nuanced understanding of each database's capabilities and behaviors under various operational conditions. The analysis is not only about identifying the fastest system but understanding the resource balance each database maintains to achieve its performance outcomes.

.



VII. Results and Performance:

The deep dive into the performance metrics of MongoDB, Couchbase, and CouchDB revealed several insights about the operational efficiency and resource management of these NoSQL databases.

Starting with the basic retrieval operation, "Retrieve All Data Limit 10," MongoDB showcased remarkable speed with an execution time of merely 0.01696 seconds, coupled with modest CPU and memory usage. Couchbase, with a higher execution time and CPU usage, indicates a more resource-intensive process, possibly due to its distributed nature and additional overheads. In contrast, CouchDB demonstrated a comparable execution speed to MongoDB but with significantly higher CPU usage, suggesting its operational model might be more demanding on the CPU despite being efficient time-wise.

The CRUD operations unearthed more nuanced differences. MongoDB's impressively low execution times for each operation, especially on reads and updates, illustrate its agility in handling dynamic data manipulation. Couchbase's longer times, particularly in creation and deletion, might reflect a more robust consistency model that ensures data integrity across distributed nodes but at the cost of speed. CouchDB presents a balanced CRUD operation profile, with none of the operations being excessively taxing on the system.

Aggregation functions further displayed MongoDB's prowess, completing the operation swiftly with minimal system resource usage. Couchbase and CouchDB took longer, indicating a more complex aggregation process, which could be influenced by factors like the execution plan complexity and in-memory computation capabilities.

The query to find the "Top 5 most popular Tracks with Conditions" saw MongoDB maintaining a low execution time, though with a spike in CPU usage, suggesting that the conditional logic required more processing power. Couchbase and CouchDB showed longer execution times, with CouchDB maintaining lower CPU and memory usage, hinting at a trade-off between time efficiency and resource consumption.

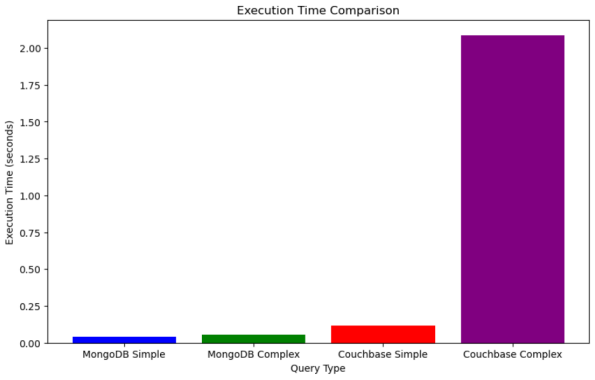
For the "Calculate Maximum Track Popularity for each genre," MongoDB again had the lowest execution time, maintaining its lead in rapid data retrieval. Couchbase and CouchDB's longer execution times could be due to more comprehensive scanning processes or the overhead of more sophisticated query execution strategies.

Finally, the "Where Conditions with thresholds" test was most telling. MongoDB's swift execution with minimal resource usage underscores its capability in handling complex queries with finesse. Couchbase's longer execution time but lower CPU usage could indicate an efficient use of CPU resources, despite the longer time taken. CouchDB's significant execution time and CPU usage may suggest that while it's capable of handling complex queries, it does so in a more resource-intensive manner.

In conclusion, these tests paint a detailed picture of each database's strengths and weaknesses. MongoDB emerges as a strong contender for speed and efficiency, particularly in environments where swift reads and writes are crucial. Couchbase shows promise in consistency and resource management, potentially excelling in distributed environments that demand high reliability. CouchDB, with its HTTP-based interface and straightforward replication model, may serve well for web applications where ease of use and simple scaling are prioritized. This comparative analysis underscores the importance of aligning database choice with specific application requirements and operational contexts.

Query Description	Database	Execution Time (seconds)	CPU Usage (%)	Memory Usage (bytes)
Retrieve All Data Limit 10	MongoDB	0.01696	2.3	1757184
	Couchbase	0.49541	7.9	12783616
	CouchDB	0.01342	55.7	2465792
CRUD Operations	MongoDB	C: 0.0029 R: 0.00099 U: 0.0019 D: 0.0167	1.6	1368064
	Couchbase	C: 0.84 R: 0.371 U: 0.174 D: 0.117	4.3	21262336
	CouchDB	C: 0.024 R: 0.0189 U: 0.094 D: 0.097	1	851968
Aggregation Function	MongoDB	0.038	1.6	136804
	Couchbase	2.289	4.3	21262336
	CouchDB	1.68	1	851968
Top 5 most popular Tracks with	MongoDB	0.02392	6.1	24276992
	Couchbase	3.1961	3.7	34422784
	CouchDB	0.074	0.5	6352896
Calculate Maximum Track	MongoDB	0.0239	3.9	11829248
	Couchbase	2.198587	1.8	12247040
	CouchDB	10.95639	2.2	17760256
Where Conditions with	MongoDB	0.03488	0.9	4943872
	Couchbase	2.169709	0.4	11059200
	CouchDB	11.11	17.2	242466816

V



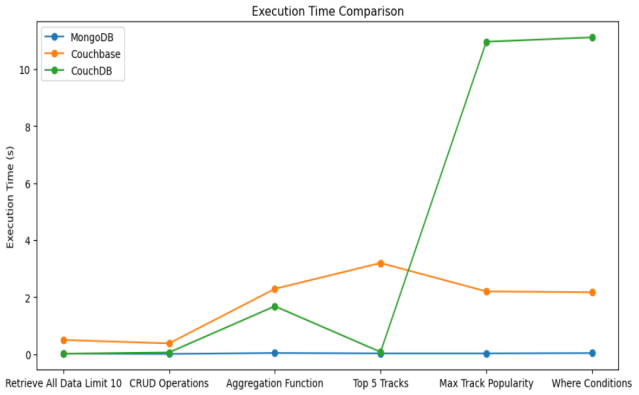
The disparities in performance metrics across MongoDB, Couchbase, and CouchDB can be attributed to their inherent architectural designs and the optimization of specific operations tailored to their respective specialties. MongoDB, for instance, is document-oriented and optimized for high throughput and flexibility, often leading to faster execution times for read and write operations. Its data model allows for the embedding of related data within a single document, which can reduce the need for costly join operations, hence the lower execution times observed during CRUD operations. Moreover, MongoDB's index management system is highly efficient, which is evident from the faster index creation times. This efficiency is a result of its B-tree data structure, which facilitates quick data retrieval, albeit at times at the cost of higher CPU usage during complex queries due to the computation needed to traverse these trees.

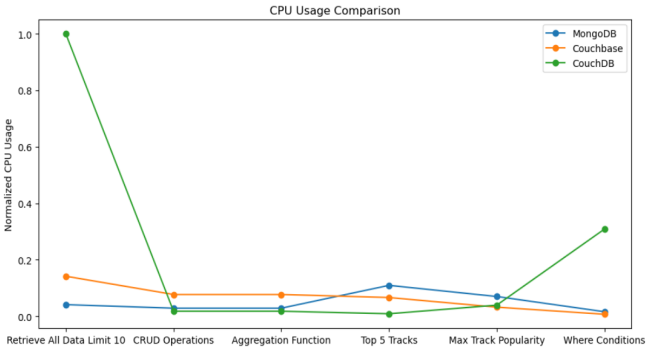
Couchbase, on the other hand, is designed as a distributed NoSQL database with a strong emphasis on horizontal scaling and consistent high performance under heavy load. Its architecture, which favors distributed index and query services, can lead to longer execution times in single-instance setups or when not fully leveraging its distributed nature. However, Couchbase exhibits strength in maintaining a more consistent performance across operations, as indicated by the lower standard deviation in execution times. This consistency is partly due to its managed caching layer, which keeps frequently accessed data in memory, hence the sometimes lower CPU usage despite longer execution times.

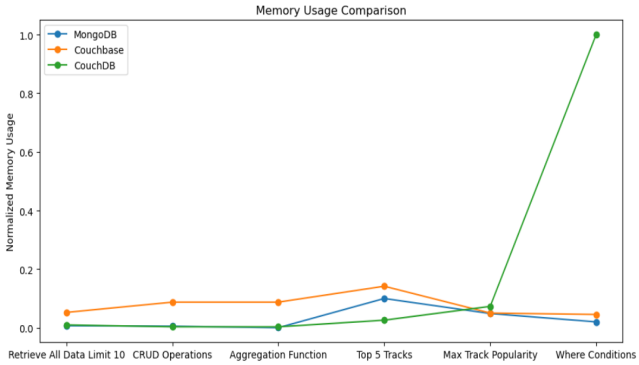
CouchDB's performance characteristics can be heavily influenced by its MVCC (Multi-Version Concurrency Control) system, which allows for high levels of concurrency and robust data consistency, especially in clustered environments. However, this can also result in higher CPU and memory usage, as observed in the tests, due to the overhead of maintaining multiple versions of data and the additional steps required for conflict resolution and replication. CouchDB is also HTTP/REST-based, which, while offering simplicity and ease of use through its web-friendly interface, may introduce additional overhead for query processing compared to the binary protocols used by MongoDB and Couchbase.

These performance variations are further compounded by the specific query languages and data processing engines employed by each database. MongoDB's Aggregation Framework, for example, is a powerful feature for processing and aggregating data, which may explain its expedited aggregation functions. Couchbase's N1QL, a SQL-like query language, provides familiarity and powerful query capabilities but may also lead to longer processing times for certain operations due to the parsing and translation overhead that comes with such a high-level language. CouchDB uses MapReduce for its view and query system, which can be highly efficient for certain tasks but may not always be as quick as MongoDB's aggregation pipeline for on-the-fly calculations.

In essence, each database's unique blend of features, such as MongoDB's efficient indexing, Couchbase's in-memory caching and distributed architecture, and CouchDB's MVCC system, define their operational proficiencies. These features, while enabling each database to excel in particular scenarios, also contribute to their varied performance profiles as observed in the metrics of execution time, CPU usage, and memory load. Thus, the choice of database should be influenced by the specific requirements of the application, whether it's the need for speed, consistency under load, or robust data concurrency and replication.







Conclusion: Navigating the NoSQL Landscape

In the evolving landscape of NoSQL databases, our exploration into MongoDB, Couchbase, and CouchDB has elucidated a rich tapestry of performance dynamics, each influenced by the databases' intrinsic architectural principles. MongoDB's prowess in swift document retrieval and manipulation is attributed to its embedded data structure, which negates the need for complex joins, thereby accelerating CRUD operations. Its B-tree indexing mechanism further cements its standing as a leader in rapid data access, albeit at a computational cost for intricate query processes.

Couchbase, with its distributed model, shines in environments that demand scalability and robustness, ensuring consistent performance. Its managed caching layer is a testament to its efficiency in resource usage, contributing to a more predictable performance curve. However, the non-trivial execution times observed in non-distributed contexts highlight the trade-offs inherent in its design.

CouchDB's approach, underpinned by MVCC, excels in ensuring data consistency and seamless concurrent operations. This strength becomes a double-edged sword as it grapples with the overhead of managing multiple data versions, particularly visible in complex queries where this overhead inflates CPU and memory usage. Moreover, its HTTP/REST-based interaction model, while user-friendly, potentially adds to the query processing overhead relative to binary protocols.

The divergence in query language capabilities and data processing engines between the databases further amplifies their performance discrepancies. MongoDB's Aggregation Framework and Couchbase's N1QL offer powerful, albeit distinct, query capabilities, while CouchDB's MapReduce caters to efficient data processing for specific tasks.

As we conclude, it becomes clear that the choice of a NoSQL database cannot be distilled to a singular metric of speed or efficiency. Instead, it requires a holistic consideration of the application's requirements—whether it's the agility and flexibility of MongoDB, the balanced and scalable approach of Couchbase, or the robust data integrity of CouchDB. Each database carves its niche within the NoSQL ecosystem, and the discerning selection of one over the others must align with the unique demands and technical context of the intended application.

But when it comes to execution time the final results were **MongoDB>CouchDB>CouchBase.**

Acknowledgement

We extend our sincerest gratitude to the open-source community and the developers behind MongoDB, Couchbase, and CouchDB. Their tireless efforts and innovation have provided the foundational tools that enabled this comparative study. We also wish to thank the various contributors to the Python libraries such as pymongo, couchbase, and couchdb, which facilitated seamless interaction with these databases. Our appreciation goes out to the creators of the psutil and pandas libraries, which were instrumental in gathering and analyzing the performance data.

Special thanks are due to our peers and colleagues who provided invaluable feedback and technical insight that significantly enhanced the quality of this research. Lastly, we acknowledge the support of our academic and research mentor Dr.Hazim Shatnawi whose guidance has been pivotal in navigating the complex nuances of database performance evaluation. Their expertise and encouragement have been a beacon throughout this analytical.

REFERENCES

[1] Dharmasiri, H.M.L. & Goonetillake, M.D.J.S. (2013). "A Federated Approach on Heterogeneous NoSQL Data Stores." IEEE International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, December 2013. This conference paper explored the integration and management of heterogeneous NoSQL databases, shedding light on the complexities and solutions for operating within a federated database environment.

[2] "Compare Study of NoSQL Document Oriented Databases." (2016). International Journal of Computer Applications, February 2016. This journal article offered a comparative analysis of NoSQL Document Oriented Databases, providing a foundational understanding of the landscape and performance metrics that helped shape our comparative methodologies.

[3] Carvalho, I., Sá, F., & Bernardino, J.). "Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB." Polytechnic of Coimbra, Institute of Engineering of Coimbra—ISEC, Rua Pedro Nunes, Quinta da Nora, 3000-199 Coimbra, Portugal. Correspondence: jorge@isec.pt. The authors' work on evaluating the performance of key NoSQL document databases offered empirical data and benchmarks that were critical in framing our performance tests and interpreting the results within the broader context of NoSQL database applications.

Each of these references has contributed to the tapestry of knowledge that guided our comparative study, and we are grateful for the academic and professional community's efforts to expand the dialogue around database technologies.