Biobank registry implementation: An evaluation of NoSQL databases for Philippine Cancer-Phenome Biobanking System

Philip John C. Sales[a,*], Michael C. Velarde[b,1], Ariel S. Betan[a,c], Alvin B. Marcelo[a]

[a]*Medical Informatics Unit, College of Medicine, University of the Philippines, Ermita, Metro Manila*
[b]*Regenerative Biology Research Laboratory, Institute of Biology, University of the Philippines, Quezon City, Diliman*
[c]*College of Arts and Sciences, Department of Social Sciences, University of the Philippines, Ermita, Metro Manila*

Abstract

Background and objective:. The field of healthcare is rapidly accumulating data of complex types and formats. The current methods of storing data relies heavily on traditional relational database management system (RDBMS) which have notable limitations in managing clinical and biomedical data. Thus, a constant demand for alternative approach is in place. Among the available solutions, NoSQL databases had been cited as a viable option due to its numerous advantages in handling healthcare data challenges.

Methods:. This study evaluated different types of NoSQL databases using an database evaluation framework tailored for healthcare context.

Results:. The application-specific evaluation criteria showed that document-based type is the best choice for extensibility, flexibility, and query readability whereas key-value pair is the most efficient in performance and scalability. Moreover, columnar-wide has the edge in storage capacity.

Conclusion:. Among the shortlisted databases, MongoDB - a document type NoSQL was found to be the recommended choice for the current implementation and immediate need of the biobank information system.

Keywords: Clinical data, relational database, NoSQL, biobanking, database evaluation framework, health information system

1. Introduction

The increasing size and complexity of data being generated in healthcare industry is reaching to the point of being unmanageable [1, 2]. This was brought about by the technological advances in medical and biomedical fields which have highlighted the importance of collecting and storing data to benefit clinical research and medical services.

Currently, storage of clinical data largely relies on relational database management systems [3, 4] and has been the most common approach of data storage since the 90's [5]. From the time of its origins in the 1970's, the relational model had been adapted widely in many database management systems [6, 7] and had remained unchanged for decades [2]. This modeling approach had become the de facto standard for most industries, but remains ineffective in the field of healthcare due to the characteristics of clinical and biomedical data.

Healthcare data are dynamic, hierarchal, sporadic, and heterogeneous in nature. It is stored in various types including free text, coded data elements, images, audios, signals or sensor data, logs, or notes that is formatted in either structured, semi-structured, or unstructured presentation [2, 4, 8, 9, 10]. The heterogeneity of formats, high volume exchange, and dynamic structure makes it difficult to model and manage health data using the traditional relational database [4, 11, 12, 13].

In addition, one of the major drawback of relational database is the need for pre-designed schema. Predefining the data structure is impractical since pre-determing the combination and variety of data fields to be collected in a medical record is unrealistic [4].

Succintly, RDBMS is not flexible, scalable, and extensible to overcome the problem of constantly changing requirements, various formats, increasing volume, and the continuous need for updates in healthcare data [3, 4, 11, 12, 13].

Against these backdrops, there is a constant demand

[*]Corresponding author at Medical Informatics Unit, UP College of Medicine, Tel. +63(2) 536 1396

Email addresses: `pcsales@up.edu.ph` (Philip John C. Sales), `mcvelarde@up.edu.ph` (Michael C. Velarde), `asbetan@up.edu.ph` (Ariel S. Betan), `admarcelo@up.edu.ph` (Alvin B. Marcelo)

[1]https://regenlab.weebly.com

and practical need for alternatives outside the conventional relational data modeling approach. Consequently, there are increasing researches and industry projects on databases that are driven by the practical experience of the conflicting static nature of relational database and dynamic characteristics of healthcare data [2].

Out of the wide spectrum of viable alternatives, this study identified NoSQL database as supported by literature to be suitable in addressing the challenges of healthcare data. NoSQL approach had attracted the attention of non-clinical industries and researchers due to its flexibility, scalability, velocity, and availability which addresses the limitations of relational databases [2, 8, 14, 15]. While there are significant advantages of using NoSQL database, there is limited research which has evaluated the use of NoSQL databases in the healthcare domain.

Additionally, it is not always straightforward to declare which database type is better than the other. Concerns are raised in selecting which among the list of 225 NoSQL databases [16] can best address the healthcare data challenge. To date, there is a paucity of literature that provides guiding principle, standard protocol, or framework on how to evaluate of database for healthcare industry.

In light of the foregoing, this study attempted to evaluate three NoSQL database types (i.e. document based, columnar wide, key-value pair) in a structured technology evaluation process customized for medical settings. Specifically, it extended an approach and develop a guidelines that focuses on healthcare domain.

The sections below are organized as follows. First, section 2 offers an overview on research project context and the evaluation approach taken. Section 3 describes the methodology performed for the testing and prototyping of each NoSQL database type. The final results of the application-specific evaluation criteria are discussed in section 4. Finally, a conclusion is given in section 5.

2. Background

2.1. Project Context

Establishing that relational database have notable limitations in managing the nature of healthcare data [3, 4, 11, 12, 13] is one thing. Proposing an alternative solution that undergoes a systematic way of evaluation is another. The latter is the undertaking that this paper aimed to implement.

In relation to Philippine scenario, one of the initiative of Philippine-California Advanced Research Institutes (PCARI) Project is to implement of Philippine Cancer Phenome-Biobanking System. The development of this pilot project will serve as a model health information system (HIS) with optimism of going national scale. The data challenges mentioned are applicable to the context of the biobank information system. Critical to the success of the system is the appropriate evaluation and selection of data modelling approach and database technology.

Technology selection or evaluation framework are constructs predominantly used in the field of business sectors with focus only upon management and financial decisions [17], but dearth of source exists when discussing technology or database evaluation for healthcare sector. Moreover, most selection practices and evaluation models in digital health technologies are informed by traditional means: word of mouth, internet search, consultant's advice, expert's opinion [18], emerging systems, and technology trend. There are limited models, empirical evidences, and guidelines that facilitate methodological evaluation of database for HIS.

Thus, this paper adapted a database evaluation framework that will evaluate NoSQL databases against context-based parameters.

2.2. Evaluation Approach

The principles of this study's evaluation framework was inspired from themes of Digital Selection Framework [18] and the context-driven component evaluation process of Technology Selection Framework for Health Interventions [17]. The aim is to provide reproducible and reusable process in evaluating database that can address healthcare data requirements. The steps are illustrated in Figure 1 and the case application are outlined below.

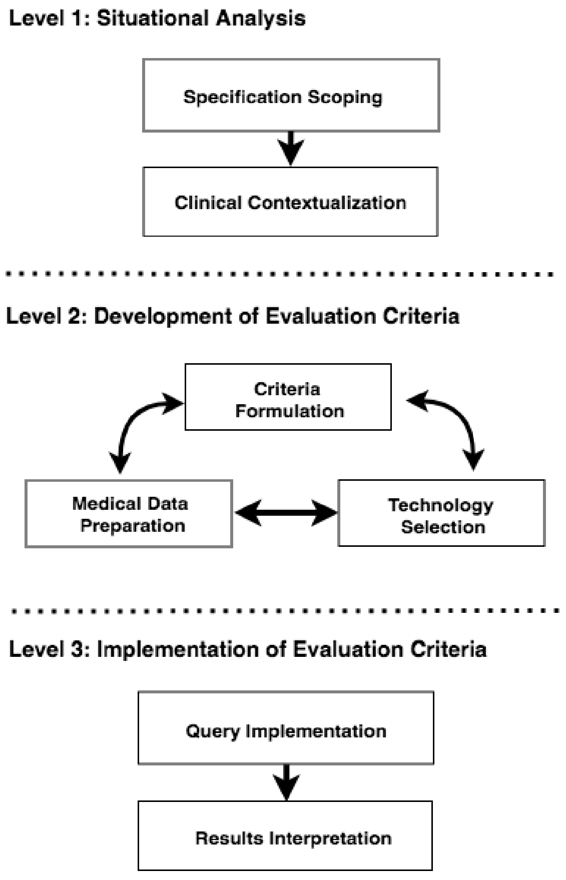

Level 1: Situational Analysis
Specification Scoping
Clinical Contextualization
Level 2: Development of Evaluation Criteria
Criteria Formulation
Medical Data Preparation
Technology Selection
Level 3: Implementation of Evaluation Criteria
Query Implementation
Results Interpretation

Figure 1: Database Evaluation for Healthcare

1. *Specification Scoping.* Identifying the type of HIS influences the evaluation given that the database operations vary relative to the nature of the application. The HIS for this project is a biological specimen inventory and clinical registry type system. This type of system is not transactional intensive but rather storage and aggregate focused. With this consideration, the study anticipated the need for a database that can manage voluminous and aggregatable data.
2. *Clinical Contextualization.* Defining the requirements within the context of a clinical domain assures that the functionalities are clinically relevant and not technology centric solutions. Multiple sessions of focus group discussions (FGD) and key informant interviews with the technical and allied health professionals were done as method of elicitation for this step.
3. *Criteria Formulation.* The study's empirical findigs is as equally important as the metrics itself. The foundation of results is pinned around the criteria. This research consolidated and prioritized the functional and non-functional requirements collated from the qualitative approach to form the basis of the evaluation criteria. The metrics measured in this study are context-based. Nevertheless, the methods applied are generalizable to other database evaluation.
4. *Technology Selection.* On the grounds that the nature of clinical data is highly evolving and dynamic, one of the immediate selection filters is the capability to adapt to schema changes. In consequence, schema-less modelling approach was the logical option; hence, focus on NoSQL was the scope of the implementation. This study evaluated only the highly represented and most popular database for each type of the NoSQL category [19]. The selected NoSQL databases are MongoDB for the document category, Cassandra for the columnar, and Redis for the key-value.
5. *Medical Data Preparation.* The research utilized 1 million unique patient records that is structured in Health Level 7 - Fast Health Interoperable Resource, release Standard Trial Use 3 (HL7 - FHIR, STU3) format. Using this FHIR standard, schema mapping and modelling of the clinical dataset into the physical data storage of each NoSQL database type is mandatory. In this step, dataset were transformed to each NoSQL physical data model requirements prior to the actual loading.
6. *Query Implementation.* The query scripts were fundamentally derived from the evaluation criteria. Each criteria, if applicable, were converted into multple database query statements or scripts. Each script was executed multiple times against different workloads on several dataset sizes in different testing configurations to allow rigorous comparison amongst each selected database.
7. *Results Interpretation.* Summary of results were displayed tabular or graphical presentation showing the comparison of each NoSQL types against the identified evaluation criteria.

3. Method

3.1. Evaluation Criteria

The data gathered from stakeholders' discussions and interviews were translated into quality attribute scenarios (QAS), functionality lists, use cases, business requirements, and software specifications. Following the guidelines of QAS workshop set by Barbacci (2003) [20] and further prioritization of functionality lists resulted into the final evaluation criteria: performance, scalability, storage, readability, flexibility, and extensibility

3.2. Clinical Datasets in HL7-FHIR

Notably, more than 95% of all data is unstructured or semi-structured [21] and thus requires additional preprocessing. To avoid this overhead, this study used clinical dataset in HL7-FHIR STU3 standard. Using structured form enhances machine processability and searchability [22]. Also, adhering to standard information modeling ensures interoperability, compatibility [4], and conceptual normalization.

FHIR specification enforces the use of modular components called "resources". The principle of modularity enables the resource to be logically exchanged and consumed by external systems either as a standalone or bundled resource.

The FHIR clinical datasets were synthetically produced using an open-source generator called Synthea. This award winning [23] synthetic patient record generator was started by the MITRE Corporation as part the Standard Health Record Collaborative program [24]. The goal of the tool is to create synthetic data that is both high-quality and realistically capable of yielding patient health records using demographic and statistical disease prevalence model. Synthea is used as the primary clinical dataset generation toolkit [25] and is capable of producing both FHIR and Clinical Document Architecture (CDA) formats [24]

Out of the 12 available resources in Synthea, only 3 resources namely: patient, condition, and diagnostic report were filtered, mapped, and stored separately in the physical data model of each selected database. The other resources that were bundled with the raw data were parsed and excluded.

These resources are sufficient to test the evaluation criteria that directly involves basic functions such as insert, upsert, scan, read, update, and delete. The database operations of clinical registry and specimen repository mainly revolve around storage and retrieval of patient's clinical personal identifiable information, diagnosis, conditions, diagnostic reports, and procedures. Hence, these resources are the bare minimum to meet the requirements of biobank HIS. The selected resources data composition are shown in Table 1 below.

Table 1: Clinical Dataset Composition

Resource Type	Number of unique records	Average size per record
Patient	1,000,000	3,980 bytes
Condition	5,430,000	5,000 bytes
Diagnostic Report	5,630,000	675 bytes

3.3. NoSQL data modelling

Considering that each NoSQL type has different approach on how data is supposedly stored, a series of modelling, mapping, and text manipulation processes were taken to satisfy the physical data model prerequisites of each database. For instance, since Redis follows a key-value convention, the dataset must be transformed to represent a key-value notation. The section below describes how the datasets were mapped according to each database technical requirement.

1. *MongoDB* stores data into collections of JSON documents. A collection is the technical equivalent of a RDBMS table, it exists within a single database and may contain single or multiple documents. A document, on the other hand is the row equivalent in relational database parlance. It is the most basic unit of data in MongoDB. It can contain flattened JSON objects with simple name-value pair, or a deeply nested nature containing multiple embedded or referenced documents. The document-oriented approach allows for the aggregation and nesting of related data via arrays and subdocuments within a single document, easily accommodating the denormalization necessary to eliminate the joins characteristic of relational database queries (Klein, et.al, 2015). The MongoDB schema in this study consists of a single database named "fhir" and multiple collections called "patient", "procedure", "diagnosticreport", and "condition". The 1 million patient records were parsed according to the FHIR resource it contained, and were stored into its corresponding collections. Figure 6.1 below shows the mapping of the raw FHIR Patient resource into the MongoDB document-oriented approach. To establish referential relationship of the FHIR.patient with FHIR.condition, the raw data of FHIR Condition resources were added with the "patientId" field in the root element alongside with other attributes (e.g. Condition, id, clinicalStatus). Figure 6.2 shows how the data transformation was done to accommodate the MongoDB referential requirements. The same data manipulation from raw FHIR JSON to MongoDB JSON document object was done to procedure and diagnostic report resources.

2. *Cassandra* stores data into tables similar to relational database. The table which commonly referred to as column families (CF) are consist of bundled or grouped columns. But unlike relational, these CF can contain primitive and complex name-value data types. Complex types like the map collection type (with the other two types - as the set type and the list type) makes Cassandra "dynamic" as it can handle unrestricted and unbounded number of columns. Each element of a map collection type is stored internally as one Cassandra column. In the relational perspective, the map collection type is comparable to a different table only and that it can be stored within a single column. Instead of having foreign key references to the separate table, Cassandra's approach is to directly contain the supposedly separate table into just one of its columns. Modeling the one-to-many relationship between patient and condition, diagnostic report, and procedure in Cassandra is different from MongoDB. Since Cassandra does not store nested JSON, additional work for data transformation is necessary. For example, if one patient lives in many addresses, the address column is transformed as a map collection data type. The use of a collection type accommodates the fact that a patient can have more than one address (e.g. primary address, mailing address. There are many approaches to the modeling details of Cassandra, design considerations on the partitioning, clustering, denormalizing, and distributing methods of data models are heavily motivated by different purposes and objectives. However, in this study the modelling goal is to maintain the granularity and independence of each FHIR resources at the same time to establish referential relationship with each other. The study intends to achieve this by storing each FHIR resources into separate tables but also allot an extra column as reference (i.e. "patientId") to each other, particularly the patient table.

3. *Redis*, everything in Redis is stored in a denormalized way. This approach provides a minimalist key-value store with neither the nested document structure of MongoDB nor the columns-within-columns structure of Cassandra. Data stores are organized in buckets within a namespace. Each namespace contains unique rows of key-value pairs where each value is attached to a unique key and can be any data type. The one-to-many relationship between patient and condition, and diagnostic report proved to be a challenged in Redis. The basic key-value pair structure is not sufficient to support the type of queries required by the prototyping. Extracting the search match of data in value column that is stored as JSON is not possible. To resolve this deficiency, the study opted to make use of Redis' built-in commands (i.e. HMSET). These commands provide a wrapper suitable for fast and efficient manipulation of key-value. Like any FHIR resources, the values are stored in keys that can be accessed with a specialized subset

of commands. With this command, similar behavior of the columnar data store like Cassandra was replicated for Redis

3.4. Testing and Prototyping

All testing was conducted using Linode servers as the third-party provider with the following configurations (see Table 2). The study used a cloud based computing environment in a centralized setup. Undoubtedly, a single node instsance seem unacceptable for industry level and production grade; however, using it can serve as baseline information for comparison against multiple node configuration. Moreover, it provide insights on the capabilities of each identified database without the extraneous variables like network latency, infrastructure technology, architectural design, and internet connectivity among others.

Table 2: Parameters for Testing Environment

Specification	Capacity	Unit
Radom Access Memory (RAM)	90	GB
Central Processing Unit (CPU)	4	Cores
Solid State Drive (SSD) Stores	90	GB
Transfer	7	TB
Network In	40	Gbps
Network Out	7000	Mbps

In addition, a measurement and evaluation tool called Yahoo! Cloud Service Benchmarking (YCSB) framework was used to manage the test execution. YCSB is an open-source benchmarking utility that served as the middleware software responsible for abstracting the technical complexity of environmental installation and configuration of the selected database technologies, datasets, and workloads [26]. It is the leading choice and industry-standard for performance benchmarking of cloud based databases [27].

YCSB has two main components: First, it contains various interfaces, integration drivers, connectors, and processes for a number of databases. As of writing, YCSB has over forty-six (46) database connectors ready for for interfacing, also it allows extension for other databases not yet in the list. Second, YCSB has core workloads and reporting framework which are libraries that manages the execution of the different types of database operation and reporting output format, respectively. The workloads refers to the standard "CRUD" database operations: Create, Read, Update, and Delete [26].

- read() — read a single record from the database, and return either a specified set of fields or all fields.
- insert() — insert a single record into the database.
- update() — update a single record in the database, adding or replacing the specified fields.
- delete() — delete a single record in the database.
- scan() — execute a range scan, reading a specified number

Other dependencies that were used in the experimentation like the database patient mass generator, NoSQL databases, and operating system are shown in Table 3.

Table 3: Technology Specifications

Dependencies	Name	Version
Operating System	Linux Ubuntu	16.04
Patient generator	Synthea	2.0.0
Database testing client	YCSB	0.14.0
NoSQL Columnar wide	Cassandra	3.11.3
NoSQL Document based	MongoDB	3.6.3
NoSQL Key-value type	Redis	4.0.11

4. Results and Discussion

In this section, the results of the three NoSQL database are discussed from the perspectives of performance, scalability, storage, query readability, flexibility, and extensibility

4.1. Storage

In table 4 Cassandra showed to be most effective in storing the raw file, which driven by its merging functionality commonly referred as compaction. The main idea of thie operational process is reusing space to improve read performance. On the opposite end, Redis consumed the most disk space. This is attributed to study's configuration of using Redis append only files (AOF) over Redis database (RDB). The AOF setup was preferred against RDB to minimize the chance of data loss - the fact that data being process in this research is clinical based. As stated, AOF files are usually bigger than the equivalent raw and RDB files for the same dataset [28]; in effect it occupied more disk space among the three NoSQL databases.

Table 4: Storage Sizes for 1 Million Patient Record

Type	Total storage size	Average size per record	Storage ratio
Raw JSON	3.98 x10^{-2} GB	3.98 KB	
Cassandra	1.54 GB	1.54 KB	+ 61.30%
MongoDB	3.45 GB	3.45 KB	+ 25.12%
Redis	10.5 GB	10.5 KB	- 38.37%

However, these remarkable storage rate differences are by no means statistically significant when subjected rank-based nonparametric test.

4.2. Scalability

Scalability is interpreted as the capacity of to handle increasing load by having simultaneous and parallel task. The measurement focus of scalability in this study pertains to the maximum number of database operations in a single second (i.e. throughput) under increasing threaded processes or parallel loads. Real-life application of threaded transactions can be seen during scenarios of multiple users using the system simultaneously.

It is noticeable in the other literatures [29, 3, 15, 4, 30] query statements were only focused on search operations or retrieval function. In contrast, this paper conducted scalability and performance evaluation using the combination of basic database functions termed as workloads (shown in Table 5).

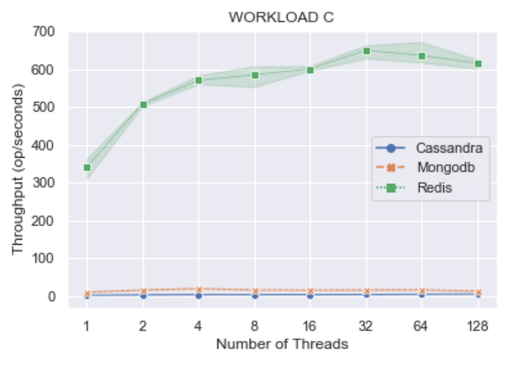
Table 5: Scalability and Performance Workload Configurations

Workload	Distribution	Name
Workload A	Read: 50% Insert: 50%	Read heavy
Workload B	Read: 95% Update: 5%	Read mostly
Workload C	Read: 100%	Read only
Workload D	Read: 95% Insert: 5%	Read latest
Workload E	Scan: 100%	Short Ranges
Workload F	Read-Insert-Update: 100%	Read-Insert-Update
Workload I	Insert: 100%	Insert only

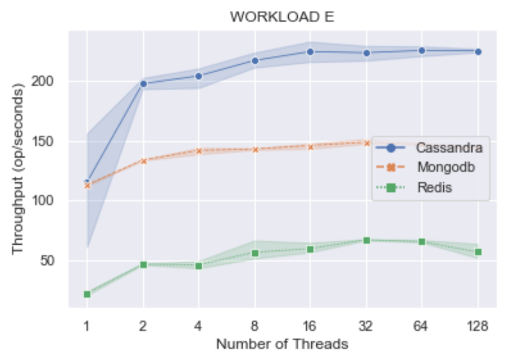
The scalability test was deployed in a single node using the configuration of 128 threads using the 1,000 records - an optmistic projection on biobanking system usage. Only workload C,E,F results was shown in Table 2 as Redis still outperformed others in all database operations (i.e. Workload A to I) except workload E.

Notable findings are: (1) Cassandra is the most effective on scan operations while Redis is the least, (2) MongoDB decreased in throughput as parallel task were increased, and (3) Redis dominated in most operations (Workload A, B, C, D, F) for 1,000 records. First, the scan efficiency of Cassandra is correlated to its storage efficiency [31]. On the other hand, the inefficiency on scans of Redis is due to its data store architecture. Since Redis is limited with only the two fields - namely the key-value column, storage of new data is appended at the last row making the hash table grow vertically as opposed having a balance to horizontal (i.e. columnar) increased. Thus a simple search for contiguous data meant traversing the whole array of key-value items it currently stores. However, this seemingly inefficient scan operation of Redis can

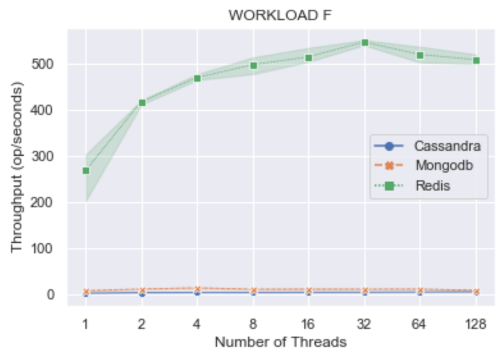
Figure 2: Scalability of NoSQL Databases for 128 threads using 1,000 records in single node



(a) Read only



(b) Short Ranges



(c) Read-Insert-Update

be augmented by sharding and clustering architecture or configuration. To cite tangible and real world analogy, it is always faster to search and scan adjacent paper documents in a properly organized and alphabetically arranged file cabinets. For the second findings on MongoDB, the deceased in throughput when the threads were increased can be attributed to its built-in locking mechanism [32]. This sort of data integrity feature is the tradeoff effect for the eventual consistency that currently exist in MongoDB. Finally, Redis was found to be the most efficient on all workloads under 1,000 to 10,000 records. This is due to the in-memory technology advantage of Redis. Data saved using in-memory are always retrieve faster than those stored in persistent storage because it uses the random access memory (i.e. RAM) as compared to utilizing in disk or hard drive.

With this configuration, it was found that there are significant differences in the scalability among the selected databases for all types of workload. In addition, results revealed that Redis is significantly the most scalable for most of the workloads (except Workload F). On the contrary, the post analysis showed that Cassandra was the least scalable. This result is supported in the performance results for 1,000 records as shown in 3 and 4; concisely, this experiment showed that Redis have the most efficient scalability.

4.3. Performance

The performance of the benchmark focuses on the latency of requests when the database is under dataset sizes. Table 3 and 4 shows the comparison of performances for the all databases when factored in against the different workloads and dataset sizes.

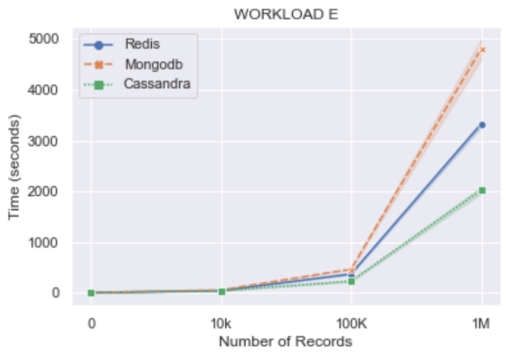
Redis is the fastest when dealing with the minimal set of data (i.e. 1,000 records). As the number of records increased, it is observed to be outperformed by MongoDB and Cassandra on the non-READ related operations (i.e. Workload E, F, and I). Moreover, it is constantly the most efficient on all READ related operations (i.e. Workload A to D) for all different dataset sizes. MongoDB fared best on both INSERT related operations (i.e. Read-Insert-Update and Bulk Load) while Cassandra topped the Short Range workload.

This experiment showed congruent results stating that Redis performed the best for performance on all READ intensive database operations (Workload A, B, C, D) like extracting and searching data. Though different datasets and configuration may have been used in this study, Redis being the most efficient on READ related operations were also supported by the following benchmarking papers [33] [34].

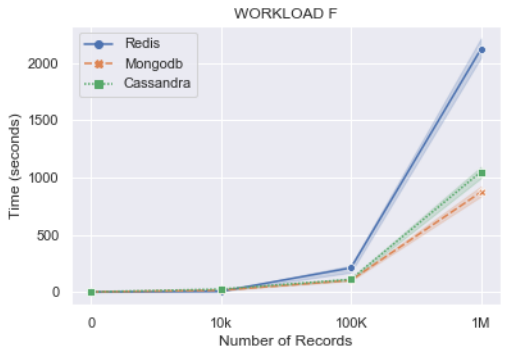
However, the difference was seen in MongoDB as it showed to be more effective on CRUD based operations (Workload F) and Bulk Loading or INSERT intensive applications (Workload I) than Redis. For Cassandra, it found to be well suitable for use cases that involves contiguous scanning (Workload E). This scan speed is attributed to the storage efficiency [31] of Cassandra which is evident in Table 2 3(b). This type of database operation is most applicable for data warehousing and analytical processing are commonly the database operations for data warehousing and analytical processing.

To infer if these latency differences were meaningful, it was computed and found that statistical differences exist for all workload when subjected to different dataset sample sizes.

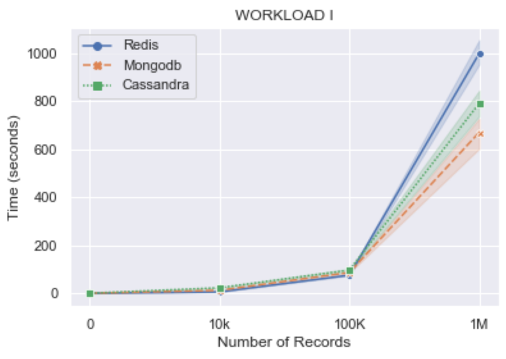
Figure 3: Performance of NoSQL Databases for Non-Read related Operations



(a) Short Ranges



(b) Read-Insert-Update

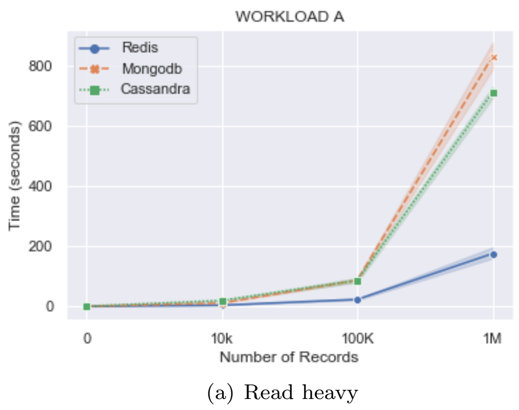


(c) Bulk Insert

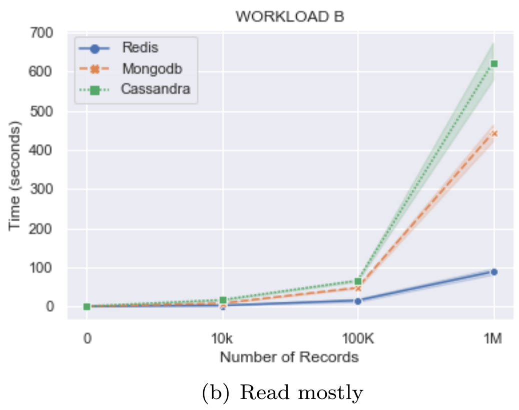
4.4. Flexibility

Flexibility can be interpreted from many perspectives, among the common are technology architecture, deployment process, and schema capabilities. In this study, discussion of flexibility is taken from the latter perspective. By definition, it is defined as the capability to store dynamically changing data without having a predefined schema [12, 3]. However, flexibility is not totally equivalent to having no schema, but rather on the capacity to accommodate changes without substantial overhead requirements. Cassandra involves the need to write an additional parser in order to import and fit the raw nested JSON data into the

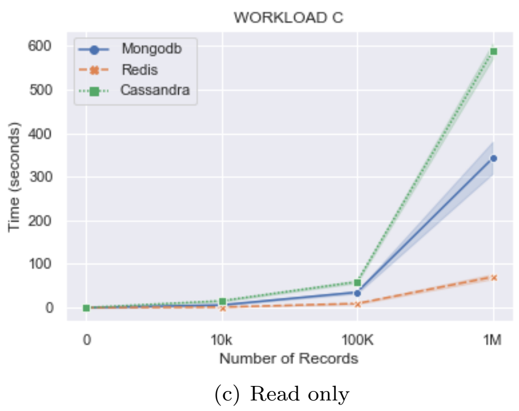
Figure 4: Performance of NoSQL Databases for Read related Operations



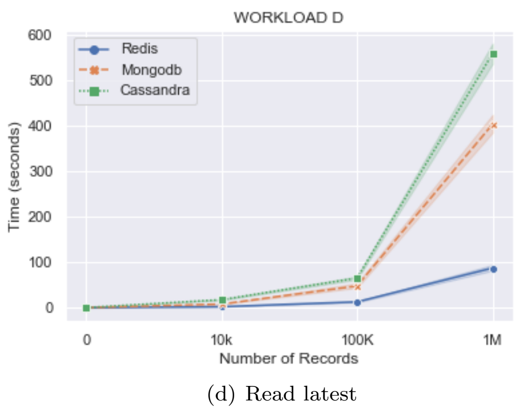
(a) Read heavy



(b) Read mostly



(c) Read only



(d) Read latest

columnar-wide type format that it currently supports.

For versions above 0.7, it was required to have columns and its data types be defined upfront [33]. From the initial approach of "schemaless", it had transitioned into a "schema-optional". Which meant, that it is heading back to where relational databases started: having predefined columns and data types. Thus, the original roadmap of being a schemaless approach was abandoned. This requirement is not necessary for native document-based databases like MongoDB among others. The raw data are simply imported using the original format of the dataset files; hence, full flexibility in data structure is experienced. In between is Redis, which requires minimal data transformation but is still considered schemaless. The data transformation is needed only for the unique identifier of each item in a data in order for it to be segregated and stored separately as the key column, the rest of the data is kept and bundled in the value column. To summary, Cassandra is the the least flexible as it requires upfront pre-design of schema while MongoDB is most flexible as it can store any raw data in a flat or deeply nested JSON format.

4.5. Extensibility

Extensibility is capability to accommodate additional revisions on attributes, columns, and data types without the overhauling need to reconstruct or create new database [35]. Though, a lot of workarounds are available to model for extensibility and cover the inefficient schema design, at a certain point if the core infrastructure is inadequate, there is only so much patching that can be done before reaching the limits of the platform [36]. Thus, a native, built-in, and inherent infrastructure is preferred to a "patch work" type of solution and temporary workaround.

When comparing extensibility within NoSQL groups, no notable difference is observed since all the data structure can be extended either vertically (i.e. columnar-wide type) or horizontally (i.e. for both document-based and key-value types). Addition of new fields, columns and nested values are allowed without the need of much overhaul to existing schema for each approaches. However, it is noted that there is constraint of extensibility particularly to the Cassandra, as it requires additional "alter table" script in order for the new changes to be committed.

4.6. Query Readability

Four (4) scenarios from QAS were selected to represent different database operations: insert, update, search-aggregate, and multiple transactions. Each operations were converted to valid but abridged query syntaxes (see Table 6). Not all variables and fields were included in order

Table 6: Query Readability of NoSQL Databases

Statement	MongoDB	Cassandra	Redis
Insert	`db.cases.insert({ "case_number": "01", "diagnosis": "breast, nos neoplasm", });`	`INSERT INTO cases (case_number, diagnosis) VALUES ("01", "breast, nos neoplasm ");`	`HSET cases:01 "case_number" "01" "diagnosis" "breast, nos neoplasm "`
Update	`db.specimens.update( { "case_number": "01" }, { "specforms.specimen.qty": 11 { upsert: true } );`	`UPDATE specimens SET'specforms.specimen.qty'=11 WHERE case_number="01"`	`HSET specimens:01 "specforms.specimen.qty" "11"`
Multiple Insert	`db.cases.insert([ { "case_number": "02", "diagnosis" : "ovary carcinoma, nos" }, { "case_number": "03", "diagnosis" : "colon, nos neoplasm" } ]);`	`BEGIN BATCH` `INSERT INTO cases (case_number, diagnosis) VALUES ( "02", "ovary carcinoma, nos");` `INSERT INTO cases (case_number, diagnosis) VALUES ( "03", "colon, nos neoplasm");` `APPLY BATCH;`	`HMSET cases:02 "case_number" "02" "diagnosis" "ovary carcinoma, nos"` `HSET conditions:03 "case_number" "03" "diagnosis" "colon, nos neoplasm"`
Search and Aggregate	`db.conditions.find({ "diagnosis": "breast, nos neoplasm" }).count();`	`SELECT COUNT(*) from conditions WHERE diagnosis="breast, nos neoplasm";`	**`Javac>>>`** `import redis.clients.jedis.*;` `private JedisCommands jedisCmd; private JedisPool jedisPool; Map<String,String>row = jedisPool.getResource();` `String diagnosis = row.get(“diagnosis”) if (diagnosis.equals("breast, nos neoplasm"){ jedisCmd.sadd(‘diagnosis’, diagnosis) }` **`redis-cli>`** `SSCAN diagnosis 0 match breast*`

to avoid technical intimidation among the study participants. The aim of the query readability evaluation was to assessed the intuitiveness and ease of understanding of each database query structure and semantics. Though the results rely on the participants' biases or lack thereof, it cannot be dismissed on grounds of subjectivity because measuring intuitiveness can also be targeted among participants with certain level or complete unfamiliarity to the particular subject of interest or topic. The participants' responses were then subjected to inferential statistics and shown below.

Table 7: Test Statistics[a,b] for Database Query Readability

	INSERT STATEMENT	SEARCH AND AGGREGATE STATEMENT	UPDATE STATEMENT	MULTIPLE INSERT STATEMENT
Chi-Square	15.453	15.333	2.516	.359
df	2	2	2	2
Asymp. Sig.	**.000**	**.000**	.284	.836

a. Kruskal Wallis Test
b. Grouping Variable: NoSQL

For Insert statement, it showed that there are significant differences between MongoDB when compared against Cassandra and between MongoDB when compared against Redis query readability. Post hoc analysis supported that MongoDB is significantly more readable than both Cassandra and Redis. For Search and Aggregate statement, there are significant differences between Cassandra when compared against Redis and between MongoDB when compared against Redis query readability. Pairwise comparison revealed that Cassandra and MongoDB is significantly more readable than Redis, while no significant difference was noted between Cassandra and MongoDB.

In summary, results showed that no unanimous NoSQL type displayed an overall superiority when benchmarked against the application specific criteria. Each NoSQL type is specialized for specific purposes and workloads. Moreover, it was found that Redis is the choice for performance and scalability, MongoDB is for query readability, schema flexibility and extensiblity, while Cassandra is for storage size allocation. The finding exposes a new question on which among the identified criteria are to be given the most weight. In the business requirement of the biobanking system, one of the major feature request and core functionality is the capability to create dynamic questionnaires in order to create a user driven forms to capture ad hoc data request. Thus, it cannot be disregarded that flexibility and extensibility are to be considered the priority for this implementation. As a result, MongoDB is the logical choice that can satisfy the data requirements of the system.

5. Conclusion

NoSQL database technology offers flexibility, extensibility, efficiency, and scalability where the traditional relational database fell short. This research adapted an evaluation framework for NoSQL databases and developed corresponding set of activities for its actual prototyping and testing. The study also described the execution of technical implementation set by the guidelines of the framework. With the foreknowledge that the type and the requirements of the application dramatically determine the appropriateness of database technology to use, the methodology highlighted the qualitative approach of eliciting the user requirements and the process of transforming into quantifiable evaluation criteria. Further, this paper aimed to make the methodology reproducible and the framework generalizable whilst adaptable to specific situations. Using the framework of evaluating data storage for healthcare domain, the study indicated that the NoSQL approach is a viable alternative to other traditional database design as it provides better query performance and scalability while still retaining certain degree of extensibility and flexibility. Among the shortlisted databases, MongoDB - a document type NoSQL was found to be the recommended choice for the current implementation and immediate need of the biobanking HIS.

Acknowledgements

Funding

The biobank project "IHITM 2016-13: Establishment of a Philippine Cancer Phenome-Biobanking System and Biomonitoring Program" led by Prof. Michael C. Velarde (MCV) was funded by the Commission on Higher Education under the Philippine-California Advanced Research Institute (PCARI) program. Also, the research itself was funded as stand-alone graduate funding under the same program.

Author's Contributors

Approved the implementation: Dr. Maria Antonia E. Habana, Dr. Rodney B. Dofitas, Designed the workflow: Philip C. Sales (PCS). Performed the experiment: PCS, Programmed the scripts: PCS, Mycar Angelo B. Chu (MBC). Analyzed the results: PCS. Reviewed the paper: MCV, Dr. Alvin B. Marcelo (ABM), Prof. Ariel S. Betan (ASB), Prof. Bryann Chua, Dr. Jun Inciong. Wrote the paper: PCS, ABM, ASB, MCV.

Statement of Reproducibility

To guarantee reproducibility of the results, all source codes, command scripts, configuration files, and Jupyter notes were deposited in a Github repository. All generated clinical datasets, benchmarking output files are also pre-generated and kept in same folder. In order to make the entire testing environment available for replication, server deployment configurations were packaged as container and is also publicly available in DockerHub link. Instructions written in the REAME.md was provided as quick-start guide for the actual implementation.

References

[1] R. Kumar and F. Fernandes, Challenges in Storage and Retrieval of Healthcare Data: Review of various NoSQL Technologies, International Journal of Innovative Research in Computer and Communication Engineering 3 (2015) 208–213.

[2] M. Ercan and M. Lane, Evaluation of NoSQL databases for EHR systems, in: Proceedings of the 25th Australasian Conference on Information Systems, ACIS, Portu, Portugal, 2014, p. 10.

[3] Z. Goli-Malekabadi and M. Sargolzaei-Javan and M.K. Akbari, An effective model for store and retrieve big health data in cloud computing, Computer Methods and Programs Biomedicine 132 (2016) 75–82. doi:10.1016/j.cmpb.2016.04.016.

[4] K. Lee and W. Tang and K. Choi, Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage, Computer Methods and Programs in Biomedicine 110 (2012) 99–109. doi:10.1016/j.cmpb.2012.10.018.

[5] P. Atzeni and V.D. Antonellis, Relational Database Theory, IHI Innovation Report (1993) 5.

[6] K. Berg and T. Seymour, History of Databases, International Journal of Management and Information Systems 17 (2013) 29–35. doi:10.19030/ijmis.v17i1.7587.

[7] D. Suciu, ACM SIGMOD, 30 30 (2001) 39–45. doi:10.1145/603867.603874.

[8] C.S. Kruse and R. Goswamy and Y. Raval and S. Marawi, Challenges and Opportunities of Big Data in Health Care: A Systematic Review , JMIR Medical Informatics 4 (2016) 38. doi:10.2196/medinform.5359.

[9] S. White, A review of big data in health care: challenges and opportunities, Open Access Bioinformatics (2014) 13–18doi:10.2147/OAB.S50519.

[10] S. Wasan and V. Bhatnagar, The impact of data mining techniques on medical diagnostics , Data Science Journal 5 (2016) 119–126. doi:10.2481/dsj.5.119.

[11] H. Al-Fatlawi and B. Al-Assadi and H. Jabardi, Dynamic database schema for Hospital Management System, Asian Journal of Information Technology 14 (2015) 122–128. doi:10.3923/ajit.2015.122.128.

[12] O. Schmitt and T. Majchrzak, Using document-based databases for medical information systems in unreliable environments, in: Proceedings of the 9th International ISCRAM Conference , ISCRAM, Vancouver, Canada, April 2012, pp. 1–6.

[13] Y. Jin and T. Deyu and Z. Xianrong, Research on the Distributed Electronic Medical Records Storage Model, in: Proceedings in 2011 IEEE International Symposium on IT in Medicine and Education, IEEE, Cuangzhou, China, December 2011, pp. 288–292. doi:10.1109/ITiME.2011.6132041.

[14] Z. Parker and S. Poe and S.V. Vrbsky, Comparing Nosql Mongodb to an Sql DB, in: Proceedings of the 51st ACM Southeast Conference, ACMSE, Savannah, Georgia, USA, April 2013, pp. 1–6. doi:10.1145/2498328.2500047.

[15] G.D. Ferreira and A. Calil and R.D. Mello, Providing DDL Support for a Relational Layer over a Document NoSQL Database, in: Proceedings of International Conference on Information Integration and Web-based Applications Services, ACM, Vienna, Austria, December 2013, pp. 125–132. doi:10.1145/2539150.2539196.

[16] S. Edlich, List of nosql databases, Retrieved from: http://nosql-database.org (accessed on September 2018).

[17] C. Chan and D. Kaufman, A technology selection framework for supporting delivery of patient-oriented health interventions

in developing countries, Journal of Biomedical Informatics. 43 (2010) 300–306. doi:300--306.

[18] A. Ostrovsky and N. Dean and A. Simon and K. Mate, A Framework for Selecting Digital Health Technology, IHI Innovation Report (June 2014) 5.

[19] DB-Engines Ranking, Database ranking, Retrieved from: https://db-engines.com/en/ranking (accessed on May 2018).

[20] M.R. Barbacci and R. Ellison and A.J. Lattanze and J.A. Stafford and C.B. Weinstock and W.G. Wood , Quality Attribute Workshop, Third Edition, Software Engineering Institute.

[21] A. Gandomi and M. Haider, Beyond the hype: Big data concepts, methods, and analytics, International Journal of Information Management 35 (2014) 137–144. doi:110.1016/j.ijinfomgt.2014.10.007.

[22] G. Weglarz, Two worlds of data – unstructured and structured , DM Review 14 (2004) 19–22.

[23] HealthDataManagement, 6 award winning FHIR application, Retrieved from: https://www.healthdatamanagement.com/list/6-award-winning-fhir-applications (accessed on November 2018).

[24] J. Walonoski and M. Kramer and J. Nichols and A. Quina and C. Moesel and D. Hall and C. Duffett and K. Dube and T. Gallagher and S. McLachlan, Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record, Journal of the American Medical Informatics Association 25 (2018) 230–238. doi:10.1093/jamia/ocx079.

[25] J.C. Mandel and D.C. Kreda and K.D. Mandl and K. Isaac and R.B. Ramoni, SMART on FHIR: A standards-based, interoperable apps platform for electronic health records. , Journal of the American Medical Informatics Association. 23 (2016) 899–908. doi:10.1093/jamia/ocv189.

[26] B.F. Cooper and A. Silberstein and E. Tam and R. Ramakrishnan and R. Sears, Benchmarking cloud serving systems with YCSB , in: Proceedings of the 1st ACM Symposium on Cloud Computing, ACM, Indianapolis, Indiana, USA, December 2010, pp. 143–154. doi:10.1145/1807128.1807152.

[27] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, Ycsb++: Benchmarking and performance debugging advanced features in scalable table stores (2011).

[28] RedisLabs, Redis Persistence, Retrieved from: https://redis.io/topics/persistence (accessed on November 30, 2018).

[29] N. Zeng and G.Q. Zhang and X. Li, and C. Licong, Evaluation of Relational and NoSQL Approaches for Patient Cohort Identification from Heterogeneous Data Sources, Journal of Health and Medical Informatics 5 (2017) 1116–1121. doi:10.4172/2157-7420.1000295.

[30] J. Gilchrist and C. Ennet and M. Frize and E. Bariciak, Performance Evaluation of Various Storage Formats for Clinical Data Repositories, IEEE Transactions on Instrumentation and Measurement 60 (2011) 3244–3252. doi:10.1109/TIM.2011.2122850.

[31] E. Chan, Apache Cassandra for analytics: A performance and storage analysis, Retrieved from: https://www.oreilly.com/ideas/apache-cassandra-for-analytics-a-performance-and-storage-analysis (accessed November 30, 2018) (2016).

[32] A.T. Kabakus and R. Kara, A performance evaluation of inmemory databases, Journal of King Saud University - Computer and Information Sciences 29 (2017) 520–525. doi:10.1016/j.jksuci.2016.06.007.

[33] J. Ellis, The evolution of schema in Cassandra, DataStax. Retrieved from: https://www.datastax.com/dev/blog/schema-in-cassandra-1-1 (accessed on November 2018).

[34] Altoros, NoSQL Performance Benchmark 2018, 2016, Retrieved from: https://www.oreilly.com/ideas/apache-cassandra-for-analytics-a-performance-and-storage-analysis (accessed November 30, 2018).

[35] S. Wang and Y. Man and T. Zhang and T.J. Wong and I. King, Data management with flexible and extensible data schema in CLANS , Procedia Computer Science 24 (2013) 268–273. doi:10.1016/j.procs.2013.10.050.

[36] N. Ramachandran and W.K. Wong and I. McNicoll, Healthcare Data Issues - Internal Data Modelling, Data Transfer, Data Model vs Dataset, Retrieved from: http://opencancer.net (accessed on April 2018).