Section 1: Week 2: Reproduce an Experiment

Nate Bachmeier

TIM-7020: Database and Business Intelligence

January 26th, 2020

North Central University

Exploring the Not Only SQL World

## Describe the Issue

Even trivial applications need to be cognizant of the differences between vectors, maps, and trees. If the application needs to perform millions of random accesses, then the vector would underperform the map, because of the lack of a key-index. The inverse is also true that for sequential reads, where the map will underperform as that is not its intended use case. These same concepts of access-pattern specific data structures apply to larger and more complex environments. As the data becomes larger and needs redundancy, availability, and persistence, these data structures need to become encapsulated into data management solutions. Just as there are multiple data structures with distinct scenario-specific optimizations, this concept recurses into the design decisions of different data management solutions. Not Only SQL (NoSQL) is a broad category of technologies that exploit these distinctions to enable built-for-purpose expert systems, in contrast to traditional general-purpose relational technologies. With the emergence of IoT, Cloud, Big Data, and Mobile (ICBM)—businesses need to adopt NoSQL solutions that are specific to the problem and not assume that one size fits all. When the business chooses the correct technology, then it becomes easier to gain insights by transforming their data into business intelligence. Organizations that are capable of making timely decisions are more productive and competitive in dynamic market conditions.

For instance, a traditional representation of a social network is a graph, and using graph databases (e.g., Apache TinkerPop and Neo4J) will likely require fewer resources than a general relational store (e.g., Azure SQL and Postgres). The primary reason for these gains comes from each node contains a local index of edges versus a relational store, which relies on global indexes that have more noise to filter. The same behavior occurs with time-series information that needs to perform sequential reads and tools like Influx and OpenTSDB focus on these specifics. Big data platforms like Azure Data Lake Analytics and Amazon Athena, attempt to negate the value of these specialized indexes by horizontally scaling partitioned data. While horizontal scaled compute can address most data management challenges, it can quickly become prohibitively expensive for smaller organizations (and doctoral students).

Martino et al. (2019) state that an official benchmark suite does not exist for time-series data management, and this introduces challenges for comparing different NoSQL solutions. To mitigate this limitation, they ingested real-world Industrial Internet of Things (IIoT) workloads into a document store (Mongo), a columnar store (Casandra), and a time-series store (Influx). The researcher then evaluated batch-ingestion times, retrieval, and disk usage of the different platforms. Balis et al. (2017) discuss a similar comparison test between a document store (Mongo), relational store (Postgres), in-memory cache (Redis), and a time-series store (Influx). They also relied on performance counters to measure memory, disk, and query times to perform random access fetches of 1,000 records. Anikin et al. (2019) also evaluate the resource utilization characteristics between various graph databases (e.g., Apache HBase, Apache TinkerPop) against a relational store (Postgres) and in-memory platform (Apache Spark).

## Describe the accomplishments and limitations

A reoccurring theme across these tests is that using an expert system outperforms a general-purpose solution. The authors collected statistical data to make comparisons between the different technologies; however, except for Balis et al., the queries are not provided. There is also little mentioned around technology-specific nuances. For instance, Amazon DynamoDB, a key-value store, can emulate fast sequential scans using the starts-with operator to retrieve multiple related items in a single query. When users do not follow these design-specific patterns, then the same operation could require multiple fetches and could be perceived as less optimal. Redis supports hosting in-memory vectors and maps, which enables developers to arbitrarily index data for different access-patterns. If there is a misalignment between the indexing and retrieval strategies, then the strengths of that store are again diminished.

One challenge across all three research groups is that nearly all tests use a single node configuration. A critical aspect of NoSQL’s scalability comes from being horizontally partitioned across multiple node clusters. By splitting the traffic across multiple nodes subsystems, of the technology can operate in isolation without fighting for finite resources. For instance, an HBase topology brings together process monitoring (Zookeeper), file system management (Hadoop), task orchestration (Yarn), among other background systems. Production environments always distribute these administrative tasks across multiple nodes, specifically because they step on each other during high-stress scenarios. Another concern with single-node configurations comes from the feasibility of continuing to vertically scale up to the demands of the big data workload. The Hadron collider generates 300MB/s (Basanta-Val et al., 2017), making a thirty-day retention 0.78Pb unlikely to fit, let alone be performant on an individual server.

Another limitation that Anikin et al. call out is that their tests use one dataset, and that makes it difficult to conclude generalized observations. This issue also applies to the work of both Martino et al. and Balis et al. NoSQL stores are expert systems that address specific-domains with a specific-access pattern. That introduces complexities to know upfront, *which* storage system in *what* configuration produces the best results. Organizations need to test their workloads at scale across multiple technologies, and then look at the telemetry to make informed architectural decisions.

## Describe the results and contributions

Martino et al. (2019) state that it is equally critical to measure all aspects of the data lifecycle, such as ingestion time, query processing, and storage requirements. Their first test of time to ingest data demonstrated that Mongo and Casandra were multiple orders of magnitude slower than Influx. The second and third shows that Influx is substantially faster to query time sequences than the general-purpose stores, though looking on a non-indexed value was significantly slower than both Mongo and Casandra. These results align with the expectations that purpose-built NoSQL stores constrain the developer’s approach to interacting with the system.

Balis et al. (2017) reduce the risk that an individual query does not align with the data store, by measuring nine variations across millions to billions of records. Their results show that Redis is orders of magnitude faster with query nine taking 0.1 seconds versus Postgres requires multiple 3.2 seconds on the same data size. Though Redis’s in-memory only model limits the tests to ten million records, while Mongo, Postgres, and Influx scaled up to one billion records. The authors also consider the memory usage for each system and call out that Redis followed by Postgres used the most. This result is slightly misleading as Redis keeps the entire database in-memory and Postgres caches frequently accessed pages to improve performance (Madusudanan, 2016). They conclude with a stress scenario of writing 300k records every 30 seconds and executing multiple queries in a loop. These results highlight that Postgre’s table locks are several orders more expensive than the eventual consistency solutions.

Anikin et al. (2019) measure the query time of various graph databases using different topology configurations, such as HBase on a single node versus clustered. Each environment ingested the same data sets, and then the same graph operations (e.g., breath-first and depth-first searches) were performed. The results suggest that a correlation exists between the query duration and traffic patterns. For instance, Postgres was the most aggressive to cache records in memory, and this causes the stress test to perform better after the system warmed up. The researchers also call out the measurable difference between solid-state drives versus hard-disk drives (SSD vs. HHD) for specific systems, due to the slower I/O channel.

# Present the Inspired Work

## What are the conclusions