The Case for NoSQL on a Single Desktop

Ryan D. L. Engle, Brent T. Langhals, Michael R. Grimaila, and Douglas D. Hodson

***Abstract*—In recent years, a variety of non-relational databases (often referred to as NoSQL database systems) have emerged and are becoming increasingly popular. These systems have overcome scaling and flexibility limitations of relational database management systems (RDBMSs). NoSQL systems are often implemented in large-scale distributed environments serving millions of users across thousands of geographically separated servers. However, smaller-scale database applications have continued to rely on RDBMSs to provide transactions enabling Create, Read, Update, and Delete (CRUD) operations. Since NoSQL database systems are typically employed for large-scale applications, little consideration has been given to examining their value in single-box environments. Thus, this paper examines potential merits of using NoSQL systems in these small-scale single-box environments.**

***Index Terms*— data storage systems, databases, database systems, relational databases, data models, NoSQL.**

I. Introduction

Ever since E.F. Codd articulated the relational database model at IBM in 1970, organizations have turned to relational databases as the primary means to store and retrieve data of perceived value [1]. For decades such a model has worked extremely well, especially given business intelligence applications tended to use data in a format conducive to storage and retrieval mechanisms afforded by the powerful Structured Querying Language (SQL) that comes standard each relational database implementation, regardless of vendor. However, by the mid-2000s and with advent of Web 2.0, organizations began to understand valuable data existed in diverse formats (blob, JSON, XML, etc.) and at such a scale (petabyte or larger) that did not easily assimilate into precise, pre-defined relational tables. Furthermore, with the globalization of the internet, organizations found the need for the data accessibility to be independent of geography, and indeed, in many cases by millions of simultaneous users with extremely low tolerance for latency. Traditional relational database models suddenly were found lacking in terms of scalability, flexibility, and speed. As a natural consequence, a new generation of data storage technologies were developed to address these shortcomings and were somewhat euphemistically given the general moniker of NoSQL or "Not Only SQL" [5]. While the meaning of the term NoSQL is imprecise and often debated, for the purposes of this paper, NoSQL is used to generally refer to all non-relational database types.

NoSQL database technologies tended to develop along four general class lines: key-value, columnar, document, or graph [3]. Each one addressed, in its own way, the issues of consistency, availability and partitionability (i.e. Brewers CAP Theorem) while simultaneously maximizing the usefulness of diverse data types by focusing on the aggregate model rather than the traditional data model used by relational databases [4]. Aggregates, in the NoSQL world, represent a collection of data that is interacted with as a unit, effectively setting the boundaries for ACID transactions [4]. As a concept, aggregates are extremely important because they define at a foundational level how data is stored and retrieved by the NoSQL database, regardless of type or class. In other words, NoSQL databases need to know a good deal about the data intended to be stored and how it will be accessed, while focusing less on the structure in which the data arrives. In doing so, NoSQL developers have created a new generation of database tools that are extremely fast, easily partitioned, and highly available, but at the cost of an ability to create independent complex queries.

Given NoSQL databases were designed to operate in an enterprise environment, at scale on large numbers of commodity hardware, while supporting a diverse array of data types, little discussion of their use has been devoted to more traditional deployments (i.e. single desktop solutions). Indeed, in the haste to develop NoSQL tools to meet emergent business use-cases, comparatively little effort has been expended to evaluate in any rigorous way the inherent advantages NoSQL databases may possess over relational databases in any context except a distributed environment. It is conceivable advantages exist to using NoSQL technology in more a limited deployment. Clearly some inherent NoSQL advantages such as ability to easily partition are minimized, if not lost, in a single box implementation, other advantages may remain. The balance of this paper illuminates such potential advantages.

II. Advantages of NoSQL (on a Small Scale)

Before deciding to use NoSQL as a single-box solution, it is useful to consider which potential advantages NoSQL retains comparative to relational databases without regard to any specific deployment strategy. While NoSQL was initially developed to address deficiencies with the relational database model regarding large data sizes and distributed environments, many of the unique attributes which make NoSQL databases desirable exist regardless of scale. The following sections describe some potential advantages depending on the needs and desires of the application using the data.

A. *Minimizing (or Eliminating) ETL Costs*

ETL (extract, transform, and load) is a database management technique designed to facilitate sharing of data

Funding for this research was provided as part of the Air Force Office of Scientific Research grant 18RT0095 in support of Dynamic Data Driven Application Systems; PI: Dr. Eric Blasch.

The contact author for this Short Research Paper is Ryan Engle, rdengle@gmail.com, ryan.engle@afit.edu. Ryan is a PhD Candidate at the Air Force Institute of Technology (AFIT), 2950 Hobson Way, Wright-Patterson AFB, OH 45433.
Dr. Brent Langhals, brent.langhals@afit.edu, is an Assistant Professor of Information Resource Management at AFIT.
Dr. Michael Grimaila, michael.grimaila@afit.edu, is a Professor and Head of the Systems Engineering and Management Department at AFIT.
Dr. Douglas Hodson, douglas.hodson@afit.edu, is an Associate Professor of Computer Engineering at AFIT.

between multiple data repositories. Historically these data stores were relational databases designed to serve specific homogeneous purposes. Given relational databases are designed to follow rigid data models with prescribed schema and structures, data inserted into or retrieved from the database often must be transformed into a new state before loading into either a new database, data warehouse or a desired end-user application. Significant costs in terms of time and effort are dedicated to ETL tasks.

NoSQL databases present an opportunity to significantly reduce ETL costs, because they store data closer to its native format and instead choose to push data manipulation to the application level. Many NoSQL databases are agnostic to what "data" is actually being stored. The ability to work with data in its native state potentially reduces coding requirements for both creating the database as well as managing it down the road. Furthermore, since NoSQL is predicated on the aggregate model, much is known *a priori* about how the target data will be used upon retrieval. Applications intended to use the data will be aware of the native state and presumably would be ready to accommodate the data in such format. Which leads to the second advantage, the ability to handle heterogeneous data.

B. Acceptance of Heterogenous Data

Much of data in existence today arrives in diverse formats, meaning it arrives as a collection of data where one record does not "look like" the other records. To visualize this, consider all the potential content of one employee's human resources folder. Such a folder may include values (name, date of birth, position title), relationships (dependents, multiple supervisor-to-employee relationships, contacts in other organizations/companies), geospatial data (addresses), metadata (travel vouchers, work history, security attributes), images (ID picture, jpeg of educational records), and free text (supervisor notes, meeting minutes). Further complicating the heterogeneous nature of the data, is that not all employees' records may contain the same type or amount of data. In a relational database, this can be dealt with by either creating a very wide table with many fields to cover every possible attribute (and accept a significant number of NULL values) or creating a large number of tables to accept every possible type of record type (and risk a significant number of complex joins during retrieval). Both solutions are highly unsatisfactory and can cause significant performance issues, especially as the database grows. NoSQL avoids such complications by storing data closer to the format in which it arrives.

Additionally, NoSQL's ability to manage such data in its native state offers particularly interesting advantages for decision making. For example, upon loading the data into the database, tools exist that can alert upon key terms of interest (i.e. people, places, or things) which can subsequently enhance categorization and indexing while incorporating the original context accompanying the data when it arrives. The result is richer information for decision making. In contrast, relational databases, in the effort to transform the data to conform to rigid data models, often lose the associated context and the potentially beneficial information it contains. Building upon the storing data natively, the next advantage of NoSQL databases is the ability to support multiple data structures.

C. Support for Multiple Data Structures

In many ways, relational databases were initially designed to address two end user needs: interest in summative reporting of information (i.e. not returning individual data) and eliminating need for the "human" to explicitly control for concurrency, integrity, consistency or data type validity. As a result, relational databases contain many constraints to guarantee transactions, schemas and referential integrity. This approach has worked well as long as the focus of the database was on satisfying the human end user and performance expectations for database size and speed were not excessive. In today's world the end user is often not simply a human sitting at the end of terminal, but rather software applications and/or analytics tools which value speed over consistency and seek to understand trends and interconnections over information summaries. Suddenly the type, complexity, and interrelatedness of the data store mattered and thus NoSQL databases evolved to support a wide-range of data structures.

- Key value stores, while simplistic in design, offer a powerful way to handle a range of data from simple binary values to lists, maps, and strings at high speed
- Columnar databases allow grouping of related information into column families for rapid storage and retrieval operations
- Document databases offer a means to store highly complex parent-child hierarchal structures
- Graph database provide a flexible (not to mention exceptionally fast) method to store and search highly interrelated information

The one constant from each of the database types and the data structures they support is the presupposition that NoSQL is driven by application-specific access patterns (i.e. what types of queries are needed). In effect, NoSQL embraces denormalization to more closely group the logical and physical storage of data of interest and uses a variety of data structures to optimize this grouping. In doing so data is stored in the most efficient organization possible to allow rapid storage and querying [6]. The tradeoff, of course, is if the queries change, NoSQL (in general) lacks the robustness to support the complex joins of SQL based relational databases. However, it should not be inferred that NoSQL equates to inflexibility.

D. Flexibility to Change

To compare the flexibility of relational vs. NoSQL databases, it is helpful to remember how each structures data. A relational database has a rigid, structured way of storing data, similar to a traditional phone book. A relational database is comprised of tables, where each row represents an entry and each column stores attributes of interest such as a name, address, or phone number. How the tables, attributes, data types permitted in each field are defined is referred to as the database schema. In a relational database, the schema is always well defined before any data is inserted because the goal is to minimize data redundancy and prevent inconsistency among tables, a feature essential to many businesses (i.e. financial operations or inventory management). However, such rigidity can produce unintended complications over time. For example, a column designed to store phone numbers might be designed to hold exactly 10 integers because 10 is the standard for phone numbers in the United States. The obvious advantage is any data entry operation which attempts to input anything other than 10 whole numbers (i.e. omits an area code or includes decimals) results in rejecting the input, resulting in highly consistent (and more likely accurate) data storage. However, if for any reason the schema needs to change (i.e. your organization expands internationally, and you need to accommodate phone number entries with more than 10 integers), then the entire database may need to be modified. For relational databases, the benefits of rigid initial organization come with compromised future flexibility.

In comparison, NoSQL databases do not enforce rigid schemas upfront. The schema agnostic nature of NoSQL allows seamless management of database operations when changes, such as inclusion of international standards for phone numbers, are required. NoSQL systems are also capable of accepting varying data types as they arrive thus, as previously discussed, the need to rewrite ETL code to accommodate changes in the structure, type or availability of data is minimized. Some NoSQL databases take this a step further and provide a universal index for the structure, values, and text found in arriving data. Thus, if the data structure changes, these indexes allow organizations to use the information immediately, rather than having to wait months while new code is tested as typically is the case with relational databases. Of course, this flexibility comes with costs, primarily in terms consistency and data redundancy issues, but some applications, even on a singles desktop, may not be concerned about attendant problems these issues cause.

E. *Independence from SQL*

Structured Query Language (SQL) is the powerful, industry standardized, programming language used to create, update, and maintain relational databases. It is also used to retrieve and share data stored in the database with users and external applications. The power of SQL is derived from its ability to enforce integrity constraints and link multiple tables together in order return information of value. However, using SQL also imposes a certain rigidness on how developers and users alike can interact with the database. Additionally, as the database increases in scale, multi-table joins can become extremely complex, thus the effectiveness of SQL is, to some degree, dependent on the skills of the database administrator.

While several databases belonging to the "NoSQL" class have developed a SQL-like interface, they typically do so to maintain compatibility with existing business applications or to accommodate users more comfortable with SQL as an access language [2]. NoSQL databases also support their own access languages with varying lesser degrees of functionality than SQL. This trade-off allows independence from SQL and permits a more developer-centric approach to the design of the database. Typically, NoSQL databases offer easy access to application programming interfaces (API) and are one of the reasons NoSQL databases are very popular among application developers. Application developers don't need to know the inner workings and vagaries of the existing database before using them. Instead, NoSQL databases empower developers to work more directly with the required data to support the applications instead of forcing relational databases to do what is required. The resulting independence from SQL represents just one of many choices offered by NoSQL databases.

F. *Application Tailored Choices (Vendor, Open Source)*

The NoSQL environment is clearly awash with choice. In 2013, over 200 different NoSQL databases options existed [4]. As previously mentioned, key-value, column, document, and graph comprise the broad categories, but each NoSQL implementation offers unique options for developers and users to choose features best suited for a given application. The key to choosing is dependent upon understanding how the relevant data is be acquired and used.

If the application is agnostic to the "value" being stored and can accept limited query capability based on primary key only, the simplicity and speed of key value databases may be the answer. Such apps may include session data, storing user preferences, or shopping cart data. Alternatively, if the developer wishes to store the data document (often as JSON, XML, or BSON) and requires the ability to search on a primary key plus some stored value, document databases are an excellent choice. Apps that are likely to take advantage of document databases would include content management systems, analytic platforms, or even e-commerce systems. Columnar databases aggregate data in related column families without requiring consistent numbers of columns for each row. Column families facilitate fast, yet flexible, write/read operations making this database type well suited for content management systems, blogging systems, and services that have expiring usage. Finally, graph databases allow storage of entities (nodes) along with corresponding relationship data (edges) without concern for what data-type is stored while supporting index-free searches. These characteristics make graph databases especially well-suited for applications involving social networks, routing info centers, or recommendation engines. These choices emphasize the power of the NoSQL aggregate model over the relational data model.

Making the choice even more appealing, is that numerous open source options exist for every NoSQL database type. While most NoSQL databases do offer paid support options, nearly all have highly scalable, fully-enabled versions, with code made freely available. Popular names like MongoDB (document), Redis (key value, in memory), Neo4j (graph), and HBase (column), among many others, all represent industry standard options that are unlikely to become unsupported due to neglect since each enjoy avid developer communities. These low cost, open source options enable users to experiment with NoSQL databases with minimal risk while allowing successful implementations to become operational (even when intended to produce profit!) without huge upfront costs. While free open source versions of relational databases exist (i.e. MySQL), they are often provided as an introductory or transitional offer to the more powerful, for profit versions or have restrictive usage agreements for no-cost versions. The net result is with NoSQL, users can not only choose the database best suited to their application needs, they can also do so with a low or no-cost options with access to the underlying code in order to remain independent of software vendors. It is very likely that users of single box or desktop solutions would find this level of choice very appealing.

III. Uses for NoSQL on the desktop

The unanswered question remains, what application or types of applications, running on a single machine, perhaps even an individual's desktop, might benefit from one or more of these potential advantages? Instead of defining a list of specific applications, it is more useful to approach the problem from the perspective of the characteristics such applications may seek. The following list, while certainly not exhaustive, highlights a few characteristics that would form a starting point for deciding which NoSQL database to choose:

- Efficient write performance – i.e. collecting non-transactional data such as logs, archiving applications
- Fast, low-latency access – i.e. such as that required for games
- Mixed, heterogeneous data-types – i.e. applications that use different media types (such as an expert system containing images, text, videos, etc.)
- Easy maintainability, administration, operation – i.e. home-grown applications without professional support
- Frequent software changes – e.g. embedded systems

In the end, each database developer must consider the goals of the database and choose the type to match the requirements needed. The key realization is, regardless of scale, there are available choices beyond relational database models.

IV. Conclusions

When choosing a database to support a specific application, the choice between relational versus NoSQL options should come down to what the database needs to accomplish to support the application. The thesis of this paper challenges the notion that NoSQL is only considered useful for applications requiring big data and distributed support, while applications residing on a single box or desktop solutions remain the domain of relational databases. Instead, the authors believe choosing the database type should be driven by expected usage and performance concerns. It is hoped the paper encourages researchers (and developers) to rigorously define methodologies and advantages related to NoSQL outside enterprise solutions. In doing so, the future applications may benefit from a richer, more understood set of choices when selecting an appropriate data storage method.

References

[1] Codd, E. F, "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, vol. 13, pp. 377–387, June, 1970.

[2] Hecht, R. and Jablonski, S., "NoSQL Evaluation: A Use Case Oriented Survey", International Conference on Cloud and Service Computing, 12-14 December, 2011.

[3] Nayak, A., Poriya, A. and Poojary, D., "Type of NoSQL Databases and its Comparison with Relational Databases", International Journal of Applied Information Systems, Vol 5, pp 16-19, March, 2013.

[4] Sadalage, P. J. and Fowler, M., NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River: Addison-Wesley, 2013.

[5] Schram, A. and Anderson, K. M., (2012) "Mysql to NoSQL: Data Modelling Challenges in Supporting Scalability", proceedings of the 3rd Annual Conferences on Systems, Programming, and Applications: Software for Humanity, 191-202, Tucson, AZ.

[6] Denning, P. J. (2005). The locality principle. Communications of the ACM, 48(7), 19-24.