Characteristics of NoSQL Analytics Systems

John A. De Goes, CTO SlamData Inc.

2015-09-14 - Version 0.8
Overview
The Nature of NoSQL Data
APIs
NoSQL Databases
Big Data
A Generic Data Model for NoSQL
Approaches to NoSQL Analytics
Coding & ETL
Hadoop
Real-Time Analytics
Relational Model Virtualization
First-Class NoSQL Analytics
Characteristics of NoSQL Analytics Systems
  1. Generic Data Model
  2. Isomorphic Data Model
     Example
  3. Multi-Dimensionality
     Example
  4. Unified Schema/Data
     Example
  5. Post-Relational
     Example
  6. Polymorphic Queries
     Example
  7. Dynamic Type Discovery & Conversion
     Example
  8. Structural Patterns
     Example
Conclusion
Overview

Semi-structured data, called NoSQL data in this paper,¹ is growing at an unprecedented rate. This growth is fueled, in part, by the proliferation of web and mobile applications, APIs, event-oriented data, sensor data, machine learning, and the Internet of Things, all of which are disproportionately powered by NoSQL technologies and data models.

NoSQL operational databases continue to gain ground on relational databases, with MongoDB recently becoming the 4th most popular database in the world. Hadoop is well on its way to becoming the de facto “data lake” for company-wide data, regardless of structure, and the rapid maturation of machine learning has provided robust ways to turn unstructured data like videos, audio, and images into NoSQL data.

Recently, there has been a flurry of discussion about the implications of the rise of NoSQL for analytics and decision-support systems. These discussions often revolve around use cases, the extent to which non-tabular data models permit analytics (and if so what kind), and whether or not NoSQL systems have the ability to participate in analytic workloads.

As expected for any early-stage technology, these discussions are often imprecise, and conflate a wide range of concerns, including semantics, architecture, performance, technology, use cases, and user interface.

In contrast, this paper carves out a single concern, by focusing on the system-level capabilities required to derive maximum analytic value from a generalized model of NoSQL data. This approach leads to eight well-defined, objective characteristics, which collectively form a precise capabilities-based definition of a NoSQL analytics system.

These capabilities are inextricably motivated by use cases, but other considerations are explicitly ignored. They are ignored not because they are unimportant (quite the contrary), but because they are orthogonal to the raw capabilities a system must possess to be capable of deriving analytic value from NoSQL data.

¹ Technically, NoSQL refers to “Not Only SQL”, and while the term has historically been used mainly to describe NoSQL operational databases, it applies equally to alternative data models.
The Nature of NoSQL Data

To discuss a NoSQL analytics system, we must first have a coherent definition of the term *NoSQL data*, as well as some assurance that this definition permits enough abstraction to formally model general-purpose analytical capabilities.

In the relational data model, the core abstraction for data is a structurally homogeneous set of tuples of atomic values. NoSQL generalizes this to *arbitrary data structures* of the type that are found in programming languages like Javascript.

NoSQL analytics, then, refers to *analytics over arbitrary data structures*. While a system capable of extracting analytic value over arbitrary data structures might sound intractably complex, the variation can be abstracted with a few building blocks.

To motivate this abstraction, and establish the type of data that a NoSQL analytics system is expected to support, the following sections review the primary sources of NoSQL data, and the data models that they employ.

**APIs**

There are well more than 200,000 APIs in the world. Without exception, each of these APIs accept and produce NoSQL data. The concept of producing relational data from an API does not even make sense, as it would require an API that could accept and produce a database.

As APIs continue to become the fabric that binds technology together, they will continue to be an inexhaustible source of NoSQL data.

APIs are not databases, but they do invariably expose database-like mechanisms for querying — including filtering, shaping, and in some cases aggregation. API data is also frequently stored in files, databases and data lakes for subsequent analysis.

The primary data formats employed by APIs are as follows:

- **JSON.** An acronym for JavaScript Object Notation, JSON is by far the most common API format, thanks to its simplicity to generate and parse.
- **XML.** An acronym for Extensible Markup Language, XML still plays a role in many APIs, particularly SOAP, and its close relative HTML is the primary medium of content for the web.

**NoSQL Databases**

NoSQL databases have existed since the 1960s, but have only proliferated in number and surged in popularity in the past decade. Originally driven solely by the need for web-scale storage, today companies adopt NoSQL databases only partially for horizontal scalability.
NoSQL databases provide an operational ease rarely seen with RDBMS. In addition, they provide agility and flexibility not possible in the relational model, and an order-of-magnitude performance improvement for certain classes of problems. Finally, because they support much richer data models than relational systems, they make it possible to build more complex applications with substantially less effort.

There is little standardization among NoSQL databases. Every database exposes its own unique set of APIs, its own data model, and its own query language (if distinct from the APIs).

Despite the heterogeneity, NoSQL databases can be classified into categories based on the type of data model they support. These categories include key/value-oriented, document-oriented, graph-oriented, data structure-oriented, and wide column-oriented (among others).

The data models of a few common NoSQL databases are presented below:

- **MongoDB**. A document-oriented database, which supports a strict superset of JSON, including arbitrary nesting of sub-documents and arrays, as well as leaf types such as integers, floating-point numbers, strings, and date/times.
- **Aerospike**. A data structure-oriented database, which supports arbitrary nesting of lists and maps, as well as leaf types such as integers and strings.
- **Redis**. A data structure-oriented database, which supports flat lists, sets, and maps, and string leaf types (in practice, these strings often store data structures such as JSON).
- **CouchDB**. A document-oriented database, which supports arbitrary JSON.
- **ElasticSearch**. A document-oriented database, which supports arbitrary JSON.
- **MarkLogic**. A document-oriented database, which supports arbitrary XML, as well as JSON via a conversion layer.
- **Clusterpoint**. A document-oriented database, which supports hierarchical documents that can encode JSON, XML, and similar content.
- **Neo4j**. A graph-oriented database, which supports values that contain typed references (edges) to other values, and a mapping from string to values (properties), such as numbers, booleans, strings, and arrays of the above.

**Big Data**

Hadoop has made commonplace the notions of “infinite” file systems and localized data computation. This, in turn, has made it increasingly common to store, archive, and process massive quantities of data in Hadoop.

Some common file formats for Hadoop include JSON, XML, ORC, Avro, and Parquet, all of which support storage of de-normalized data (some heterogeneous, some homogeneous).
These data formats are self-describing and self-contained, so a single file can contain a complete description of any kind of non-cyclic data. As a result, a large percentage of the data in big data file systems is stored in such formats.

A Generic Data Model for NoSQL

As can be seen from the preceding review, NoSQL is an amalgamation of everything non- and post-relational. Instead of standardization and uniformity, there are a multitude of databases, data models, and data formats represented by the moniker.

In practice, however, a few building blocks are sufficient to represent nearly all NoSQL data:

- **A heterogeneous ordered map from value to value.** When the keys are strings, this is often called a record, object, or document in NoSQL systems. In the general case, however, the keys need not be strings and can themselves be arbitrary values.
  - While many systems do not care about or provide ordering, some do, so the more general notion is an ordered map (i.e. a map whose key-value pairs have some user-defined ordering).
  - Maps can also represent sets, as a mapping from a unique identifier to a value.
  - Neither the keys nor the value need have the same type, which allows a direct encoding of heterogeneity.

- **A heterogeneous ordered array of values.** Unlike arrays in relational systems (which are poorly supported and not used much), arrays in NoSQL systems are generally first-class, and can contain arbitrary values of completely different types.
  - Ordered arrays can also be used to represent unordered collections.

- **A value reference.** A reference is a link to another value. This is called a foreign key in relational systems, an edge in NoSQL graph systems, and a reference in most programming data models.

- **Atomic values.** Atomic values do not contain any other value; they are the primitive types of a data model. Usually, they include things like booleans, numbers, characters, dates, times, date/times, intervals, and so forth.
  - Text is actually not atomic, as it can be represented using arrays of characters.

Any NoSQL analytics system that abstracts across different NoSQL data models will inevitably end up using a generalization that is similar (if not identical) to this one.
Approaches to NoSQL Analytics

In the history of NoSQL data, there have been many approaches to solving the problem of analytics on NoSQL data. Not all of these approaches are able to solve all problems in NoSQL analytics — they vary greatly in the expressiveness and flexibility.

This section will survey some of these approaches, and conclude by highlighting some of the recent work in open source aimed at making NoSQL analytics truly first-class.

Coding & ETL

NoSQL storage systems first arose in the 1960s. Despite the existence of data interface languages in products like IBM’s IMS (a hierarchical database built in 1966), analytics on non-relational data has historically required one of two approaches:

1. **Custom coding.** Data is pulled out of a NoSQL source and filtered, shaped, and aggregated by hand-written code. This approach is common today with NoSQL operational databases, especially by smaller companies with less sophisticated analytical needs.

2. **ETL.** Data is pulled out of a NoSQL source, transformed and flattened to a simpler relational data model. This approach is also common today, primarily among larger companies who have advanced analytical needs, a scarcity of development resources, and heavy investment in legacy relational analytics tooling.

Hadoop

The rise of Hadoop made semi-structured data much more common. This, in turn, created the need for analytical capabilities on semi-structured data.

As a general-purpose computing platform, Hadoop’s Map/Reduce framework has supported near arbitrary analytics on NoSQL data from the beginning. Originally, however, these capabilities could be leveraged only by skilled big data engineers.

The engineering burden led to the creation of Pig and Hive, two complementary (but overlapping) technologies that support basic analytics over semi-structured data.

Pig adopted an expressive NoSQL data model supporting bags, tuples, maps, and more, and exposed functionality sufficient for many common analytic scenarios. For more advanced analytic scenarios, Pig supported a pluggable UDF architecture.

Hive, meanwhile, provided a simple, if first-class, concession to nested data in the form of lateral views — a feature that, while unfamiliar to those coming from a relational background — proved indispensable to the highly nested world of NoSQL data.
Other technologies in use for analytics on semi-structured data in Hadoop include Spark, Cascading, and other computational frameworks that rely on hand-written code.

**Real-Time Analytics**
Many NoSQL operational databases have acquired the ability to perform atomic operations, such as increment and decrement on numeric values. Combined with dynamic data models, this allows NoSQL systems to perform so-called *real-time analytics*, in which various aggregations are built dynamically.

While the flexibility of such real-time analytical systems is poor, they scale easily, and provide a basic level of insight into simple event-oriented systems. The analytics produced by such systems are already aggregated, but further filtering and aggregation are possible, so real-time analytics is at best a partial solution to the problem of NoSQL analytics.

**Relational Model Virtualization**
As NoSQL data systems have slowly entered mainstream, there has been growing demand for the ability to connect relational analytics tools (such as Tableau, Qlik, Cognos, MicroStrategy, and BusinessObjects) to these NoSQL systems.

This has spawned so-called *relational model virtualization adapters*. Usually speaking the JDBC or ODBC protocols, these adapters expose a virtual relational model on top of a NoSQL data system. The most sophisticated of these drivers expose virtual tables for arrays and data nesting, and use null-padding to encode heterogeneity.

These drivers are not without application, but customer satisfaction has been poor — partially because of performance issues, but mostly because of the impedance mismatch between relational and NoSQL data models.

As NoSQL analytics systems, it will become apparent that virtualization suffers from an inability to answer many types of analytic questions over NoSQL data.

**First-Class NoSQL Analytics**
Recently, the industry has entered a new era for NoSQL analytics. The need for analytical capabilities over semi-structured data no longer requires justification. Instead, these needs are assumed, and the only point of contention is the expressiveness of such capabilities.

In the past few years, many relational systems have added one or more new column types for semi-structured data (typically JSON or XML). Some, such as PostgreSQL, allow indexing on the inner structure of this data. All expose basic capabilities for accessing such data, but the capabilities fall short of the eight characteristics of NoSQL analytics systems.

Beyond these perfunctory concessions from relational systems, we have seen a new generation of analytics systems enter the scene, such as Drill, Quasar, and FORWARD.
These systems were natively designed for performing analytics on semi-structured data. Drill adopts a JSON-like data model, Quasar strives for full generality and expressiveness, and FORWARD lands somewhere in the middle. Though they differ on their approaches and level of expressiveness, all recognize the need for truly first-class NoSQL analytics.

On the standards front, SQL++ (FORWARD), SQL² (Quasar), N1QL (Couchbase), and Impala’s SQL extensions are among many efforts at generalizing the relational query model to semi-structured data. While a standard query interface has yet to emerge, the wealth of work being done in the space suggests that eventually the industry will see convergence, and that it will look a lot like SQL, but with a richer data model and multi-dimensional semantics.
Characteristics of NoSQL Analytics Systems

The preceding sections have outlined the numerous approaches to the problem of NoSQL analytics. For each approach, there are many different solutions.

Not all of these solutions are equal. In order to derive maximum analytic value from arbitrary NoSQL data, a solution must possess the following eight characteristics:

1. Generic Data Model
2. Isomorphic Data Model
3. Multi-Dimensionality
4. Unified Schema/Data
5. Post-Relational
6. Polymorphic Queries
7. Dynamic Type Discovery & Conversion
8. Structural Patterns

These characteristics may be used to judge whether or not any given system is capable of generalized NoSQL analytics as described in this paper.

1. Generic Data Model

**Characteristic:** NoSQL analytics systems must support a generic model of NoSQL that abstracts across the differences between different sources of NoSQL data.

To the extent that a NoSQL analytics system is truly general-purpose, and capable of deriving analytic value from post-relational data models, it is necessary the system be capable of working across complex NoSQL data, such as edges in a NoSQL graph database or maps with complex keys in a data structure-oriented database.

2. Isomorphic Data Model

**Characteristic:** NoSQL analytics systems must support queries across the data as it is actually structured, or across an invertible view of the data that preserves all features of the original (and which is hence isomorphic to the original data model).

NoSQL data models are rich, and the ways in which these models are used to capture and process information differs substantially from the relational world.

The strategy adopted by some systems of exposing NoSQL data under a relational model fails, in part, because the virtual relational models do not contain the same information as the original data. They both *lose* information present in the original data, and *add* other “fake” data, resulting in a poor approximation of the original.
In order to preserve the maximum amount of analytic value from NoSQL data, NoSQL analytics systems must expose a completely lossless view of the original data.

In the ideal scenario, a NoSQL analytics system exposes and allows analytics across the data as it is actually structured, with no changes to the rich data structures or heterogeneity present in the original data set.

Example
In the case of a content management system build on an operational NoSQL database, the data model may consist of individual pieces of content (represented as semi-structured HTML), which have arrays of comments, each of which has information on the author of the comment. The content may also include a histogram of website visitors, broken down by day and browser type.

A NoSQL analytics system should reflect and allow analytics on this structure exactly as it exists in the NoSQL database, or at minimum, reflect a view of this structure which preserves all information content of the original.

3. Multi-Dimensionality

Characteristic: NoSQL analytics systems must support unrestricted lifting of set-level analytic operations to arbitrary dimensions of nested data.

The analytic utility of relational systems comes from their ability to perform set-level operations, such as filtering, grouping, and aggregation. In the relational world, the data model is always flat, and there exists a single dimension over which set-level operations may be applied: namely, the set of tuples under consideration.

In contrast, NoSQL data is inherently multi-dimensional. These dimensions of data are nested and have irregular shapes. In order to derive analytic value from them, a NoSQL analytics system must allow performing all set-level operations on arbitrary dimensions of nested data.

Example
In the case of a behavioral analytics application built on an operational NoSQL database, the data model may consist of one value per user, which contains an array of sessions. Inside each session might be an array of all events comprising the session. Events might have ad hoc structure (generated by Javascript or code running on smartphones), and may include further nesting such as a sorted list of possible locations as derived by geo IP.

A NoSQL analytics system must allow arbitrary and unrestricted analytics on any of these nested dimensions of data. For example, the system must support returning a per-user histogram of events, broken down by hour of day, and also a per-province histogram of events, across all users, broken down by hour of day.
4. Unified Schema/Data

**Characteristic:** NoSQL analytics systems must support the full range of analytic capabilities on the “schema”, without any difference in analytic expressiveness between “schema” and value.

One of the most unique properties of NoSQL systems, which makes them strictly more powerful than their relational counterparts, is that the “schema” of NoSQL data is itself data. In fact, the very notion of “schema” breaks down in many NoSQL systems, because the “schema” refers to string keys in a map-like data structure. Although these keys may be used in a fashion similar to column names in a relational system, they may also be used for storing heterogeneous data, which has no direct parallel in a relational system.

As a consequence, a NoSQL analytics system must support completely arbitrary, ad hoc analytics on “schema”. A pleasing consequence of this support is that several operations which are classically impossible or extremely difficult to do in a relational system become trivially easy in a NoSQL analytics system (such as pivots).

**Example**
In the case of a real-time analytics application built on a NoSQL operational database, the keys in a map may represent date/times, while the values may be numbers that are incremented using atomic counters that are common to NoSQL databases.

A NoSQL analytics system must be capable of pulling out the date/time values encoded in the “schema”, filtering them, joining them with other date/time histograms, and aggregating across the joined dataset for the same date/times.

5. Post-Relational

**Characteristic:** NoSQL analytics systems must be strictly more expressive than relational analytics systems.

In a NoSQL analytics system, the need for data denormalization is lessened, because NoSQL data models permit storing de-normalized data directly. However, even with a high-degree of denormalization, any given data set is still related to many others, and for analytic purposes, tying them together is essential.

Thus, a NoSQL analytics system must be post-relational rather than non-relational, supporting the full expressive power of relational algebra, including joins, filters, groups, and aggregates.

**Example**
In the case of a dump of API responses for an online store, the data model might consist of heterogeneous product catalog data, each entry containing not only information on the product...
(which varies depending on whether represents an event, subscription, product, or electronic product), but user reviews and ratings.

A NoSQL analytics system should be capable of joining the user review data, which is nested inside the product entries, to a user profile data set that maps from user ids to profile information.

6. Polymorphic Queries

**Characteristic:** NoSQL analytics systems must support queries across structurally polymorphic data.

In the limit, a collection of values may share absolutely no structure, with every value having a completely different structure from every other value. However, in many common cases, there are common structural elements across values that have the same semantic.

NoSQL analytics systems must be able to query across such common structural elements of a collection of values, even when they possess arbitrarily large amounts of structural heterogeneity.

**Example**

In the case of a multi-tenant CRM application built on a NoSQL database, the data model for customer contacts may include common elements such as contact name and email, but may also include arbitrary user-defined data structures (possibly nested and heterogeneous depending on the contact type).

A NoSQL analytics system must support queries across the common structural elements of these contacts despite the large degree of heterogeneity.

7. Dynamic Type Discovery & Conversion

**Characteristic:** NoSQL analytics systems must support runtime type identification and conversion so that custom business logic can be used to dictate analytic treatment of variation.

Heterogeneity is a defining characteristic of NoSQL data. Values may have completely different structures. Elements that have the same semantic may have different structures, while sometimes, elements that have the same structure may have different semantics.

In order to enable a business to leverage its domain knowledge of the data, it is necessary for a NoSQL analytics system to expose the type (and therefore structure) of data at runtime, and to enable conversion between different types according to custom logic.
Most relational analytics systems already support type conversion, but NoSQL analytics systems must go beyond that to support type identification, as well as far richer conversions and identifications than would be necessary in a relational system, due to the richness of NoSQL data models.

Example
Over the lifetime of an application, a field in a record may have different types: for example, a comma separated list of values embedded in a string, or an array of values.

A NoSQL analytics system must be capable of allowing queries to inspect the type / structure of the field, and then to convert the structure as business logic dictates, splitting the string into an array based on the position of the commas.

8. Structural Patterns

**Characteristic:** NoSQL analytics systems must support structural pattern matching that is capable of filtering and extracting from variable length, multi-dimensional patterns.

NoSQL data frequently represents content (such as web pages, resumes, health forms, and so forth), events, and relationships. In such cases, many analytical use cases require the identification and extraction of user-defined patterns.

These use cases are not all restricted to NoSQL data. Most relational analytics systems have a way to identify and extract user-defined patterns in event-oriented data (for example, MATCH in Vertica, NPATH in Aster; and the SQL window functions). Indeed, all relational systems can identify simple character patterns in strings with SQL’s LIKE clause.

For NoSQL data, these use cases are far more common, and they are substantially more complex because of the much richer data model.

Example
In the case of a static snapshot of the HTML for an entire website, the data model might consist of raw HTML pages. This data may be linked to transactional data.

A NoSQL analytics system must support computing the purchase rate as a function of how far a purchase link is from its nearest (topside) header element. This requires the ability to match on a variable length pattern consisting of a header, followed by zero or more intermediate nodes, followed by a block element, which contains (at some unspecified but bounded depth) a link target that matches the pattern for a purchase link.

Conclusion
As NoSQL data has risen in popularity, so also has interest in “NoSQL analytics”. While there is much discussion around analytics and BI for semi-structured data, before real progress can be
made, it is necessary to lay a foundation that precisely defines the capabilities required to extract maximum analytic value from arbitrary NoSQL data; i.e. to precisely define what a NoSQL analytics system is capable of.

This challenge is complicated by the fact that there is no such thing as a single “NoSQL data model”. However, the common NoSQL data models can all be unified with an abstract data model containing maps, arrays, references, and a variety of common atomic types.

There are many different approaches to the problem of NoSQL analytics, from ETL to coding to next-generation analytics systems.

In evaluating these approaches, eight characteristics stand out, which collectively enable a system to derive maximum analytic value from a generic model of NoSQL data:

- A generic data model capable of abstracting across a wide range of NoSQL data models
- Reflecting back a lossless view of the data
- Supporting set-level operations on arbitrary nested dimensions
- Allowing arbitrary analytics on “schema”
- Supporting all the relational operators (including joins)
- Allowing queries across structurally polymorphic data
- Enabling dynamic type identification and conversion
- Supporting multi-dimensional pattern matching

Together, these eight characteristics form a robust, capabilities-based definition of what it means for a system to support generalized NoSQL analytics. Additionally, they serve as a guide for companies evaluate competing approaches to NoSQL analytics.

With this foundation of NoSQL analytics systems laid, there is broad room for exploration, differentiation, and innovation around other relevant dimensions, semantics, architecture, performance, technology, use cases, and user interface.

Welcome to the era of NoSQL analytics.