NoSQL: Web Relevance of CouchDB, BigTable and Dynamo

"Databases are tools; they are the means to an end. Each database has its own story and its own way of looking at the world. The more you understand them, the better you will be at harnessing the latent power in the ever-growing corpus of data at your disposal." (pg.13)

What is NoSQL?

When discussing databases the first things that come to mind are popular relational databases such as MySQL and Oracle, but one has to remember that datastores come in a broad variety of types and genres, each suited for a particular class of problems.

The original NoSQL databases were born out of necessity, as web giants like Google and Amazon struggled to scale their traditional relational databases to massive new scales. Google published a paper on their solution, called BigTable, in 2006, and Amazon followed with a paper on their database Dynamo in 2007. Meanwhile, an open-source project called CouchDB that was first released in 2005 began to draw attention, and became a top-level Apache project in 2008.

The common property that bound these databases together was that they rejected, not only the language SQL, but the entire relational model that has been dominant for decades. The common realization was that, in order to maintain acceptable performance in the face of enormous amounts of data, the strong guarantees of atomicity and consistency provided by relational databases, as well as their powerful query language, would have to be abandoned in favour of simpler insertions and retrievals across a distributed cluster.

However, there were significant differences between these three approaches, so much so in fact that each one became the prototype for its own genre of database within the loosely defined NoSQL family. BigTable spawned the 'columnar' genre, which includes popular databases such as HBase, Cassandra, and Hypertable, and is structurally the most similar to traditional relational models. Dynamo inspired the 'key-value' genre, which is the simplest of the three and includes databases such as Riak, Redis, and Voldemort. And CouchDB is the original 'document-oriented' NoSQL database, which is very similar to the key-value model and includes databases such as MongoDB and SimpleDB.

Nowadays, it is generally accepted that NoSQL stands for 'Not only SQL', in the sense that these databases should be used to complement, rather than replace, relational databases. Besides the companies already mentioned, some of the biggest internet companies in the world, such as Facebook and Twitter, rely on NoSQL databases. There are currently, according to http://nosql-database.org/, approximately 132 different NoSQL databases, each of which being roughly classified into one or more of the three groups described above (or an additional genre called ‘graph’ databases that won’t be discussed here). There are no formal criteria for inclusion under NoSQL other than not using the relational model, but some properties that are generally shared are that they are open-source, schemaless, distributed, and provide easy replication and simple APIs.

Strengths of NoSQL

The primary strength of NoSQL is the reason it was created in the first place: scalability. With the increasing popularity of web applications and the emergence of massive data sets, scalability has become more important than ever. Traditional RDBMS are designed to run on a single machine, so scaling must be done 'vertically' (i.e. more powerful servers must be bought). This can be extremely expensive, and there are technical limits as to how powerful a single machine can be. NoSQL, on the other hand, has been designed to be 'horizontally' scalable, which means it can take advantage of a cluster of commodity hardware. This scalability means that some previously impossible problems become possible, and some previously expensive problems become feasible.

While the primary motivation behind NoSQL has always been scalability, it does possess another important property that is much more subtle. Relational databases store records as rows of data in a table, and columns are not permitted to store multiple values such as lists or hashes. Since the rise of relational databases, however, complex data structures and object oriented programming have swept the development world. The objects that programmers are dealing with commonly result in deeply nested hierarchies of aggregated data. Converting between these objects and the flat rows of a table can create large amounts of development work. This fundamental difference has been termed the "Object-relational impedance mismatch". NoSQL databases, on the other hand, typically store values either as blobs of data, or in formats such as JSON, which more closely models these data structures.

Weaknesses of NoSQL

The disadvantages can be divided into two categories that make them different in an important way: those that are fundamental to the way NoSQL works, and those that are simply an unfortunate by-product of the fact that NoSQL is still very new.

All new products and technologies suffer from a similar set of problems. First, new products have had less time to be optimized for performance and to be thoroughly tested under real-world conditions. Second, users will have a harder time finding professional support, unlike the relational databases that are backed by companies such as Oracle, Microsoft, and IBM. And lastly, companies will find it more difficult to find developers that are proficient in using that product. Fortunately, all of these problems can be solved as the product matures and becomes more popular, but many companies are not willing to adopt it until after the fact.

As the adage goes, there’s no such thing as a free lunch, and NoSQL databases are only able to achieve their advantages by making sacrifices. While relational databases stress data consistency, in order to maintain performance and high availability when dealing with massive data sets, NoSQL databases typically do not provide the same consistency guarantees. For example, this means that if a user wants to ensure atomicity for multiple updates to the database, they will have to implement this themselves in their application code. Also, while relational databases' rigid data model and schema might make it inflexible for inserting ad-hoc data, it lends itself to being queried using relational algebra, a powerful formalism for retrieving data in a very flexible way. NoSQL's flexibility and convenience for inserting ad-hoc and unstructured data results in it not having a flexible and convenient tool for retrieving data like SQL.

Arguably a strength or a weakness of NoSQL

When it comes to creating software, there is always a tension between safety and flexibility, and an attempt to balance those two conflicting interests in a way that will maximize programmer productivity. A classic example of this is the debate between static and dynamic typing, where dynamic typing proponents claim that their run-time flexibility increases programmer productivity, while static typing proponents argue that as systems become larger and more complex, their better ability to detect errors at compile time results in greater productivity. A similar situation exists between relational and NoSQL databases. Traditional databases force the user to provide a schema before the database can be used, including the names and data types of the columns in each table, and enforce that schema every time the user attempts to store new data. NoSQL, on the other hand, typically allows the user to insert ad-hoc data of any type and structure. Not surprisingly, this difference provokes the same kind of debate. Some argue that the schemaless NoSQL databases save time because of their better flexibility when dealing with non-uniform data and changing requirements, while others argue that declaring a schema during table creation allows the database to enforce constraints that save developers time in the long run. Which approach actually leads to better productivity will undoubtedly depend on both the specific application as well as the application developers.

What is its web relevance?

Lets start with what we all know, relational database model systems (RDBMS). These are commonly found behind many websites. However, if a website begins to grow and acquires more and more users, then it will need to grow its RDBMS so that the performance will stay equivalent. But how is this done with RDBMS? This means that a larger server needs to be purchased which is high in cost, highly complex and proprietary, especially when compared to web and cloud based architectures.

So how is NoSQL any better? For starters it does not need a schema, so when new information needs to be collected it can easily be done. And it does this in a way such that it enables cost-effective management of the the data coming from the web and mobile applications. This is important when it comes to analysis of a company's statistics. There may be a requirement for certain data types to be collected from the web pages that has been missed, but it can simply be added into the database, with little to no difficulties, so that in the future, that type of data can be included in the company analysis.

Not only is web management important, but todays technology also allows cell phones, ipads, playbooks, etc, to access the internet and interact. These types of technologies bring a large range of data with them. Is it easy to adapt to all this new incoming data? The hurdle with these mobile devices is that they require synchronization as the connection or even the device itself may be lost. NoSQL is starting to support the synchronization of data between mobile devices and database clusters deployed in the cloud, so to speak.

In summary, this tool is good when it comes to doing the analysis with large, copious amounts of data that is continuously growing. The big internet giants incorporated NoSQL into their systems for just those reasons and their systems are running efficiently. The NoSQL is also quite tailored to each company's needs. Each company had problems that they had to overcome, however, there are always some bumps in the road.

Three Specific NoSQL Databases

These three databases were chosen for more detailed analysis because of their historical importance in the NoSQL movement, and because they each represent a fundamentally different approach to NoSQL that gives rise to the different genres.

CouchDB

An open-source project started by Damien Katz in the hopes of becoming "the database of the Internet". CouchDB stores data as a series of JSON documents, and supports devices going offline and resyncing later when they reconnect. For this reason, it seems particularly suited to mobile applications, as these devices frequently see use while not connected to a data network.

The major advantages of CouchDB come from the storage schema, or lack thereof. Since data in CouchDB is stored as a series of JSON queries, 'database design' consists of inserting your JSON object directly into the database. This eliminates the need for schema design which plagues RDBMS and makes for easier schema changes/updates as well as faster load/store functionality.

Since CouchDB stores data as JSON documents and knows little about the contents of this JSON, it has a number of drawbacks as well. It does not support update functionality, requiring a delete/update combination. It is extremely difficult to enforce constraints upon the data set (i.e. unique keys). It does not support transactions as an atomic operation. Because data is stored in JSON, the size of the database grows extremely quickly. Occasionally, replication to previously-offline devices can fail when working with large data sets. It has no way to capture relationships between different 'types' of data within the database.

So far, CouchDB has failed to capture widespread acceptance. A page of "CouchDB in the wild" exists tracking major projects that incorporate CouchDB, but none of these projects are particularly noteworthy; only a couple dozen websites seem to be built off of CouchDB in any capacity and none of them are names I recognize. It may be worthwhile for use over large amounts of simple data, but for use in most database applications CouchDB does not seem to be worth adopting.

BigTable

BigTable is Google's in-house storage system. It is not readily available to users outside of Google applications. A BigTable table stores data indexed on "row keys", "column keys", and timestamps. The values in the table are arbitrary bytes, and are not interpreted by BigTable to conform to any particular format.

Research on BigTable is confounded by the fact that it is not used outside of Google. However, Google claims they adopted the system because BigTable is extremely scalable, experiencing minimal slowdown over extremely large data sets and distributing load across multiple servers with ease.

There are two main disadvantages of BigTable, beyond those faced by all NoSQL databases. The first is that it requires you to use the "column key" to group similar data together, a change in thinking that many database designers may find unnatural. The second is that, unless we're working with or for Google, we can't use BigTable anyway.

DynamoDB

DynamoDB is Amazon's proprietary storage system. A table in DynamoDB is a collection of items. Each item consists of a key and a series of values. DynamoDB comes with a very simple API (create/delete tables, get/put items into tables) that allows basic search functionality.

The primary advantage of DynamoDB is the lightweight query system, which passes minimal queries and return values. It supports in-place updates on numerical values. Amazon claims that their DynamoDB service has a high cost for storage on the backend but a low cost for querying. This makes it ideal for applications where a relatively small amount of data is constantly requested, updated, and changed. Amazon gives the example of a video game, where the database may need to store only the positions and inventories of characters; other game assets may be stored locally or elsewhere. Getting this information out of DynamoDB is easy and fast, as is storing it back into DynamoDB. Tables in DynamoDB are schemaless, so the grouping of data items in a table is semantic rather than necessary.

The disadvantages of DynamoDB are the same as with any NoSQL database. Relationships between data are not captured. The free nature of items within a table increases storage cost. Searches on elements other than primary key are possible in DynamoDB but prohibitively expensive. The query language is exceedingly simplistic which means that complicated queries on the data set are impossible. The freeform structure of tables makes error checking a realistic concern, as tables will not reject malformed data. Queries are still parsed, which makes them slightly more expensive than in CouchDB.

DynamoDB seems like a realistic alternative to RDBMS when a reasonably small data set is being frequently accessed and updated. It requires a bit more work on the application side than RDBMS to ensure the sanctity of the data, though seemingly less than other NoSQL programs.

Comparison of CouchDB, BigTable and Dynamo

	BigTable	**Dynamo**	**CouchDB**
Creator	Google	Amazon	Apache
Date	Released - 2006	Released - Jan 2012	Created - Apr 2005 Taken over by Apache - Feb 2008
Type	Database	Database	Database
License	Proprietary	Proprietary	Apache License
Model	Column Oriented, Key Value	Key Value Model	Document Oriented, Key Value Model
Data Storage		SSD	File System
Query Language	API Calls	API Calls	JavaScript, REST, Erlang
Indexing		Does not use Composite Keys for indexing	Uses Composite Keys for indexing
Text Search	Can do full text search	Cannot do full text search	Cannot do full text search
Scalability	Horizontally Scalable	Horizontally Scalable	Not horizontally scalable
Programming Language	C	Java	Erlang
Data Partition	ordered	random	
Consistency	atomic	eventual	
Architecture	hierarchical	decentralized	

Pros and Cons of the 3 Compared Database types

	BigTable	**Dynamo**	**CouchDB**
Pros	Caching a block avoid seeks for reads with locality	Highly available key value storage system	Uses JSON
	fast and sequential	scalable and decentralized	Uses Javascripts/jQuery

			which makes it easier for new users to use.
	highly scalable	Tight control over tradeoffs between availability, consistency, performance	
	fault tolerable		
Cons	small random reads have high overhead and waste memory	sacrifices consistency and achieves high availability	need to create views for each and every query
	complex recovery - recovers tablets independently yet their logs are mixed		will either have redundant data, or end up implementing join and sort logic yourself on client side
	strong need for monitoring tools		complex joins

How each of the 3 database types have contributed to web development

- BigTable is currently being used in the services such as Google Analytics and Google Earth. Google Analytics is a service that helps web masters analyze traffic patterns on their websites. It then summarizes the data and makes it available. Google Earth, a service that gives users access to high-resolution satellite imagery of the world's surface, uses one table to store raw imagery. The imagery is cleaned and consolidated into final serving data. This table contains 70 terabytes of data, with each row corresponding to a single geographic segment.
- DynamoDB is being used for IMDb.com (Internet Movie Database), an online database of information related to films, television programs, and video games. This includes actors, production crew personnel, and fictional characters featured in these three visual entertainment media. It is one of the most popular online entertainment destinations, with over 100 million unique users each month and a solid and rapidly growing mobile presence.
- DynamoDB is also used in a number of other applications, such as:
  - AdRoll, a quickly growing advertising company
  - Amazon Cloud Drive, a web storage application from Amazon
  - SmugMug, a digital photo sharing website.
- CouchDB is used by several companies and applications including:
  - BBC, a British public service broadcasting corporation, uses CouchDB for its

dynamic content platforms
  - Ajatus, a CRM that runs as a local AJAX Web application on your own computer
  - Emberlight, an application on flickr.com that allows a user to publish spatial hypertext (information-rich flowcharts) on the web
  - Spreadlyrics, a song lyrics sharing application for Android
  - BerrySki, a BlackBerry app that offers GPS Ski Maps for Europe and North America
  - Several facebook applications such as Horoscope, Birthday Greeting cards, Friend Dashboard and I Play WoW.

Conclusion

NoSQL is very different from the commonly used RDBMS. For starters, it does not use SQL for its data query or manipulation. As well, the structure is not built on relations between tables. Will NoSQL fit within our company and work with our existing programs, or will we have to change everything that we currently have in place? The benefits of NoSQL are highly dependent on the context in which it is used. Do we have an extremely large amount of data? Is this data expected to continuously grow? Do we have hard real-time performance requirements? Is the performance of our current system becoming an issue? In some cases, the answers to these questions will warrant a further investigation into a potential NoSQL solution. In extremely rare cases, such as those achieved by the likes of Google, the answers to these questions will make using NoSQL unavoidable. However, in most cases, traditional relational databases are more than performant enough, and NoSQL databases should only be considered as possible strategic adjuncts to the main RDBMS. Their ACID compliance, maturity, tools, support, available expertise, and of course, SQL, make relational databases too valuable to be passed over unless necessary.

Works Cited

http://nosql-database.org/
Seven Databases in Seven Weeks
http://en.wikipedia.org/wiki/NoSQL
http://newtech.about.com/od/databasemanagement/a/Nosql.htm
http://www.couchbase.com/why-nosql/nosql-database
research.**google**.com/archive/**bigtable**-osdi06.pdf
http://en.wikipedia.org/wiki/CouchDB
http://wiki.apache.org/couchdb/CouchDB_in_the_wild
http://aws.amazon.com/dynamodb
http://vschart.com/compare/bigtable/vs/dynamo-db/vs/couchdb
http://www.cs.berkeley.edu/~brewer/cs262/BigTableLecture.pdf
http://www.slideshare.net/kingherc/bigtable-and-dynamo