NOSQL stores and Data analytics tools

Advances in Data Management, 2012

Petra Selmer,
Birkbeck College (PhD) and UCL Institute of Neurology (Software Architect)
Agenda

- The wider landscape
- NOSQL
- Data Analytics
Preamble

• The area is HUGE
• The area is ever-changing!
• Plenty of links throughout this presentation for further reading...
The wider landscape

The evolving database landscape

© 2012 by The 451 Group. All rights reserved
The wider landscape

- Several dimensions in one picture:
  - Relational vs. Non-relational
  - Analytic (batch, offline) vs. Operational (transactional, real-time)
  - NOSQL vs. NewSQL

- Increasingly difficult to categorise these data stores:
  - Everyone is now trying fiercely to integrate features from databases found in other spaces.
  - NewSQL: implements core NOSQL features.
  - NOSQL: trying more and more to implement 'classic' features as SQL support or ACID or at least often configurable persistence.

- The emergence of “multi-model” data stores:
  - ArangoDB, OrientDB and others.
  - One may start with one data model (e.g. Document model) and add other models (graph or key-value) as new requirements emerge.
NOSQL

NOSQL:
- “Not Only SQL”, not “No SQL”
- Basically means “not relational” – however this also doesn't quite apply, because graph data stores are very relational; they just track different forms of relationships than a traditional RDBMS. A more precise definition would be the union of different data management systems differing from Codd’s classic relational model
- The name is not a really good one, because some of these support SQL and SQL is really orthogonal to the capabilities of these systems. However, tricky to find a suitable name.
- A good way to think of these is as “the rest of the databases that solve the rest of our problems”

Scalability:
- **Horizontal** (scale out): the addition of more nodes (commodity servers) to a system (cluster) - NOSQL
- **Vertical** (scale up): the addition of more resources – CPU, memory – to a single machine
NOSQL vs. Relational

- What’s wrong with relational DBs? They’re great!
  - ACID
  - Enforcement of referential integrity and constraints
  - SQL
  - Excellent support by many languages and technology stacks
  - Excellent tooling
  - Well-understood operational processes (DBAs): backups, recovery, tuning etc
  - Good security management (user access, groups etc)
NOSQL vs. Relational

● BUT...there are problems:
  ◦ Scaling with large and high-velocity data
    • ‘Big Data’
    • Expensive / difficult / impossible to scale reads and writes vertically and horizontally
  ◦ Complexity of data
    • Impedance mismatch
    • Performance issues (joins)
    • Difficult to develop and maintain
  ◦ Schema flexibility and evolution
    • Not trivial
    • Application downtime
Of hammers and nails...

The Law of the Hammer

If the only tool you have is a hammer, everything looks like a nail.

Abraham Maslow - The Psychology of Science - 1966
Of hammers and nails (cont)..

The Law of the Relational Database

If the only tool you have is a relational database, everything looks like a table.

A Walk in Graph Databases - 2012
Why did these limitations not become problems before now?

- In the past, there have been non-relational databases: Object Databases, XML Databases and proprietary formats; IBM’s IMS, Lotus Notes...and Matlab
- Types of systems – mostly operational and highly structured, and thus simpler:
  - Payroll, inventory, stock management etc
  - A simplified data model: employees previously only had one ‘phone number, one title etc
NOSQL vs. Relational

- Some datasets can be mapped easily to key-value pairs - flattening the data doesn't make it any less meaningful, and no reconstruction of its relationships is necessary.
- For other datasets, the *relationship* to other items of data is as important as the items of data themselves.
- Relational databases are based on *relational algebra (set theory)*:
  - Many datasets have relationships based on set theory, so an RDBMS is a good fit
  - However, for datasets where hierarchical or distance of relationships are required, set theory is not the best solution. In these cases, *graph theory* is a better match.
- **Summary:**
  - RDBMS are too complex for data that can be effectively used as key-value pairs: we lose *scalability*
  - RDBMS are not complex enough for data that needs more context: we lose *performance*
NOSQL

- Not intended as a replacement for RDBMS
- One size doesn’t fit all
- **Use the right tool for the job**
- Just as we shouldn't try to solve all of our problems with an RDBMS, we shouldn't try to solve all of our maths problems with set theory.
  - Today's data problems are getting complicated: the scalability, performance (low latency), and volume needs are greater.
  - In order to solve these problems, we're going to have to use an alternative data store or use more than one database technology.
Relational vs. Aggregate Data Model

Relational

- Data are divided into rows (tuples) with pre-defined columns (attributes)
- There is no nesting of tuples
- There is no list of values

Aggregate

- Think of this as a collection of related objects, which should be treated as a unit
Relational vs. Aggregate Data Model

Relational Instance

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th></th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NAME</td>
<td>ID</td>
</tr>
<tr>
<td>1</td>
<td>Guido</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPod Touch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monster Beat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BILLING_ADDRESS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CUSTOMER_ID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>STREET</td>
</tr>
<tr>
<td>55</td>
<td>Chaumontweg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORDER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CUSTOMER_ID</td>
</tr>
<tr>
<td>90</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORDER_ITEM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ORDER_ID</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

Aggregate Instance

```json
{
  "id": 1,
  "name": "Guido",
  "billingAddress": [{
    "street": "Chaumontweg",
    "city": "Spiegel",
    "postCode": "3095"
  }]
}

{
  "id": 90,
  "customerId": 1,
  "orderItems": [
    {
      "productId": 1000,
      "price": 250.55,
      "productName": "IPod Touch"
    },
    {
      "productId": 1020,
      "price": 199.55,
      "productName": "Monster Beat"
    }
  ],
  "shippingAddress": [{
    "street": "Chaumontweg",
    "city": "Spiegel",
    "postCode": "3095"
  }]
}
```
NOSQL Families

Key-Value

Column Store  (also known as Big Table)

Graph Store
NOSQL Families
# NOSQL Families

<table>
<thead>
<tr>
<th>Design</th>
<th>Key/Value Store</th>
<th>Column Store</th>
<th>Document Store</th>
<th>Graph Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key/Value pairs; indexed by Key</td>
<td>Columns and Column Families. Directly accesses the column values</td>
<td>Multiple Key/Value pairs form a document. Values may be nested documents or lists as well as scalar values</td>
<td>Focus on the connections between data and fast navigation through these connections</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scalability / Performance</th>
<th>+++</th>
<th>+++</th>
<th>++</th>
<th>++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate-Oriented</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Complexity</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>Inspiration / Relation</td>
<td>Berkley DB, Memcached, Distributed Hashmaps</td>
<td>SAP Sybase IQ, Google BigTable</td>
<td>Lotus Notes</td>
<td>Graph Theory</td>
</tr>
</tbody>
</table>

- **NOSQL Products**
  - Voldemort
  - Redis
  - Riak
  - HBase
  - Cassandra
  - Hypertable
  - MongoDB
  - CouchDB
  - Couchbase
  - Neo4j
  - OrientDB
  - DEX
  - InfiniteGraph [Triple and Quad Stores]
NOSQL Families

Size

Key-Value Stores

Column Stores

Document databases

Graph databases

Relational databases

Complexity
1) Key-Value Stores

- History – Amazon decided that they always wanted the shopping basket to be available, but couldn’t take a chance on RDBMS. So they built their own…
  - “Dynamo: Amazon’s Highly Available Key-Value Store” (2007)
- A key-value store is a simple hash table
- Generally used when all access to the data is via a primary key
- Simplest NoSQL data store
- Value is a BLOB → data store does not care or necessarily know what is ‘inside’
- Aggregate-oriented
- Simplest NoSQL data stores to use (from an API perspective)
- Accessing and writing the data: PUT, GET, DELETE (matches REST)
- Data model:
  - Global key-value mapping
  - Big scalable HashMap
  - Highly fault tolerant (typically)
- Examples:
  - Riak, Redis, Voldemort
1) Key-Value Stores

- **Strengths**
  - Simple data model
  - Great at scaling out horizontally for reads and writes
    - Scalable
    - Available
    - No database maintenance required when adding / removing columns

- **Weaknesses:**
  - Simplistic data model – moves a lot of the complexity of the application **into** the application layer itself
  - Poor for complex data
  - Querying is simply by a given key: more complex querying not supported
1) Key-Value Stores

Suitable Use Cases
- Storing Session Information
- User Profiles, Preferences
- Shopping Cart Data
- Sensor data, log data, serving ads
1) Key-Value Stores - Riak

- Developed by Basho
- Riak is a distributed database architected for:
  - Availability: replication of data means it is available for read and write operations, even in failure conditions;
  - Fault-tolerance: loss of access to many nodes owing to network partition or hardware failure does not mean a loss of data;
  - Operational simplicity: new machines can be added to the Riak cluster easily without incurring a larger operational burden;
  - Scalability: Riak automatically distributes data around the cluster and yields a near-linear performance increase as you add capacity.
- See http://basho.com/company/production-users/ for a list of users
1) Key-Value Stores - Riak

- Querying Riak:
  - Most of the interactions you'll have with Riak will be setting or retrieving the value of a key – using Riak’s HTTP API (using REST: GET/PUT etc) and/or client API
  - MapReduce can be used as a method for non-key-based querying
  - Riak Search:
    - Syntax based on Lucene: title:"The Right Way" AND text:go
    - Term and field searching
    - Exact match queries and wildcards
    - Inclusive/exclusive range queries
    - Boolean operators: AND/OR/NOT
    - Grouping and scoring
    - Prefix matching
    - Proximity searches
    - Lexicographical range searches
2) Column Stores

- Google’s “Bigtable: A Distributed Storage System for Structured Data” (2006). Sometimes this family is called Big Table, Wide Column etc

- Data model:
  - Rows are split across multiple nodes through sharding on the primary key
  - A big table, with column families. Column families are groups of related data, often accessed together
  - New columns may be added within the column family on a per-record basis, when needed. Lists of values may be stored in the column.
  - MapReduce for querying/processing
  - The records may be partitioned horizontally (sharded) across multiple servers, or parts of a SINGLE record may be stored on multiple servers (vertical partitioning)

- Examples:
  - HBase, HyperTable, Cassandra

- Aggregate-oriented
2) Column Stores - Example

- One row for Customer 1234
- Customer table partitioned into 2 column families: profile and orders
- Each column family has columns (e.g. name and payment) and supercolumns (have a name and an arbitrary number of associated columns)
- Each column family may be treated as a separate table in terms of sharding:
  - Profile for Customer 1234 may be on Node 1
  - Orders for Customer 1234 may be on Node 2

Source: NoSQL Distilled
2) Column Store

- Many key-value stores offer some form of grouping for columns and can be considered "column" stores as well.
- Some databases – like HBase – were designed as column stores from the beginning:
  - This is a more advanced form of a key-value pair database. Essentially, the keys and values become composite.
  - Think of this as a hash map crossed with a multidimensional array. Essentially each column contains a row of data.
2) Column Stores

- **Strengths**
  - Data model supports (sparse) semi-structured data
  - Naturally indexed (columns)
  - Good at scaling out horizontally
  - MapReduce is very often used on these, so they can be good analytical stores for semi-structured data
  - Can see results of queries in real time

- **Weaknesses:**
  - Uns suited for interconnected data: if the relationships between the data are as important as the data itself (such as distance or path calculations), then *don't* use a column store
  - Uns suited for complex data reads
  - Require maintenance – when adding / removing columns and grouping them
  - Queries need to be pre-written; no ad-hoc queries defined “on the fly” : NOT for use for non real-time, unknown queries
2) Column Store

- Use cases
  - Netflix use it for logging and customer analytics, among others
  - Ebay use it for search optimisation
  - Adobe use it for structured data processing and Business Intelligence (BI)
  - Used for ‘firehose’ data for TV shows such as BGT, The X Factor etc (audience and viewer voting): high amount of writes, and fast real-time basic analytics (Cassandra)
  - Event Logging
  - Counters
2) Column Stores - Cassandra

- Apache project; also distributed by third-parties such as Acunu and Datastax (who provide extra functionality on top of the base technology)

**Model:**
- Column oriented, key value.
- The values are split into columns which are pre-indexed before the information can be retrieved.
- Eventually consistent.
- This makes it better for highly distributed use cases or ones where the data is spread over an unreliable networks – lends itself well to geographically-distributed networks, in particular.

- See [http://www.datastax.com/cassandrausers](http://www.datastax.com/cassandrausers) for a list of users

- Robin Schumacher (VP of products for DataStax): "A popular use case for Cassandra is time series data, which can come from devices, sensors, websites (e.g., Web logs), financial tick data, etc. The data typically comes in at a high rate of speed, can come from multiple locations at once, adds up quickly, and requires fast write capabilities as well as high-performance reads over time slices."

- Real-time query examples at which Cassandra excels:
  - Give me X
  - How many Y?
  - What is the top K?
  - How many distinct P in Q?
3) Document Stores

- Documents are the main concept
- Data model
  - Collections of documents
  - A document is a key-value collection
  - Index-centric: primary as well as secondary
- Stores and retrieves documents, which can be XML, JSON, BSON..
- Documents are self-describing, hierarchical tree data structures which can consist of maps, collections and scalar values, as well as nested documents
- Documents stored are similar to each other but do not have to be exactly the same
- Aggregate-oriented
- Examples
  - CouchDB, MongoDB, Couchbase

```json
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```
3) Document Stores

- **Strengths**
  - Simple but powerful data model – able to express nested structures
  - Good scaling (especially if sharding supported)
  - No database maintenance required to add / remove ‘columns’
  - Powerful query expressivity (especially with nested structures) – able to pose fairly sophisticated queries

- **Weaknesses:**
  - Uns suited for interconnected data
  - Query model limited to keys (and indexes)
    - MapReduce for larger queries (thus, might be slow)
3) Document Stores

- Suitable Use Cases
  - Event Logging
  - Content Management Systems
  - Web Analytics or Real-Time Analytics
  - Product Catalogue
  - Operational Systems
3) Document Stores

- Both key-value stores and document stores talk about “key-value” pairs – what is the difference?
  - From clustering to accessing data, document stores and key-value stores are exactly the same, except in a document store, the store understands the documents in the data store - the values are JSON, and the elements inside the JSON document can be indexed for better querying and search.
  - Because of this, the querying semantics within the document store will be much richer.

- When to consider using a document store (from a data model point of view)?
  - Your schema is changing quickly over time and hence becomes too complex to model in a relational database.
  - If you were considering using an XML store, a document store will probably be a good fit.
3) Document Stores - MongoDB

- The best-known document store: 10gen
- Powerful, dynamic querying mechanism:
  - Find all documents where j is not equal to 3 and k is greater than 10:
    - `db.things.find({j: {$ne: 3}, k: {$gt: 10}});`
  - Check for existence (or lack thereof) of a field:
    - `db.things.find( { a : { $exists : true } } );` // return object if a is present
    - `db.things.find( { a : { $exists : false } } );` // return if a is missing
- Also has equivalents to IN, NOT IN, AND, OR
- Regular expression matching (of string values)
- Aggregate functions too: COUNT, GROUP etc
3) Document Stores - MongoDB

- Example of enterprise uses of MongoDB:
  - eBay: with 250M writes/day, 10M updates, 500M to 1B reads, 2ms (yes, millisecond) response time 24/7/365
  - Craigslist: Craigslist’s initial MongoDB deployment was designed to hold over 5 billion documents and 10TB of data. (Craigslist Customer Profile)
  - Disney: More than 1,400 MongoDB instances, adding new instances every day, via a custom-built self-service portal, to test, stage and host new games.
  - Wordnik: Tens of billions of documents with more always being added; more than 20 million REST API calls per day; mapping layer supports 35,000 records per second.
  - Financial Services use cases (http://www.10gen.com/presentations/webinar-how-and-why-leading-investment-organisations-are-moving-mongodb):
    - High Volume Data Feeds
    - Tick Data capture
    - Risk Analytics & Reporting
    - Product Catalogs & Trade Capture
    - Portfolio and Position Reporting
    - Reference Data Management
    - Portfolio Management
    - Quantitative Analysis
    - Automated Trading
3) Document Stores - MongoDB

- More examples of use cases with MongoDB:
  - CERN’s Large Hadron Collider – DAS (Data Aggregation System):
    - [http://www.slideshare.net/vkuznet/mongodb-at-the-energy-frontier](http://www.slideshare.net/vkuznet/mongodb-at-the-energy-frontier)
  - Project at UCL’s Institute of Neurology: managing and finding heterogeneous data files
3) Document Stores – use case

- National Archives project:
  - [http://www.slideshare.net/AleksDrozdov/from-sql-server-to-mongo-d-bv10](http://www.slideshare.net/AleksDrozdov/from-sql-server-to-mongo-d-bv10)
  - The National Archives is one of the world’s largest records repositories, holding more than 11 million records spanning the Magna Carta to modern government papers – all of which is available to the public
  - Very interesting case of how they ran into massive problems with an RDBMS, and had to change the architecture to cope with the volume and heterogeneity of the data

- Their systems encompassed:
  - Metadata
  - Digital images
  - e-Commerce; inventory; orders
  - User accounts; history
  - User participation

- They have many searches through the catalogue
- They had 2 000 tables and 56 000 attributes in SQL Server!
- [http://discovery.nationalarchives.gov.uk](http://discovery.nationalarchives.gov.uk)
3) Document Stores

RELATIONAL MODEL
Aggregate-Oriented Databases

- [http://martinfowler.com/bliki/AggregateOrientedDatabaseDatabase.html](http://martinfowler.com/bliki/AggregateOrientedDatabaseDatabase.html)
- “There’s a big similarity between [key-value stores, column stores and document stores] - all have a fundamental unit of storage which is a rich structure of closely related data: for key-value stores it’s the value, for document stores it’s the document, and for column-family stores it’s the column family. This group of data is an aggregate. An aggregate makes a lot of sense to an application programmer. If you’re capturing a screenful of information and storing it in a relational database, you have to decompose that information into rows before storing it away. An aggregate makes for a much simpler mapping - which is why many early adopters of NoSQL databases report that it’s an easier programming model.”
4) Graph Stores

- “Odd man out” in the NOSQL group
- Designed for **COMPLEX** data – richer data, a lot of expressive power
- Data model – nodes and edges:
  - Nodes (with properties)
  - Edges are named relationships between nodes (with properties)
- A query on the graph is also known as traversing the graph
- Traversing the relationships is very fast
- Not aggregate-oriented
- Examples:
  - Neo4j, OrientDB, InfiniteGraph, AllegroGraph
- Graph theory:
  - People talk about Codd’s relational model being mature because it was proposed in 1969: 43 years old.
  - Euler’s graph theory was proposed in 1736: 276 years old!
- Semantic Web technologies: RDF, ontologies, triple stores and SPARQL
4) Graph Stores

- **Strengths**
  - Powerful data model
  - Fast
    - For connected data, can be many orders of magnitude faster than RDBMS
  - Good, well-established querying models: Tinkerpop stack (provides a common set of interfaces allowing the various graph computing technologies to work together, which the developer uses as and when needed), SPARQL and Cypher

- **Weaknesses:**
  - Sharding
    - Though they *can* scale reasonably well
    - And for some domains you can shard too!
4) Graph Stores

- Suitable Use Cases:
  - Recommendation engines
  - Business intelligence
  - Social computing
  - Geospatial
  - Systems management
  - Web of things
  - Genealogy
  - Product catalogue
  - Life Sciences and scientific computing (especially bioinformatics)
  - *Connected* data
  - *Hierarchical* data
  - Routing, Dispatch and Location-Based Services
  - Financial services – finance chain, dependencies, risk management, fraud detection etc. For example, if you want to find out how vulnerable a company is to a bit of "bad news" for another company, the directness of the relationship can be a critical calculation. Querying this in several SQL statements takes a lot of code and won't be fast, but a graph store excels at this task.
4) Graph Stores

Property Graph Model

first name: Rose
late name: Tyler

name: the Doctor
age: 907
species: Time Lord

vehicle: tardis
model: Type 40

Courtesy of Neo Technologies
4) Graph Stores – Neo4j

- **Neo4j** – a well-known, popular graph database; developed by Neo Technology
- **Cypher**: powerful declarative query language, able to read and mutate the data, as well as perform various aggregate functions like count and so on.
- **Graph data (see diagram):**
  - People $\rightarrow$ “works_in” Department
  - Department $\rightarrow$ “parent” Department
- **Cypher query:**
  
  ```cypher
  start department = node:Department(name='Widgets')
  match employee -[:works_in]-> sub_department -
  [0..*]:parent]-> department
  return employee
  
  Results: David, Liz, Dan, Mary
  ```
4) Graph Stores – use case
4) Graph Stores – use case

asset management & access control

[Diagram of a graph database showing relationships between user, group, master, customer, account, subscription, product, and sub-product.]
4) Graph Stores – use case

network impact analysis

Courtesy of Neo Technologies
4) Graph Stores

- Less about the volume of data or availability
- More about how your data is related and what calculations you're attempting to perform
- Especially useful when the data set is fundamentally interconnected and non-tabular.
- Use in any case where the relationship between the data is just as important as the data itself.
- Don’t use if your data is simple or tabular.
4) Graph Stores – Triple Stores

- Comments from the web: “Semantic Web and RDF is ‘legacy’ / not interesting etc” - WRONG
- Triple stores:
  - The foundation of many Semantic Web systems
  - Encoded in RDF format
  - Each row is a ‘node – link – node’ structure (subject – predicate - object)
  - They also focus on the ability to join graphs together automatically by matching the identifiers of nodes.
  - By merging two graphs from unrelated systems joins can be performed automatically. For example the first graph stores node A links to B and a second graph links B to C, and the union of these graphs shows a relationship of A to C.
- Examples: Virtuoso, Sesame, Jena
- RDF data is queried via the protocol and query language SPARQL, which incorporates the use of ontologies for inferencing (designed by the W3C RDF Data Access Working Group)
4) Graph Stores – LOD

Linked Data cloud

US DoD (Department of Defense) is using SW technologies...
4) Graph Stores – SW: why should you care?

- **In Life & Health Science** –
  - Genome projects
  - Generating disease models
  - New patient treatments
  - New drugs

- **Financial Services**
- **New trading strategies**

- **Intelligence & security industries**
  - New fraud patterns

- **Space sciences (NASA’s Jet Propulsion Laboratory) and clinical trials: provenance of data and experiments**

- **Netherlands ‘actionable intelligence’ example:**
  - Using an ontology to model a wedding in Afghanistan
  - Can be used to detect whether an event is really a wedding or a cover for a bombing: red-flag it if a component is missing
  - Similar for detecting anomalies in airline passengers to counter terrorism

- **Google Knowledge Graph**
4) Graph Stores – future

- Internet → net of computers
- Word Wide Web → web of documents
- (GGG) Giant Global Graph → graph of metadata

“I called this graph the Semantic Web, but maybe it should have been Giant Global Graph.” - Tim Berners-Lee - 2007
“There is a significant downside - the whole approach works really well when data access is aligned with the aggregates, but what if you want to look at the data in a different way? Order entry naturally stores orders as aggregates, but analyzing product sales cuts across the aggregate structure. The advantage of not using an aggregate structure in the database is that it allows you to slice and dice your data different ways for different audiences.” – Martin Fowler
“This is why aggregate-oriented stores talk so much about MapReduce - which is a programming pattern that's well suited to running on clusters. MapReduce jobs can reorganize the data into different groups for different readers - what many people refer to as materialized views. But it's more work to do this than using the relational model.”

“This is part of the argument for Polyglot Persistence - use aggregate-oriented databases when you are manipulating clear aggregates (especially if you are running on a cluster) and use relational databases (or a graph database) when you want to manipulate that data in different ways.” -- (Martin Fowler)
Polyglot Persistence

- Polyglot Programming and Persistence:
  - Coined by Neal Ford in 2006
  - Expresses the idea that applications should be written in a mix of languages to take advantage of the fact that different languages are suitable for tackling different problems: i.e. defines a hybrid approach to data storage
  - Complex applications combine different types of problems, so picking the right language for the job may be more productive than trying to fit all aspects into a single language --- (recall the “hammer and nails” idea!)
  - So, we want to avoid pushing a square peg into a round hole....
  - Use multiple data storage technologies; particularly when there is no need for same properties of availability, consistency or backup requirements
  - Selection based on the way data is being used by individual applications
  - Can occur both over the enterprise as well as within a single application

- http://martinfowler.com/bliki/PolyglotPersistence.html
Polyglot Persistence

Using a variety of data technologies for different kinds of data, applications and use-cases

```
Polyglot Persistence Model

E-commerce Application

Shopping cart data  User Sessions  Completed Order  Product Catalog  Recommendations

RDBMS

Key-Value  RDMBS  Document  Graph

Traditional Persistence Model

E-commerce Application

Shopping cart data  User Sessions  Completed Order  Product Catalog  Recommendations

RDBMS

Courtesy of Trivadis
```
Polyglot Persistence

- Polyglot persistence usually comes about in one of the following ways:
  - We have an existing relational system, and find that we want to query the data along ‘connectedness’ lines (as an example). We find that this gets harder to code over time, and that it becomes less and less performant in real time. This is the time to look at replacing those parts of the system with a graph store (for this example). It is more cost-effective to just move parts of the system over, as opposed to everything – and far less risky for the business.
  - We are building a new system, and realise that the data fits into distinct categories, for example: huge volumes of simple time-series data that don't need to be inter-related, giant multimedia files, and then something closely knit and highly interconnected. As the data volumes grow and SLAs become more rigorous, it starts to make sense to store data in a place that's optimised for that type of data. In this example, you might use something like Cassandra and Neo4j.
- An example is Telenor, one of the world's 10 largest telecommunications firms, replaced part of a Sybase application with Neo4j for hierarchical queries that needed to run very fast, but kept much of their existing database around.
Pros and Cons of NOSQL vs. RDBMS

- **Pros**
  - Highly performant and scalable (reads and writes) for many use cases
  - No O/R impedance mismatch – applications may become simpler to write
  - Can easily evolve schemas and data structures: no application downtime
  - Can represent semi-structured info
  - Can represent graphs/networks (with performance)
  - Cost-effective with large volumes of data and processing (scale-out with commodity hardware)

- **Cons**
  - Lacks in tool and framework support
  - Few other implementations → potential lock in?
  - Little / no support for ad-hoc queries in many of the products
  - Another/A new database in production to take care of
  - Developer ecosystem: skilling up required / hiring
  - Lack of standards
  - Lack of ‘easy’ interoperability
NOSQL Challenges

- Immaturity
  - NOSQL tools are still young, full of the rough edges that new tools have
  - Not much experience: we don’t know how to use them well
  - No patterns and best practices exist yet

- Organisational Change
  - How will the different data groups in an enterprise react to this new technology
  - IT ecosystem

- Dealing with the eventual consistency paradigm
  - Reaction of different stakeholders to the fact that data could be stale
  - How to enforce rules to sync data across systems
NOSQL Challenges

• Lack of standards:
  ◦ Hard to provide: NOSQL covers a really wide range of models and requirements. So, unified languages for all major areas such as Column, Key/Value, Document and Graph stores will not be available for a long time because it's impossible to cover all areas.
  ◦ The graph domain has some standards, however - Tinkerpop blueprints, Gremlin, SPARQL and Cypher.
  ◦ With the power of Hadoop, however, many projects are working on bridging famous ETL languages such as Pig or Hive to other NOSQL stores
  ◦ Importing data from one system into another (such as from NOSQL store X into MS Excel, for example)
  ◦ Re-using tools across the stores
• No security features per se (such as roles, groups etc) – these need to be handled within the application – work is being done here, though...
Choosing – Design Mindset

- Don’t think of RDBMS vs. NOSQL, think of the “right tool for the right job”
  - Flexibility: Do you benefit from having schema-less design? Do you need the flexibility of exploratory queries?
  - Performance: How much data? How many writes? How many reads? Connectedness of data important?
  - Transactions? Can you live without them?
  - Do I need complex queries and sorts?
  - Am I mapping a complex relationship tree?
  - What is more important: Consistency or Availability?
  - Am I updating records, or just inserting new records?
  - Could I wait a few seconds if a primary node failed?

- Excellent links at

- Some more use cases towards the end of this presentation

- Emergence of “multi-model” databases, such as OrientDB, which supports graph, document, key-value functionality combined with transactions and SQL

“The whole point of seeking alternatives [to RDBMS systems] is that you need to solve a problem that relational databases are a bad fit for.” – Eric Evans, Rackspace
Using NOSQL - Strategies

- Good to use the "Goldilocks Pilot Project Strategy"
  - Not too big, not too small, just the right size
  - Duration
  - Sponsorship
  - Importance
  - Skills
  - Mentorship
- Remember that a flexible schema does not mean there is an excuse for a badly-designed application!
- When using NOSQL, and when one doesn’t know about the schema beforehand, use a “schema strategy” to ensure you adhere to a good design throughout
Data Analytics – a few uses...

- eCommerce optimisation (which basically means "how do I get more ££ from the customers")
- Targeted advertisements
- Financial services, such as risk-modelling
- Detecting anomalies
- Automated metadata generation – generating graphs. Graph distance algorithms to determine how related two entities are, and updated in real-time
- Recommendation systems:
  - Recently heard a talk about Amazon's recommendation system and how 'flat' it was - the fact that, within one household, there may be multiple user types. Thus, it would be cleverer if Amazon could distinguish between these, so the parents, browsing at night, do not get recommendations about colouring books because the kids looked at those during the day...
- System monitoring. An example would be detecting whether there are emerging problems with an oil rig.
Data Analytics

• Modes:
  ◦ **Real-time** – transactional processing; showing a ‘fairly suitable’ advert to a customer browsing your web site.
  ◦ **Offline** – batch processing; improvements and refinements to the model over time, so that in the future, you can show the ‘best’ advert to a customer browsing your web site.
# Data Analytics – Evolutionary roadmap?

<table>
<thead>
<tr>
<th>Google</th>
<th>Open-source project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004 – GFS &amp; MapReduce</td>
<td>2006 – Hadoop</td>
<td>Batch programs</td>
</tr>
<tr>
<td>2005 – Sawzall</td>
<td>2008 – Pig and Hive</td>
<td>Batch queries</td>
</tr>
<tr>
<td>2006 – BigTable</td>
<td>2008 – HBase</td>
<td>Online column</td>
</tr>
<tr>
<td>2010 – Dremel / F1</td>
<td>2012 – Impala (and Drill)</td>
<td>Online queries</td>
</tr>
<tr>
<td>2012 – Spanner</td>
<td>?</td>
<td>Transactions etc</td>
</tr>
</tbody>
</table>
Data Analytics – Hadoop

- Synonymous with NoSQL
- Apache – open source
- A combination of:
  - A distributed file manager – HDFS (Hadoop Distributed File System)
  - A parallel programming framework known as MapReduce (original paper from Google)
- Usually runs on a cluster of commodity nodes
- Batch-oriented: not suitable for ad-hoc querying!
- Integration with many data stores – and not only the NOSQL ones...
- Two thoughts regarding Hadoop:
  - “I will spread your data over many servers and keep it safe”
  - “I will facilitate a new idea that you should send the work to the data and not the other way around”
Data Analytics - Hadoop

- Great for “boil the ocean”-type processing and analytics:
  - **Big Data Analytics:**
    - Running queries over large semi-structured datasets
    - Makes filtering and aggregation-type jobs very easy
  - **Big Data Processing:**
    - Efficient and effective way to write data pipelines
    - Easy way to parallelise computationally complex queries
    - Scales nicely with the amount of data and cluster size

- Not great for the following:
  - Real-time analytics or processing
    - Even small queries take time
    - Can't build into real-time data flows
  - Algorithms which are difficult to parallelise
    - Almost anything can be expressed in a number of MR steps
    - Almost always, MR is SUB-OPTIMAL! This is the case even if it is easier to abstract

![Hadoop Logo](image)
Data Analytics – Hadoop tooling

- Writing MapReduce jobs in Hadoop natively is very tricky
- Instead, it is recommended to use tools to do this for you:
  - **Hive**
    - Provides a SQL interface to Hadoop’s MapReduce
    - Thus, MapReduce jobs are written in a SQL-like language
  - **Pig**
    - Language is also very simple and high-level
    - UDFs can be written in Java, Python or Javascript – useful
  - Other tools: Cascading, Crunch etc.
Data Analytics - Streaming data

- Collecting and processing streaming data
  - **Storm** (Twitter)
    - Free and open source distributed real-time computation system
    - Reliably processes unbounded streams of data in real-time
    - STORM topology consumes streams of data and processes those streams in arbitrarily complex ways
  - Other tools include **Kafka** (LinkedIn: messaging) and **Flume**:
    - Different models but the general idea is the same
    - Data collected from many sources
    - The data is aggregated
    - The data is fed to a database, a system like Hadoop, or other clients.
Data Analytics – Machine Learning

- **Apache Mahout:**
  - A library of ML (Machine Learning) algorithms

- **Uses:**
  - Log analysis
  - Data warehouses
  - NLP (Natural Language Processing)
  - Search indexing (creation of indices offline)
  - Machine learning
Data Analytics - Impala

- Cloudera’s Impala recently released:
  - Real-time query engine
  - Interactive queries
  - Complements MapReduce
  - Runs on HBase and HDFS.
  - Large-scale, ad-hoc querying of data possible
  - Especially suited to data exploration.
  - SQL predicates like SELECT, JOIN and aggregate functions
NOSQL – general use cases

- Volume
- Massive write performance
- Fast key-value access
- Flexibility: schema, data types, applications
- Write availability
- No single point of failure
- Generally available parallel computing (especially with MapReduce)
NOSQL – specific use cases

- Managing large streams of non-transactional data: logs, clickstreams, etc.
- Fast response times under all loads.
- Avoiding heavy joins for when the query load for complex joins become too large for a RDBMS.
- Soft real-time systems where low latency is critical. Games are one example.
- Applications where a wide variety of different write, read, query, and consistency patterns need to be supported.
- Load balance to accommodate data and usage concentrations and to help keep microprocessors busy.
- Real-time inserts, updates, and queries.
- Hierarchical data like threaded discussions and parts explosion.
- Dynamic table creation.
- Two tier applications where low latency data is made available through a fast NOSQL interface, but the data itself can be calculated and updated by high latency Hadoop applications or other low priority apps.
- Slicing off part of service that may need better performance/scalability onto its own system. For example, user logins may need to be high performance and this feature could use a dedicated service to meet those goals.
- Caching - a high performance caching tier for web sites and other applications (such as the CERN LHC project).
- Voting
- Real-time page view counters.
- User registration, profile, and session data.
- Document, catalogue management and content management systems (CMS). These are facilitated by the ability to store complex documents as a whole rather than organised as relational tables. Similar logic applies to inventory, shopping carts, and other structured data types (such as the National Archives project).
NOSQL – specific use cases

- Archiving. Storing a large continual stream of data that is still accessible online. Document-oriented databases with a flexible schema that can handle schema changes over time.
- Analytics. Use MapReduce, Hive, or Pig to perform analytical queries and scale-out systems that support high write loads.
- Working with heterogeneous types of data - for example, different media types at a generic level.
- Federal law enforcement agencies tracking Americans in real time using credit cards, loyalty cards and travel reservations.
- Fraud detection by comparing transactions to known patterns in real-time.
- Helping diagnose the typology of tumours by integrating the history of every patient.
- In-memory database for high update situations, like a web site that displays everyone's "last active" time (for chat maybe).
- Handling lower-frequency multi-partition queries using materialised views while continuing to process high-frequency streaming data.
- Running calculations on cached data, using a program-friendly interface, without have to go through an ORM.
- To keep querying fast, values can be rolled-up into different time slices.
- Computing the intersection of two massive sets, where a join would be too slow.
- A timeline a la Twitter.
Links

- Meetups: www.meetup.com
- Skillsmatter: http://skillsmatter.com/go/nosql (podcasts)
- http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/ - good one
- http://www.opensource-it.com/open_source_nosql_databases
- Free online training - MongoDB - http://education.10gen.com/
- Couchbase: http://www.couchbase.com/library
- http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- http://neo4j.org
- http://cassandra.apache.org/
- http://basho.com/
- http://datasciencelondon.org/
- http://www.infoq.com
- http://www.dzone.com/mz/nosql
Thank you for your attention!