Overview of NoSQL in Big Data Management with a closer look on graph databases

Bernd
Institute for Clarity in Documentation
webmaster@marysville-ohio.com

Florian The Thørväld Group
larst@affiliation.org

Lin Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Patrick Saeuerl
TU Vienna
Bacc Studium
e1125492@student.tuwien.ac.at

ABSTRACT

The Abstract that we are going to write.

1. INTRODUCTION

In data growth in the last years was intense. Social Systems like Facebook, LinkedIn and Twitter produce more and more data each day. The same applies to the increased use of mobile devices and the Internet of Things. In 2012 over 2.8 zettabytes of data existed. This new amount of data and the rate at which it is generated leads to new challenges for companies and their storage models. The challenges include how to store the data and how to analyze the data. We will have a closer look on the Big Data movement and the new data-storage systems that developed in the last years, broadly named NoSQL systems. Because graphs are a natural structure for social media, we will introduce some of the most used graph databases. As the topic of big data is huge, we will not cover analyzation techniques like Map-Reduce and Bulk-Synchronous-Processing, although we recommend reading them up.

2. BIG DATA

The term Big Data was first used in 2001 by the consulting company Mckinsey [ref overview big data]. There is no general definition about the term Big Data, but there is a common aggrement on some characteristics of Big Data. In general, the amount of data is too much as that it could be managed by traditional IT systems.

In 2010, Apache Hadoop defined big data as âĂIJdatasets which could not be captured, managed, and processed by general computers within an acceptable scope". This means that traditional relational database management systems have reached their limits.

A basic characterization of Big data was done by IDC. This characterization are the "4Vs":

- Variety - describing the different data and data structures
- Veolocity - the speed of data creation
- Volume - the amount of data
- Value - the worth derived from the data

The following picture from [big data - related technologies] illustrates very well the Velocity, Volume and Value of Big Data.

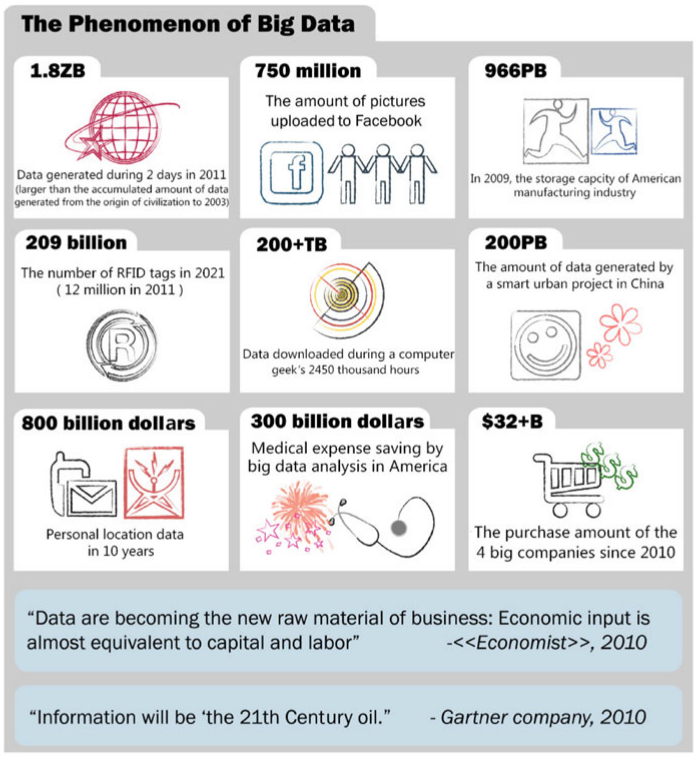
The Phenomenon of Big Data
1.8ZB
Data generated during 2 days in 2011 (larger than the accumulated amount of data generated from the origin of civilization to 2003)
750 million
The amount of pictures uploaded to Facebook
966PB
In 2009, the storage capcity of American manufacturing industry
209 billion
The number of RFID tags in 2021 (12 million in 2011)
200+TB
Data downloaded during a computer geek's 2450 thousand hours
200PB
The amount of data generated by a smart urban project in China
800 billion dollars
Personal location data in 10 years
300 billion dollars
Medical expense saving by big data analysis in America
$32+B
The purchase amount of the 4 big companies since 2010
"Data are becoming the new raw material of business: Economic input is almost equivalent to capital and labor" -<<Economist>>, 2010
"Information will be 'the 21th Century oil." - Gartner company, 2010

Figure 1: Phenomenon of Big Data

Traditionally, most of the data was stored in relational database management systems (RDBMS). RDBMS usually store structured data and are therefor not good in handling the point of variety. Because of the structured way and the integrity constraints in RDBMS, they do not scale very well horizontally which makes them not good for storing large volumes of data. The NoSQL ("Not only SQL") family of databases tackle those challenges.

3. NOSQL OVERVIEW

"NoSQL" stands for "Not only SQL" and describes a broad family of database management systems. They tackle a variety of problems that were introduced by Big Data - mainly variety and volumne. The aspect of volumn is covered by enabling the databases to scale horizontally and reducing constraints by different approach to the CAP Theorem compared to RDBMS. The aspect of variety is covered by the wide range of different NoSQL systems. We will look at the CAP Theorem first as with it we can explain the horizontal scalability of NoSQL systems.

3.1 CAP-Theorem

The CAP or Brewer Theorem[**?**] was formulated in 2000. The theorem states, that it is impossible for any data sharing system, to guarantee the following three properties all the time

- **Consistency**
  this means that all clients see the same data. In an environment with a huge amount of data, this is hard to achieve as it might be stored in different parts of the cluster
- **Availability**
  this means that the service is basically available. To achieve this availability, the system must have some kind of fault tolerance to handle situations like servers going down. In our case with the amounts of data that Big Data is concernd with, multiple machines will be used for the system, therefor availability is needed.
- **Partition tolerance**
  this means that the data is partitioned to several parts of the database. This enables horizontal scalability and also fault tolerance.

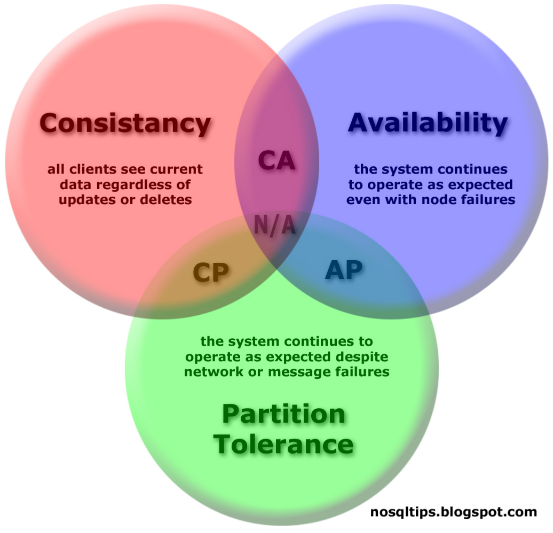
Consistancy
all clients see current data regardless of updates or deletes
CA
Availability
the system continues to operate as expected even with node failures
N/A
CP
AP
the system continues to operate as expected despite network or message failures
Partition Tolerance
nosqltips.blogspot.com

Figure 2: CAP Theorem, Source: blog.nosqltips.com

Big Data is too big to be handeld on a single machine or a small cluster. Therefor, partition tolerance is needed. Traditional RDBMS focus on the properties C and A, NoSQL systems vary if they focus on CP or on AP, but basically the are horizontal scaleable. We will now see what kind of NoSQL databases exist.

3.2 Types of NoSQL systems

There are several different classifications of "NoSQL" databases. Most of them contain the following classes: Key-Value stores, Column-Stores, Document-Stores und Graph Databases. Other types like Object-Oriented Databases and XML-Databases exist but they will not be covered here. We have a short look on the basic characterists of those types.

3.2.1 Key-Value Stores

The data model is simpel. Each key is unique and it has an associated value. The storage has no knowledge about the value that is stored. Ad-Hoc query features and joins are usually ommited. They have very good scaling due to their easy lookup and storage model. A popular example is Amazon Dynamo DB, another ones are Apache Cassandra and BerkeleyDB.

3.2.2 Column-Stores

In RDBMS, the important unit is a row. Column-Stores, as the name implies, lay their focus on columns instead of rows. The columns are groupd together to a column-family. A row consists out of column-families which need to be predefined. Which columns are in a column-family is open. The columns that belong to a column family should relate together. For example, Address would be a column family containing the columns that represent the whole address. Columns of a column family are stored together, which makes it easier to retrieve those columns and process them together.

To achieve the performance needed, those databases normally allow temporary inconsistency. They usually store updates to data as a new versio instead of updating the old one.

Popular examples of this type of storage are Goolge BigTable and Apache HBase. Apache Cassandra also is in this section as it is a hybrid system having prominent features of key-value and column-stores.

[Cite NoSQL systems for big data]

3.2.3 Document Stores

This kind of NoSQL Databases consider documents as their unit of information. A document can be seen like a record in an RDBMS, but they are schemaless. This gives them more flexibility and they do not include null values. Therefore those systems are very well suited systems that have unstructured data, so they tackle the issue of variety in Big Data. The documents are normally stored in JSON or XML. Usually, document stores are fully searchable. A typical element of a document store looks like this:

```
{
        FirstName: "Jonathan",
        Address: "15 Wanamassa Point Road",
        Children: [
                {Name: "Michael", Age: 10},
                {Name: "Jennifer", Age: 8},
                {Name: "Samantha", Age: 5},
                {Name: "Elena", Age: 2}
        ]
}
```

Prominent members for document stores are MongoDB and CouchDB. MongoDB also offers support for Hadoop, therefor suiting good for Map/Reduce applications. They both also provide good horizontal scaling.

An example of a document store in the Big Data scenario was done by[Insert name of Crime Paper]. They have utilized MongoDB to caputre unstructered data from various ressources likes blogs, social media and RSS feed. Afterwards, they classified and analyzed the data to predict areas which have high crime rate.

3.3 Graph Databases

The dominant feature of graph databases is, that they represent their data in an actual graph. The importance is here on the relationship between the data. The nodes and edges can have properties, describing the node or the relationship between the nodes. By the way the data is stored internally, they perform much better than relational databases on graphs, although there has been research on how relational databases could handle graphs better [add ref]. From all the data-models in the NoSQL family, the data model of graph-databases is the most complex and expressive one.

When thinking of social media like Facebook, Xing and LinkedIn, one can see very easily how it would feel natural to model the data as graphs. User in those systems represent nodes and the edges between them if they are friends, for example.

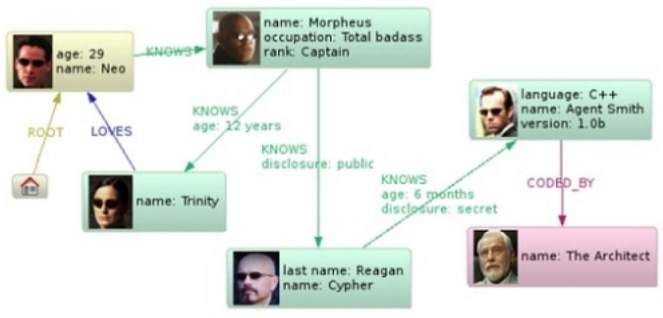

Figure 3: a social graph, Source: /www.infoq.com/articles/graph-nosql-neo4j

Besides representing social-graphs, they have also use in scientific computing. A request for databases optimized for graphs dates back to 1995 as they can represent genomes easier [reference to genome]. Another nice example for the use of graph databases is in the support of cloud management. [Soundararajan and Kakaraddi, needs ref] used a graph database to support system admins with managing their cloud environment by mapping their virtual infrastructure to a graph.

[...nosql vs rdbms paper] used Neo4J, a popular graph database, to implement a large-scale healthcare system. They compared it's performance to the implementation with a MySQL version of the system. An interesting finding of theirs was, that the performance of Neo4J degrades with respect to the degree of a node, while in the MySQL case it degrades with respect to the overall size.

As those databases have the most expressive data-system, we will have a closer on look on some of the available graph databases and compare them.

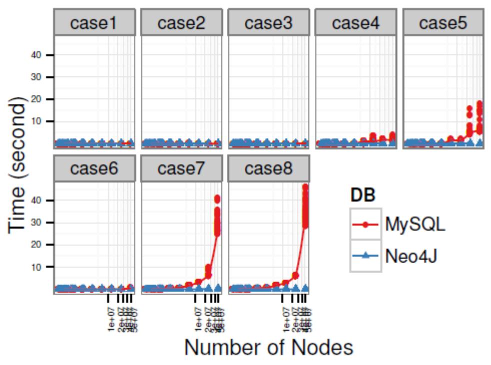

Figure 4: Data Size vs. Query Processing Time

4. GRAPH DATABASE COMPARISION

We will now look on three graph databases, Neo4J, HyperGraphDB and Dex a comparision between them done by [ref to choosing between].

4.1 Neo4J

Neo4J is written in Java and was released in 2007. It is fully ACID compliant. The nodes and edges in the graph can have properties. Edges also have a type.

To access data, one has a wide variaty of access methods. It is possible to query data via it's native Java API, or to use SPARQL (normally used for RDF systems), Gremlin or Cypher. [...]. An example for a cypher query finding all connections between a and b with three hops or less.

```
START a=node:indexName(name="A")
MATCH (a)-[*1..3]-(b)
WHERE b.node_type = "B"
RETURN distinct b;
```

4.2 HypergraphDB

HypergraphDB uses hypergraphs as a storage model. In hypergraphs, more edges correspond to a single hyperedge. Those hyperedges connect orderd sets of nodes. The HypergraphDB is built on top of the BerkeleyDB, a Key-Value Store.

For accessing the data a simple, but efficient querying API where data is loaded on demand with lazy join, bi-directional (whenever possible) cursors exists.

4.3 DEX / Sparksee

DEX storage model is a labeled and attributed multigraph. The edges between the nodes can be directed or undirected. In this kind of graph, every edge and node has a label representing the type. Each node and edge acn also contain various attribtues. Because of the multigraph structure, multiple edges are allowed between two nodes.

The core engine of DEX was written in C++. A Java library with complete API access was also created. The name was changed in February 2014 to Sparksee.

DEX does not offer a specific query language but instead relies on basic graph operations. The result of those operations are again graphs.

4.4 Performance

[Ref] used datasets representing road-networks that were used 9th DIMACS Implementation Challenge - Shortest Paths. Interestingly, when initially loading the data, the HyperGraphDB timed out after 24-hours on one of the test-instances.

They compared the performance on two basic graph search-operations: Depth-First-Search and Breadth-First-Search, running internally and externally. For external algorithmns, they relied on adjacency functions of the APIs.

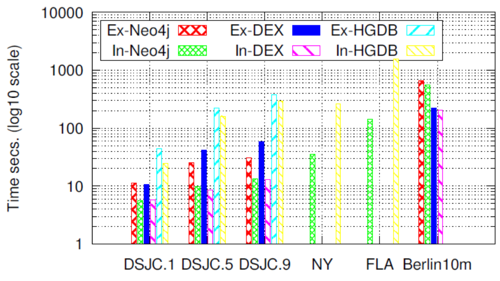

Figure 5: BFS performance, non-shown bars indicate that the corresponding engine timed out after 1 hour

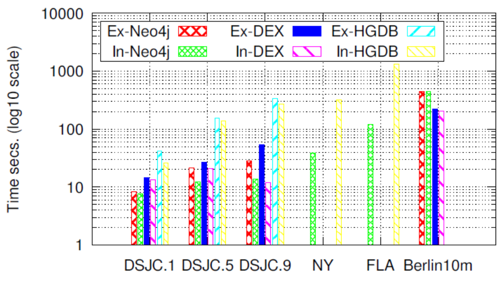

Figure 6: dfs performance, non-shown bars indicate that the corresponding engine timed out after 1 hour

They also compared the performance for the following two use-cases: Densest-Subgraph and Graph-Summarization. Densest-Subgraph tries to look for a subgraph with the highest density. Graph-Summarization produces an aggregated graph of the initial graph where each node corresponds to a set of nodes and a single edge represent the set of edges between two node sets.

The core output is that Neo4J and DEX both outperform Hyper-GraphDB. Also Neo4J seems to have better performance on sparse graphs and DEX works better on dense graphs. This correlates with the finding of [nosql -rdbms stuff] where Neo4Js performance drops with the degree of nodes.

5. CONCLUSIONS

We have seen how Big Data changed the landscape for data-storage systems and how it gave birth to a whole new family of storage systems, broaldy desribed as "NoSQL"-Systems. We seen which properties they have in general and how they can be classified. At last, we took a closer look on Graph Database Management systems as graphs have widespdread use. A closer comparision between

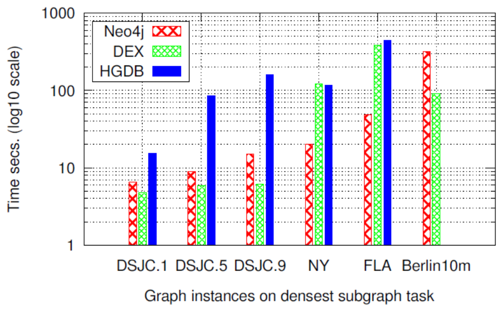

Figure 7: Denses Subgraph, non-shown bars indicate that the corresponding engine timed out after 1 hour

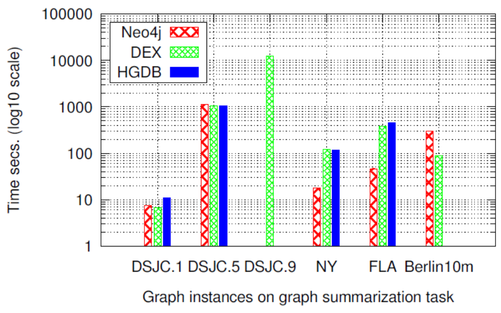

Figure 8: Graph Summarization performance, non-shown bars indicate that the corresponding engine timed out after 1 hour

Neo4J, DEX and Hypergraph showed us, that in this specialised field the performances can vary and one has to adept to the given situation. We hope to see the spread of Graph Databases and the emerge of standardized bench-marks.

6. REFERENCES