New database architectures: steps toward Big Data processing

J. Pokorný
Faculty of Mathematics and Physics, Charles University, Prague
Czech Republic
Where Prague is?
Prague
Faculty of Mathematics and Physics
Big Data Movement

**Something from Big Data Statistics**
- M. Lynch (1998): 80-90% of (business) data is unstructured
- D. Feinberg (2011): worldwide information volume is growing at a minimum rate of 59% annually.
- eBay study (2012): The volume of business data worldwide, across all companies, doubles every 1.2 years

**Problem:** our inability to utilize vast amounts of information and their users effectively. It concerns:
- data storage and processing at low-level (different formats, transactions, consistency, …)
- analytical tools on higher levels (difficulties with data mining algorithms).
Big Data Movement

- User options:
  - traditional parallel database systems ("shared-nothing"),
  - distributed file systems (GFS, HDFS) + Hadoop technologies,
  - key-value data stores (so called NoSQL databases),
  - new architectures.

- Applications:
  - both transactional and analytical,
  - they require usually different architectures

- Goal of the talk:
  - discuss possibilities of NoSQL technologies, i.e., their pros and cons, in this movement, present some other alternatives, trends
Content

- Big Data movement
- Database architectures
- NoSQL databases
  - data models
  - architectures
  - representatives
- Big Data
  - Big Data characteristics
  - Big Data processing
- Trends
- Conclusions
## Database architectures

The five-layered DBMS mapping hierarchy (Härder and Reuter, 1983)

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Objects</th>
<th>Auxiliary mapping data</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>non-procedural access or algebraic access</td>
<td>tables, views, rows</td>
</tr>
<tr>
<td>L4</td>
<td>record-oriented navigational approach</td>
<td>records, sets, hierarchies, networks</td>
</tr>
<tr>
<td>L3</td>
<td>records and access path management</td>
<td>physical records, access paths</td>
</tr>
<tr>
<td>L2</td>
<td>propagation control</td>
<td>segments, pages</td>
</tr>
<tr>
<td>L1</td>
<td>file management</td>
<td>files, blocks</td>
</tr>
</tbody>
</table>
Database architectures

Significant feature of usual parallel DBMSs: only one way how to access the system’s architecture

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Objects</th>
<th>Auxiliary mapping data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL</strong></td>
<td>tables, views, rows</td>
<td>logical schema description</td>
</tr>
<tr>
<td>record-oriented navigational approach</td>
<td>records, sets, hierarchies, networks</td>
<td>logical and physical schema description</td>
</tr>
<tr>
<td>records and access path management</td>
<td>physical records, access paths</td>
<td>free space tables, DB-key, translation tables</td>
</tr>
<tr>
<td>propagation control</td>
<td>segments, pages</td>
<td>buffers, page tables</td>
</tr>
<tr>
<td>file management</td>
<td>files, blocks</td>
<td>directories</td>
</tr>
</tbody>
</table>
Scalability of DBMSs in context of Big Data

- traditional **scaling up** (adding new expensive big servers)
  - requires higher level of skills
  - is not reliable in some cases
- new architectural principle: **scaling out** (or horizontal scaling) based on data partitioning, i.e. dividing the database across many (inexpensive) machines
  - technique: **data sharding**, i.e. horizontal partitioning of data (e.g., hash or range partitioning)
  - compare: manual or user-oriented data distribution (DDBSs) vs. automatic data sharding (clouds, web DB, NoSQL DB)

**Consequences of scaling out:**
- scales well for both reads and writes
- manage parallel access in the application
  - scaling out is not transparent, application needs to be partition-aware
## New DB architectures

The three-layered Hadoop software stack

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Data processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5 non-procedural access</td>
<td>HiveQL/PigLatin/Jaql</td>
</tr>
<tr>
<td>L2-L4 record-oriented, navigational approach</td>
<td>Hadoop MapReduce Dataflow Layer</td>
</tr>
<tr>
<td></td>
<td>HBase Key-Value Store</td>
</tr>
<tr>
<td>L2-L4 records and access path management</td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>L2-L4 propagation control</td>
<td></td>
</tr>
<tr>
<td>L1 file management</td>
<td></td>
</tr>
</tbody>
</table>
New DB architectures

The three-layered Hadoop software stack – data access through more layers

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Data processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5 non-procedural access</td>
<td>HiveQL/Pig/Jaql</td>
</tr>
<tr>
<td>L2-L4 record-oriented, navigational approach</td>
<td>Hadoop MapReduce Dataflow Layer</td>
</tr>
<tr>
<td></td>
<td>M/R jobs Get/Put ops</td>
</tr>
<tr>
<td></td>
<td>HBase Key-Value Store</td>
</tr>
<tr>
<td></td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>L1 file management</td>
<td></td>
</tr>
</tbody>
</table>

Griffith Uni, Feb 13, 2014
Relaxing ACID properties

ACID properties in RDBMS
- Atomicity: All or nothing.
- Consistency: Consistent state of data and transactions.
- Isolation: Transactions are isolated from each other.
- Durability: When the transaction is committed, state will be durable.

ACID is hard to achieve, moreover, it is not always required, particularly C (e.g., for blogs, status updates, product listings, etc.)
Relaxing ACID properties

Alternative properties in large networks:

- **Availability**
  - Traditionally, thought of as the server/process available 99.999% of time
  - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning

- **(Network) partition tolerance**
  - ensures that write and read operations are redirected to available replicas when segments of the network become disconnected
Relaxing ACID properties

- **Eventual Consistency**
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent.
  - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service.

- **New conceptual proposal replacing ACID: BASE properties**
  - (Basically Available, Soft state, Eventual consistency)
  - Soft state: copies of a data item may be inconsistent.
  - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item.
  - Basically Available – possibilities of faults but not a fault of the whole system.
CAP Theorem

- Suppose three properties of a system
  - Consistency
  - Availability
  - Partition tolerance

- Brewer’s CAP “Theorem”: for any system sharing data it is impossible to guarantee simultaneously all of these three properties

- Very large systems will partition at some point, i.e. supposing P, it is necessary to decide between C and A.
CAP Theorem

- Traditional DBMS prefer C over A and P
- Most Web applications choose A (except in specific applications such as order processing)

Consequences:
- relaxing C makes replication easy, facilitates fault tolerance,
- relaxing A reduces (or eliminates) need for distributed concurrency control.

Examples:
- ATMs chose A with weak C
- Datacenters chose A and C (probability of a network failure is small)
NoSQL databases

- The name stands for Not Only SQL

Common features:
- non-relational
- usually do not require a fixed table schema
  - more flexible data model
- horizontal scalable
  - scalability and performance advantages
- relax one or more of the ACID properties (see CAP theorem)
  - mostly AP systems
- replication support
- easy API (if SQL, then only its very restricted variant)
- mostly open source
NoSQL databases

- Do not fully support usual relational features:
  - no join operations (except within partitions),
  - no referential integrity constraints across partitions.

- Use cases:
  - massive write performance,
  - fast key value look ups,
  - no single point of failure,
  - fast prototyping and development,
  - out of the box scalability,
  - easy maintenance.
Categories of NoSQL databases

- key-value stores
- column-oriented
- document-based
- graph databases (e.g. neo4j, InfoGrid - primarily focus on social network analysis)
- XML databases (e.g. myXMLDB, Tamino, Sedna)
- RDF databases (support the W3C RDF standard internally)
Categories of NoSQL databases

- key-value stores
- column-oriented
- document-based
- graph databases (e.g. neo4j, InfoGrid - primarily focus on social network analysis)
- XML databases (e.g. myXMLDB, Tamino, Sedna)
- RDF databases (support the W3C RDF standard internally)
Potential weaknesses: as the volume of data increases, maintaining unique values as keys may become more difficult.
Column-oriented*

- store data in column order
- allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added
  - storing principle: big hashed distributed tables
  - properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application

* Better: extendible records
Column-oriented

- Example: BigTable
  - indexed by row key, column key and timestamp. i.e. (row: string, column: string, time: int64) → String.
  - rows are ordered in lexicographic order by row key.
  - row range for a table is dynamically partitioned, each row range is called a tablet.
  - columns and column families
  - access to columns: syntax is family:qualifier
## Column-oriented

A table representation of a row in BigTable

Example: Grandchildren:GCH1 returns (Claire 7)
Column-oriented

Example: Cassandra

- **keyspace**: Usually the name of the application; e.g., 'Twitter', 'Wordpress'.
- **column**: a tuple with name, value and time stamp
- **column family**: structure containing an unlimited number of rows (key + set of columns)
- **key**: a name of row
- **super column**: a named set of columns
- **super column family**: a set of columns and super columns
Document-based

- based on JSON format: lists, maps, dates, Boolean with nesting are supported
- Really: indexed semistructured documents
- Example: Mongo
  - `{Name:"Jaroslav",
    Address:"Malostranské nám. 25, 118 00 Praha 1",
  }`
Typical NoSQL API

- reduced access: CRUD operations – create, read, update, delete

Examples:

- `get(key)` -- extract the value given a key
- `put(key, value)` -- create or update the value given its key
- `delete(key)` -- remove the key and its associated value
- `execute(key, operation, parameters)` -- invoke an operation to the value (given its key) which is a special data structure (e.g., List, Set, Map, ..., etc.).
- `multi-get(key1, key2, ..., keyN)`
### Representatives of NoSQL databases key-valued

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riak</td>
<td>Basho Technologies</td>
<td>set of couples (key, value)</td>
<td>put, get, post, and delete functions</td>
</tr>
<tr>
<td>Redis</td>
<td>S. Sanfilippo</td>
<td>set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value</td>
<td>primitive operations for each value type</td>
</tr>
<tr>
<td>Memcached</td>
<td>B. Fitzpatrick</td>
<td>set of couples (key, value) in associative array</td>
<td>get, put and remove operations</td>
</tr>
<tr>
<td>Dynamo</td>
<td>Amazon</td>
<td>binary objects identified by unique key</td>
<td>simple get operation and put in a context</td>
</tr>
<tr>
<td>Voldemort</td>
<td>Linkeld</td>
<td>like SimpleDB</td>
<td>similar to Dynamo</td>
</tr>
</tbody>
</table>
Representatives of NoSQL databases

### Key-valued

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riak</td>
<td>Basho Technologies</td>
<td>set of couples (key, value)</td>
<td>put, get, post, and delete functions</td>
</tr>
<tr>
<td>Redis</td>
<td>S. Sanfilippo</td>
<td>set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value</td>
<td>primitive operations for each value type</td>
</tr>
<tr>
<td>Memcached</td>
<td>B. Fitzpatrick</td>
<td>set of couples (key, value) in associative array</td>
<td>get, put and remove operations</td>
</tr>
<tr>
<td>Dynamo</td>
<td>Amazon</td>
<td>binary objects identified by unique key</td>
<td>simple get operation and put in a context</td>
</tr>
<tr>
<td>Voldemort</td>
<td>Linkeld</td>
<td>like Dynamo</td>
<td>similar to Dynamo</td>
</tr>
</tbody>
</table>
## Representatives of NoSQL databases

### column-oriented

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigTable</td>
<td>Google</td>
<td>set of triples (key, column value, timestamp)</td>
<td>selection (by combination of row, column, and time stamp ranges)</td>
</tr>
<tr>
<td>HBase</td>
<td>Apache</td>
<td>columns corresponding to a key (a BigTable clone)</td>
<td>JRUBY IRB-based shell (similar to SQL)</td>
</tr>
<tr>
<td>Hypertable</td>
<td>Hypertable</td>
<td>like BigTable</td>
<td>HQL (Hypertext Query Language)</td>
</tr>
<tr>
<td>CASSANDRA</td>
<td>Apache (originally Facebook)</td>
<td>columns, groups of columns corresponding to a key (supercolumns)</td>
<td>simple selections on key, range queries, column or columns ranges</td>
</tr>
<tr>
<td>PNUTS</td>
<td>Yahoo</td>
<td>(hashed or ordered) tables, typed arrays, flexible schema</td>
<td>selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)</td>
</tr>
</tbody>
</table>
Representatives of NoSQL databases

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigTable</td>
<td>Google</td>
<td>set of triples (key, column value, timestamp)</td>
<td>selection (by combination of row, column, and timestamp ranges)</td>
</tr>
<tr>
<td>HBase</td>
<td>Apache</td>
<td>columns corresponding to a key (a BigTable clone)</td>
<td>JRUBY IRB-based shell (similar to SQL)</td>
</tr>
<tr>
<td>Hypertable</td>
<td>Hypertable</td>
<td>like BigTable</td>
<td>HQL (Hypertext Query Language)</td>
</tr>
<tr>
<td>CASSANDRA</td>
<td>Apache (originally Facebook)</td>
<td>columns, groups of columns corresponding to a key (supercolumns)</td>
<td>simple selections on key, range queries, column or columns ranges; CQL</td>
</tr>
<tr>
<td>PNUTS</td>
<td>Yahoo</td>
<td>(hashed or ordered) tables, typed arrays, flexible schema</td>
<td>selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)</td>
</tr>
</tbody>
</table>

Key-player on the NoSQL market
## Representatives of NoSQL databases

### column-oriented

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigTable</td>
<td>Google</td>
<td>set of couples (key, {value})</td>
<td>selection (by combination of row, column, and timestamp ranges)</td>
</tr>
</tbody>
</table>

**Example: Google development:**

|------|------|------|------|------|------|------|------|------|------|------|

- Google
- MapReduce
- BigTable

- Dremel
- Percolator
- Tenzing
- MegaStore
- Spanner
- F1

- query API
- transactions
- SQL DBMS built on Spanner
- scalable DBMS
### Representatives of NoSQL databases

**document-based**

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Data model</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>10gen</td>
<td>object-structured documents stored in collections; each object has a primary key called ObjectId</td>
<td>manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)</td>
</tr>
<tr>
<td>CouchDB(^1)</td>
<td>Apach</td>
<td>document as a list of named (structured) items (JSON document)</td>
<td>views via Javascript and MapReduce; UnQL</td>
</tr>
</tbody>
</table>

\(^1\) similar but not the same as Couchbase from CouchOne, Inc.
Representatives of NoSQL databases

document-based

<table>
<thead>
<tr>
<th>Name</th>
<th>Producer</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>10gen</td>
<td>object-structured documents stored in collections; each object has a primary key called ObjectId</td>
</tr>
<tr>
<td>CouchDB(^1)</td>
<td>Apach</td>
<td>document as a list of named (structured) items (JSON document)</td>
</tr>
</tbody>
</table>

\(^1\) similar but not the same as Couchbase from CouchOne, Inc.
Consistency in DBMSs

Types of consistency:

- Strict Consistency – RDBMS.
- Tunable Consistency – Cassandra.
- Eventual Consistency – Dynamo, SimpleDB.

- E.g. SimpleDB:
  - will acknowledge a client’s write before the write has propagated to all replicas.
  - offers even a range of consistency options.

- Configurable Consistency - Oracle NoSQL DB (CP or AP)
Categories of DBMS by CAP

CA: RDBMS, Vertica

CP: Redis, BigTable, Hypertable, HBase, MongoDB

AP: Dynamo, Voldemort, SimpleDB, Cassandra, PNUTS, CouchDB

Vertica – column-oriented relational DBMS in massive parallel system with linear scalability
Usability of NoSQL DBMSs

- parts of data-intensive cloud applications (mainly Web applications).
  - Web entertainment applications,
  - indexing a large number of documents,
  - serving pages on high-traffic websites,
  - delivering streaming media,
  - data as typically occurs in social networking applications,
- Examples:
  - Digg's 3 TB for green badges (markers that indicate stories upvoted by others in a social network)
  - Facebook's 50 TB for inbox search
  - Google uses BigTable in over 60 applications (e.g. Earth, Orkut)
Usability of NoSQL DBMSs

- applications do not requiring transactional semantics
  - address books, blogs, or content management systems
  - analyzing high-volume, real time data (e.g., Web-site click streams)
- enforcing schemas and row-level locking as in RDBMS — unnecessarily over-complicate these applications
- mobile computing makes transactions at large scale technically infeasible
- absence of ACID allows significant acceleration and decentralization of NoSQL databases
Usability of NoSQL DBMSs

- Unusual and often inappropriate phenomena in NoSQL approaches:
  - have little or no use for data modeling
  - developers generally do not create a logical model
  - query driven database design
  - unconstrained data
  - different behavior in different applications
  - no query language standard
  - complicated migration from one such system to another.

- http://www.nosql-database.org/ lists currently 150 NoSQL databases (including OO, Graph, XML, …)
Usability of NoSQL DBMSs

Examples of problems with NoSQL
- Hadoop stack: poor performance except where the application is "trivially parallel"
  - reasons: no indexing, very inefficient joins in Hadoop layer, "sending the data to the query" and not "sending the query to the data"
- Couchbase: replications of the whole document if only its small parts is changed.

Inadvisability of NoSQL DBMSs for
- most of the DW and BI querying
  - few facilities for ad-hoc query and analysis. Even a simple query requires significant programming expertise, and commonly used BI tools do not provide connectivity to NoSQL.
  - E.g., HBase – fast analytical queries, but on column level only
- applications requiring enterprise-level functionality (ACID, security, and other features of RDBMS technology). NoSQL should not be the only option in the cloud.
Big Data „V“ characteristics

- **Volume**    data at scale - size from TB to PB
- **Velocity**  how quickly data is being produced and how quickly the data must be processed to meet demand analysis (e.g., streaming data)
- **Variety**   data in many formats/media
- **Veracity**  uncertainty/quality – managing the reliability and predictability of inherently imprecise data.
Big Data „V“ characteristics

- **Value** worthwhile and valuable data for business (creating social and economic added value – see so called information economy).

- **Visualization** visual representations and insights for decision making.

- **Variability** the different meanings/contexts associated with a given piece of data (Forrester)

- **Volatility** how long is data valid and how long should it be stored (at what point is data no longer relevant to the current analysis.

Discussions: are some Vs really definitional and not only confusing?
Big Data „V“ characteristics

Gardner’s definition:

**Big data** is high-volume, -velocity and -variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making.

3Vs are only 1/3 of the definition!

Remark: three Vs are by Gartner analyst Doug Laney from 2001
Big Data processing

- general observation:
  - data and its analysis becoming more and more complex
  - now: problem with data volume - it is a speed (velocity), not size!
  - necessity: to scale up and scale out both infrastructures and standard data processing techniques

- types of processing:
  - real-time processing of data-in-motion
  - interactive processing and decision support processing of data-at-rest
  - batch oriented analysis (mining, machine learning, e-science)
Big Data processing

Today’s possibilities:

- **Data Stream Management Systems** for streaming data analysis
- NoSQL systems rather for interactive data serving environments
- Hadoop based on MapReduce for decision support
  - problems with complex data access patterns,
  - not appropriate for ad hoc analyses but rather for batch oriented analysis
- highly scalable platforms supporting these technologies: **Big Data Management System (BDMS)**
  - Example: ASTERIX (Vinayak *et al*, 2012) uses fuzzy matching for analytical purposes
<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Data processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>non-procedural access</td>
</tr>
<tr>
<td>L2-L4</td>
<td>algebraic approach</td>
</tr>
<tr>
<td>L1</td>
<td>file management</td>
</tr>
</tbody>
</table>

- ASTERIX DBMS
- Other HLL Compilers
- Algebricks Algebra Layer
- Hadoop M/R Compatibility
- Hyracks Data-parallel Platform
The three-layered Asterix software stack – data access in more layers

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Data processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5 non-procedural access</td>
<td>Asterix QL</td>
</tr>
<tr>
<td>L2-L4 algebraic approach</td>
<td>ASTERIX DBMS</td>
</tr>
<tr>
<td>L1 file management</td>
<td>Hyracks Data-parallel Platform</td>
</tr>
</tbody>
</table>

- Asterix QL
- HiveQL Piglet: subset of Pig
- M/R jobs: Algebricks Algebra Layer, Hadoop M/R Compatibility
- Hayracks jobs: Hyracks Data-parallel Platform
Other trends - NewSQL

- Next generation of highly scalable and elastic RDBMS: NewSQL databases (from April 2011):
  - designed to scale out horizontally on shared nothing machines,
  - still provide ACID guarantees,
  - applications interact with the DB primarily using SQL (with joins),
  - employ a lock-free concurrency control,
  - provide higher performance than available from the traditional systems.
Other trends - NewSQL

- General purpose DB: Google’s Spanner, ClustrixDB, NuoDB, VoltDB (in-memory DB)
- Hadoop-relational hybrids (HadoopDB, parallel DB with Hadoop connectors – e.g. Vertica)
- Layer Concept: supports many data models (e.g. FoundationDB)
- Transparent sharding: MySQL Cluster, (ClustrixDB), ScaleArc, …
- MySQL-like: MemSQL (in-memory DB)
Conclusions

- Key problems are in decisions with NoSQL:
  - choosing the right product
  - design of appropriate architecture for a given class of applications
  - S. Edlich suggests choosing a NoSQL DB after answering about 70 questions in 6 categories, and building a prototype

- New alternatives for Big Data analytics with NewSQL:
  - MemSQL, VoltDB (automatic cross-partition joins), but performance is still a question
  - ClustrixDB (for TP and real-time analytics)

- Near future:
  - shipping to „No Hadoop“ DBMSs (MapReduce layer is eliminated)