Database Architectures: Current State and Development

Jaroslav Pokorný[1] and Karel Richta[2]
[1]Department of Software Engineering, Charles University, Prague, Czech Republic
[2]Department of Computers, Czech Technical University, Prague, Czech Republic

Keywords: Database Architectures, Distributed Databases, Scalable Databases, MapReduce, Big Data Management Systems, NoSQL Databases, NewSQL Databases, SQL-on-Hadoop, Big Data, Big Analytics.

Abstract: The paper presents shortly a history and development of database management tools in last decade. The movement towards a higher database performance and database scalability is discussed in the context to requirements of practice. These include Big Data and Big Analytics as driving forces that together with a progress in hardware development led to new DBMS architectures. We describe and evaluate them mainly in terms of their scalability. We focus also on a usability of these architectures which depends strongly on application environment. We also mention more complex software stacks containing tools for management of real-time analysis and intelligent processes.

1 INTRODUCTION

In 2007 we published the paper (Pokorny, 2007) about database architectures and their relationships to requirements of practice. In some sense, a development of database architectures reflects always requirements from practice. For example, earlier Database Management Systems (DBMS) were focused mainly on OLTP applications typical for business environment. However, the requirements changed also according to the progress in hardware development and sometimes go hand in hand with a theoretical progress. Today, hardware technology makes it possible to scale higher data volumes and workloads, often very specialized, in the way that was not possible in middle 2000s. As a consequence some new database architectures have emerged. Some of them are scalable with some technical parameters appropriate for Big Data storage and processing as well as for cloud-hosted database systems. In other words, they reflect the challenge of providing efficient support for Big Volume of data in data-intensive high performance computing (HPC) environments. Moreover, the architectures of traditional OLTP databases have been proven also obsolete. Already in 2007, (Stonebraker, *et al.*, 2007) described in-memory relational DBMS (RDBMS) overcoming former traditional DBMSs by nearly two orders of magnitude. We could also observe that high performance application requirements shifted from transaction processing and data warehousing to requirements posed by Web and business intelligence applications.

This paper aims to discuss the basic characteristics and the recent advancements of these technologies, illustrate the strengths and weaknesses of each technology and present some opportunities for future work. Particularly, we will describe movement in the development of database architectures towards scalable architectures. Obviously, this movement is related to the advent of the Big Data era, where data volume, velocity, and variety influence significantly its processing. Although Big Data can be viewed from various perspectives and in various dimensions, e.g. economical, legal, organizational, and technological one, we mention only the last one, i.e., Big Data storage and processing. Big Data processing can be of two categories – namely Big Analytics and online read-write access to large volume of rather simple data. This leads to the development of different types of tools and associated architectures.

In Section 2, we describe a history of DBMS architectures by binding to the work (Pokorný, 2007). Section 3 is devoted to scalable architectures, namely NoSQL databases, Hadoop and MapReduce, Big Data Management Systems, NewSQL DBMSs, NoSQL databases with ACID transactions, and SQL-on-Hadoop systems. Section 4 summarizes these approaches and concludes the paper.

Pokorny J. and Richta K..
Database Architectures: Current State and Development.
DOI: 10.5220/0005512001520161
In *Proceedings of 4th International Conference on Data Management Technologies and Applications* (DATA-2015), pages 152-161
ISBN: 978-989-758-103-8

2 DBMS ARCHITECTURES – A HISTORY

In this section we review two main phases of database evolution which influenced the development of the database software. We start with the universal architecture consisting from a hierarchy of layers where the layer *k* is defined by the layer *k*-1. The proposed layers are not fine grained enough to map them exactly to DBMS components but they became the basis for the implementation of all traditional DBMSs in the past. We describe also the development of special purpose database servers which were a response to the some inappropriate properties of the universal architecture. Here we will often use the more common short term database instead of DBMS.

2.1 Universal Architecture

The concept of a multi-layered architecture consisting of five layers L1,…, L5 was proposed in (Härder and Reuter, 1983). There L1 ensures a file management and L5 enables an access to data with a high-level language, typically SQL. The middle layers use auxiliary data structures for mapping higher-level objects to more simple ones in a hierarchical way. In other words, they enable to transform logical data (tables, rows) and SQL operations to physical records and sequences of simple access operations over them. A typical property of the universal architecture is that users can see only this outermost layer. In practice, the number of layers is often reduced, e.g. to three, due to some techniques enabling a more effective performance (Härder, 2005a).

The same model can be used in the case of *distributed databases* for every node of a network together with a connection layer responsible for communication, adaptation, or mediation services. Also, a typical *shared-nothing parallel RDBMS* can be described by this architecture. Layers L1-L4 are presented usually at each machine in a cluster.

Extension of database requirements to process other data types than relational, e.g. VITA (video, image, text, and audio) has led to a solution to use common maximally the layer L1. The others had to be implemented for each type separately. Solutions with *universal servers* or *universal DBMS* so popular in 90s were based on adding loosely coupled additional modules (components) for each new data type. Their data models owned some object-orientation features, but not standardized yet. Vendors of leading DBMSs called these components extenders, data blades, and cartridges, respectively. Remind, that e.g. spatial and text components had a background in software of specialized vendors built on a file system equipped by a sophisticated functionality for processing spatial objects and texts, respectively. Integration of such components with relational engines presented a problem and effectiveness of the resulted system was never fully resolved. Any efficient solution of optimization problems resulted usually in modification of the DBMS kernel that was very expensive, time-consuming, and tending to errors.

Development of the L5 layer during 90s peaked in a specification of so-called object-relational (OR) data model. Its standardization in SQL started in the version SQL:1999. Tables can have structured rows; their columns can even be of user-defined types or of new built-in data types. Relations can be sets of objects (rows) linked via their IDs. For new built-in data types there is a standardized set of predicates and functions for manipulation of their instances. Although the ORDB technology is already available for use in all the major RDBMS products, its industrial adoption rate is due to its complexity not very high.

2.2 Special Purpose Database Servers

In 2005, (Stonebraker, 2005) and other scientists claimed “the one size fits all” model of DBMS had ended and raised a need to develop new DBMS architectures reminding rather separate database servers tailored to requirements of particular applications types. Not only functionality was considered, but also the way how data is stored. Candidates for special-purpose database servers have been found in application areas, namely:

- data warehousing, OLAP
- XML processing,
- data streams processing,
- text retrieval,
- processing scientific data.

We consider also mobile and embedded DBMS. This application area produces large and complex datasets that requires more advanced database support, than that one offered by universal DBMS.

2.2.1 Data Warehousing and OLAP

Without doubts data warehouses belong to the oldest special purpose databases. Based principally on RDBMS techniques, data warehouse architecture has to include explicit specialized support for a specialized logical model (denormalized, multidimensional), historical, summarized and

consolidated data operations for ad hoc analysis queries, and efficient access and implementation methods for such operations. Optimizations using column stores, pipelining operations, and vectorising operations are used here to take advantage of commodity server hardware. For example, column-stores use "once-a-column" style processing, which is aimed at better I/O and cache efficiency. Sybase IQ (today in Version 16) belongs to pioneering implementations of column-stores. An experience with column-stores from the past emphasizes their high performance and scalability for complex queries against large data sets. That is why the variants of column-stores have become a popular platform for managing and analysing Big Data using SQL-based analytic methods.

Oracle produces a remarkable solution for data warehouses, so called Oracle Exadata Database Machine (https://www.oracle.com/engineered-systems/exadata/index.html, 2015). It is a complete, optimized, hardware and software solution that delivers extreme performance and database consolidation for data warehousing and reporting systems. It uses very efficient hybrid columnar compression.

Today, the column-oriented HP Vertica Analytic Database with *massively parallel processing* (MPP) and shared-nothing architecture is most popular (Lamb et al., 2012). This tool is cluster-based and integrated with Hadoop. Its SQL has built-in many analytics capabilities.

An example of a progress in OLAP database technology is standalone OLAP server EssBase (Anantapantula and Gomez, 2009) which stores data in array-like structures, where the dimensions of the array represent columns of the underlying tables, and the values of the cells represent precomputed aggregates over the data.

2.2.2 XML Processing

In late 90s, XML started to become more popular for representing semi-structured data. New XML query languages and XML DBMSs occurred. A significant role in storing XML data in a database way is whether the data is data-centric or document-centric.

One possibility how to store XML data is an *XML-enabled* database. It means to map (shred) the XML documents into data structures of the existing RDBMS. Experiments show that such database is most feasible if only simple XPath operations are used or if the applications are designed to work directly against the underlying relational schema. Data-centric documents using XML as a data transport mechanism are suitable for this approach.

A more advanced solution was to develop a DBMS with a native XML storage (*native XML database* or *NXD*). These databases are suitable for document-centric XML data. An implementation of NXD was a challenge in 2000th both for developers and researchers of DBMSs. In database architectures, NXDs provide a nice example when a DBMS needs a separate engine. We can distinguish two main approaches to NXD implementations:

- NXD DBMS as a separate engine (Tamino XML Server, XHive/DB, XIndice, eXist, etc.),
- adding native XML storage to RDBMS (e.g., XML Data Synthesis by Oracle),
- hybrid solutions, i.e. a RDBMS natively stores and natively processes XML data (e.g., DB2 9 pureXML, ORACLE 11g with more storage models for XML data, SQL Server 2012). These approaches include also possibility to parse and shred the XML documents to an XML-data-driven relational schema.

An advantage of the latter is the possibility to mix XML with relational data and/or with other types of data (e.g. textual, RDF) within one DBMS. XML data can be accessed via the combined use of SQL/XML and full XQuery language. While critical data is still in a relational format, the data that do not fit the relational data model is stored natively in XML. Härder shows in (Härder, 2005b) how the layered architecture described in Section 2.1 can be used to implement NXD DBMS.

Unfortunately, it seems in the last years, that research in XML database area is not too intensive and that XML databases become rather a niche topic. The more lightweight, bandwidth-non-intensive JSON (JavaScript Object Notation) is now emerging as a preferred format in web-centric, so-called NoSQL databases (Section 3.1). Even, PostgreSQL from version 9.2 has a JSON data type.

2.2.3 Data Stream Processing

Data streams occur in many modern applications as, e.g., network traffic analysis, collecting records of transactions and their analysis, sensor networks, application exploiting RFID tags, telephone calls, health care applications, financial applications, Web logs, click-streams, etc. Data transmission poses continuous streaming data, which are indexed in time dimension to be filtered to individual (possibly mobile) users. Applications require near real-time querying and analyses. These requirements justify the

existence of special DBMSs because the relational ones are not effective in this area.

Special purpose *Data Stream Management Systems* (DSMS) have been implemented to deal with these issues. These systems are not based primarily on loading data into a database, except transiently for the duration of certain operations. For example, data streams containing RFID tags generated from events obtained by RFID readers are filtered, aggregated, transformed, and harmonized so that events associated with them can be monitored in real-time. Events processing is data centric, it requires in-memory processing. Associated query languages are based on SQL, e.g., StreamSQL (www.streambase.com/developers/docs/latest/streamsql/, 2004). Queries executed continuously over the data passed to the system are called *continuous queries*. Typical for DSMS are *windows* operators that only select a part of the stream according to fixed parameters, such as the size and bounds of the window. For example, in the *sliding windows*, both bounds move.

A pioneering STREAM project (http://ilpubs.stanford.edu:8090/641/, 2004) started in 2002 and has officially wound down in 2006. A number of other DSMS have occurred since then. For example, Odysseus (http://odysseus.informatik.uni-oldenburg.de/, 2007) belongs to this category. Many big vendors such as Microsoft, IBM, or Oracle, have their own data stream management solution. However, there is neither a common standard for data streaming query languages nor an agreement on a common set of operators and their semantics until today.

Although these systems proved to be an optimal solution for on the fly analysis of data streams, they cannot perform complex reasoning tasks.

2.2.4 Text Retrieval

Previous full text search engines did not have too many database features. They provided typical operations like index creation, full text search, and index update. Again some hybrid solutions are preferred today. Some projects use current DBMS as backend to existing full text search engines. For example, the index file is stored in a relational database.

A significant representative of this development is the search engine Apache Lucene (now in Version 4.7.1) (http://lucene.apache.org/, 2011). An associated open source enterprise search platform Apache Solr is now with Lucene integrated (Lucene/Solr). Its major features include full text search, hit highlighting, faceted search, dynamic clustering, database integration, and rich document (e.g., Word, PDF) handling.

Based on the standard SQL/MM from 2003 we can find text retrieval features in SQL (similarly as spatial objects and still images), e.g., in RDBMS PostgreSQL. But performance tests (Lütolf, 2012) have shown that full text search queries with a large result set are fast with Apache Lucene/Solr but with PostgreSQL are very slow.

2.2.5 Processing Scientific Data

A special category of database servers is used for scientific data storage and processing. Biomedical sciences, astronomy, etc., use huge data repositories often organized in grid or cloud architectures. However, a use of commercial cloud services raises issues of governance, cost-effectiveness, trust and quality of service. Consequently, some frameworks use hybrid cloud, combining internal institutional storage, cloud storage and cloud-based preservation services into a single integrated repository infrastructure. It seems, that universal and hybrid servers are effective especially when the data requirements can be simply decomposed into relatively independent parts evaluated separately in database kernel and in the module built for the given special data type.

A notable representative of this data processing category is DBMS SciDB proposed in 2009. Now, SciDB is a DBMS optimized for multidimensional data management of Big Data and for so called Big Analytics (Stonebraker et al., 2013). Data structures of SciDB include arrays and vectors as first-class objects with built-in optimized operations. SciDB is usable also for geospatial, financial, and industrial applications.

2.2.6 Mobile and Embedded DBMSs

Embedded DBMSs are a special case of embedded applications. Typically, they are single application DBMS that are not shared with other users, their management is automatic, and their functions are substantially limited. Their self-management includes at least backups, error recovery, or reorganization of tables and indices. Such applications run often on special devices, mostly mobile. The client and server have wireless connections. We talk then about *mobile and embedded DBMS*.

A sufficient motivation for movement to this new data management seems to be applications as healthcare, insurance or field services, which use

mobile devices. In the case data is in backend databases, and/or generally somewhere on Web, or accessible, e.g., in form of cloud computing. In many applications, databases are needed in a very restricted version directly on mobile devices, e.g., on sensors. An important property of these architectures is synchronization with backend data source. Current most mobile DBMSs only provide limited prepackaged SQL functions for the mobile application.

Typical commercial solutions for mobile and embedded DBMSs include, e.g., Sybase's Ultralite, IBM's Mobile Database, SQL Server Express, MS-Pocket Access, SQL Server Compact, and Oracle Lite.

Now mobile and embedded DBMSs typically:

- use a component approach, i.e. a possibility to configure the database functionality according to application requirements and minimize thereby the size of application software;
- are often in-memory databases, even without requirements on any persistence. It means they use special query techniques and indexing methods.

3 SCALABLE DATABASES

The authors of (Abiteboul et al., 2005) emphasized two main driving forces in database area: Internet and particular sciences, as the physics, biology, medicine, and engineering. Said in today's words, it means a shift towards Big Data. The former led to development of Web databases, the latter to scientific databases that require today not only a relevant data management (see Section 2.2.5) but also tools for advanced analytics. Similar requirements occur in Big Data and Big Analytics trends today. In the course of the last eight years, the classical DBMS architecture was challenged by a variety of these requirements and changes, i.e. data volume and heterogeneity, scalability and functional extensions in data processing.

It is typical, that traditional distributed DBMS are not appropriate for these purposes. They are many reasons for it, e.g.:

- database administration may be complex (e.g. design, recovery),
- distributed schema management,
- distributed query management,
- synchronous distributed concurrency control (2PC protocol) decreases update performance.

DBMS architectures suitable for Big Data are mostly built on a new hardware. A rather traditional solution is based on a single server with very large memory and multi-core multiprocessor. A popular infrastructure for Big Data is a HPC cluster (a.k.a. supercomputer). Some RDBMS installations use SSD data storage that is 100 times faster in random access to data than the best disks. Such installations are able to process PBytes of data. HPC cluster as well as grid is appropriate especially for Big Analytics in context of scientific data. We can scale vertically such databases, i.e. *scale-up*, by adding new resources to a single server in a system. This approach to data scalability is appropriate for corporate cloud computing (Zhao et al., 2014).

Architectures suitable rather for customer cloud computing scale DBMSs across multiple machines. This technique, *scale-out*, uses well-known mechanism called *database sharding*, which breaks a database into multiple shared-nothing groups of rows in the case of tabular data and spreads them across a number of distributed servers. Sherds are not necessarily disjunctive. Each server acts as the single source of a data subset. Sharding is just another name for *horizontal data partitioning*. However, database sharding reminding classical distributed databases cannot provide high scalability at large scale due to the inherent complexity of the interface and ACID guarantees mechanisms. This had an influence on development of so called NoSQL databases (Cattell, 2010).

3.1 NoSQL Databases

A new generation of distributed database products labelled *NoSQL* emerged from 2004. Obviously, the term NoSQL is misleading. Though not based on the relational data model, some of these products offer a subset of SQL data access capabilities. To be able to scale-out, NoSQL architectures differ from RDBMS in many key design aspects (Pokorny, 2013):

- simplified data model,
- database design is rather query driven,
- integrity constraints are not supported,
- there is no standard query language,
- unneeded complexity is reduced (simple API, weakening ACID semantics, simple get, put, and delete operations).

Most NoSQL databases have been designed to query over high data volumes and provide little or no support for traditional OLTP based on ACID properties. Indeed, CAP theorem (Brewer, 2005) has shown that a distributed database system can only choose at most two out of three properties: Consistency, Availability and tolerance to Partitions.

Then preferring P in our non-reliable Internet environment, NoSQL databases support A or C. For example, Cassandra and Hbase databases have chosen, AP and CP, respectively. Cassandra uses also tuneable consistency, enabling different degrees of balance between C, A, P.

In practice, mostly strict consistency is relaxed to so-called *eventual consistency*. Eventually consistent emphasizes that after a certain time period, the data store comes to a consistent state.

There are many of NoSQL databases. Some well-known lists, e.g. (http://nosql-database.org/, 2015), of them consider also XML, object-oriented, and graph databases and others in this category. Today, mainly key-value stores and their little more complex column-oriented and document-oriented variants represent the NoSQL category. Remind that the column-oriented data model uses the column as a basic term and it is implemented similarly as relational column stores (see Section 2.2.1). A management of graphs is possible with *graph databases* (see, e.g., Neo4j (http://neo4j.com/, 2015).

Every NoSQL database has some special features and functionality which makes it different to decide for using in an application. Diversity of query tools in these databases is also very high and it seems that it will be very difficult to develop a single standard for all categories (Bach and Werner, 2013). A system administrator must thus consider carefully which type of database best suits user needs before committing to one implementation or the other. Today, NoSQL evolve and take on more traditional DBMS-like features. However, their ad-hoc designs prevent their wider adaptability and extensibility.

A special problem is quality and usability of these engines. Partially interesting information can be obtained from DB-Engines Ranking (http://db-engines.com/en/ranking, 2015) where score of a DBMS product expresses its popularity. Considering NoSQL, the document store MongoDB, column-oriented Cassandra, and key-value store appear in the top ten rated database engines in April 2015.

Concerning the application level, NoSQL systems are recommended for newly developed applications, particularly for storage and processing Big Data, but not for migrating existing applications which are written on top of traditional RDMSs.

An important point is that the term NoSQL has come to categorize a set of databases that are more different than they are the same. Today, scaling is certainly the first requirement but modern databases must meet more, at least, the must:

- adapt to change, e.g. mine new data sources and data types without the database restructuring,
- be able to offer tools for formulating rich queries, creating indexes, and searching over multi-structured and quickly changing data.

Unfortunately, only some of NoSQL databases meet all three requirements.

3.2 Hadoop and MapReduce

Many NoSQL databases are based on *Hadoop Distributed File System* (*HDFS*) (http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf, 2007), which is a part so called *Hadoop software stack*. This stack enables to access data by three different sets of tools in particular layers which distinguishes it from the universal DBMS architecture with only SQL API in the outermost layer. The NoSQL HBase is available as a key-value layer with Get/Put operations as input. Hadoop MapReduce (M/R) system server in the middle layer enables to create M/R jobs. Finally, high-level languages HiveQL, PigLatin, and Jaql are at disposal for some users at the outermost layer. HiveQL is an SQL-like language; Jaql is a declarative scripting language for analysing large semi-structured datasets. Pig Latin is not declarative. Whose programs are series of assignments similar to an execution plan for relational operations in a RDBMS.

On the analytics side, M/R emerged as the platform for all analytics needs of the enterprise, i.e. as an effective tool to pre-process unstructured and semi-structured data sources such as images, text, raw logs, XML/JSON objects etc. A special attention belongs to above mentioned HiveQL. It is originally a part of infrastructure (data warehousing application) Hive (https://hive.apache.org, 2011), which is the first SQL-on-Hadoop solution (see Section 3.6) providing an SQL-like interface with the underlying M/R. Hive converts the query in HiveQL into sequence of M/R jobs.

Not all from M/R is perfect. For example, its implementation is sub-optimal, it uses brute force instead of indexing. Despite of the critique of many aspects of M/R, there are many approaches to its improvement (Doulkeridis and Nørvåg, 2014). For example, often it is emphasized the one main limitation of the M/R framework is that it does not support the joining multiple datasets in one task. However, this can still be achieved with additional M/R steps (Zhao et al., 2014). An approach for extending M/R for supporting real-time analysis is introduced at Facebook (Borthakur et al., 2011).

On the other hand, rapid implementation of majority of the data discovery and data science attempts requires strong support for SQL with, e.g.

embedded analytical capabilities normally available in an MPP-based analytic database.

Other solution is offered by so called Hadoop-relational hybrids. For example, above mentioned column-oriented HP Vertica Analytic Database is cluster-based and integrated with Hadoop. Its SQL has built-in many analytics capabilities.

3.3 Big Data Management Systems

Hadoop software stack dominates today in industry, but for Big Analytics it is not too practical. To address Big Analytics challenges, a new generation of scalable data management technologies has emerged in the last years. A *Big Data Management System* (BDMS) is a highly scalable platform which supports requirements of Big Data processing. Now, BDMS is considered as a new component of more general information architecture in an enterprise into which Big Data solutions should be integrated.

Example: Considering Hadoop software stack as BDMS architecture of a first-generation, a representative of BDMS of a second-generation is ASTERIX system (Vinayak et al., 2012). It is fully parallel, able to store, access, index, query, analyse, and publish very large quantities of semi-structured data (represented in the JSON format). The ASTERIX architecture (see Table 1) is a software stack also with more layers for data access with some new platforms. For example, Pregel (Malewicz et al., 2010) is a system for large-scale graph processing on distributed cluster of commodity machines. Algebrick's algebra layer is independent on a data model and is therefore able to support high-level data languages like HiveQL and Piglet (a subset of PigLatin in Hadoop software Stack)) and others. Partitioned parallel platform Hyracks for data-intensive computing allows users to express a computation as a directed acyclic graph of data operators and connectors. IMRU provides a general framework for parallelizing a large class of machine learning algorithms, including batch learning, based on Hyracs.

Microsoft also developed a Big Data software stack targeted for Big Analytics (Chaiken et al., 2008). Data are modelled by relations with a schema. A declarative and extensible scripting language called SCOPE (Structured Computations Optimized for Parallel Execution) is a clone of SQL. Lower layers of the stack contain a distributed platform Cosmos designed to run on large clusters consisting of thousands of commodity servers.

There are other software stacks addressing Big Data challenges, e.g. Berkeley Data Analytics Stack, Stratosphere Stack, etc.

Finally, note that to manage and analyse non-relational Big Data not only NoSQL, BDMS, even no DBMS is needed. Non-DBMS data platforms such as low-level HDFS can be sufficient, e.g. for a batch analysis.

3.4 NewSQL Databases

NewSQL is a subcategory of RDBMS preserving SQL language and ACID properties. These tools achieve high performance and scalability by offering architectural redesigns that take better advantage of modern hardware platforms such as shared-nothing clusters of many-core machines with large or non-volatile in-memory storage. Their architectures provide much higher per-node performance than traditional RDBMS.

Obviously the NewSQL architectures can distinguish significantly. Some of them are really new, e.g., VoltDB (http://voltdb.com/, 2015), Clustrix (www.clustrix.com/, 2015), NuoDB (www.nuodb.com/, 2015), and Spanner (Corbett, 2012), some are improved versions of MySQL. These DBMSs are trying to be usable for applications already written for an earlier generation of RDBMSs. However, Spanner uses a little different relational data model enabling to create hierarchies of tables. Often, SQL features are not so strict, e.g. SQL in VoltDB does not use NOT in WHERE clause. The horizontal scalability of NewSQL databases is high. They are appropriate for data volumes up to the PByte level. An excellent review of above mentioned four NewSQL systems and a lot of NoSQL DBMSs is offered by the paper (Grolinger et al., 2013).

A high performance is often achieved by in-memory processing approach. Thanks to considerable technological advances during the last 30 years these DBMSs finally become available in commercial products. For example, above mentioned VoltDB is in-memory, parallel DBMSs optimized for OLTP applications. It is a highly distributed, relational database that runs on a cluster on shared-nothing, in-memory executor nodes.

The fundamental problem with in-memory DBMSs, however, is that their improved performance is only achievable when the database is smaller than the amount of physical memory available in the system. To overcome the restriction that all data fit in main memory, a new technique, called *anti-caching* (DeBrabant et al., 2013), was proposed for H-store (the academic version of VoltDB). An anti-caching

Table 1: The ASTERIX Software Stack.

	Level of abstraction	*Data processing*
L5	non-procedural access	Asterix QL
L2-L4	algebraic approach	ASTERIX DBMS; HiveQL, Piglet, …; Other HLL compilers; M/R jobs; Pregel job; Algebricks Algebra Layer; Hadoop M/R compatibility; Pregelix; IMRU; Hyracks jobs
L1	file management	Hyracks Data-parallel Platform

DBMS reverses the traditional hierarchy of disk-based systems, i.e. all data initially resides in memory, and when memory is exhausted, the least-recently accessed records are collected and written to disk.

3.5 NoSQL Databases with ACID Transactions

A notable new generation of NoSQL databases we mention enables ACID transactions. Many NoSQL designers are therefore exploring a return to transactions with ACID properties as the preferred means of managing concurrency for a broad range of applications. Using tailored optimizations, designers are finding that implementation of ACID transactions needs not sacrifice scalability, fault-tolerance, or performance. Such NoSQL databases are also called *Enterprise NoSQL*. They are database tools that have the ability to handle the volume, variety, and velocity of data like all NoSQL solutions, and have the necessary features to run inside the business environment.

These DBMSs:

- maintain distributed design, fault tolerance, easy scaling, and a simple, flexible base data model,
- extend the base data models of NoSQL,
- have monitoring and performance tools,
- are CP systems with global transactions.

A good example of such DBMS is a key-value FoundationDB (https://foundationdb.com/, 2015) with scalability and fault tolerance (and an SQL layer). Oracle NoSQL Database provides ACID complaint transactions for full CRUD (create, read, update, delete) operations, with adjustable durability and consistency transactional guarantees. The in Section 3.4 mentioned Spanner is also NoSQL which can be considered as NewSQL as well.

MarkLogic (Cromwell S., 2013) is a document database that can store XML, JSON, text, and large binaries such as PDFs and Microsoft Office documents. MarkLogic run on HDFS, has full-text search built directly into the database kernel. MarkLogic is a true transactional database. Similarly, the distributed graph database OrientDB guarantees also ACID properties.

3.6 SQL-on-Hadoop Systems

Although Hadoop software stack seems to be sufficiently mature now for some applications, we have seen that there are still possibilities to improve and optimize tools based on it. These concerns especially SQL-on-Hadoop systems that evolve to more database-like architectures, mainly towards SQL.

Obviously, Hive mentioned in Section 3.2 belongs to this category. Technologies such as Hive are designed for batch queries on Hadoop by providing a declarative abstraction layer (HiveQL), which uses M/R processing framework in the background. Hive is used primarily for queries on very large data sets and large ETL jobs.

SQL processed by a specialized (Google-inspired) SQL engine on top of a Hadoop cluster is Cloudera Impala (www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html, 2015) - designed as MPP SQL query engine that runs natively in Hadoop. Impala provides interactive query capabilities to enable traditional business intelligence and analytics on Hadoop-scale datasets.

Splice Machine (Splice Machine, 2015) is a Hadoop RDBMS. It is tightly integrated into Hadoop, using HBase and HDFS as the storage level. Splice Machine supports real-time ACID transactions.

The Vertica mentioned in Section 3.2 is often categorized as a Hadoop-relational hybrid because it can work with Hadoop together.

Table 2: New Database Architectures in last 15 years.

Milestone	*Category*	*Subcategory*	*Representatives*
2009	NoSQL	Key-value	Redis
		Column-oriented	Cassandra
		Document-oriented	MongoDB
		Graph databases	Neo4j
2005	BDMS	1. generation	Hadoop software stack
2010		2. generation	Asterix software stack
2011	NewSQL	General purpose	NuoDB, VoltDB, Clustrix
		Google's hybrids	Spanner
		Hadoop-relational	Vertica
		SQL-on-Hadoop	Hive, Impala, Presto, Splice
		NoSQL with ACID	FoundationDB, MarkLogic

There is also a technology of "connectors". In this architecture Hadoop and a DBMS product are connected in a simple way so that data can be passed back and forth between these two systems.

As a representative of this approach we can mention Presto (http://prestodb.io/, 2013) - an open source distributed SQL query engine for running interactive analytic queries. Presto supports HiveQL as well. A user has also connectors Hadoop/Hive, Cassandra, and TPC-H at disposal that provide data to queries. The last connector dynamically generates data that can be used for experimenting with and testing Presto. Unlike Hive, Presto and Impala follow the Google distributed query engine processor inspired by Google Dremel (Melnik, et al., 2010) – a query system for analysis of read-only nested data.

4 CONCLUSIONS

Summarizing, the current approaches to DBMS architecture turn in towards:

- improvements of present database architectures,
- radically new database architecture designs.

The former includes challenges given by new hardware possibilities and Big Analytics, i.e. some low-level data processing algorithms have to be changed. This influences also approaches to query optimization. But (Zhao, et al., 2014) believe that it is unlikely that MapReduce will completely replace DBMSs even for data warehousing applications.

Both SQL-on-Hadoop systems and NoSQL DBMS aimed at Big Data management require much more care in a design and tuning of application environment. It is necessary to select a database product according to the role it is intended to play and the data over which it will work. This causes higher complexity of new architectures, particularly of their hybrid variants, that are becoming dominant. We have mentioned some query tools of today's new DBMSs, which are partially SQL-like. To use more data manipulation languages in one hybrid platform is certainly possible, but also complicated. Any abstraction/virtualization layer would be useful. However, the concept of a unifying language is not yet real. The database history in the last 15 years is summarized in Table 2.

We have discussed horizontal layers coming from special software stacks. In real Big Data environment it is necessary to consider explicitly applications layers as well. They are vertical and include at least universal information management, real-time analytics, and intelligent processes, which are so important to most organizations today. Behind we can find data flows between particular data stores.

However, this all complicates the design of enterprise information systems and create particular challenges for research and development not only of new database architectures.

ACKNOWLEDGEMENTS

This research has been partially supported by the grant of GACR No. P103/13/08195S, and also partially supported by the Avast Foundation.

REFERENCES

Anantapantula, S., Gomez, J.-S., 2009. *Oracle Essbase 9 Implementation Guide*, Packt Publishing. Birmingham.

Abiteboul, S., Agrawal, R., Bernstein, Ph., Carey, M., Ceri, S., et. al., 2005. Lowell Database Research self-assessment, *Comm. of ACM*, 48(5) pp.111-118.

Bach, M., Werner, A., 2014. Standardization of NoSQL Database Languages. In *BDAS 2014, 10th International Conference*, Ustron, Poland, pp. 50-60.

Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., et al., 2011. Apache Hadoop goes real-time at Facebook. In *ACM SIGMOD Int. Conf. on Management of Data*, Athens, pp. 1071-1080.

Brewer, E., 2000. Towards robust distributed systems. Invited Talk on *PODC 2000*, Portland, Oregon.

Cattell, R., 2010. Scalable SQL and NoSQL Data Stores. *SIGMOD Record*, 39(4) 12-27.

Chaiken, R., Jenkins, B., Larson, Per-Åke, Ramsey, B., Shakib, D., et al., 2008. SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In *VLDB '08 – Conference proceedings*, Auckland.

Corbett, J. C., Dean, J., Epstein, M., Fike, A., et al., 2012. Spanner: Google's Globally-Distributed Database. In *OSDI 2012, 10th Symposium on Operating System Design and Implementation - Conference Proceedings*, Hollywood, pp. 1-14.

Cromwell S., 2013. Top 5: MarkLogic topped last week's Silicon Valley fundings. Silicon Valley Business Journal, April 15, 2013.

DeBrabant, J., Pavlo, A., Tu, S., Stonebraker, M., and Zdonik, S., 2013. Anti-Caching: A New Approach to Database Management System Architecture. In *VLDB Endowment – Conference Proceedings*, 6(14), pp. 1942-1953.

Doulkeridis and Nørvåg, K., 2014. A survey of large-scale analytical query processing in MapReduce, *The VLDB Journal*, 23:355-380.

Grolinger, K., Higashino, W. A., Tiwari, A., and Capretz, M. AM, 2013. Data management in cloud environments: NoSQL and NewSQL data stores, *Journal of Cloud Computing: Advances, Systems and Applications - a Springer Open Journal*, 2:22.

Härder, T., Reuter, A., 1983. Concepts for Implementing and Centralized Database Management System. In *Int. Computing Symposium on Application Systems Development - Conference Proceedings*, Nürnberg, pp. 28-10.

Härder, T., 2005. DBMS Architecture – the Layer Model and its Evolution, *Datenbank-Spektrum*, 13, pp. 45-57.

Härder, T., 2005. XML Databases and Beyond-Plenty of Architectural Challenges Ahead. *Proceedings of ADBIS 2005*, LNCS 3631, Springer, pp. 1 – 16.

Lamb, M., Fuller, R., Varadarajan, N., Tran, B., Vandiver, et al., 2012. The Vertica Analytic Database: C-Store 7 Years Later. *Proceedings of VLDB Endowment*, 5(12), pp. 1790-1801.

Lütolf, S., 2012. PostgreSQL Full Text Search - An Introduction and a Performance Comparison with Apache Lucene /Solr. Seminar Thesis, Chur. (http://wiki.hsr.ch/Datenbanken/files/Full_Text_Search_in_PostgreSQL_Luetolf_Paper_final.pdf), (& May 2015).

Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., and Czajkowski, G., 2010. Pregel: a system for large-scale graph processing. In *SIGMOD '10 - Int. Conf. on Management of Data*, Indianopolis, pp. 135-146.

Melnik, S., Gubarev, A., Long, J.J., Romer, G., et al, 2010. Dremel: Interactive Analysis of Web-Scale Datasets. In *VLDB - 36th Int'l Conf on Very Large Data Bases*, pp. 330-339.

Pokorný, J., 2007. Database Architectures: Current Trends and Their Relationships to Requirements of Practice. In *Advances in Information Systems Development: New Methods and Practice for the Networked Society. Information Systems Development Series*, Springer Science+Business Media, New York, pp. 267–277.

Pokorný, J., 2013. NoSQL Databases: a step to databases scalability in Web environment. *International Journal of Web Information Systems*, 9 (1), 69-82.

Splice Machine (2015) White Paper: Splice Machine: The Only Hadoop RDBMS [online]. Available from: http://www.splicemachine.com/ [Accessed: 15th January, 2015].

Stonebraker, M., Cetintemel, U., 2005. One size fits all: An idea whose time has come and Gone. In *ICDE '05 - 21st Int. Conf. on Data Engineering*, pp. 2–11.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., et al., 2007. The End of an Architectural Era (It's Time for a Complete Rewrite). In *VLDB 2007 – Conference Proceedings*, ACM, pp. 1150-1160.

Stonebraker, M., Brown, P., Zhang, D., and Becla, J., 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering*, 15(3) 54-62.

Vinayak, R., Borkar, V., Carey, M.-J., and Li, Ch. Ch., 2012. Big data platforms: what's next? *ACM Cross Road*, No. 1 pp. 44-49.

Zhao, L., Sakr, S., Liu, A., and Boughuettaya, A., 2014. *Cloud Data Management*. Springer.