database
research group

PS Non-Standard Database Systems

Summer term 2019

Overview

Daniel Kocher

March 4, 2019

This document gives a brief overview on three categories of non-standard database systems (DBS) in the context of this class. For each category, we provide you with a motivation, a discussion about the key properties, a list of commonly used implementations (proprietary and open-source/freely-available), and recommendation(s) for your project. For your convenience, we also recap some relevant terms (e.g., ACID, OLTP, ...).

Terminology

OLTP *Online transaction processing*. OLTP systems are highly optimized to process (1) short queries (operating only on a small portion of the database) and (2) small transactions (inserting/updating few tuples per relation). Typically, OLTP includes insertions, updates, and deletions. The main focus of OLTP systems is on low response time and high throughput (in terms of transactions per second). Databases in OLTP systems are highly normalized [5].

OLAP *Online analytical processing*. OLAP systems analyze complex data (usually from a data warehouse). Typically, queries in an OLAP system may run for a long time (as opposed to queries in OLTP systems) and the number of transactions is usually low. Furthermore, typical OLAP queries are read-only and touch a large portion of the database (e.g., scan or join large tables). The main focus of OLAP systems is on low response time. Databases in OLAP systems are often *de-normalized* intentionally [5].

ACID Traditional relational database systems guarantee ACID properties:

- ***A**tomicity*
  All or *no* operation(s) of a transaction are reflected in the database.
- ***C**onsistency*
  Isolated execution of a transaction preserves consistency of the database.
- ***I**solation*
  Concurrently executed transactions must be *unaware of one another* and must not see results of other concurrently executed transactions.

- ***D**urability*
  After successful completion of a transaction, the changes it has made to the database *persist* even in case of a system failure.

BASE Replication-based distributed database systems often trade consistency for availability. They are said to satisfy the BASE properties which are looser than the ACID guarantees [10]:

- ***B**asically **A**vailable*
  The database is available almost all the time (in terms of the CAP theorem). For example, updates are allowed even if the network is partitioned.
- ***S**oft state*
  Due to network partitioning, replicas may have different states. Thus, the state of the database may not be precisely defined at all times.
- ***E**ventually consistent*
  Consistency will be established across all replicas once a network partitioning is resolved.

CAP The CAP theorem states that it is impossible to guarantee all three desirable properties at the same time in a distributed database system that replicates data to multiple nodes in a cluster [5]. The three desirable properties are

- ***C**onsistency*
  All nodes have the same copy of a replicated data item visible for various transactions (reads/writes).
- ***A**vailability*
  Every read/write operation on a certain data item is either processed successfully or results in an error message (requested operation is *not* completed).
- ***P**artition tolerance*
  Even if the network is partitioned into two or more partitions (communication of nodes only possible within a partition), the system continues operation.

SQL SQL is a powerful *declarative* language to formulate queries in relational database systems, which support complex queries like joins.

Vertical Scalability Expand the capacity (e.g., main memory) of individual nodes in a distributed system without interrupting the system [5].

Horizontal Scalability Increase or decrease the number of nodes in a distributed system without interrupting the system [5].

Sharding In a distributed environment, a relation is split into subrelations (subsets of rows) and the subrelations are distributed horizontally on many nodes [5].

NoSQL Database Systems

NoSQL is also commonly referred to as *Not only SQL* and represents a class of *non-relational* database systems that emerged to meet new requirements: they focus on *high availability* and *high scalability*. Typically, NoSQL database systems are designed to operate in a distributed environment and one of their main goals is horizontal scalability. They operate on a non-relational data model and are schemaless (as opposed to relational databases). Some NoSQL database systems support SQL, some offer a custom programmatic/query interface.

To achieve high availability and high scalability, NoSQL database systems usually trade guarantees/capabilities (compared to typical relational database systems) [5]:

Drop of ACID Guarantees NoSQL database systems are only guaranteed to be *eventually consistent*, meaning that updates are propagated to other nodes eventually Moreover, consistency guarantees for reads are limited [1]. With respect to the CAP theorem, NoSQL database systems usually favor availability and partition tolerance over consistency to achieve the desired performance and scalability. They are said to require the BASE properties.

Limited Query Language Capabilities As opposed to SQL, NoSQL database systems typically provide a limited set of operations, often referred to as *CRUD* operations: *Create*, *Read*, *Update*, and *Delete*. However, for most applications that use a NoSQL database systems, this restricted set of operations is sufficient.

Amongst the most popular types of NoSQL database systems are the following:

Key-Value Stores As the name suggests, key-value stores store key-value pairs (k, v). In other words, a key-value store can be viewed as large, distributed hash table (like `std::unordered_map` in C++, `dict` in Python3, or `HashMap` in Java). Hence, a key is required to be unique and data items are accessed through the key. The value associated with a key can be anything and does not have to satisfy some schema (like it is the case in relational databases). Key-value stores typically support insertion/deletion of key-value pairs as well as accessing values by specifying a key. More complex operations need to be implemented by the application [12].

Examples include Amazon DynamoDB [1], Riak [2], and Redis [3].

(Wide) Column Stores Traditional relational database systems store their data items (tuples) row-wise, i.e., rows are stored on disk consecutively. Contrarily, column stores use a columnar data representation on disk, i.e., columns are stored on disk one after another. Logically, the data is still viewed as table just as in relational database systems. For some workloads, storing a relation column-wise may be beneficial in terms of performance. However, it is important to notice that this is not the case for every query workload. *Wide* column stores (also referred to as *extensible record stores*) introduce the notion of *column families*. Column families are sets of columns that are expected to be accessed simultaneously. This concept adds a third dimension for data access (in addition to row and column). Typically, data in a column family is stored consecutively on disk [12]. Google's BigTable [2] is the most popular wide column store.

Examples include Google BigTable [4], Apache HBase [5], and Apache Cassandra [6].

Document Stores Document stores are somewhat similar to key-value stores as documents are associated with and accessed through a unique key. Typical document formats are XML and JSON, and a document has to satisfy the corresponding specification. Therefore, document stores are said to store semi-structured data [12].

Examples include MongoDB [7] and Apache CouchDB [8].

[1]`https://aws.amazon.com/dynamodb/`
[2]`http://basho.com/products/riak-kv/`
[3]`https://redis.io/`
[4]`https://cloud.google.com/bigtable/`
[5]`https://hbase.apache.org/`
[6]`https://cassandra.apache.org/`
[7]`https://www.mongodb.com/`
[8]`https://couchdb.apache.org/`

Graph Stores In graph stores, the underlying data model is tailored to represent data items connected by semantically meaningful links. Data (typically key-value pairs) may be stored alongside both nodes and links. Due to their data model, graph stores offer good performance when data items need to be traversed by their links [8, 12].

Examples include Neo4J [9] and Apache TinkerPop [10].

Recommendation Generally speaking, you can choose any type of NoSQL system. Regarding specific implementations, we recommend to use

- Riak or Redis (key-value stores),
- Apache Cassandra (column stores),
- MongoDB (document stores), and
- Neo4J (graph stores)

for your project.

NewSQL Database Systems

NewSQL database systems attempt to provide NoSQL-like performance and scalability while preserving the ACID guarantees and offering a query language that is as powerful as SQL. The main driving force for NewSQL systems are new requirements of OLTP read-write workloads [11]:

- Much higher OLTP throughput
- Real-time analytics

Another motivation for NewSQL database systems is the fact that some applications cannot give up strong transactional and consistency guarantees. Nevertheless, they should perform and scale horizontally as well as NoSQL database systems do. Hence, NewSQL database systems are required to support a large number of concurrent transactions. According to [7], NewSQL database systems can be categorized into three main classes:

New Architectures New database systems built from scratch without relying on any legacy codebase. Typically, they operate in a shared-nothing environment and provide important features such as multi-node concurrency control, fault tolerance through replication, distributed query processing, and a tailored storage engine. A tailored storage engine allows the systems to optimize the network traffic: the query is sent, rather than the data. That is, the query *travels* to the data on which it operates.

Examples include Google Spanner [3], HyPer [6], SAP Hana [9], CockroachDB [11], H-Store [12], MemSQL [13], NuoDB [14], and VoltDB [15].

Transparent Sharding Middleware The database is divided into shards and the shards are distributed across many nodes in a cluster. Each node runs the same database systems. A (centralized) middleware coordinates queries, transactions, replication, and partitioning. The communication between nodes and middleware is done via an additional layer on top of the

[9] `https://neo4j.com/`
[10] `https://tinkerpop.apache.org/`
[11] `https://www.cockroachlabs.com/`
[12] `http://hstore.cs.brown.edu`
[13] `https://www.memsql.com/`
[14] `https://www.nuodb.com`
[15] `https://www.voltdb.com`

database system that is deployed on each node. Thus, for this class of NewSQL systems, the underlying database system remains untouched. This is considered the main advantage of this class of NewSQL systems: it is possible to replace an existing single-node database system without changing the application code. However, recall that most single-node database systems operate disk-based. However, scalability and performance may still be limited compared to *new architectures*, which often exploit new technologies.

Examples include AgilData [16] and MariaDB MaxScale [17].

Database-as-a-Service Essentially, these are NewSQL databases offered by cloud computing providers.

Examples include Amazon Aurora [18] and ClearDB [19].

Recommendation For NewSQL systems, we recommend *new architecture* systems. Currently, there are not many open-source/freely-available implementations of NewSQL database systems. For your project, we recommend to use H-Store, VoltDB (the commerical fork of H-Store), or CockroachDB.

Large-Scale Processing Frameworks

Lots of enterprises collect large amounts of data (Big Data). Therefore, new large-scale processing frameworks had to be developed that are able to handle this amount of data. Typically, large-scale processing frameworks operate in a distributed environment (a cluster) and provide a convenient way of processing distributed data, i.e., users need little knowledge about the underlying architecture to exploit the power of distributed computing. The *MapReduce* paradigm [4] is one popular example of such an abstraction.

Large-scale processing framework are usually classified into 2 categories.

Batch Processing Frameworks Batch processing frameworks operate on large a (but bounded), static dataset (a so-called *batch* and return only the result of *their* computation after processing the entire batch. As a consequence, batch processing is *stateful*. Aggregations (e.g., computating the average) are typical operations that fit batch processing. Consequently, pure batch processing systems are not applicable for highly interactive application scenario.

An example of a batch processing framework is Apache's Hadoop [20].

Stream Processing Frameworks Stream processing frameworks operate on an unbounded number of independent data items. In true stream processing frameworks, the data items *stream* into the system one after another and operations are defined on individual data items rather than on the entire dataset (cf. batch processing). The results of the operations are available immediately after processing a data item. Generally speaking, true stream processing is stateless. In contrast, there also exist so-called *micro-batch* processing frameworks that are still considered stream processing frameworks but process data in small batches of data items. That is, micro-batch processing systems also maintain a *micro-state*. Stream processing is well-suited for applications having (near) real-time requirements.

[16]`http://www.agildata.com/`
[17]`https://mariadb.com/products/technology/maxscale`
[18]`https://aws.amazon.com/rds/aurora/`
[19]`http://w2.cleardb.net/`
[20]`https://hadoop.apache.org/`

Examples include Apache Storm [21] and Apache Samza [22].

Hybrid Frameworks There also exist hybrid processing systems that are capable of handling batched and streamed data.

Examples include Apache Spark [23] and Apache Flink [24].

Recommendation We consider Apache Spark and Apache Flink to be the best choices for batch *and* stream processing.

References

[1] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's Globally-distributed Database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[5] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 7th edition, 2015.

[6] A. Kemper and T. Neumann. HyPer: A Hybrid OLTP amp;OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *2011 IEEE 27th International Conference on Data Engineering*, pages 195–206, April 2011.

[7] A. Pavlo and M. Aslett. What's Really New with NewSQL? *SIGMOD Rec.*, 45(2):45–55, Sept. 2016.

[8] E. Redmond and J. R. Wilson. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012.

[9] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 731–742, New York, NY, USA, 2012. ACM.

[10] A. Silberschatz, H. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5 edition, 2006.

[11] M. Stonebraker. New Opportunities for New SQL. *Commun. ACM*, 55(11):10–11, Nov. 2012.

[21] `https://storm.apache.org/`
[22] `https://samza.apache.org/`
[23] `https://spark.apache.org/`
[24] `https://flink.apache.org/`

[12] L. Wiese. *Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases.* DeGruyter, 2015.