Scalable Data Management: NoSQL Data Stores in Research and Practice

Felix Gessert, Norbert Ritter
Database and Information Systems Group
University of Hamburg
{gessert,ritter}@informatik.uni-hamburg.de

***Abstract*—The unprecedented scale at which data is consumed and generated today has shown a large demand for scalable data management and given rise to non-relational, distributed "NoSQL" database systems. Two central problems triggered this process: 1) vast amounts of user-generated content in modern applications and the resulting requests loads and data volumes 2) the desire of the developer community to employ problem-specific data models for storage and querying. To address these needs, various data stores have been developed by both industry and research, arguing that the era of one-size-fits-all database systems is over. The heterogeneity and sheer amount of these systems – now commonly referred to as NoSQL data stores – make it increasingly difficult to select the most appropriate system for a given application. Therefore, these systems are frequently combined in polyglot persistence architectures to leverage each system in its respective sweet spot.**

This tutorial gives an in-depth survey of the most relevant NoSQL databases to provide comparative classification and highlight open challenges. To this end, we analyze the approach of each system to derive its scalability, availability, consistency, data modeling and querying characteristics. We present how each system's design is governed by a central set of trade-offs over irreconcilable system properties. We then cover recent research results in distributed data management to illustrate that some shortcomings of NoSQL systems could already be solved in practice, whereas other NoSQL data management problems pose interesting and unsolved research challenges.

I. Introduction

Traditional relational database management systems (RDBMSs) provide powerful mechanisms to store and query structured data under strong consistency guarantees and have reached an unmatched level of reliability, stability and support through decades of development. In recent years, however, the amount of useful data in some application areas has become so vast, that it simply cannot be stored or processed by traditional database solutions [1]. User-driven content in social networks or data retrieved from large sensor networks are only two examples of this phenomenon commonly referred to as Big Data [2]. A class of novel data storage products able to cope with Big Data are subsumed under the term NoSQL databases, many of which offer horizontal scalability and higher availability than traditional relational databases or other useful properties by sacrificing querying options and consistency guarantees [3] .

There are dozens of NoSQL database systems, and it is very hard to keep an overview over what these systems provide, where they fail and where they differ. Beyond a mere presentation of prominent NoSQL representatives and their respective features, this tutorial intends to give an overview over the requirements typically posed to NoSQL database systems, the techniques used to fulfill these requirements and the trade-offs that have to be made in the process.

The main problem NoSQL database systems seek to solve, is providing storage and query capabilities for specific problem domains. This encompasses specialization in *data models*, *query languages*, *consistency*, *scalability* and *availability* properties, *transactional guarantees*, *schema management*, *low latency*, *analytical* and *real-time processing* as well as *durability*, *reliability* and *elasticity*. While the foundational techniques are often similar (e.g., asynchronous master-slave replication), the resulting systems exhibit very different behavior in both functional and non-functional aspects. Since very different requirements are often found in different parts of the same application, a recent trend is the consolidation of different database systems within a single application (*polyglot persistence*).

Our tutorial discusses the characteristics of NoSQL databases and introduces approaches proposed to address their challenges. Furthermore, we highlight open problems that provide opportunities for contributing to the emerging area of scalable data management and polyglot persistence.

II. Tutorial Outline

Our 1.5 hours tutorial is divided into fours parts and structured as follows.

As background (Section II-A), we recall the basics of distributed data management, in particular partitioning, replication, eventual consistency and the different NoSQL data models. We also present the most important impossibility results for distributed databases, in particular the widely used CAP theorem.

The core survey of NoSQL databases is outlined in Section II-B and covers the discussion and classification of the different systems. Each system is described in depth and relations to current research are given. We include open-source and commercial NoSQL systems, research systems as well as cloud-based database-as-a-service systems.

978-1-5090-2020-1/16/$31.00 © 2016 IEEE

The third part (Section II-C) reviews the integration of the discussed systems in polyglot persistence environments, in particular challenges and potential benefits. The final part of the tutorial (Section II-D) highlights open issues.

A. Foundations

The NoSQL movement [4] is motivated by *horizontal scalability* to leverage clusters of commodity hardware and minimization of the *impedance mismatch* between the data model of the application and the database. Horizontal scalability is addressed through *partitioning*[1] and *replication* [5], [6], [7]. Partitioning can be performed either *hash-based* or *range-based*, favoring either fast-lookups and even data distribution or efficient, single-site range queries. The applicability of hash-based partitioning that is now widely used in NoSQL databases is enabled by consistent hashing [8], [9]. Replication can be either performed synchronously or asynchronously, allowing a trade-off between low latency with higher availability and stronger consistency [7]. Quorum-based replication protocols allow tuning the desired level of consistency at runtime [10], [11], [12].

Consistency-Availability Trade-Off. Since neither serializability nor strong consistency are achievable when network partitions may occur [13] RDBMSs usually choose ACID over high availability [14], [15]. Brewer's *CAP theorem* states that of the three properties availability (A), consistency (C) and partition tolerance (P), any distributed database system can achieve at most two [16]. CAP thus classifies NoSQL database into the three categories AP, CP and CA. Though CAP has been criticized as being an overly coarse-grained consistency classification scheme [5], [17] it still succinctly captures whether a particular system focuses more on availability or consistency (*linearizability* [18]).

Eventual Consistency. Another important impossibility result governing the design and implementation of NoSQL stores is the *FLP theorem* [19]. It states that in an asynchronous network, no consensus protocol can guarantee termination if one or more nodes can fail by stopping. Since consensus protocols such as Paxos, Raft and ZAB are widely used in NoSQL systems for group membership, atomic commitment and conditional updates, these systems either have to relax availability or consistency guarantees [20], [21], [22]. Many models for *eventual consistency* have been proposed in the literature [23], [24], [25]. NoSQL systems adopting these relaxed models are usually referred to as *BASE* (Basically Available, Soft-State, Eventually Consistent) [26]. In eventually consistent NoSQL systems, strong safety guarantees can be achieved through automatic conflict resolution and conflict-free data types [27].

Data Models. NoSQL databases are typically classified according to their data model. Our focus lies on key-value, document and wide-column stores; other NoSQL database systems such as graph or object databases are not covered in detail. *Key-value stores* expose data through arbitrarily structured values accessible through a primary key. As the nature of the stored value is opaque to the database, pure key-value stores do not support operations beyond simple CRUD (Create, Read, Update, Delete) [9], [28]. *Document stores* are similar to key-value stores, but instead of arbitrary values, they only accept values with a certain structure, e.g. JSON documents [29], [30]. Through this restriction, document stores can arrange data in a way that allows performing complex queries [31]. *Wide-column stores* use a multi-level data model, in which lexicographically ordered row keys map to values categorized by column families, columns and timestamps, effectively forming a sparse, multi-dimensional map [32]. Wide-column stores achieve horizontal scalability similar to key-value stores while allowing for additional structure.

[1] also referred to as *sharding* or *fragmentation*

B. Survey and Classification of NoSQL Databases

Classification scheme. We categorize each system according to non-functional and functional properties that we map to the respective techniques that each NoSQL systems uses to achieve them. *Functional* persistence requirements we review are [33], [34], [3], [7]:

- ACID transactions with different isolation levels
- Atomic, conditional and/or set-oriented updates
- Queries: point access, scans, aggregation, selections, projections, joins, subqueries, map-reduce, graph queries, batch analytics, search, continuous queries
- Partial and/or commutative updates
- Data structures: graphs, lists, hash tables, trees, queues, documents, etc.
- Structured, semi-structured or implicit schemata
- Semantic integrity constraints

Surveyed *non-functional* requirements are [35], [36], [5], [37]:

- Throughput for reads, writes and queries
- Read and write latency
- Availability for writes and/or reads during network partitions and/or sever failures
- Scalability of data volume, reads, writes and queries
- Consistency guarantees: strong consistency, read-your-writes, causal consistency, monotonic reads, monotonic writes, writes follow reads, etc.
- Durability
- Elastic scale-out and/or scale-in

Dynamo-style systems. Many NoSQL systems are based on the design of the key-value store *Dynamo* [9] or the wide-column store Bigtable [32]. Dynamo achieves high availability through quorum-based replication with consistent hashing and is thus an AP-system. By allowing concurrent writes, it achieves high availability but has to allow inconsistent data. These inconsistencies are detected through vector clocks and exposed to the application, which has to perform semantic reconciliation. Through its fast asynchronous replication, *read repair* and *anti-entropy* mechanisms, Dynamo nonetheless quickly converges to a consistent state [11]. *Riak* [38] is a popular open-source database that extends Dynamo's model by search, analytics and conflict-free data types.

Bigtable-style systems. *Bigtable* (CP) is a wide-column store that employs range-partitioning to provide scan queries and an append-only I/O model to provide high write throughput [32]. It is built on the synchronously replicated Google File System (GFS) [39] and can hence guarantee row-level atomicity. As both GFS and Bigtable rely on a master to coordinate the distribution of data partitions, the system is susceptible to unavailability during network partitions and crashes of the master. *HBase* adopts the Bigtable model using the Hadoop ecosystem [40]. Cassandra leverages the data distribution model of Dynamo while providing the data model of Bigtable [41]. *MegaStore* is built on top of BigTable to provide multi-key transactions across data centers and a strict schema [42].

Other NoSQL databases. *MongoDB* is a replicated document store based on master-slave replication [43]. It provides relatively powerful query capabilities, document-level atomicity as well as hash- and range-based data partitioning. Similar to Bigtable, MongoDB's newest storage engine is based-on log-structured merge trees to support high write throughputs. The in-memory key-value store *Redis* is optimized for low latency [44]. This is achieved by storing data in main memory while replicating and logging updates according to configurable policies. Database-as-a-Service (DBaaS) systems like *Azure Tables* [45] and can be categorized similarly. Aspects that are particularly relevant for a DBaaS are service-level agreements, multi-tenancy and workload management [3], [46], [47].

C. Polyglot Persistence Architectures

The architectural pattern *polyglot persistence* describes the use of different databases for different requirements [1]. The key idea is that designing applications without a monolithic database can tremendously increase developer productivity and performance. Three different types of polyglot persistence can be found in research and industry approaches. In the *application-coordinated polyglot persistence* pattern the application is solely responsible for distributing data and queries to specialized databases. With the *microservices* pattern, storage decisions are encapsulated by loosely coupled services. *Polyglot database services* offer database APIs and transparently forward requests based on distribution rules.

D. Open Challenges

There are many open challenges for NoSQL data management. NoSQL systems need to support novel application architectures (e.g., single-page or real-time apps) and deliver low latency in face of distributed storage and application tiers. There are currently no means to turn the functionality-performance trade-off into a tunable runtime configuration. Polyglot database services lack the capability to automate, optimize and learn the best choice of given database systems. They can neither route queries and data to minimize SLA violations nor preserve consistency and transaction guarantees.

III. Intended Audience

The target audience is anyone with interest in learning the trade-offs provided by NoSQL data stores. We expect the tutorial to appeal to a large portion of the ICDE community:

- Students who are looking for novel research topics and orientation
- Experienced Researchers in the fields of database systems, cloud computing or distributed systems interested in open challenges and a comparative overview of the NoSQL landscape
- Industry practitioners tackling data management problems who are looking for a survey and classification of existing systems and their respective sweet spots

The tutorial is aimed at balancing between practical aspects (e.g., APIs and deployments models) and research approaches (e.g., novel architectures and transaction protocols) so that both researchers and practitioners gain insights into the challenges of the respectively other side.

IV. Relationship To Prior Tutorials

A related tutorial was held at the BTW 2015 database conference [48] on non-relational cloud data management in March 2015. Unlike this tutorial, there was a stronger focus on cloud database systems and differences in their architectures and SLAs. We incorporate the fundamental insights from that prior tutorial but extend it with a more rigorous and extensive treatment of the NoSQL data stores themselves. We previously presented the practical aspects of NoSQL systems in several internal conferences at software companies as well as different graduate and undergraduate courses at the University of Hamburg. Though this tutorial also discusses APIs and programming models the stronger focus lies on reliability, consistency and performance properties as well as approaches for polyglot persistence. To the best of our knowledge there are currently neither books nor other other tutorials that systematically categorize and comprehensively survey the vast landscape of NoSQL database systems.

References

[1] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

[2] D. Laney, "3D data management: Controlling data volume, velocity, and variety," META Group, Tech. Rep., February 2001.

[3] W. Lehner and K.-U. Sattler, *Web-Scale Data Management for the Cloud*. Springer, 2013.

[4] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[5] M. Kleppmann, *Designing Data-Intensive Applications*, 1st ed. O'Reilly Media, Jan. 2016.

[6] B. Lindsay, P. Seilinger, C. Galtieri, J. Gray, R. Lorie, T. Price, F. Putzulo, I. Traiger, and B. Wade, "Notes on Distributed Databases," 1980.

[7] M. T. Ozsu and P. Valduriez, *Principles of distributed database systems*. Springer-Verlag New York Inc, 2011.

[8] D. R. Karger, E. Lehmanl, T. Leightonl, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *ACM Symposium on Theory of Computing*, 1997, pp. 654–663.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SOSP*, ser. 17, vol. 14, 2007, pp. 205–220.

[10] D. K. Gifford, "Weighted voting for replicated data," in *ACM Symposium on Operating Systems Principles*, 1979, pp. 150–162.

[11] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, "Quantifying eventual consistency with PBS," *The VLDB Journal*, vol. 23, no. 2, pp. 279–302, Apr. 2014.

[12] D. Bermbach, *Benchmarking Eventually Consistent Distributed Storage Systems*. Karlsruhe, Baden: KIT Scientific Publishing, 2014.

[13] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in a partitioned network: a survey," *ACM Computing Surveys (CSUR)*, vol. 17, no. 3, pp. 341–370, 1985.

[14] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era:(it's time for a complete rewrite)," in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 1150–1160.

[15] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Scalable Atomic Visibility with RAMP Transactions," in *ACM SIGMOD Conference*, 2014.

[16] S. Gilbert and N. A. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[17] D. J. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, no. 2, pp. 37–42, 2012.

[18] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 463–492, 1990.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[20] L. Lamport, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[21] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," *Draft of October*, vol. 7, 2013.

[22] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems." in *USENIX Annual Technical Conference*, vol. 8, 2010, p. 9.

[23] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[24] A. Adya, "Weak consistency: a generalized theory and optimistic implementations for distributed transactions," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[25] S. Friedrich, W. Wingerath, F. Gessert, and N. Ritter, "NoSQL OLTP Benchmarking: A Survey," in *DMC 2014*, ser. LNI, E. Pldereder, L. Grunske, E. Schneider, and D. Ull, Eds., vol. 232. GI, 2014, pp. 693–704.

[26] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, *Cluster-based scalable network services*. ACM, 1997, vol. 31, no. 5.

[27] M. Shapiro, N. Pregui\cca, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," 2011.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[29] "CouchDB," http://couchdb.apache.org/.

[30] "RethinkDB," https://www.rethinkdb.com/.

[31] L. Qiao, K. Surlaker, S. Das, T. Quiggle, B. Schulman, B. Ghosh, A. Curtis, O. Seeliger, Z. Zhang, A. Auradar, and others, "On brewing fresh espresso: LinkedIn's distributed data serving platform," in *Proceedings of the 2013 international conference on Management of data*. ACM, 2013, pp. 1135–1146.

[32] 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," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[33] M. Schaarschmidt, F. Gessert, and N. Ritter, "Towards Automated Polyglot Persistence," in *Datenbanksysteme fr Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme"*, 2015.

[34] U. Strl, T. Hauf, M. Klettke, S. Scherzinger, and O. T. H. Regensburg, "Schemaless NoSQL Data StoresObject-NoSQL Mappers to the Rescue?" in *Proc. BTW*, vol. 15, 2015.

[35] E. Brewer, "CAP twelve years later: How the" rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[36] S. Babu and H. Herodotou, "Massively parallel databases and mapreduce systems." *Foundations and Trends in Databases*, vol. 5, no. 1, pp. 1–104, 2013.

[37] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigümüş, "Intelligent management of virtualized resources for database systems in cloud environment," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 87–98.

[38] "Riak," http://docs.basho.com/riak/latest/.

[39] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, 2003, pp. 29–43.

[40] "HBase," https://hbase.apache.org/.

[41] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[42] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, J.-M. Lon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services." in *CIDR*, vol. 11, 2011, pp. 223–234.

[43] "MongoDB," https://www.mongodb.org/.

[44] "Redis," http://redis.io/.

[45] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, and others, "Windows Azure Storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[46] F. Gessert, F. Bucklers, and N. Ritter, "Orestes: A scalable database-as-a-service architecture for low latency," in *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 - April 4, 2014*. IEEE, 2014, pp. 215–222.

[47] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational cloud: A database-as-a-service for the cloud," in *Proc. of CIDR*, 2011.

[48] F. Gessert and N. Ritter, "Skalierbare NoSQL- und Cloud-Datenbanken in Forschung und Praxis," in *Datenbanksysteme für Business, Technologie und Web (BTW 2015) - Workshopband, 2.-3. März 2015, Hamburg, Germany*, ser. LNI, vol. 242. GI, 2015, pp. 271–274.

BIOGRAPHIES

Felix Gessert is a Ph.D. student at the databases and information systems group at the University of Hamburg. His main research fields are scalable database systems, transactions and web technologies for cloud data management. His thesis addresses caching and transaction processing for low latency mobile and web applications. He is also founder and CEO of the startup Baqend that implements these research results in a cloud-based backend-as-a-service platform. Since their product is based on a polyglot, NoSQL-centric storage model, he is very interested in both the research and practical challenges of leveraging and improving these systems. He is frequently giving talks on different NoSQL topics.

Norbert Ritter is a full professor of computer science at the University of Hamburg, where he heads the databases and information systems group. He received his Ph.D. from the University of Kaiserslautern in 1997. His research interests include distributed and federated database systems, transaction processing, caching, cloud data management, information integration and autonomous database systems. He has been teaching NoSQL topics in various database courses for several years. Seeing the many open challenges for NoSQL systems, he and Felix Gessert have been organizing the annual SCDM[2] workshop for three years to promote research in this area.

[2]Scalable Cloud Data Management Workshop: www.scdm2015.com