Chapter 10

Architectures

A DBMS is a complex system that actually has to serve different purposes and hence provide multiple and independent functionalities. Thus, it can neither be studied nor built monolithically as an atomic unit. Oppositely, there are different software inter-dependent components that interact in different ways to achieve the global purpose. Similarly to DBMS, in a Big Data system, we have to understand how our system is going to collect data; how these are going to be used; where they are going to be stored; how they are going to be related to the corresponding metadata; if we are going to use any kind of master data, where these will come from and how they will be integrated; how are the data going to be processed; how replicas are going to be managed and their consistency guaranteed; etc. All these functionalities will be performed by different, independent components, and it is crucial to understand and establish how they interact. The blueprint of components of a software system[1] is known as system architecture ([Kru95] shows different kinds of such blueprints, depending on the information they provide).

10.1 Functionalities of a DBMS

If we want to study the functional components required to manage data, we firstly need to see which are the required functionalities. Thus, a DBMS is defined in [ACPT99] as a software system able to manage collections of data that are large, shared and persistent, and to ensure their reliability and privacy. Like any software product, a DBMS must be efficient and effective.

Out of the many functionalities provided by a DBMS, we highlight:

Storage implies safely keeping big amounts of data for long periods of time, which assumes the existence and usage of formats/structures that facilitate query and modification (e.g., hash, B-tree, LSM-tree, heap file), as well as mechanisms to guarantee durability.

Modelling includes allowing users to specify the logical structure of data and arrange them within the available structures (e.g., normalization, partitioning).

Ingestion refers to insert/upload data to the system (e.g., ORACLE SQL*Loader), including mechanisms to provide atomicity, guarantee isolation and avoid interferences between users.

Processing provides specific means to manipulate data (e.g., PL/SQL).

Querying/fetching allows users to efficiently retrieve data (e.g., SQL, relational algebra).

We can see a Big Data Management System as a DBMS that has to provide exactly those same functionalities, but at the same time face the specific challenges of Big Data.

10.1.1 Volume: Distribution and parallelism

The first of the three "V"s representing the main challenges of Big Data is for Volume. As we have already discussed, this entails distributing the data and parallelizing the processing. From the architectural point of view,

[1] `https://en.wikipedia.org/wiki/Component_(UML)`

the main consequence of this is that the components that we use must allow to natively deal with them (e.g., HDFS and Spark). It would not be a good idea to try to build such mechanisms on top of centralized systems as part of our project, because of their complexity. However, it is important to be aware of the limitations of the specific components being used (e.g., we could add a wrapper to split full table scans in HBase in different intervals to overcome its limitations in offering parallelism) and avoid creating communications bottlenecks (e.g., using external libraries or different execution environments generates data transfer between them). Also, we should think of the consequences of our decisions to avoid unnecessarily wasting resources (e.g., replica sets in MongoDB are a multiplying factor of the number of machines required).

10.1.2 Velocity: Batch vs Stream processing

The second "V", standing for Velocity, highlights the challenge created by a high arrival rate of data. Nevertheless, this does not mean everything must be done in real time. Indeed, some execution flows do not feel such pressure and do not require extremely low latency (i.e., response time). However, this does not mean we should not aim to high efficiency and throughput (i.e., processing as much data as possible per time unit). It is important to realize that, as it is for humans, executing batches is usually much more efficient and easier to do also for computers.

Unfortunately, some processes do require an extremely short time of reaction and cannot work in batches. For this, latency is a priority and throughput can be sacrificed to some extent. This means that in many projects both batch and stream processing have to coexist in the same architecture, which complicates it in terms of number of components, and also hinders their communication and data sharing, as well as the consistency of independent processing branches.

10.1.3 Variety: Kinds of data

BigBench, presented in [RFD$^+$15], defines a benchmark representative of real uses cases of Big Data. We can observe that the main differences with traditional data intensive projects are (i) the presence of external sources and (ii) the relevance of non-structured data (i.e., not typed and not tabular, which today represents the majority of data being generated). These correspond to the third "V" of Big Data. This does not mean that internal and structured data are not relevant anymore. It just means, we have new difficulties to overcome and benefits to obtain.

Michele Nemschoff classifies data external to the organization in seven categories:[2]

Structured Data

- **Created Data** are just those that businesses purposely create, generally for market research (e.g., customer surveys or focus groups, created by loyalty programmes).
- **Provoked Data** defined by Forbes as those "Giving people the opportunity to express their views" (e.g., star rates).
- **Transacted Data** are those businesses collect on every transaction completed (e.g., purchase order, but also web clicks).
- **Compiled Data** are databases collected by some companies to better profile customers (e.g., credit scores, location, demographics).
- **Experimental Data** are created when businesses experiment with different marketing pieces and messages to see which are most effective with consumers (e.g., A/B test).

Unstructured Data

- **Captured Data** are created passively due to a person's behavior (e.g., keywords posed in a search engine, or GPS positions generated by our smartphones).
- **User-generated Data** consist of all of the data individuals are putting on the Internet every day (e.g., tweets, Facebook posts, comments on news stories, videos on YouTube).

It is obvious that to make use of all these kinds of information sources, we need specific and specialized architectural components that will have to interact with many others to transform such complex data into actionable knowledge.

[2]`http://smartdatacollective.com/7-important-types-big-data`

10.2 Big Data Architectures

Those challenges require a complete reconsideration of classical DBMS architecture and components that date back to the 70s, and are based on the System-R project, introduced in [ABC$^+$76]. Thus, the approach taken by the databases community has been that of developing independent components addressing each one of the required functionalities (i.e., Storage, Modeling, Ingestion, Processing, and Querying) as efficiently as possible. For this, they implement well-known techniques (like distributed trees, dynamic hashing, or columnar storage), but leveraging on new hardware and Cloud technologies. This way, several big players (e.g., Google, Amazon, Facebook) independently implemented different tools for diverse purposes, resulting in the consequent heterogeneity and interoperability problems (luckily, some of these proprietary tools were reimplemented with the same specifications, but in the form of open source software by the Apache foundation, under the Hadoop project).

In complex projects, where many of these tools need to be connected, it is definitely not wise to do it arbitrarily. Such erroneous approach would give rise to what is known as an "spaghetti architecture".[3] Alternatively, there are some architectural patterns already studied and implemented in many other projects that we can follow to address new Big Data challenges.

First of all, it is good to remember the centralized RDBMS architecture in Figure 3.4. It is easy to identify there the Querying component (corresponding to the Query Manager), the Processing component (corresponding to the Execution Manager and Scheduler), and underneath the Storage Component (corresponding to the Data Manager working on the Operating System).

Similarly, we can identify in Figure 3.5 the same Storage component, but now the Querying and Processing ones are a bit more complex, because they both require Global and Local managers (not for the Scheduler, which remains only global). Relevantly, we need to highlight the importance of Modeling and managing metadata in a distributed architecture, requiring both a Global and Local catalog.

In a similar way, we can group the different components of a Big Data architecture, where we will have to find some devoted to Storage of the huge amount of data, some to Modeling and metadata management to cope with distribution and heterogeneity, some to Ingestion of the data from the diverse and external sources, some to the Processing in both batch and stream fashion, and some to Querying and exposing results to users in a friendly and efficient manner. Nevertheless, there are tens of different tools for the management of Big Data, each of them with very concrete functionalities optimal for different use cases. Thus, we need to choose the most appropriate ones depending on our concrete requirements (you can find examples of Big Data tools selection in [NHR$^+$17]). This creates an added problem to orchestrate the execution of all those heterogenous components. Hence, we need to consider also some new component (not explicitly present in RDBMS) that helps to coordinate the execution of complex jobs involving several tools.

Possible choices inside the boundaries of the Hadoop ecosystem would be:

Storage $\rightarrow$ HDFS & HBase

Modeling $\rightarrow$ HCatalog

Ingestion $\rightarrow$ Sqoop

Processing $\rightarrow$ Spark

Querying $\rightarrow$ Spark SQL

Orchestrator $\rightarrow$ Oozy

This simple classification of tools can help us to better understand the components we need and the choices we have to make. However, we still have to see how they are to be connected.

10.2.1 New storage pattern

The first difference we find between traditional analytics and Big Data is that the former works on predefined and well established Key Performance Indicators (KPI), sometimes defined at the company level or even by some external regulatory institution. Thus, given their existence *a priori*, it is possible to firstly group them by subject (a.k.a. Fact) and also know which are main factors that can influence them (a.k.a. Dimensions). Subject of

[3]`https://www.confluent.io/blog/event-streaming-platform-1`

analysis and their dimensions conform what is known as the multidimensional model (see [AR18]), and results in Star Schemas, which are typically used to integrate and shape the cleaned data in the Data Warehouse.

Oppositely, Big Data analysis is much more exploratory and, far from being pre-defined, finding predictors is part of the problem itself. Therefore, integrating and structuring the data *a priori* in one way or another would be a mistake, and a waste of time and resources. Instead, it is better to simply keep the raw data as they are found in the sources in what is called a Data Lake (roughly a collection of independent files). Then, whenever a data scientist requires some of these data for a concrete purpose, we can spend the time to integrate, clean and structure them in the right format and schema for the problem at hand. This is known as the "Load-first, Model-later" approach.

The risk with this approach is that files can be simply massively accumulated without any order, resulting in what is called a "Data Swamp", where just finding the relevant data would be a challenge. The solution for this is creating some organization of files and semantically annotate them with metadata. Thus, to the already existing mappings in the catalog, we should add links from each file to a domain ontology[4] containing the relevant concepts for our business. In this way, users would be able to perform guided searches over the ontology instead of blindly navigating the files. If properly done, a semantic approach can even facilitate automation of integration and queries (see [NRA+19]).

10.2.2 New processing pattern

The Data Warehousing architecture composed of Sources-ETL-DataWarehouse-DataMarts is the most used to serve descriptive analytics. However, it is well known that it falls short for predictive analytics. Indeed, descriptive analytics typically study how the business is going at different levels of granularity (e.g., regions, cities or districts), and how this has evolved with time. Timeliness of data is usually not an issue for long term trends, and days or even weeks are acceptable for the current data to be processed and made ready for the analysis. Oppositely, predictive analytics typically try to foresee how a given entity (e.g., customer) is going to behave in the near future. Obviously, since the purpose of a prediction is to react or at least be ready to take some action, data freshness and response time is typically crucial in this case. Minutes or even seconds are relevant.

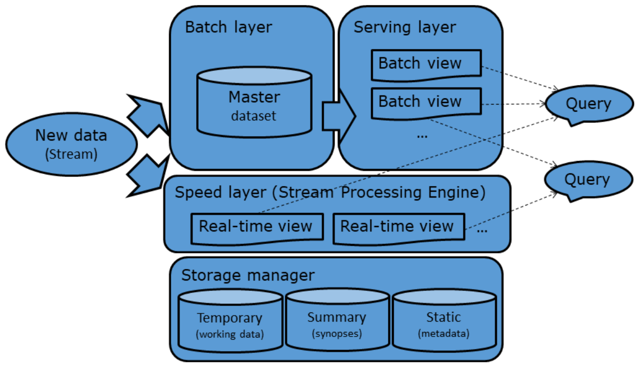

Figure 10.1: λ-architecture, [MW15]

Consequently, since requirements are completely contradictory, we have to distinguish both precessing flows (which will be used and involve different software components), giving rise to what is known as λ-Architecture.[5] This, depicted in Figure 10.1, consists of two execution branches fed from the same sources. The one on top focuses on batch processing (offering a serving layer facilitating creation and maintenance of views), and the one underneath focuses on stream processing (relying on sliding window mechanisms, synopses and other kinds of statistics).

Nevertheless, maintaining such potentially redundant flows generates some management risks. To understand where the problem lays, we should pay attention to Figure 10.2 showing the different phases and processes data go through. Firstly, we have raw data coming from our source. Then, these data are transformed including formatting, normalization, cleaning and potentially integration (if we had more than one source). This generates processed data that are then prepared for a specific algorithm (e.g., using one-hot encoding, or some kind of discretization).

[4]`https://en.wikipedia.org/wiki/Ontology_(information_science)`
[5]The name comes from the resemblance of the divergent flows and the shape of the Greek letter.

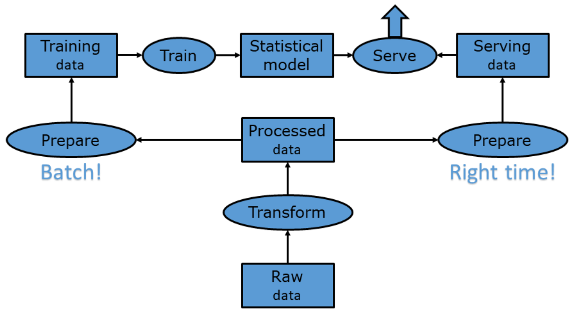

Figure 10.2: Model building and using

Finally, a statistical model is created by some machine learning algorithm. This usually includes not only the training of the model, but also some kind of evaluation that potentially triggers further iterations of the training, preparation, or even transformation tasks (not depicted in the figure for the sake of simplicity). Eventually, the model will be accepted and pushed to production. At this moment, we switch from batch and iterative processing of the whole dataset (i.e., LHS of the figure) to evaluation of one tuple at a time (i.e., RHS of the figure), in order to predict its associated value according to the previously generated model, so it can be used in the corresponding business flow. It is obvious, that for the prediction to be accurate, the new arriving tuples have to go through exactly the same transformation and preparation tasks as the training data went through (otherwise, the validity of the prediction would be compromised). Consequently, our architecture must guarantee that and avoid potential bugs and inconsistences in sibling data flows.

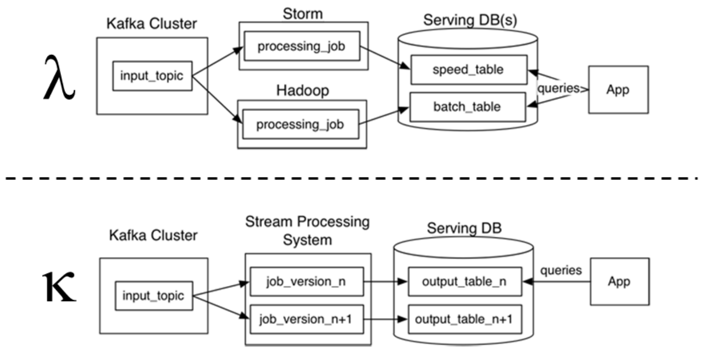

Figure 10.3: Comparison of the λ and κ architectures[6]

As an evolution of the λ-architecture, Jay Kreps introduced the κ-architecture,[6] whose comparison we can see in Figure 10.3. Firstly, the data in a κ-architecture is an append-only immutable log. From the log, data is streamed through a computational system and fed into auxiliary stores for serving. As shown in the figure, the problem with the λ-architecture is that we have to maintain the implementation of data transformations in different systems (e.g., Storm and Hadoop) depending on whether we require streaming or batch, and this is clearly prone to inconsistences between them. Thus, κ-architecture is a simplification, where we only have a single execution engine (hence a single implementation of the transformations). The batch processing is replaced by simply playing the data through the streaming system quickly. If for some reason, we require different versions of the transformations for different statistical models, we can keep all of them in the same system and choose the most appropriate one at every moment, independently of whether it is for training or production.

[6]`https://www.oreilly.com/radar/questioning-the-lambda-architecture`

Bibliography

[ABC$^+$76] Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim Gray, Patricia P. Griffiths, W. Frank King III, Raymond A. Lorie, Paul R. McJones, James W. Mehl, Gianfranco R. Putzolu, Irving L. Traiger, Bradford W. Wade, and Vera Watson. System R: relational approach to database management. *ACM Trans. Database Syst.*, 1(2):97–137, 1976.

[Abi97] S. Abiteboul. Querying Semi-Structured Data. In *ICDT*, 1997.

[ACPT99] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, and Riccardo Torlone. *Database Systems - Concepts, Languages and Architectures*. McGraW-Hill International, 1999.

[AG08] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1995.

[AKM$^+$16] Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. Designing access methods: The RUM conjecture. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, pages 461–466. OpenProceedings.org, 2016.

[Amb03] S. Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. Wiley& Sons, 2003.

[AMH08] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 967–980. ACM, 2008.

[AMR$^+$11] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management*. Cambridge University Press, 2011.

[AR18] Alberto Abelló and Oscar Romero. *Online Analytical Processing*. Springer, 2018.

[BL11] Antonio Badia and Daniel Lemire. A call to arms: revisiting database design. *SIGMOD Rec.*, 40(3):61–69, 2011.

[BM03] Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003.

[Car75] Alfonso F. Cardenas. Analysis and performance of inverted data base structures. *ACM Commun.*, 18(5):253–263, 1975.

[Cat10] Rick Cattell. Scalable SQL and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, 2010.

[CDG$^+$06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[CDKB11] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[CIF02] *Corporate Information Factory*. John Wiley&Sons, 2002.

[CK85] George P. Copeland and Setrag Khoshafian. A decomposition storage model. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pages 268–279. ACM, 1985.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Comm. ACM*, 13(6):377–387, 1970.

[CZ14] C. L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.*, 275:314–347, 2014.

[DG92] David J. DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150. USENIX Association, 2004.

[Dha13] Vasant Dhar. Data science and prediction. *Commun. ACM*, 56(12):64–73, 2013.

[DN14] Christos Doulkeridis and Kjetil Nørvåg. A survey of large-scale analytical query processing in mapreduce. *VLDB J.*, 23(3):355–380, 2014.

[FCKK20] Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. A survey on the evolution of stream processing systems. *CoRR*, abs/2008.00842, 2020.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP*, pages 29–43. ACM, 2003.

[GGR16] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 2016.

[GMUW09] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, 2009.

[GÖ03] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[GRH$^+$13] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: towards an industry standard benchmark for big data analytics. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1197–1208. ACM, 2013.

[Gun07] Neil J. Gunther. *Guerrilla capacity planning - a tactical approach to planning for highly scalable applications and services*. Springer, 2007.

[HAR16] Victor Herrero, Alberto Abelló, and Oscar Romero. NOSQL design for analytical workloads: Variability matters. In Isabelle Comyn-Wattiau, Katsumi Tanaka, Il-Yeol Song, Shuichiro Yamamoto, and Motoshi Saeki, editors, *Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*, volume 9974 of *Lecture Notes in Computer Science*, pages 50–64, 2016.

[HAVZ20] Moditha Hewasinghage, Alberto Abelló, Jovan Varga, and Esteban Zimányi. Docdesign: Cost-based database design for document stores. In Elaheh Pourabbas, Dimitris Sacharidis, Kurt Stockinger, and Thanasis Vergoulis, editors, *SSDBM 2020: 32nd International Conference on Scientific and Statistical Database Management, Vienna, Austria, July 7-9, 2020*, pages 27:1–27:4. ACM, 2020.

[HCC$^+$13] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. More effective distributed ML via a stale synchronous parallel parameter server. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 1223–1231, 2013. `http://www.cs.cmu.edu/%7Eqho/ssp_nips2013.pdf`.

[HNA20] Moditha Hewasinghage, Sergi Nadal, and Alberto Abelló. On the performance impact of using json, beyond impedance mismatch. In Jérôme Darmont, Boris Novikov, and Robert Wrembel, editors, *New Trends in Databases and Information Systems - ADBIS 2020 Short Papers, Lyon, France, August 25-27, 2020, Proceedings*, volume 1259 of *Communications in Computer and Information Science*, pages 73–83. Springer, 2020.

[Jar77] Donald Jardine. *The ANSI/SPARC DBMS Model*. North-Holland, 1977.

[Joh09] Ryan Johnson. Pipeline. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 2116–2117. Springer US, 2009.

[JQD11] Alekh Jindal, Jorge-Arnulfo Quiané-Ruiz, and Jens Dittrich. Trojan data layouts: right shoes for a running elephant. In *SOCC*, page 21, 2011.

[Kip15] Scott Kipp. Will ssd replace hdd? `http://www.ieee802.org/3/CU4HDDSG/public/sep15/Kipp_CU4HDDsg_01a_0915.pdf`, 2015.

[Kru95] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, 1995.

[KSB$^+$99] David R. Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. *Comput. Networks*, 31(11-16):1203–1213, 1999.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[LBKS16] Arezki Laga, Jalil Boukhobza, Michel Koskas, and Frank Singhoff. Lynx: a learning linux prefetching mechanism for SSD performance model. In *NVMSA*, pages 1–6, 2016.

[MAR$^+$20] Rana Faisal Munir, Alberto Abelló, Oscar Romero, Maik Thiele, and Wolfgang Lehner. A cost-based storage format selector for materialized results in big data frameworks. *Distributed Parallel Databases*, 38(2):335–364, 2020.

[MB11] Erik Meijer and Gavin M. Bierman. A co-relational model of data for large shared data banks. *Commun. ACM*, 54(4):49–58, 2011.

[Moh13] Chandrasekaran Mohan. History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. In *EDBT*, 2013.

[MRA$^+$16] Rana Faisal Munir, Oscar Romero, Alberto Abelló, Besim Bilalli, Maik Thiele, and Wolfgang Lehner. Resilientstore: A heuristic-based data format selector for intermediate results. In *MEDI*, volume 9893 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 2016.

[MW15] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, 2015.

[NHR$^+$17] Sergi Nadal, Victor Herrero, Oscar Romero, Alberto Abelló, Xavier Franch, Stijn Vansummeren, and Danilo Valerio. A software reference architecture for semantic-aware big data systems. *Inf. Softw. Technol.*, 90:75–92, 2017.

[NRA$^+$19] Sergi Nadal, Oscar Romero, Alberto Abelló, Panos Vassiliadis, and Stijn Vansummeren. An integration-oriented ontology to govern evolution in big data ecosystems. *Inf. Syst.*, 79:3–19, 2019.

[OCGO96] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[Olt20] Dan Olteanu. The relational data borg is learning. *Proc. VLDB Endow.*, 13(12):3502–3515, 2020.

[ÖV20] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, 4th Edition*. Springer, 2020.

[Pit18] Evaggelia Pitoura. Pipelining. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.

[PRS$^+$16] F. Pezoa, J. L. Reutter, F. Suárez, M. Ugarte, and D. Vrgoc. Foundations of JSON Schema. In *WWW*, 2016.

[PSV$^+$08] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and Nils-Erik Frantzell. Meshing streaming updates with persistent data in an active data warehouse. *IEEE Trans. Knowl. Data Eng.*, 20(7):976–991, 2008.

[PZ11] Hasso Plattner and Alexander Zeier. *In-Memory Data Management*. Springer, 2011.

[RFD$^+$15] Tilmann Rabl, Michael Frank, Manuel Danisch, Hans-Arno Jacobsen, and Bhaskar Gowda. The vision of bigbench 2.0. In Asterios Katsifodimos, editor, *Proceedings of the Fourth Workshop on Data analytics in the Cloud, DanaC 2015, Melbourne, VIC, Australia, May 31 - June 4, 2015*, pages 3:1–3:4. ACM, 2015.

[RHAF15] Oscar Romero, Victor Herrero, Alberto Abelló, and Jaume Ferrarons. Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem. *Inf. Syst.*, 54:336–356, 2015.

[SAB$^+$05] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: A column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 553–564. ACM, 2005.

[SF12] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition, 2012.

[SS20] Stefanie Scherzinger and Sebastian Sidortschuck. An Empirical Study on the Design and Evolution of NoSQL Database Schemas. *CoRR*, abs/2003.00054, 2020.

[Sto08] Michael Stonebraker. Technical perspective - one size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12):76, 2008.

[TADP21] Ciprian-Octavian Truic, Elena-Simona Apostol, Jrme Darmont, and Torben Bach Pedersen. The forgotten document-oriented database management systems: An overview and benchmark of native xml dodbmses in comparison with json dodbmses. *Big Data Research*, 25:100205, Jul 2021.

[Tan18] Kian-Lee Tan. *Distributed Database Systems*, pages 894–896. Springer US, 2018.

[TCGM17] Ran Tan, Rada Chirkova, Vijay Gadepally, and Timothy G. Mattson. Enabling query processing across heterogeneous data models: A survey. In Jian-Yun Nie, Zoran Obradovic, Toyotaro Suzumura, Rumi Ghosh, Raghunath Nambiar, Chonggang Wang, Hui Zang, Ricardo Baeza-Yates, Xiaohua Hu, Jeremy Kepner, Alfredo Cuzzocrea, Jian Tang, and Masashi Toyoda, editors, *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 3211–3220. IEEE Computer Society, 2017.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[W3C04] W3C. *Extensible Markup Language (XML) 1.0*, 3rd edition, February 2004.

[Zah13] Matei A. Zaharia. *An Architecture for and Fast and General Data Processing on Large Clusters*. PhD thesis, University of California, Berkeley, USA, 2013.

Appendix A

Acronyms

ACID Atomicity, Consistency, Isolation and Durability

BASE Basically Available, Soft state, Eventual consistency

BSP Bulk Synchronous Parallel model

CSV Comma-separated Values

DAG Directed Acyclic Graph

DB Database

DBA Database Administrator

DBMS Database Management System

DE Data Engineering

DFS Distributed File System

DS Data Science

DW Data Warehousing

FK Foreign Key

FS File System

HTAP Hybrid Transaction/Analytical Processing

I/O Input/Output

JVM Java Virtual Machine

KPI Key Performance Indicator

KV Key/Value

LAN Local Area Network

LHS Left Hand Side

LRU Least Recently Used

NOSQL Not Only SQL

OLAP On-Line Analytical Processing

OLTP On-Line Transactional Processing

OS Operating System

PK Primary Key

RDBMS Relational DBMS

RDD Resilient Distributed Dataset

RHS Right Hand Side

SQL Structured Query Language

WAN Wide Area Network