



Deliverable D1: Survey

Coupled Storage System for Efficient Management of Self-Describing Data Formats (CoSEMoS)

Kira Duwe
kira.duwe@ovgu.de

Michael Kuhn
michael.kuhn@ovgu.de

November 15, 2021

Contents

1. Introduction	2
2. File Systems	2
2.1. Kernel File Systems	3
2.2. User Space File Systems	4
2.3. Miscellaneous Systems	5
3. Evolution of Databases	7
3.1. Relational Queries vs. Non-Relational Queries	9
3.2. Data Models	9
4. Relational Database Systems	10
5. Non-Relational Database Systems	12
5.1. Key-Value Stores	14
5.2. Wide-Column Stores	16
5.3. Document Stores	17
5.4. Graph Stores	18
5.5. Time-Series DBMS	19
6. Object Stores	20
References	21

1. Introduction

This survey contains the state of the art in parallel file systems and database technology, with the latter being focused on structured storage (T2.1, T2.2).¹ It will provide a substantial basis for selecting appropriate representatives for CoSEMoS. The following introduction and section two are excerpts from our storage survey [Lüttgau et al., 2018]. The text is complemented with newer developments as well.

In current supercomputers, storage is typically provided by parallel distributed file systems for hot data and tape archives for cold data. These file systems are often compatible with local file systems due to their use of the POSIX interface and semantics. It eases development and debugging because applications can easily run both on workstations and supercomputers. There is a wide variety of file systems to choose from, each tuned for different use cases and implementing various optimizations. However, the overall application performance is often held back by I/O bottlenecks due to the insufficient performance of file systems or I/O libraries for highly parallel workloads. Performance problems are dealt with by using novel storage hardware technologies as well as alternative I/O semantics and interfaces. These approaches have to be integrated into the storage stack seamlessly to make them convenient to use. Upcoming storage systems abandon the traditional POSIX interface and semantics in favor of alternative concepts such as object and key-value storage; moreover, they heavily rely on technologies such as NVRAM and burst buffers to improve performance. Additional tiers of storage hardware will increase the importance of hierarchical storage management.

2. File Systems

Providing reliable, efficient, and easy-to-use storage and file systems is one of the main issues today in HPC because a wide variety of scientific applications produces and analyzes enormous data volumes. File systems offer an interface to the underlying storage device. They link an identifier such as the file name to the corresponding physical addresses of the storage hardware. Thereby, more comfortable and simplified use of storage devices is enabled. Traditionally, they realize the concept of hierarchical structuring through the use of directories and files. Besides the actual file content, metadata such as the file size and access times are managed. Over the years, several file systems have been proposed and established, offering a wide range of functionality. Especially HPC systems deploy parallel distributed file systems, allowing to spread data across numerous storage devices and combining the particular features to increase the throughput as well as the system's capacity. However, due to the rapidly increasing data sizes, more sophisticated and specialized approaches are required for handling the enormous amount of information. At the same time, new and more powerful storage and network technologies are developed, posing challenges to exploit the respective capabilities. Besides the old file system concepts, other approaches have found their way into HPC systems.

Figure 1 gives an overview of a typical HPC system with several different storage systems and technologies. Hence, the need for high-throughput concurrent read and write capabilities of HPC applications led to the development of parallel and distributed file systems.

¹For an overview of the project's work packages and tasks, please see Deliverable D1: Report.

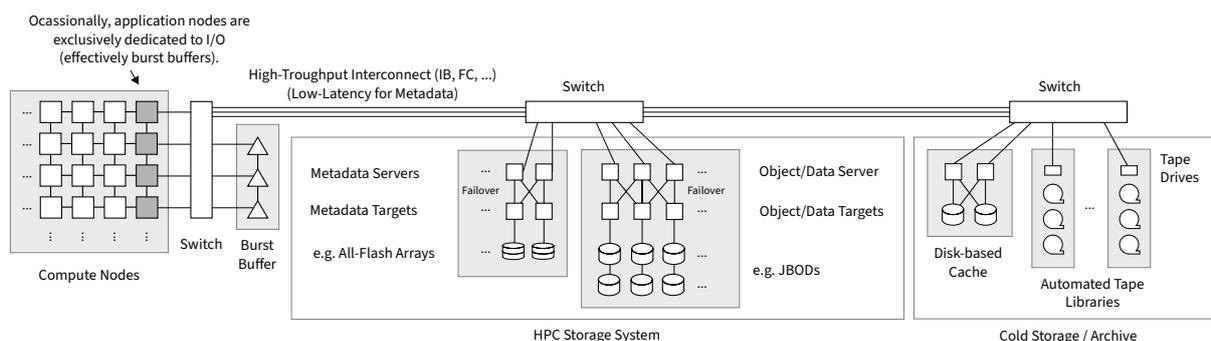


Figure 1: Typical HPC storage system [Lüttgau et al., 2018]

2.1. Kernel File Systems

In this section, we discuss the most popular and widely used systems and their characteristics. Among them are Lustre, Spectrum Scale, BeeGFS, OrangeFS, Ceph, and GlusterFS.

Spectrum Scale Spectrum Scale is a scalable high-performance data management solution developed by IBM for enterprises that need to process and store massive amounts of unstructured data [Potnis, 2018]. It is based on the former General Parallel File System (GPFS) [IBM, 2016]. GPFS provides concurrent data access with the ability to perform data analysis and archiving in one place. Spectrum Scale unifies different storage tiers like SSDs, HDDs, and tapes as well as analytics into a single scale-out solution. This combination enables users to choose optimal storage for their files or object data and move them as quickly as possible with low costs. Spectrum Scale is fully POSIX-compliant, which allows it to support many traditional HPC applications.

The file system helps to avoid a performance bottleneck for metadata-intensive applications by configuring dedicated servers for metadata updates. Shared devices stores both metadata and data on the same devices [Hildebrand and Schmuck, 2015]. Another bottleneck regarding single-server performance is also avoided in Spectrum Scale as all servers and clients can access and share the data without moving it. Thus, even a client can play the role of a server. Among the advanced features of Spectrum Scale are a declustered software RAID algorithm, storage pools, and the integration into OpenStack services. Vef et al. performed extensive tracing using the GPFS tracing tool and their new tracing interface prototype FlexTrace, shown to have negligible overhead [Vef et al., 2018]. FlexTrace offers user-defined tracing profiles to allow fine-grained trace control.

Even though Spectrum Scale is a capable file system and has great support from IBM with the integration of various useful tools, it is still quite expensive, especially for non-profit clusters with research purposes.

Lustre Lustre is a parallel distributed file system used on supercomputers. It is licensed under the GNU General Public License (GPLv2) and can thus be extended and improved by interested developers. Because of its high performance, Lustre is used on more than half of the 100 fastest supercomputers in the world.

The file system's architecture distinguishes between clients and servers. Clients use RPC messages to communicate with the servers, which perform the actual I/O operations. While all

clients are identical, the servers can have different roles: Object Storage Servers (OSS) manage the file system's data in the form of objects; clients can access byte ranges within the objects. Metadata Servers (MDS) manage the file system's metadata; after retrieving the metadata, clients are able to contact the appropriate OSSs independently. Each server is connected to possibly multiple targets (OSTs/MDTs) that store the actual file data or metadata, respectively.

Lustre runs in kernel space. Therefore, most functionality has been implemented in the form of kernel modules, having advantages and disadvantages. On the one hand, by using the kernel's virtual file system (VFS), Lustre can provide a POSIX-compliant file system compatible with existing applications. On the other hand, each file system operation requires a system call, which can be expensive when dealing with high-performance network and storage devices.

In line with its open approach to Lustre development, Intel has funded five Intel Parallel Computing Centers to integrate new features into Lustre. Among others, these centers are working on the quality of service for I/O performance, file system compression as well as better integration of TSM storage backends and big data workflows. An in-depth description of the Lustre architecture can be found in [Braam, 2019]. To make full use of node-local storage, Qian et al. proposed a hierarchical Persistent Client Caching (LPCC) mechanism, building either a local read-write cache on one SSD or a read-only cache over the SSDs of multiple clients [Qian et al., 2019].

BeeGFS The parallel and POSIX-compliant cluster file system BeeGFS was developed for I/O-intensive HPC applications [Fraunhofer ITWM, 2018]. Its architecture has a client-server design and consists of three key components: clients, metadata servers, and storage servers. The scalability and flexibility of BeeGFS are reached simply by increasing the number of servers and disks required for specific users. All their data is transparently distributed across multiple servers using striping. Besides data distribution, metadata is also striped over several metadata servers on a directory level, with each server storing a part of the complete file system tree. In this way, fast access to the data is provided. BeeGFS also enables load balancing for metadata.

The client kernel module of the BeeGFS system is free and under the GNU General Public License (GPL). The server module is covered by the BeeGFS EULA. Hence, commercial support is optionally available. Abramson et al. proposed a prototype file caching layer that makes use of the data locality across storage tiers [Abramson et al., 2020]. It also increases the data sharing between compute nodes and applications, while using data striping and metadata partitioning to support fast parallel I/O.

2.2. User Space File Systems

In contrast to kernel space file systems such as Lustre, user-level file systems do not require any kernel modules to run. This typically makes it easier to use such file systems in a supercomputer environment, where users normally do not have root privileges.

OrangeFS OrangeFS is a parallel distributed file system that runs completely in user space. It is open-source and licensed under the GNU Lesser General Public License (LGPL). It provides excellent MPI-IO support through a native ADIO backend and provides a wide range of user interfaces, including several Direct Interface libraries, a FUSE file system, and an optional

kernel module [Vilayannur et al., 2004]. Similar to other file systems, OrangeFS has dedicated servers for data and metadata storage. OrangeFS uses arbitrary local POSIX file systems for data and can use either Berkeley DB (BDB) or Lightning Memory-Mapped Database (LMDB) for metadata.

GlusterFS GlusterFS is another POSIX-compliant, free, and open-source distributed file system [Depardon et al., 2013]. Like other traditional storage solutions, it has a client-server model but does not need a dedicated metadata server. All data and metadata are stored on several devices (called volumes), which are dedicated to different servers. GlusterFS locates files algorithmically using an elastic hashing algorithm. This no-metadata server architecture ensures better performance, linear scalability, and reliability [Selvaganesan and Liazudeen, 2016]. At the same time, GlusterFS is a network file system that provides file-based storage only. Block and object interfaces must be built on top of it.

Ad Hoc File Systems Ad hoc file systems offer a way to include node-local SSDs or NVRAMs into a temporary storage system [Brinkmann et al., 2020]. There are still a lot of open questions regarding this approach, for example, how to use these file systems in combination with a present scheduling environment. The lifetimes of ad hoc file systems vary greatly and can be as short as the runtime of a single job. They are mostly implemented in user space. Prominent examples are BeeOND (BeeGFS-On-Demand), GekkoFS and BurstFS [Vef et al., 2020].

BeeOND was created to use BeeGFS as an ad hoc file system. It works as an intermediary between the application and the storage systems and is often used for data buffering. Its main idea is to distribute data and metadata evenly across a cluster. GekkoFS employs hashing at the level of file inodes to determine the cluster node managing it. Spreading the metadata across all participating cluster nodes required significantly relaxed POSIX directory semantics. Therefore, applications frequently listing or renaming directories should not be run on GekkoFS. Furthermore, it does not handle conflicts like overlapping file regions and shifts the responsibility to application developers. GekkoFS outperformed Lustre by a factor of $\sim 1,400$ for file creation regardless of whether a single or unique directory was used [Brinkmann et al., 2020]. BurstFS is very similar to GekkoFS, the main difference being the log-structured writing of BurstFS. Thereby, the write performance is not limited by the network latency. However, metadata directories are required to reconstruct multi-client writes to a single file.

2.3. Miscellaneous Systems

In the following, a variety of miscellaneous systems are presented.

ECFS and MARS Other storage systems focused on data archival developed by the European Centre for Medium-Range Weather Forecasts (ECMWF) are ECMWF's File Storage System (ECFS) and Meteorological Archival and Retrieval System (MARS) [Grawinkel et al., 2015]. A High Performance Storage System (HPSS) manages the tape archives for both systems as well as the disk cache for ECFS, where the files are accessed using a unique path. MARS is an object store providing an interface similar to a database. By using queries in a custom language, a list of relevant fields can be set, which are then joined into a package and stored in the system. The field database (FDB) stages and caches fields that are often accessed. ECFS contains relatively

few files that are used concurrently and experiences mainly write calls. In MARS, however, the files are equally relevant and mostly read [Smart et al., 2017a]. Thus, both systems provide powerful storage management for researchers interested in weather modeling. MARS allows HPC users to access large amounts of meteorological data stored only in GRIB and BUFR formats collected over the last 30 years.

Ceph Ceph is a free and open-source platform that offers file-, block- and object-based data storage on a single distributed cluster [Weil et al., 2006]. The system implements distributed object-storage on using the Reliable Autonomic Distributed Object Store (RADOS) system [Weil et al., 2007]. It is responsible for data migration, replication, failure detection and failure recovery to the cluster. Ceph also provides a near-POSIX-compliant file system. Its integration makes the benefits and features of the scalable environment available to applications. Ceph makes use of intelligent Object Storage Devices (OSDs) that provide file I/O (reads and writes) for all clients which interact with them. Data and metadata are independent because the Metadata Servers (MDSs) perform the altering metadata operations. Ceph dynamically distributes the metadata management and responsibility for the file system directory hierarchy among tens or even hundreds of those MDSs.

However, Ceph still has some drawbacks, for example, the limitation that a cluster can only deploy one CephFS. In addition, Ceph is designed with HDDs as its basis and needs performance improvements when disks are replaced with SSDs and data access pattern is random [Oh et al., 2016]. The Ceph developers turned away from building distributed storage backends on local file systems after the experiences with FileStore and KStore [Aghayev et al., 2020]. They developed BlueStore which works directly on raw block storage, thereby reducing the I/O layers significantly [Aghayev et al., 2019]. Furthermore, this decision give them more control over the complete stack which in turn leads to less performance variability.

DAOS The Fast Forward Storage and IO (FFSIO) project aims at providing an exascale storage system that is capable of dealing with the requirements of HPC applications as well as big data type workloads. It aims at introducing a new I/O stack and supporting more complex basic data types like containers and key-arrays [Lofstead et al., 2016]. Its functionality ranges from a general I/O interface at the top over an I/O forwarding and an I/O dispatcher layer to the Distributed Application Object Store (DAOS) layer, which offers a persistent storage interface and translates the object model visible to the user to the demands of the underlying infrastructure. DAOS will require large quantities of NVRAM and NVMe devices. Therefore it will not be suitable for all environments since high prices for these relatively new technologies will limit DAOS's use, both in data centers and in research, at least in the near future.

In addition to introducing a novel user space storage system, DAOS will also support new ways of performing I/O in a scalable way [Breitenfeld et al., 2017]. From an application developer's point of view, DAOS will provide a rich I/O interface in the form of key-array objects with support for both structured and unstructured data. Additionally, established I/O interfaces such as a legacy POSIX interface and an HDF5 interface will be supported natively. Similar to databases, DAOS has support for transactions. Multiple operations can be batched in a single transaction, which becomes immutable, durable, and consistent once committed as an epoch. On the one hand, this allows multiple processes to perform asynchronous write operations without having to worry about consistency problems. On the other hand, read operations will always have a consistent view because they are based on a committed (and thus

immutable) epoch. First performance evaluations indicate that DAOS has a stable performance across different IO500 workloads, whereas other file systems exhibit great variation between individual tests [Liang et al., 2020].

Sirocco Sirocco takes a very different approach to any of the above. It aims at redesigning the complete storage system by using a concept similar to unstructured peer-to-peer networks [Curry et al., 2016]. They position themselves against existing systems like Lustre, GPFS, and PVFS because these all are inspired by the Zebra file system, meaning that they use data striping [Hartman and Ousterhout, 1995]. Sirocco has some very unconventional design principles as it values write performance and scalability over most other objectives. First, there is no central indexing for data and data can be stored by any reachable server. Furthermore, data is continually moved within the system to offer integrity and longevity. Clients are not updated to store where the data is moved to. They can, however, specify a data protection level. Due to the write-back caching, explicit synchronization calls are required to persist data. The focus on writing comes at the cost of losing track of where data is stored and in which state. In the worst case, an extensive search over the storage servers is required. This need for a full scan is the most severe drawback of Sirocco.

3. Evolution of Databases

The following overview of database technology evolution is based on the comprehensive work of Davoudian et al. [Davoudian et al., 2018]. We find this brief insight into the history of database systems greatly benefits the discussion of current challenges as the previous requirements and problems provide plenty of important considerations already. A time-line of database generations is depicted in Figure 2.

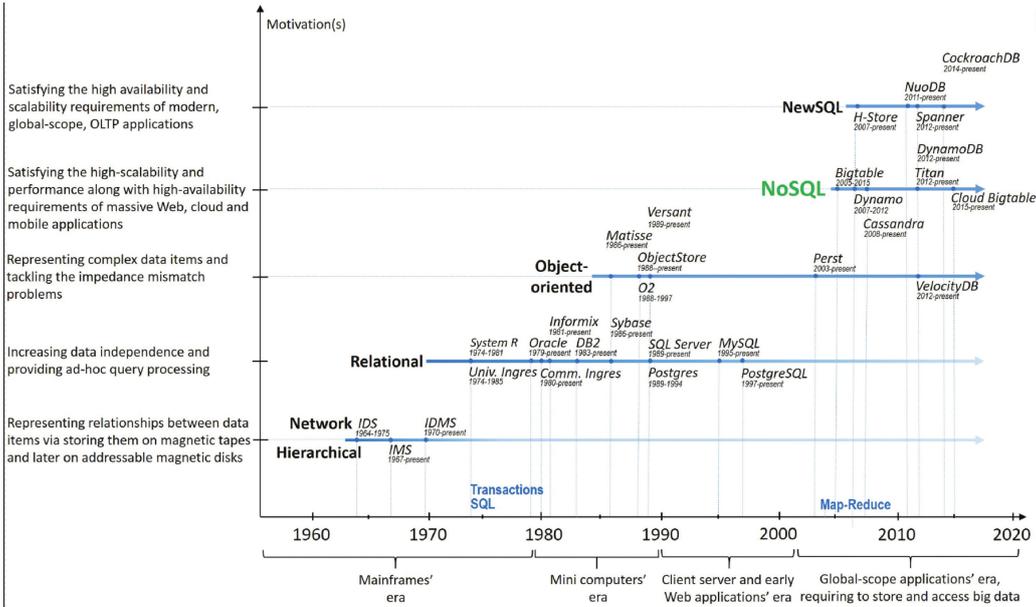


Figure 2: Evolution of database technologies from [Davoudian et al., 2018]. Some corresponding database systems are depicted as well.

First Generation (Hierarchical and Network Data Models) The first generation of database technology was aimed to provide convenient and efficient data access in the form of general-purpose database systems. They offered hierarchical and network data models in which data entries are linked together. This data dependence made application programming very challenging, fuelling the development of the second generation.

Second Generation (Relational Data Model) To counter the data dependence problems of the first generation, a relational data model was developed that used tuples to describe relations. By separating the logical model and the physical data organization, the data independence increased. That is when SEQUEL (Structured English Query Language) was developed. It is based on relational algebra and was then adapted by Chamberlin and Boyce to be more accessible to users without intensive mathematical training [Chamberlin and Boyce, 1974]. SEQUEL was later renamed SQL due to copyright problems. SQL was standardized in ANSI X3.135 in 1986 and adopted by ISO as ISO 9075-1987. Even today, SQL is still an active research topic as can be seen in the endeavors to parse natural language to SQL (NL2SQL) [Kim et al., 2020].

Third Generation (Object-Oriented Data Model) The spreading of object-oriented programming resulted in the need for an appropriate database model. Combined with the limited modeling capabilities of the relational data model, this led to the third generation of database systems. Following the programming trends, they were designed to conform to the object-oriented approach, identifying objects via object IDs and using a hierarchical structure to model the object feature inheritance. However, the high investments into relational database (RDB) systems hindered the adoption of object-oriented databases. Through extensions to prominent RDB systems, such as DB2 and Oracle, important object-oriented features were incorporated, leading to object-relational database systems.

Fourth Generation (Flexible Data Models) Applications in the area of web technologies and the Internet of Things (IoT) started to present another set of challenging requirements. These are in part mutually exclusive and listed in the following.

- Horizontal Scalability: Making use of additional resources to deal with the increasing data sizes.
- High Availability and Fault Tolerance: Guaranteeing stable service even in the face of hardware failure.
- Transaction Reliability: Providing strong consistency guarantees.
- Database Schema Maintainability: Making the evolution of database schemas less costly.

These requirements cannot be easily satisfied through Relational Database Management Systems (RDBMSs). For one, the pre-defined schema hinders the database evolution. Also, fixed schemas are not suitable for the agile development approaches found, for example, in the area of Big Data. Additionally, scaling up needs dedicated and often specialized hardware that introduces the need for costly data movement. Lastly, scaling out complicates the joining of distributed data. The use of ACID transactions also brought challenges regarding availability and performance. Sacrificing some requirements for the sake of others, more essential to the demands of the application, led to the fourth database generation and the development of numeral non-relational storage systems, namely NoSQL stores.

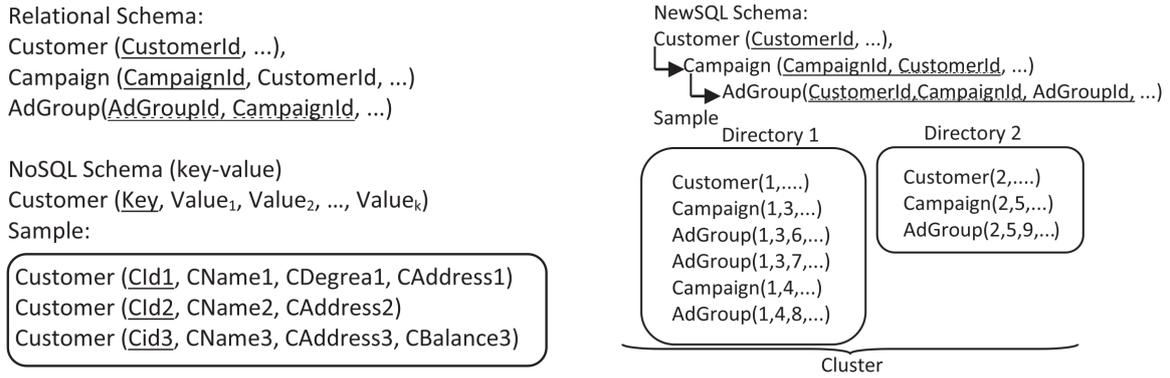


Figure 3: Relational (top left), NoSQL (bottom left) and NewSQL (right) schemas from [Mansouri et al., 2018]. Mansouri et al. used the NewSQL query example by [Shute et al., 2013].

Fifth Generation (NewSQL) Further high scalability and reliability requirements of OLTP applications arose in the late 2000s. They motivated the evolution of database technologies to handle the large data volumes that now span over multiple datacenters. The resulting solutions are called NewSQL stores as they offer SQL queries and ACID transactions without the need for a join operation.

3.1. Relational Queries vs. Non-Relational Queries

Figure 3 shows the difference between relational queries, NoSQL queries and NewSQL queries. In contrast to NoSQL databases, RDBs offer fixed and predefined fields for each entry. NoSQL stores build upon the key-value model or similar variants such as graphs and documents where the entry is linked to a key-value pair. NewSQL databases use directories in the top-level of their hierarchy, e.g., Customer, where each row of the directory table and all the entries in the included tables, e.g., Campaign and AdGroup, form a directory itself [Mansouri et al., 2018]. Thereby, NewSQL builds child-parent relations between all pairs where the primary key of the parents is used as a prefix in the child's primary key.

3.2. Data Models

The separation of different data models shown in the following was introduced by [Mansouri et al., 2018]. We highlight these models and the concise description by Mansouri et al. because they help to categorize the massive number of database systems in different ways. Figure 4 depicts this data model taxonomy.

Data Structure The data structure significantly influences retrieval performance. There are three categories. (1) In *structured data* relationship is defined between fields identified through name and value, for example, in an RDB. (2) *Semi-structured data* provides a specific form of structuring that is known to the application. Primitive operations are supported similarly to structured data. (3) *Unstructured data* does not offer any pre-defined data model and uses the key-value access providing high scalability and performance at the expense of data consistency and transaction support.

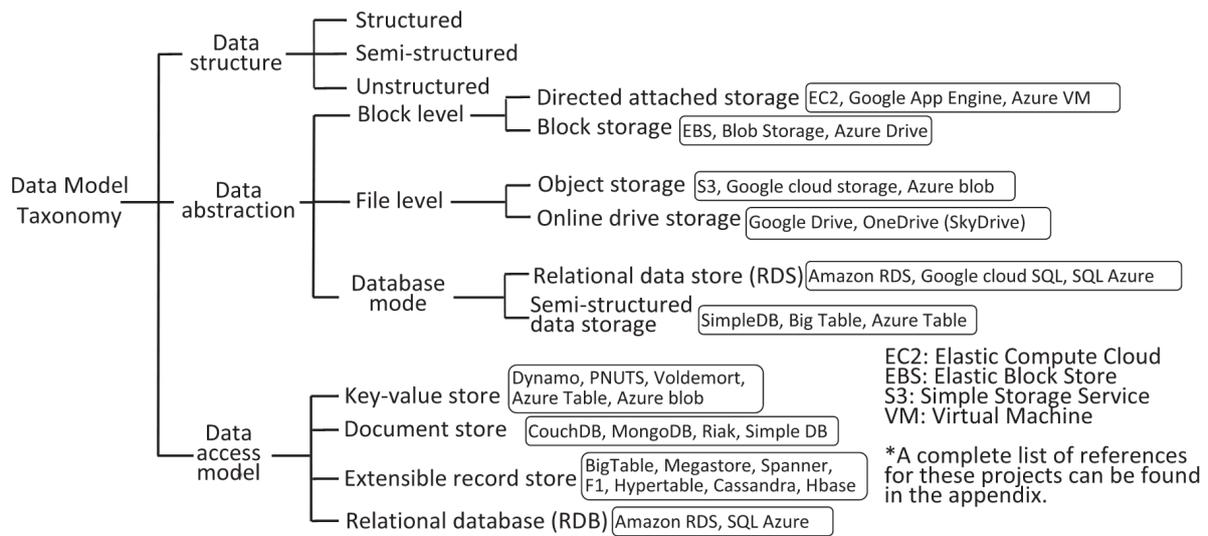


Figure 4: Data model taxonomy from [Mansouri et al., 2018]

Data Abstraction Data abstraction is the storage level abstraction in data stores. There are three levels. (1) The *block level* provides the fastest access that is split into *directed-attached storage* and *block storage*. (2) The most common one is the *file level*. Its widespread use is due to the simple API that eases management. It comes either as *object storage* that provides storage of large binary objects or as *online drive storage*. (3) The last level is the *database mode* either as *relational storage* or *semi-structured data storage*. It offers RDBs and NoSQL/NewSQL databases, respectively, to the users. The former cannot be scaled easily but provides strong consistency guarantees, while the latter offers horizontal scalability.

Data Access The data access model affects the offered consistency and the transaction management. It has four main categories. (1) *Key-value databases*, as mentioned before, provide a simple interface based on unique keys and arbitrary data values. As key-value stores are NoSQL databases, they support the features discussed in the third data abstraction level. (2) *Document databases* offer storage for several kinds of documents that can be indexed. They do not provide ACID guarantees for primitive operations. Therefore, they support *eventual consistency*. (3) *Extensible record databases* (also called wide-column stores) are similar to tables where columns can be grouped, while the rows are distributed over several storage nodes depending on the primary key. Each field offers multiple data versions ordered by timestamps. This approach is called NewSQL. (4) As discussed above, *relational databases* use pre-defined schemes and offer SQL queries and ACID transactions.

4. Relational Database Systems

There have been various studies trying to identify the best database system for a specific application type. Makris et al. evaluate the response time of PostgreSQL in comparison with MongoDB and find that PostgreSQL outperforms MongoDB for all queries on vessel tracking [Makris et al., 2019]. Performance bottlenecks of relational database systems because of, for example, the read amplification and high cache miss rates, can be mitigated by employing 3D XPoint storage [Yang and Lilja, 2018].

Feser et al. propose a promising approach, Castor, by developing a new layout algebra to combine the data layout and the queries to be performed [Feser et al., 2020]. Thereby, they are able to transform both the layout and the query in a single rule and allow drastic layout transformations. Besides the layout algebra, they provide an automated deductive optimizer, a type-driven layout compiler, and a high-performance query compiler. This combination of features comes along with a significant limitation. The constructed databases are read-only and are, therefore, tailored towards specific use cases.

In the following, the 15 most popular relational database systems as of July 2021 are listed. The complete list consists of 143 entries. The order is determined by the DB-Engines ranking².

While MonetDB is a relational database, it can also be considered a wide-column store because it manages data in columns [solid IT, 2021]. This vertical storage allows more performant queries because CPU cache lines are used more efficiently, and new cache-conscious algorithms are provided [Boncz et al., 2008]. Furthermore, the reduced query algebra makes faster hardware implementations possible. Idreos et al. state that it is a column store that feels like a relational database to the user [Idreos et al., 2012]. Index concepts such as Elf have been integrated into MonetDB to optimize efficient querying on scientific data [Blockhaus et al., 2020]. Another variation, MonetDBLite, was developed to cater to the needs of machine learning and classifications tasks [Raasveldt and Mühleisen, 2018]. It offers fast data transfer between analytical tools and the database while also guaranteeing the ACID properties of relational systems. Systems offering a hybrid approach of both in-memory and persistent solutions are marked with two asterisks (**).

1. Oracle^{**3}
2. MySQL^{**4}
3. Microsoft SQL Server^{**5}
4. PostgreSQL⁶
5. IBM DB2⁷
6. SQLite^{**8}
7. Microsoft Access⁹
8. MariaDB^{**10}
9. Apache Hive¹¹
10. Microsoft Azure SQL Database¹²
11. Teradata^{**13}
12. SAP HANA^{**14}

²<https://db-engines.com/en/ranking/relational+dbms>

³<https://www.oracle.com/database/>

⁴<https://www.mysql.com/>

⁵<https://www.microsoft.com/en-us/sql-server/>

⁶<https://www.postgresql.org/>

⁷<https://www.ibm.com/analytics/db2>

⁸<https://www.sqlite.org/index.html>

⁹<https://www.microsoft.com/en-us/microsoft-365/access>

¹⁰<https://mariadb.com/>

¹¹<https://hive.apache.org/>

¹²<https://azure.microsoft.com/en-us/products/azure-sql/database/>

¹³<https://www.teradata.com/>

¹⁴<https://www.sap.com/products/hana.html>

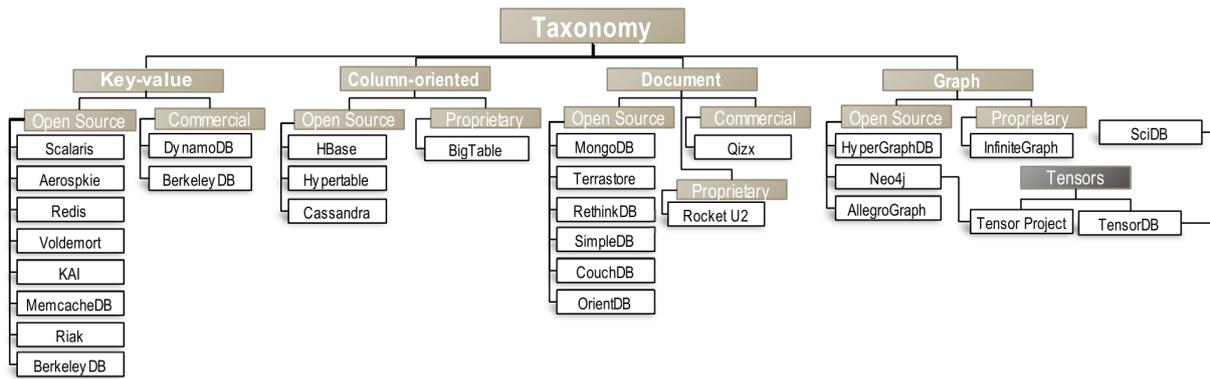


Figure 5: Taxonomy of big data storage technologies taken from [Siddiq et al., 2017]

13. FileMaker¹⁵
14. SAP ASE (Adaptive Server Enterprise)¹⁶
15. Google BigQuery¹⁷

5. Non-Relational Database Systems

The NoSQL Databases List¹⁸ currently includes over 225 databases. They can be divided into several subcategories, of which we will highlight key-value stores, wide-column stores, document stores, graph stores, and object stores. While we are mostly interested in systems suitable for HPC, big data plays an increasingly important role. Therefore, we included the taxonomy of storage systems used in big data by [Siddiq et al., 2017], which is shown in Figure 5. Most of the systems are included in the listings in the following subsections. A comprehensive overview of data storage and data placement in cloud environments was published by [Mazumdar et al., 2019].

NoSQL stores have a wide variety of consistency models ranging from strong consistency to eventual consistency. A good overview is provided by [Davoudian et al., 2018] in Figure 6.

Log-Structured Merge-Trees (LSM-Trees) Log-structured merge-trees or short LSM-trees are a very popular index structure for NoSQL databases. They are often used as an indexing data structure for key-value stores or wide-column stores. They offer high performance for batched sequential write access on both SSDs or HDDs. There have been large efforts to optimize and improve LSM-trees for different workloads and settings. A comprehensive survey has been performed by [Luo and Carey, 2020]. They developed a taxonomy of improvements that is shown in Figure 7.

In the following, some optimizations are presented. Prominent approaches that aim at minimizing I/O amplification are WiscKey [Lu et al., 2016] and LSM-trie [Wu et al., 2015]. Cheng et al. researched the impact of LSM-trie performance by parallel search on the read performance

¹⁵<https://www.claris.com/filemaker/>

¹⁶<https://www.sap.com/products/sybase-ase.html>

¹⁷<https://db-engines.com/en/system/Google+BigQuery>

¹⁸<https://hostingdata.co.uk/nosql-database/>

	Characteristics	Suitable applications	Suitable NoSQL store data models
Strong consistency	<ul style="list-style-type: none"> - Dictating a total, real-time ordering of all operations/transactions - Providing a single, consistent image of the whole data - Severe impact on availability and performance (especially in wide-area networks) - Using a synchronous primary/ update-anywhere replication strategy 	Scenarios requiring ACID reliability, such as financial services	It can be practically preserved over one or more <i>aggregates</i> (along with their corresponding replicas). An <i>aggregate</i> can be a key-value pair in <i>key-value stores</i> , a column-family, super column-family, the whole row of a table in <i>wide-column stores</i> , or a document in <i>document stores</i> .
Causal consistency	<ul style="list-style-type: none"> - Enforcing a partial ordering (called happens-before relation) among causally dependent operations - The efficiency of implementation is more than strong consistency and less than the eventual one. - The possibility of conflicting writes - Read operations may not always access the latest versions of read data items. - Using an asynchronous primary/ update-anywhere replication strategy 	Online human-facing services, such as commenting services in social networks	It can be preserved in all data models while enforcing in a lot of academic NoSQL stores. However, owing to its communication overhead (of dependency tracking), no industrial NoSQL store enforces this model.
Per-object Timeline consistency	<ul style="list-style-type: none"> - Ensuring a total ordering of all operations on each data item along with respecting their order as issued by each client - Read operations may access stale data items. - A client's read operation never returns the new version of a data item before an old one. - Using an asynchronous primary replication strategy 	Online human-facing services, such as the update of photos/albums	It can be practically preserved over an <i>aggregate</i> in a key-value, wide-column or document stores. It is enforced in some industrial NoSQL stores, such as Yahoo Pnuts (Cooper et al. 2008).
Parallel snapshot isolation	<ul style="list-style-type: none"> - Enforcing snapshot isolation on transactions executed in each datacenter - Preserving the causal ordering of transactions executed across datacenters - Using an asynchronous primary/ update-anywhere replication strategy 	Similar to causal consistency	Preserved in some academic NoSQL stores, such as Walter (Sovran et al. 2011)
Eventual consistency	<ul style="list-style-type: none"> - It does not dictate any ordering of operations. - It ensures the gradual and eventual convergence of replicas to identical values after receiving the same set of asynchronously propagated updates. - It allows anomalous behaviors of applications. - It suffers from conflicting write operations on the same data item. - It uses an asynchronous primary/ update-anywhere replication strategy. 	It suffices for many services in web applications for whom the high availability of data requests is so critical that even a tiny impact on it causes user dissatisfaction and loss of revenue.	It can be preserved in all data models. It is enforced in a lot of NoSQL stores.

Figure 6: Consistency models of NoSQL stores from [Davoudian et al., 2018]

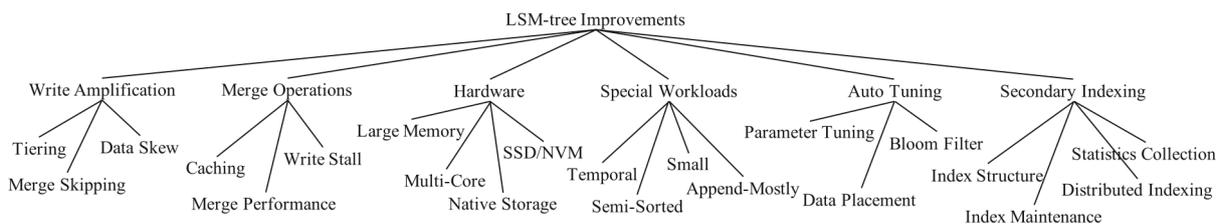


Figure 7: Taxonomy of LSM-tree improvements taken from [Luo and Carey, 2020]

and found that it can be improved up to 98 % and on average up to 72 % [Cheng et al., 2020]. Mäscher et al. proposed Hyperion, a trie-based main-memory key-value store that improves memory usage considerably [Mäscher et al., 2019]. Papagiannis et al. observe two major bottlenecks [Papagiannis et al., 2021]. First, the compaction of LSM-trees requires constant merging and sorting; second, using an I/O cache for device access. CPU overhead and I/O amplification are mitigated by Kreon, a key-value store for flash-based storage using memory-mapped I/O.

5.1. Key-Value Stores

The concept of key-value stores is very straightforward and popular. It consists of a value uniquely identified by a key, for example, a name or a hash. The value can be data of any type with any structure and size. Data is stored as byte arrays requiring serialization and deserialization that is often performed on the clients. Therefore, indexing or querying on the values is not supported. Using this simple model, access is only possible through the key, limiting the potential applications. Therefore, advanced key-value stores may support data lookup based on the value, for example, by providing list data types. Thereby, Redis and Aerospike offer data updates that do not require the replacement of the whole value. Other extensions are tabular data types with Spinnaker¹⁹, while Riak also supports documents such as JSON. This development blurs the line between key-value stores and other approaches like document stores (see Section 5.3).

Key-value stores can be grouped according to different criteria. Often in-memory, persistent and hybrid approaches are distinguished. The first approach is typically used as a caching layer for transient information such as session information. Facebook uses Memcached while LinkedIn employs Voldemort²⁰. In our research on storage systems, we often come across LevelDB [Ghemawat and Dean, 2020], LMDB and RocksDB [Yang et al., 2015, Cao et al., 2020].

The DB-Engines ranking has the following top 25 (of 59) as of July 2021²¹. Key-value stores that provide an in-memory-only solution are printed boldly and marked with an asterisk (*), hybrid approaches with two asterisks (**).

1. Redis^{**22}
2. Amazon DynamoDB²³
3. Microsoft Azure Cosmos DB²⁴
4. **Memcached**^{*25}
5. etcd²⁶
6. **Hazelcast**^{*27}
7. **Ehcache**^{*28}

¹⁹<https://spinnaker.io/>

²⁰<https://project-voldemort.com/voldemort/>

²¹<https://db-engines.com/en/ranking/key-value+store>

²²<https://redis.io/>

²³<https://aws.amazon.com/dynamodb/>

²⁴<https://azure.microsoft.com/en-us/services/cosmos-db/>

²⁵<http://www.memcached.org/>

²⁶<https://etcd.io/> and <https://github.com/etcd-io/etcd>

²⁷<https://hazelcast.com/>

²⁸<https://hazelcast.com/>

8. Aerospike**²⁹
9. Riak KV³⁰
10. ArangoDB³¹
11. Apache Ignite**³²
12. OrientDB³³
13. Oracle NoSQL Database**³⁴
14. RocksDB**³⁵
15. ScyllaDB**³⁶
16. LevelDB³⁷
17. Oracle Berkeley DB**³⁸
18. InterSystems Caché**³⁹
19. **Infinispan** *⁴⁰
20. **Oracle Coherence** *⁴¹
21. LMDB**⁴²
22. Amazon SimpleDB⁴³
23. Apache Geode**⁴⁴
24. GridGain *⁴⁵⁴⁶
25. InterSystems IRIS**⁴⁷

²⁹<https://aerospike.com/>

³⁰<https://riak.com/products/riak-kv/>

³¹<https://www.arangodb.com/>

³²<https://ignite.apache.org/>

³³<https://www.arangodb.com/>

³⁴<https://db-engines.com/en/ranking/key-value+store>

³⁵<https://rocksdb.org/>

³⁶<https://www.scylladb.com/>

³⁷<https://github.com/google/leveldb>

³⁸<https://www.oracle.com/database/technologies/related/berkeleydb.html>

³⁹<https://www.intersystems.com/products/cache/>

⁴⁰<https://infinispan.org/>

⁴¹<https://www.oracle.com/middleware/technologies/coherence.html>

⁴²<https://symas.com/lmdb/>

⁴³<https://aws.amazon.com/simplydb/>

⁴⁴<https://geode.apache.org/>

⁴⁵Built on Apache Ignite

⁴⁶<https://aws.amazon.com/simplydb/>

⁴⁷<https://www.intersystems.com/products/intersystems-iris/>

5.2. Wide-Column Stores

Wide-column stores are a database concept inspired by Google Bigtable [Chang et al., 2008]. The main concept is to manage data using rows and a fixed number of column families. Column families consist of logically connected columns that are stored together instead of rows [Davoudian et al., 2018]. Wide-column stores provide a more flexible schema offering to add or remove columns at runtime. Davoudian et al. call them extended key-value stores where the value is a number of nested key-value pairs.

Google Bigtable is built for structured data and to provide flexibility and high performance for the variety of Google products [Chang et al., 2008]. The most popular wide-column store by far is Apache Cassandra [solid IT, 2021]. It was developed to meet the reliability and scalability needs of Facebook [Lakshman and Malik, 2010]. Another prominent wide-column store is Apache HBase which is built on top of Hadoop Distributed File System (HDFS) [Vora, 2011]. Recently, ScyllaDB has been promoted to function as a drop-in replacement of Apache Cassandra, promised to outperform it by up to a factor of 10. One key aspect contributing to the improvement is ScyllaDB's implementation in C++ instead of Java [Suneja, 2019]. ScyllaDB provides large data volumes, high availability, and high performance. Mahgoub et al. report some performance instability of ScyllaDB with server replication [Mahgoub et al., 2017]. Replacing file systems with wide-column stores is extensively studied. Santamaria et al. compare HDF5 on GPFS with Cassandra or Scylla through Hecuba [Santamaria et al., 2019].

Below is the complete list of wide-column stores, ordered according to the DB-Engines ranking⁴⁸. Wide-column stores that provide an in-memory-only solution are printed boldly and marked with an asterisk (*), hybrid approaches with two asterisks (**).

1. Cassandra⁴⁹
2. HBase^{**50}
3. Microsoft Azure Cosmos DB⁵¹
4. Datastax Enterprise^{**52,53}
5. Microsoft Azure Table Storage⁵⁴
6. Apache Accumulo^{55,56}
7. Google Cloud Bigtable⁵⁷
8. ScyllaDB^{**58}
9. HPC Ezmeral Data Fabric⁵⁹

⁴⁸<https://db-engines.com/en/ranking/wide+column+store>

⁴⁹<https://cassandra.apache.org/>

⁵⁰<http://hbase.apache.org/>

⁵¹<https://azure.microsoft.com/en-us/services/cosmos-db/>

⁵²Built on Apache Cassandra

⁵³<https://www.datastax.com/products/datastax-enterprise>

⁵⁴<https://azure.microsoft.com/en-us/services/storage/tables/>

⁵⁵Uses Hadoop's HDFS and ZooKeeper

⁵⁶<https://accumulo.apache.org/>

⁵⁷<https://cloud.google.com/bigtable/>

⁵⁸<https://www.scylladb.com/>

⁵⁹<https://www.hpe.com/us/en/software/data-fabric.html>

10. Elassandra^{60,61}
11. Amazon Keyspaces (for Cassandra)⁶²
12. Alibaba Cloud Table Store⁶³
13. SWC-DB (Super Wide Column Database)⁶⁴

5.3. Document Stores

Document stores are also an extension of key-value stores where the value is a document [Davoudian et al., 2018]. Typical formats include XML, JSON, and BSON. The schema is flexible as attributes can be added to the document at runtime. It also enables indexing and search functionality, using the attribute names and values. Some document stores like ArangoDB and orientDB also offer an additional graph representation alongside the document data model by referencing documents. Schultz et al. analyze the tunable consistency model for MongoDB [Schultz et al., 2019].

Below is the list of the top 10 of all ranked 50 document stores, ordered according to the DB-Engines ranking⁶⁵. Document stores that provide an in-memory-only solution are print bold and marked with an asterisk (*), hybrid approaches with two asterisks (**). Solutions offering a multi-model approach are marked with a plus (+).

1. MongoDB^{**+}⁶⁶
2. Amazon Dynamo DB⁺⁶⁷
3. Microsoft Azure Cosmos DB⁺⁶⁸
4. Couchbase^{**+}⁶⁹
5. Firebase Realtime Database⁷⁰
6. Apache CouchDB⁺⁷¹
7. Realm^{**}⁷²
8. MarkLogic^{**+}⁷³
9. Google Cloud Firestore⁷⁴
10. ArangoDB⁺⁷⁵

⁶⁰Combination of Cassandra and Elasticsearch

⁶¹<https://www.elassandra.io/>

⁶²<https://aws.amazon.com/keyspaces/>

⁶³<https://www.alibabacloud.com/product/table-store>

⁶⁴<https://github.com/kashirin-alex/swc-db>

⁶⁵<https://db-engines.com/en/ranking/document+store>

⁶⁶<https://www.mongodb.com/>

⁶⁷<https://aws.amazon.com/dynamodb/>

⁶⁸<https://azure.microsoft.com/en-us/services/cosmos-db/>

⁶⁹<https://www.couchbase.com/>

⁷⁰<https://firebase.google.com/products/realtime-database/>

⁷¹<https://couchdb.apache.org/>

⁷²<https://realm.io/>

⁷³<https://www.marklogic.com/>

⁷⁴<https://firebase.google.com/products/firestore/>

⁷⁵<https://www.arangodb.com/>

5.4. Graph Stores

With the increase in graph-oriented data like the Semantic Web, a data model allowing the efficient relationship traversal was required [Davoudian et al., 2018]. This led to the concept of graph stores where vertices represent entities and edges the relationships. Property graphs are a mixture of various graph types like directed, labeled, and multigraphs, which makes them very flexible.

Below the top 15 of all ranked 32 graph stores are listed, ordered according to the DB-Engines ranking⁷⁶. Graph stores that provide an in-memory-only solution are printed boldly and marked with an asterisk (*), hybrid approaches with two asterisks (**). Solutions offering a multi-model approach are marked with a plus (+).

1. Neo4J⁷⁷
2. Microsoft Azure Cosmos+⁷⁸
3. ArangoDB+⁷⁹
4. OrientDB+⁸⁰
5. Virtuoso**+⁸¹
6. GraphDB+⁸²
7. Janusgraph^{83,84}
8. Amazon Neptune+⁸⁵
9. TigerGraph⁸⁶
10. Stardog**+⁸⁷
11. Dgraph⁸⁸
12. Fauna+⁸⁹
13. AllegroGraph+⁹⁰
14. Apache Giraph⁹¹
15. Nebula Graph**⁹²

⁷⁶<https://db-engines.com/en/ranking/graph+dbms>

⁷⁷<https://neo4j.com/>

⁷⁸<https://azure.microsoft.com/en-us/services/cosmos-db/>

⁷⁹<https://www.arangodb.com/>

⁸⁰<https://orientdb.org/>

⁸¹<https://virtuoso.openlinksw.com/>

⁸²<https://www.ontotext.com/products/graphdb/>

⁸³Janusgraph is an open-source fork of Titan, a previously prominent solution that was decommissioned after a takeover by Datastay (see <https://db-engines.com/de/system/Titan>).

⁸⁴<https://janusgraph.org/>

⁸⁵<https://aws.amazon.com/neptune/>

⁸⁶<https://www.tigergraph.com/>

⁸⁷<https://www.stardog.com/>

⁸⁸<https://dgraph.io/>

⁸⁹<https://fauna.com/>

⁹⁰<https://allegrograph.com/>

⁹¹<http://giraph.apache.org/>

⁹²<https://nebula-graph.io/>

5.5. Time-Series DBMS

With the rise of sensor storage, a new data model was required for Cloud computing and Big Data [Mazumdar et al., 2019]. Time-series DBMS typically extend key-value stores or column-stores by building an additional data model on top.

Below the top 15 of all ranked 31 time-series DBMS are listed, ordered according to the DB-Engines ranking⁹³. Time-series DBMS that provide an in-memory only solution, are print bold and marked with an asterisk (*), hybrid approaches with two asterisks (**). Solutions offering a multi-model approach are marked with a plus (+).

1. InfluxDB^{**+}⁹⁴
2. Kdb+/KX^{**+}⁹⁵
3. Prometheus⁹⁶
4. Graphite⁹⁷
5. TimescaleDB⁺⁹⁸
6. Apache Druid⁺⁹⁹
7. RRDTool (Round Robin Database)¹⁰⁰
8. OpenTSDB¹⁰¹
9. Fauna⁺¹⁰²
10. GridDB^{**+}¹⁰³
11. DolphinDB^{**}¹⁰⁴
12. KairosDB¹⁰⁵
13. eXtremeDB^{**+}¹⁰⁶
14. Amazon Timestream¹⁰⁷
15. QuestDB^{**+}¹⁰⁸

⁹³<https://db-engines.com/en/ranking/time+series+dbms>

⁹⁴<https://www.influxdata.com/products/influxdb-overview/>

⁹⁵<https://kx.com/>

⁹⁶<https://prometheus.io/>

⁹⁷<https://github.com/graphite-project/graphite-web>

⁹⁸<https://www.timescale.com/>

⁹⁹<https://druid.apache.org/>

¹⁰⁰<https://oss.oetiker.ch/rrdtool/>

¹⁰¹<http://opentsdb.net/>

¹⁰²<https://fauna.com/>

¹⁰³<https://griddb.net/>

¹⁰⁴<https://www.dolphindb.com/>

¹⁰⁵<https://github.com/kairosdb/kairosdb>

¹⁰⁶<https://www.mcobject.com/extremedbfamily/>

¹⁰⁷<https://aws.amazon.com/timestream/>

¹⁰⁸<https://questdb.io/>

6. Object Stores

In contrast to previous database approaches, object stores provide management solutions for unstructured data. They offer storage of variable size for so-called objects that typically store data as BLOBs (binary large objects). A first standardization was undertaken in late 2004 [Factor et al., 2005]. The indexing approaches of object stores are comparable to those of other databases like key-value stores. While object stores were not designed for HPC environments, they are increasingly employed on large-scale clusters, for example, to store meteorological and climate data [Smart et al., 2017b]. In an effort to mitigate the performance limitations of current file systems, data is extracted from self-describing data formats like NetCDF and HDF5 and stored in object stores. Chu et al. demonstrate the benefits of mapping HDF5 datasets to object storage [Chu et al., 2020].

There is also a growing interest from the IoT community to use object storage in fog and edge computing infrastructure as shown by [Confais et al., 2017]. They studied RADOS, Cassandra, and InterPlanetary File System (IPFS) with a focus on access time for writing and reading objects, the network traffic exchanged between different geographical sites, and the influence of network latency.

Lee et al. propose SwimStore, a storage architecture providing a high write performance, low write amplification, and stable performance on top of a file system [Lee et al., 2018]. It was integrated into Ceph and performed better than other Ceph storage backends like FileStore and KStore but similar to BlueStore. BlueStore is a drastic redesign compared to FileStore and KStore. It directly operates on raw block storage, using only a thin user space file system called BlueFS in combination with RocksDB for the metadata [Aghayev et al., 2019].

Since there is no ranking of object stores in the DB-Engines ranking, we have compiled a list of prominent object stores in no particular order.

- Amazon S3¹⁰⁹
- OpenStack Swift¹¹⁰
- MinIO¹¹¹
- OpenIO¹¹²
- RADOS¹¹³
- BlueStore¹¹⁴

¹⁰⁹<https://docs.aws.amazon.com/s3/>

¹¹⁰<https://wiki.openstack.org/wiki/Swift>

¹¹¹<https://min.io/>

¹¹²<https://www.openio.io/>

¹¹³<https://ceph.io/en/>, <https://docs.ceph.com/en/latest/radosgw/>

¹¹⁴<https://ceph.io/en/> and

<https://docs.ceph.com/en/latest/rados/configuration/bluestore-config-ref/>

References

- [Abramson et al., 2020] Abramson, D., Jin, C., Luong, J., and Carroll, J. (2020). A BeeGFS-Based Caching File System for Data-Intensive Parallel Computing. In Panda, D. K., editor, *Supercomputing Frontiers - 6th Asian Conference, SCFA 2020, Singapore, February 24-27, 2020, Proceedings*, volume 12082 of *Lecture Notes in Computer Science*, pages 3–22. Springer. (Cited on page 4)
- [Aghayev et al., 2019] Aghayev, A., Weil, S. A., Kuchnik, M., Nelson, M., Ganger, G. R., and Amvrosiadis, G. (2019). File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In Brecht, T. and Williamson, C., editors, *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 353–369. ACM. (Cited on pages 6 and 20)
- [Aghayev et al., 2020] Aghayev, A., Weil, S. A., Kuchnik, M., Nelson, M., Ganger, G. R., and Amvrosiadis, G. (2020). The Case for Custom Storage Backends in Distributed Storage Systems. *ACM Trans. Storage*, 16(2):9:1–9:31. (Cited on page 6)
- [Blockhaus et al., 2020] Blockhaus, P., Broneske, D., Schäler, M., Köppen, V., and Saake, G. (2020). Combining Two Worlds: MonetDB with Multi-Dimensional Index Structure Support to Efficiently Query Scientific Data. In Pourabbas, E., Sacharidis, D., Stockinger, K., and Vergoulis, T., editors, *SSDBM 2020: 32nd International Conference on Scientific and Statistical Database Management, Vienna, Austria, July 7-9, 2020*, pages 29:1–29:4. ACM. (Cited on page 11)
- [Boncz et al., 2008] Boncz, P. A., Kersten, M. L., and Manegold, S. (2008). Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85. (Cited on page 11)
- [Braam, 2019] Braam, P. (2019). The Lustre Storage Architecture. *CoRR*, abs/1903.01955. (Cited on page 4)
- [Breitenfeld et al., 2017] Breitenfeld, M. S., Fortner, N., Henderson, J., Soumagne, J., Chaarawi, M., Lombardi, J., and Koziol, Q. (2017). DAOS for Extreme-scale Systems in Scientific Applications. *CoRR*, abs/1712.00423. (Cited on page 6)
- [Brinkmann et al., 2020] Brinkmann, A., Mohror, K., Yu, W., Carns, P. H., Cortes, T., Klasky, S., Miranda, A., Pfreundt, F., Ross, R. B., and Vef, M. (2020). Ad Hoc File Systems for High-Performance Computing. *J. Comput. Sci. Technol.*, 35(1):4–26. (Cited on page 5)
- [Cao et al., 2020] Cao, Z., Dong, S., Vemuri, S., and Du, D. H. C. (2020). Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In Noh, S. H. and Welch, B., editors, *18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020*, pages 209–223. USENIX Association. (Cited on page 14)
- [Chamberlin and Boyce, 1974] Chamberlin, D. D. and Boyce, R. F. (1974). SEQUEL: A Structured English Query Language. In Rustin, R., editor, *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Michigan, USA, May 1-3, 1974, 2 Volumes*, pages 249–264. ACM. (Cited on page 8)
- [Chang et al., 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26. (Cited on page 16)

- [Cheng et al., 2020] Cheng, W., Guo, T., Zeng, L., Wang, Y., Nagel, L., Süß, T., and Brinkmann, A. (2020). Improving LSM-trie performance by parallel search. *Softw. Pract. Exp.*, 50(10):1952–1965. (Cited on page 14)
- [Chu et al., 2020] Chu, X., LeFevre, J., Montana, A., Robinson, D., Koziol, Q., Alvaro, P., and Maltzahn, C. (2020). Mapping Datasets to Object Storage System. *CoRR*, abs/2007.01789. (Cited on page 20)
- [Confais et al., 2017] Confais, B., Lebre, A., and Parrein, B. (2017). Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure. *Trans. Large Scale Data Knowl. Centered Syst.*, 33:40–79. (Cited on page 20)
- [Curry et al., 2016] Curry, M. L., Ward, H. L., Danielson, G., and Lofstead, J. F. (2016). An Overview of the Sirocco Parallel Storage System. In Taufer, M., Mohr, B., and Kunkel, J. M., editors, *High Performance Computing - ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P³MA, VHPC, WOPSSS, Frankfurt, Germany, June 19-23, 2016, Revised Selected Papers*, volume 9945 of *Lecture Notes in Computer Science*, pages 121–129. (Cited on page 7)
- [Davoudian et al., 2018] Davoudian, A., Chen, L., and Liu, M. (2018). A Survey on NoSQL Stores. *ACM Comput. Surv.*, 51(2):40:1–40:43. (Cited on pages 7, 12, 13, 16, 17, and 18)
- [Depardon et al., 2013] Depardon, B., Le Mahec, G., and Séguin, C. (2013). Analysis of Six Distributed File Systems. Research report. (Cited on page 5)
- [Factor et al., 2005] Factor, M., Meth, K., Naor, D., Rodeh, O., and Satran, J. (2005). Object storage: The future building block for storage systems. In *2005 IEEE International Symposium on Mass Storage Systems and Technology*, pages 119–123. IEEE. (Cited on page 20)
- [Feser et al., 2020] Feser, J. K., Madden, S., Tang, N., and Solar-Lezama, A. (2020). Deductive optimization of relational data storage. *Proc. ACM Program. Lang.*, 4(OOPSLA):170:1–170:30. (Cited on page 11)
- [Fraunhofer ITWM, 2018] Fraunhofer ITWM, T. (2018). BeeGFS - The Leading Parallel Cluster File System. <https://www.beegfs.io>. Accessed: 2020-12-08. (Cited on page 4)
- [Ghemawat and Dean, 2020] Ghemawat, S. and Dean, J. (2020). LevelDB, A fast and lightweight key/value database library by Google. <https://github.com/google/leveldb>. Accessed: 2020-12-08. (Cited on page 14)
- [Grawinkel et al., 2015] Grawinkel, M., Nagel, L., Mäsker, M., Padua, F., Brinkmann, A., and Sorth, L. (2015). Analysis of the ECMWF Storage Landscape. In Schindler, J. and Zadok, E., editors, *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*, pages 15–27. USENIX Association. (Cited on page 5)
- [Hartman and Ousterhout, 1995] Hartman, J. H. and Ousterhout, J. K. (1995). The Zebra Striped Network File System. *ACM Trans. Comput. Syst.*, 13(3):274–310. (Cited on page 7)
- [Hildebrand and Schmuck, 2015] Hildebrand, D. and Schmuck, F. B. (2015). On Making GPFS Truly General. *login Usenix Mag.*, 40(3). (Cited on page 3)
- [IBM, 2016] IBM (2016). General Parallel File System. <https://www.ibm.com/docs/en/gpfs>. Accessed: 2020-12-08. (Cited on page 3)

- [Idreos et al., 2012] Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K. S., and Kersten, M. L. (2012). MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull.*, 35(1):40–45. (Cited on page 11)
- [Kim et al., 2020] Kim, H., So, B., Han, W., and Lee, H. (2020). Natural language to SQL: Where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750. (Cited on page 8)
- [Lakshman and Malik, 2010] Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.*, 44(2):35–40. (Cited on page 16)
- [Lee et al., 2018] Lee, E., Han, Y., Yang, S., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2018). How to Teach an Old File System Dog New Object Store Tricks. In Goel, A. and Talagala, N., editors, *10th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2018, Boston, MA, USA, July 9-10, 2018*. USENIX Association. (Cited on page 20)
- [Liang et al., 2020] Liang, Z., Lombardi, J., Chaarawi, M., and Hennecke, M. (2020). DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory. In Panda, D. K., editor, *Supercomputing Frontiers - 6th Asian Conference, SCFA 2020, Singapore, February 24-27, 2020, Proceedings*, volume 12082 of *Lecture Notes in Computer Science*, pages 40–54. Springer. (Cited on page 7)
- [Lofstead et al., 2016] Lofstead, J. F., Jimenez, I., Maltzahn, C., Koziol, Q., Bent, J., and Barton, E. (2016). DAOS and friends: a proposal for an exascale storage system. In West, J. and Pancake, C. M., editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, pages 585–596. IEEE Computer Society. (Cited on page 6)
- [Lu et al., 2016] Lu, L., Pillai, T. S., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2016). WiscKey: Separating Keys from Values in SSD-conscious Storage. In Brown, A. D. and Popovici, F. I., editors, *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016*, pages 133–148. USENIX Association. (Cited on page 12)
- [Luo and Carey, 2020] Luo, C. and Carey, M. J. (2020). LSM-based storage techniques: a survey. *VLDB J.*, 29(1):393–418. (Cited on pages 12 and 13)
- [Lüttgau et al., 2018] Lüttgau, J., Kuhn, M., Duwe, K., Alforov, Y., Betke, E., Kunkel, J. M., and Ludwig, T. (2018). Survey of Storage Systems for High-Performance Computing. *Supercomput. Front. Innov.*, 5(1):31–58. (Cited on pages 2 and 3)
- [Mahgoub et al., 2017] Mahgoub, A., Ganesh, S., Meyer, F., Grama, A., and Chaterji, S. (2017). Suitability of NoSQL systems - Cassandra and ScyllaDB - For IoT workloads. In *9th International Conference on Communication Systems and Networks, COMSNETS 2017, Bengaluru, India, January 4-8, 2017*, pages 476–479. IEEE. (Cited on page 16)
- [Makris et al., 2019] Makris, A., Tserpes, K., Spiliopoulos, G., and Anagnostopoulos, D. (2019). Performance Evaluation of MongoDB and PostgreSQL for Spatio-temporal Data. In Papotti, P., editor, *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019*, volume 2322 of *CEUR Workshop Proceedings*. CEUR-WS.org. (Cited on page 10)
- [Mansouri et al., 2018] Mansouri, Y., Toosi, A. N., and Buyya, R. (2018). Data Storage Management in Cloud Environments: Taxonomy, Survey, and Future Directions. *ACM Comput. Surv.*, 50(6):91:1–91:51. (Cited on pages 9 and 10)

- [Mäscher et al., 2019] Mäscher, M., Süß, T., Nagel, L., Zeng, L., and Brinkmann, A. (2019). Hyperion: Building the Largest In-memory Search Tree. In Boncz, P. A., Manegold, S., Ailamaki, A., Deshpande, A., and Kraska, T., editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1207–1222. ACM. (Cited on page 14)
- [Mazumdar et al., 2019] Mazumdar, S., Seybold, D., Kritikos, K., and Verginadis, Y. (2019). A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. *J. Big Data*, 6:15. (Cited on pages 12 and 19)
- [Oh et al., 2016] Oh, M., Eom, J., Yoon, J., Yun, J. Y., Kim, S., and Yeom, H. Y. (2016). Performance Optimization for All Flash Scale-Out Storage. In *2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016*, pages 316–325. DOI: 10.1109/CLUSTER.2016.11. (Cited on page 6)
- [Papagiannis et al., 2021] Papagiannis, A., Saloustros, G., Xanthakis, G., Kalaentzis, G., González-Férez, P., and Bilas, A. (2021). Kreon: An Efficient Memory-Mapped Key-Value Store for Flash Storage. *ACM Trans. Storage*, 17(1):7:1–7:32. (Cited on page 14)
- [Potnis, 2018] Potnis, A. (2018). Illuminating Insight for Unstructured Data at Scale. <https://www.ibm.com/downloads/cas/Z2ZBAY6R>. Accessed: 2020-12-08. (Cited on page 3)
- [Qian et al., 2019] Qian, Y., Li, X., Ihara, S., Dilger, A., Thomaz, C., Wang, S., Cheng, W., Li, C., Zeng, L., Wang, F., Feng, D., Süß, T., and Brinkmann, A. (2019). LPCC: hierarchical persistent client caching for lustre. In Taufer, M., Balaji, P., and Peña, A. J., editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, pages 88:1–88:14. ACM. (Cited on page 4)
- [Raasveldt and Mühleisen, 2018] Raasveldt, M. and Mühleisen, H. (2018). MonetDBLite: An Embedded Analytical Database. *CoRR*, abs/1805.08520. (Cited on page 11)
- [Santamaria et al., 2019] Santamaria, P., Oden, L., Gil, E., Becerra, Y., Sirvent, R., Glock, P., and Torres, J. (2019). Evaluating the benefits of key-value databases for scientific applications. In *International Conference on Computational Science*, pages 412–426. Springer. (Cited on page 16)
- [Schultz et al., 2019] Schultz, W., Avitabile, T., and Cabral, A. (2019). Tunable Consistency in MongoDB. *Proc. VLDB Endow.*, 12(12):2071–2081. (Cited on page 17)
- [Selvaganesan and Liazudeen, 2016] Selvaganesan, M. and Liazudeen, M. A. (2016). An Insight about GlusterFS and Its Enforcement Techniques. In *International Conference on Cloud Computing Research and Innovations, ICCCRI 2016, Singapore, Singapore, May 4-5, 2016*, pages 120–127. IEEE Computer Society. (Cited on page 5)
- [Shute et al., 2013] Shute, J., Vingralek, R., Samwel, B., Handy, B., Whipkey, C., Rollins, E., Oancea, M., Littlefield, K., Menestrina, D., Ellner, S., Cieslewicz, J., Rae, I., Stancescu, T., and Apte, H. (2013). F1: A Distributed SQL Database That Scales. *Proc. VLDB Endow.*, 6(11):1068–1079. (Cited on page 9)
- [Siddiqa et al., 2017] Siddiqa, A., Karim, A., and Gani, A. (2017). Big data storage technologies: a survey. *Frontiers Inf. Technol. Electron. Eng.*, 18(8):1040–1070. (Cited on page 12)

- [Smart et al., 2017a] Smart, S. D., Quintino, T., and Raoult, B. (2017a). A Scalable Object Store for Meteorological and Climate Data. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '17*, pages 13:1–13:8, New York, NY, USA. ACM. DOI: 10.1145/3093172.3093238. (Cited on page 6)
- [Smart et al., 2017b] Smart, S. D., Quintino, T., and Raoult, B. (2017b). A Scalable Object Store for Meteorological and Climate Data. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2017, Lugano, Switzerland, June 26 - 28, 2017*, pages 13:1–13:8. ACM. (Cited on page 20)
- [solid IT, 2021] solid IT (2021). DB-Engines Ranking. <https://db-engines.com/en/ranking>. Accessed: 2021-07-08. (Cited on pages 11 and 16)
- [Suneja, 2019] Suneja, N. (2019). Scylladb optimizes database architecture to maximize hardware performance. *IEEE Softw.*, 36(4):96–100. (Cited on page 16)
- [Vef et al., 2020] Vef, M., Moti, N., Süß, T., Tacke, M., Tocci, T., Nou, R., Miranda, A., Cortes, T., and Brinkmann, A. (2020). GekkoFS - A Temporary Burst Buffer File System for HPC Applications. *J. Comput. Sci. Technol.*, 35(1):72–91. (Cited on page 5)
- [Vef et al., 2018] Vef, M., Tarasov, V., Hildebrand, D., and Brinkmann, A. (2018). Challenges and Solutions for Tracing Storage Systems: A Case Study with Spectrum Scale. *ACM Trans. Storage*, 14(2):18:1–18:24. (Cited on page 3)
- [Vilayannur et al., 2004] Vilayannur, M., Ross, R. B., Carns, P. H., Thakur, R., Sivasubramaniam, A., and Kandemir, M. T. (2004). On the Performance of the POSIX I/O Interface to PVFS. In *12th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2004), 11-13 February 2004, A Coruna, Spain*, pages 332–339. DOI: 10.1109/EMPDP.2004.1271463. (Cited on page 5)
- [Vora, 2011] Vora, M. N. (2011). Hadoop-HBase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 1, pages 601–605. (Cited on page 16)
- [Weil et al., 2006] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006). Ceph: A Scalable, High-Performance Distributed File System. In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 307–320. (Cited on page 6)
- [Weil et al., 2007] Weil, S. A., Leung, A. W., Brandt, S. A., and Maltzahn, C. (2007). RADOS: a scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2nd International Petascale Data Storage Workshop (PDSW '07), November 11, 2007, Reno, Nevada, USA*, pages 35–44. DOI: 10.1145/1374596.1374606. (Cited on page 6)
- [Wu et al., 2015] Wu, X., Xu, Y., Shao, Z., and Jiang, S. (2015). LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items. In Lu, S. and Riedel, E., editors, *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, pages 71–82. USENIX Association. (Cited on page 12)
- [Yang et al., 2015] Yang, F., Dou, K., Chen, S., Hou, M., Kang, J., and Cho, S. (2015). Optimizing NoSQL DB on Flash: A Case Study of RocksDB. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated*

Workshops (UIC-ATC-ScalCom), Beijing, China, August 10-14, 2015, pages 1062–1069. IEEE Computer Society. (Cited on page 14)

[Yang and Lilja, 2018] Yang, J. and Lilja, D. J. (2018). Reducing Relational Database Performance Bottlenecks Using 3D XPoint Storage Technology. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 1804–1808. IEEE. (Cited on page 10)