Semantic Hybrid Multi-Model Multi-Platform (SHM3P) Databases

Sven Groppe

Institute of Information Systems (IFIS), University of Lübeck, Ratzeburger Allee 160, D-23562 Lübeck, Germany

Abstract

Today's companies have to handle a zoo of data of different models. Multi-model databases promise to simplify data administration for the parallel usage of different data models. Compared to the other data models, semantic data models introduce an additional abstraction layer for reasoning purposes, such that semantic data models provide superior capabilities. Hence semantic multi-model databases use the semantic data model as main glue between the different data models. Furthermore, applications as well as databases are today running on different platforms like mobile devices, web, desktops, servers, clouds and post-clouds (e.g., fog and edge computing). Hybrid multi-model multi-platform (HM3P) databases and its semantic counterpart (SHM3P databases) integrate the different platforms in order to offer their advantages and benefits for data distribution, query processing and transaction handling to their users. In this paper we introduce and discuss the novel concept of SHM3P databases and its open challenges.

Keywords

Semantic Web, databases, multi-platform, multi-model, cloud, post-cloud, edge computing, fog computing, dew computing, hardware acceleration, Internet-of-Things, mobile database, parallel database, main-memory database

1. Introduction

Today companies have to deal with and process data in various data formats: The backends of their web shops with databases about customers and their orders are typically connected to relational databases. Product catalogs of companies are often exchanged using XML, JSON or RDF. The boom of social networks leads to a high demand to process their graph data, other social media like wikis offer their data as unstructured data. Key-value stores are often used whenever data must be accessed in a simple way just via keys. However, there is also a need for schema-free or schema-less databases, which don't ask the data to stay in the inflexible corset of a schema, but still working on complex data formats like document stores. The data is hence stored according to and processed using different models (*multi-model data* [1]). The big challenge for today's companies are the synchronization and integration of their multi-model data into a single view of and for the customer [2]. **Multi-Model Database Management Systems (MM-DBMSs)** offer the management of different data models in one single database [1] in order to overcome the disadvantages of polyglot persistence, where applications use several databases at the same time to handle multi-model data [3] hindering optimizations down to the physical layer of connected DBMSs [4]. Furthermore, we propose the semantic data model in order to unify the other data models, because the semantic data model offers the ontology layer as additional abstraction layer, which can be utilized for data integration purposes of the other data models.

While in the past database management systems (DBMSs) run mainly on parallel servers, there are today various different platforms like mobile devices, web, desktops, servers (maybe additionally hardware accelerated by GPUs, FPGAs and in future scenarios even quantum computing), clouds and post-clouds (e.g., fog and edge computing) offering execution environments for running a DBMS[1].

Multi-platform development (as supported by e.g. the programming language Kotlin [5]) allows to share common code between different platforms like desktop, server, web, mobile and IoT. Multi-platform development reduces the development costs for a DBMS running on multiple platforms drastically.

Puzzling all pieces together we propose the following definitions ((H)M3P DBMS are defined according to [4]):

Definition 1 (M3P/HM3P/SHM3P DBMS). *A Multi-Model Multi-Platform Database Management System (M3P DBMS) is a MM-DBMS that can be executed on different platforms. A hybrid M3P (HM3P) DBMS spans over different platforms in operation. A Semantic HM3P*

[1]Note that clients of DBMSs typically run on different platforms, but we are considering the database server here.

ISIC'21: International Semantic Intelligence Conference, February 25–27, 2021, New Delhi, India

groppe@ifis.uni-luebeck.de (S. Groppe)

0000-0001-5196-1117 (S. Groppe)

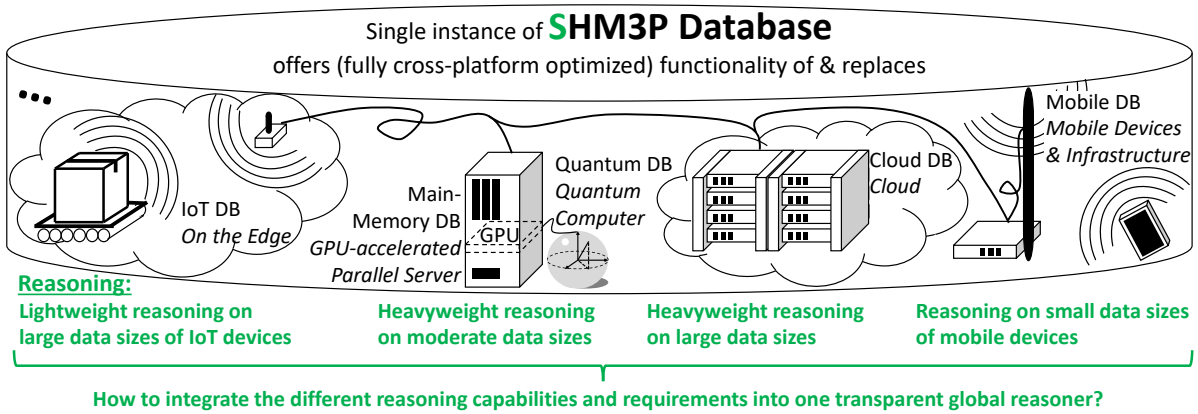

Figure 1: SHM3P database spanning over multiple platforms. Here, an SHM3P database replaces an IoT database in an Industry 4.0 scenario (*using edge-computing*), a GPU-accelerated parallel database (*on a parallel server*) for archiving and generating long-term statistics of the IoT data, which is further supported by a *quantum computer* for query and reasoning optimization, a database *in the cloud* for natural language processing tasks and a mobile database (*on mobile devices and infrastructure*) for monitoring and controlling of the production line in the company. Platforms are marked with an *italic font*. Green text marks discussion about reasoning in these scenarios. Figure is based on [4] and extended by the discussion on reasoning.

(SHM3P) DBMS supports a (global) semantic layer (for querying and reasoning purposes) over all platforms of an HM3P DBMS.

Whereas today's M3P DBMSs are typically developed for platforms of the same type (like windows and linux servers, see Section 2.1), some other even span over a (locally installed) private cloud and a public cloud (in a so called *hybrid cloud*[2]). In contrast, we envision SHM3P DBMSs over platforms of *different type* (like IoT and hardware-accelerated parallel servers) integrating the features of databases developed for these platforms (like energy-savings on IoT devices and high throughput on servers) while offering advanced global reasoning capabilities over all platforms. Hence SHM3P databases support any data model at any platform by tightly integrating them with a semantic layer. For an example installation, see Figure 1.

Our main contributions are:

- the introduction of SHM3P DBMS as new type of DBMS,
- a detailed discussion of the current state of the art about and comparative analysis of DBMS designed for different platforms with special attention to Semantic Web DBMS, and
- a discussion about open research challenges for HM3P DBMS and SHM3P DBMS.

The remainder is as follows: Section 2 describes the basics and an analysis of current state-of-the-art concerning MM-DBMSs, multi-platform development, databases running on different platforms, polyglot persistence and further related work. Section 3 introduces SHM3P DBMSs and explores the advantages, and analyses envisioned platforms and common properties of their combinations. Finally we summarize the results and provide an overview of future work in Section 4.

2. Basics

2.1. Databases for Multi-Model Data

Polyglot persistence uses different databases supporting different data models (and maybe running on different platforms) within one application [3]. Federated query languages enable polyglot persistence by supporting queries over heterogeneous data stores within one single query. One example of such a query language is CloudMdsQL [6], with which one can formulate queries over SQL and NoSQL databases. The proposed prototype even optimizes the queries globally and pushes operations down to the integrated SQL and NoSQL databases as much as possible. A similar approach is taken by [7] offering to query cloud-based NoSQL like Google's Bigtable and relational databases with the Google Bigtable query language GQL. The focus of Apache Drill[3] is interactive ad-hoc analysis of large-scale datasets with low-latency handling up to petabytes of data spread across thousands of servers.

[2] Please note that private and public clouds are platforms of the same type.

[3] https://drill.apache.org/ (accessed on 17.12.2020)

Drill optimizes a query plan to leverage the datastore's internal processing capabilities and by considering data locality. Commercial multi-store products like IBM BigInsights, Microsoft HDInsight and Oracle Bigdata Appliance as well as open source projects like PrestoDB[4] integrate diverse data sources by using database connectors (like JDBC drivers). Tatooine [8] uses a semantic layer as glue between databases for different data models supporting a semantic integration. However, all these polystores also don't support to fully optimize queries across the integrated, but independent data sources, which limit data processing.

Federation Databases [9] and multidatabases [10] place a mediator between different autonomous databases for integration purposes by reformulating queries according to a global schema to the native schemes of the integrated databases, which afterwards execute these queries. Today, some research focus on federating databases following the polyglot persistence approach: For example, DBMS+ [11] provides unified declarative processing for the integration of several processing and database platforms. BigDAWG [12] offers location transparency while running queries against the three different integrated systems PostgreSQL, SciDB and Accumulo.

Multi-Model Databases: A multi-model database is one single database for multiple data models, which fully integrates a backend to offer advanced performance, scalability and fault tolerance [13]. One of the first of this type are Object-Relational DataBase Management Systems (ORDBMSs), which support various data models like relational, text, XML, spatial and object. ORDBMSs use the relational technology for implementing the support of their data models, i.e., the relational model is the first-class citizen. In comparison and in general, in multi-model databases the different models can be all first class citizens and supported in a native way (utilizing e.g. specialized indices for them). The authors in [14] propose to use a semantic layer as glue between the different data models in order to support global querying and reasoning over all data. We extend this idea to multi-platform databases integrating the technologies and features of different types of databases.

[4] contains an overview of current state-of-the-art multi-model databases, their type of extension, their supported data models, query languages and platforms. The investigated multi-model databases support at most 5 from 8 data models, such that no multi-model database offers all data models to their users. From the investigated 21 MM DBMS only 5 support RDF as data model, where most of which, i.e. 4 of these 5 MM DBMS with RDF support, also manage graph data. The graph model seems to be more popular (12 from 21 MM DBMS). MM DBMS with RDF support typically don't support reasoning at all or only in a rudimentary way, such that users should look for native semantic DBMS if reasoning is needed. Hence reasoning seems to be challenging in the MM DBMS context. Most multi-model databases run SQL, SQL-like or extensions of SQL queries. Binaries of these databases are offered in machine code (often compiled from C/C++) or for the Java virtual machine (JVM). They usually run on all or a big subset of the major desktop operating systems linux, windows, macOS, unix and their variants. Few multi-model databases like IBM DB2 run on mainframes operating e.g. z/OS. While all offer to run in the cloud, some are also enabled for the hybrid cloud. In the hybrid cloud, a (locally installed) private cloud is together used with a public cloud. Hybrid clouds decrease costs spent to the public cloud provider while still having on-demand resources with the illusion of infinite capacity at the public cloud for a surprising high resource demand.

While all multi-model databases run on different platforms, they don't integrate database instances on different types of platforms and different types of databases. Databases in hybrid clouds combining the resources of a locally installed private cloud with a public cloud are approximations of the idea of operating on multiple platforms of different types. An HM3P DBMS extends this idea and supports multiple types of platforms like main-memory, cloud, Internet-of-Things (with e.g. edge computing) and hardware-accelerated databases using their different advantages at runtime for database tasks like data distribution, transaction handling and query processing. A SHM3P DBMS offers a semantic layer as glue between the different data models and supports global semantic querying and reasoning by tightly integrating local query engines and reasoners.

2.2. Multi-Platform Development

There are several programming languages like C/C++ available compiling to various platform targets in their native machine code best suitable for high performance programs. Calls to the operating system for disk accesses or developing a (native) graphical user interface must be ported to the different platforms. There is no special support for multi-platform development like code-sharing of common code and allowing to define platform-specific modules to code the differences between the different platforms. Java was one of the first programming languages for developing one code run-

[4] https://prestodb.io/ (accessed on 17.12.2020)

ning on different platforms, which is still the key for the success of Java. It has been implemented by compiling to bytecode, which is processed in the Java virtual machine (JVM) available for many platforms. The JVM introduces an intermediate abstraction layer, but also some performance overhead, although the bytecode is often just-in-time (JIT) compiled to native machine code. Scripting languages like JavaScript also run on different platforms (i.e., wherever browsers and Node.js environments can be started). JavaScript besides HTML 5 is the basis of cross-platform libraries like React Native and PhoneGap. Advanced multi-platform support introducing a module concept for sharing common code between the different platforms, and platform-specific modules for coding remaining differences, is introduced by modern programming languages like Kotlin [5]. Kotlin offers multi-platform support for the JVM (Desktop, Server and Android), JavaScript engines (browser and server via Node.js) and via LLVM Windows, Linux, Android (arm32/64), MacOS, iOS, Raspberry Pi and WebAssembly.

Many DBMSs are implemented in C/C++ for performance reasons and run in native machine code for operating systems like Windows, Linux, Unix and MacOS (see [4]). Some modern DBMSs and most Semantic Web tools (see [15]) are implemented in Java further decreasing development costs, but still running on clusters and servers operating Windows, Linux, Unix and MacOS. Real multi-platform tools by e.g. using Kotlin multi-platform projects are missing so far for Semantic Web tools.

2.3. Databases for different Platforms

Most DBMSs and their clients run on different platforms. There exist usually also numerous language bindings for APIs calling database functionalities from database applications.

Multi-Platform DBMSs are typically either implemented in C/C++ or in Java. Ports are often available for Windows, Linux, Unix (sometimes for Solaris) and MacOS (see [4]). Only few DBMSs still run on mainframes. Modern DBMSs run in the Cloud and sometimes they are offered only as managed service in the Cloud (e.g., Cosmos DB). Some few are also running in a *Hybrid Cloud*, where the DBMS is running in a local installation of a cluster (private cloud) as well as in a public cloud (of a cloud provider). [15] contains a selection of 18 widely-used Semantic Web tools including triple stores and Semantic Web databases. Over half of these tools are implemented in Java (i.e., 6 of these tools run on any platform, which supports java) or support java language bindings (4 of these tools). Semantic Web tools with native binaries run usually on any desktop and server computers, some only on linux operating systems.

Hence these DBMSs can be called Multi-Platform DBMSs, but don't bring the multi-platform approach to its full potential. They are typically developed for one type of platform: server, cluster or cloud. DBMSs designed for different types of platforms like cluster, mobile, IoT and the web are not considered so far. HM3P DBMSs span over different platforms at runtime, which may be the case for hybrid cloud installations, but which are also not deployed at different platform types. Hence, full-fledged HM3P DBMSs have to consider various different properties (e.g., availability of nodes, storage and computing resources), the data (like security concerns) and queries (like one-time versus continuous queries) of the supported platforms at runtime for data distribution and processing. Reasoning support is not available for all platforms and types of queries [16]: While many contributions exist for RDFS and OWL support during one-time query processing on server and desktop computers, there exist only few approaches for the cloud and for P2P networks. There exist only few approaches for trigger and continuous queries with RDFS and OWL support on server and desktop computers as well as for the cloud. Ontology inference for trigger and continuous queries in P2P networks haven't been considered so far. The development of an SH3MP database may help to support ontology inference in trigger and continuous queries with reasonable efforts also on these platforms.

Multi-Platform Clients offering to set up queries and displaying their results are available for all DBMSs[5]: DBMSs typically offer clients for platforms like the Web, major desktop operating systems like Windows, Linux, Unix and MacOS, mobile apps like android and iOS. Some clients are even implemented as cross-platform application[6], which also support different DBMSs. The situation is quite comfortable for the Semantic Web: The W3C standardized the protocol to query SPARQL endpoints in [17]. The protocol [17] is widely supported and hence the Semantic Web DBMSs as well as the clients can be easily exchanged.

The user may have the impression that a database may be running on different platforms, because (s)he gets in touch with clients for the database available for different platforms. However, the DBMS does neither store nor process the data on the clients' computer, but only transfers the query result to it. We envision a SHM3P DBMS, where the advantages of the different

[5] We consider PostgreSQL and its clients as example here.

[6] For example, DBeaver available at https://dbeaver.io/ (accessed on 17.12.2020).

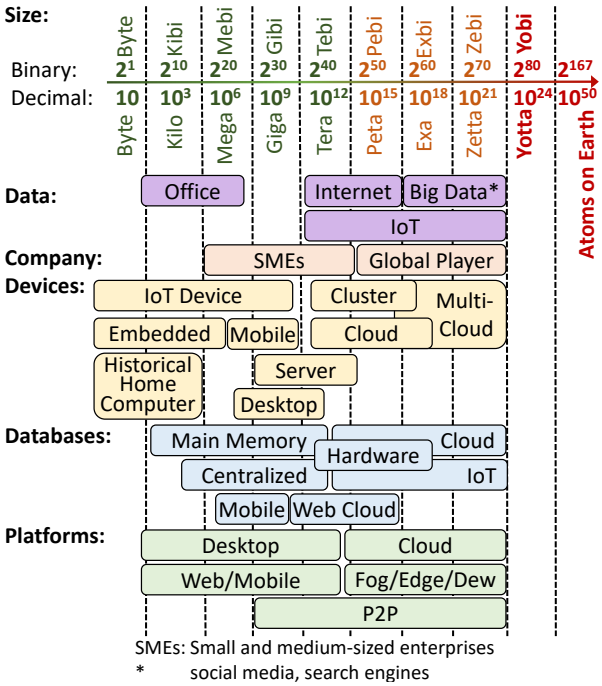

Figure 2: Data sizes in companies, devices, databases and platforms. See [18] for the estimation of atoms on earth.

platforms are utilized for data storing and processing, and the overall best approaches are chosen according to the platform properties.

3. Multi-Platform Multi-Model Databases

Figure 2 provides an overview over data sizes of different types of data used in companies, devices, databases and platforms. It already becomes obvious that some types of databases fit better to the considered types of data and company, used devices and platforms than the others. Hence the different types of data are stored on and processed at different platforms dependent on their size, the devices they are generated at and other properties like their velocity. Integrating these data sets implies to support multiple models and also different platforms at the same time. This also requires to support and integrate different types of databases running on different platforms. For example, one might combine the data of IoT devices (stored in an IoT database running on the edge of the network) with the accounting data containing the remaining time for charging off (stored in a main memory database running on an employee's desktop computer). These different types of databases have different properties and advantages because they have been developed for different application scenarios, devices, properties of their indexed data (velocity, heterogeneity, size etc.) and so on. Table 1 contains a rough evaluation of these databases. Databases have tailored their architectures according to the properties of the different platforms, but often also to the required properties coming from their applications. Especially distributed databases cannot offer all: The well known PAC theorem [19] describes trade-offs, where developers of distributed systems (and hence also distributed databases) can choose to fully support only two features with high efficiency out of three: **P**artition-tolerance, **A**vailability and **C**onsistency. For example, if the system works correctly also in the case of network partitions and is highly available, then consistency must be relaxed, such that some replicas may contain older states and not the most recent ones. The PACELC theorem [20] refines the PAC theorem and states that in the case of network **P**artitions only **A**vailability or **C**onsistency is guaranteed. In case of no failures when the databases run normally (**E**lse), then there is a trade-off between **L**atency and **C**onsistency, i.e., only small latency or high consistency can be guaranteed, but not both at the same time. Distributed triple stores, which are built on top of NoSQL databases, inherit the properties of their underlying systems: For example, D-SPARQ [21] supports PA/EC, because it is based on MongoDB[7]. Jena-HBase [22], H2RDF [23] and H2RDF+ [24] inherit the PC/EC properties of HBase[8]. CM-Well[9] is based on Cassandra[10] supporting PA/EL. Remaining research challenges include hybrid approaches supporting PA and PC (as well as EL and EC) for different fragments of the data at the same time according to their applications.

Hence there is a need to run these different types of databases at the same time, but there might be also the need for integrating the data of these databases (like in the scenario of combining the data of IoT devices with accounting data). For an advanced processing of this different types of data stored in different databases and other database tasks it is indispensable to break the boundaries of single installations of these DBMSs and to run one single DBMS. Furthermore, it is desirable that this single DBMS provides a semantic layer for advanced processing and reasoning capabilities and for a tight integration of the different data models. This would also allow to offer the best features

[7] https://www.mongodb.com/ (accessed on 17.12.2020)
[8] https://hbase.apache.org/ (accessed on 17.12.2020)
[9] https://github.com/CM-Well/CM-Well (accessed on 17.12.2020)
[10] https://cassandra.apache.org/ (accessed on 17.12.2020)

of the different types of databases to applications and users "under one hood" transparently or with an intelligent integration into one query language and API. This single SHM3P DBMS installation runs over all platforms at the same time offering the advantages of all the different types of DBMSs (to the data that has been previously processed by the single installations) tightly integrated in a semantic layer, but to have e.g. a global optimization of data distribution, transaction handling and global queries and reasoning tasks with full potential by having freedom of processing down to the physical layer (e.g., index accesses)[11]. One single SHM3P DBMS would also reduce development costs of applications and periods of vocational adjustment of developers by offering one API and query language with an additional semantic layer for all different platforms. A very big challenge for SHM3P DBMSs is to provide a global distributed reasoner, which integrates different types of reasoners to be processed on the different platforms, where reasoning is optimized for this heterogeneous environment minimizing overall costs combining weighted costs of different types (communication, processing, lifetime of IoT devices etc.).

3.1. Platforms

We describe shortly the different platforms running execution environments for different types of DBMSs here.

Server Platforms are typical platforms for database servers of small to medium-sized enterprises (SMEs). The DBMSs running on servers are usually centralized databases, which are operating in parallel on multi-core and sometimes many-core systems, often in virtual machines. Relational DBMSs, most Semantic Web DBMSs and Reasoners are typically running on server platforms, and all other types of DBMSs usually offer a local mode to run on a single server.

Hardware-Accelerated Servers speed up database tasks by utilizing the massive parallelism of special hardware behind today's multi-core CPUs.

Modern Graphical Processing Units (GPUs) consist of several thousand computing cores, which follow the single-instruction multiple-data paradigm, i.e., the same instruction is executed on different data on different cores at the same time. GPUs are often regarded as special form of many-core CPUs. Hence, neither all parallel algorithms are suitable for nor benefit from GPUs. However, the massive parallel processing of execution plans are ideal for many-core CPUs and GPUs as well as whenever the best possibilities among enumerated ones must be found (like in query optimization and multi-version concurrency control (MVCC)). Complex operations like joins processing large data inputs are very suitable for GPU-acceleration, too (see e.g. [25] for especially designed joins for SPARQL processing on GPUs).

Field-programmable gate arrays (FPGAs) can reconfigure interconnects for connecting programmable logic blocks with each other. This property makes FPGAs ideal suitable for data-flow-driven algorithms (like processing an execution plan for evaluating queries in a streaming way without block-wise materialization of intermediate steps like it is the case for many-core CPUs and GPUs), but also any arbitrary type of parallelism can be offered by FPGAs. FPGA-acceleration of SPARQL query processing as discussed in e.g. [26] achieves scalable speedups even increasing with larger data sets. Dynamic partial reconfiguration enables FPGAs to dynamically exchange their configurations to process different queries at runtime [26].

Universal quantum computers try to combine the full power of classical computers with quantum computers that manipulate (some few) qubits in super position by applying quantum logic gates. In comparison, quantum annealers - operating on up to several thousand qubits - only run special types of quantum algorithms to solve adiabatic (as special form of combinatorial) optimization problems, which is e.g. the case for traffic control[12], selecting the execution plan with the best estimated costs (from a set of enumerated plans) [27], concurrency control between transactions [28] as well as optimizing transaction schedules [29, 30].

Cloud Databases are designed to be run in the cloud, where (storage and computing) resources can be dynamically allocated and freed according to users' demands. Hence, cloud databases must consider that nodes (for storing and computing) are joining and leaving, such that it may be necessary to redistribute data and to react for processing jobs on leaving nodes. Furthermore, as the nodes are typically not high-end hardware like servers with redundant components and clouds consist of many more nodes (up to several thousand nodes), hardware and communication failures may occur more often. Hence, cloud computing architectures apply simple fault-tolerance mechanisms by repeating crashed jobs. Table 2 contains an overview

[11] Note that single installations of DBMSs can only be accessed via their offered APIs or by setting up subqueries (of the global query) to them, which hinders the full potential of optimized processing of e.g. joins between the data of the different DBMSs.

[12] investigated by Volkswagen, see https://www.volkswagenag.com/en/news/stories/2018/11/intelligent-traffic-control-with-quantum-computers.html (accessed on 17.12.2020)

Table 1
Rough Evaluation of different Types of Databases.

Feature \ DBMS	Main Memory	Parallel	Distributed	Federated	Cloud	Web Cloud	Mobile	IoT
Scalability	−	o	+	+	+ +	+ + +	+	+ +
Transaction rates	+ + +	+ +	o / +	o	+ +	+	−	− −
Intra-Transaction Parallelism	+ + +	+ +	o / +	− / o	+	o	−	−
Atomicity	+ + +	+ + +	+ +	+	+	+	+	+
Durability	+	+	+ + +	+ +	+ +	−	o	−
Consistency	+ + +	+ + +	+ +	+	+	+	+	+
Extensibility	−	+	o / +	o	+ +	+ + +	−	+ + +
Schemaless	− − −	− − −	− − −	−	+ + +	+ + +	+	+ + +
Availability	+ +	+	+	−	−	− − −	− −	− − −
Transparency of Distribution	+ +	+ +	+	o	+ +	−	−	− −
Geographical Distribution	− −	−	+	+	+ +	+ + +	+ +	+ +
Mobility	−	−	−	o	o	o	+ +	+
Node Autonomy	− −	−	o	+	o	− −	+ +	+
Heterogeneity of DBMS	− −	−	−	+	−	−	+ +	+ + +
Administration	o	o	−	− / − −	−	+ +	− −	− − −
Hardware Costs	−	− −	−	−	+ +	+ + +	−	+ + +
Reasoning	+ + +	+ + +	+	− −	+ +	+	− −	− − −

Table 2
Evolution of Big data analytics engines. Based on [16] and extended by the rows "Impact on Databases" and "Impact on Reasoning".

Generation:	1	2	3	4
Features:	Batch	+ Interactive	+ Near-Real-Time + Iterative Processing	+ Real-Time Streaming + Native Iterative Processing
Processing Model:	MapReduce	DAG Dataflows	Resilient Distributed Datasets (RDD)	Cyclic Dataflows
Impact on Databases:	Long-Running Queries	Query Answering with lower latency	+ Continuous Queries	+ Real-Time Continuous Queries
Impact on Reasoning:	Long-Running Reasoning	Reasoning with lower latency	+ Capabilities for Stream Reasoning	+ Cap. for Real-Time Stream Reasoning
Engine:	Hadoop	TEZ	Spark	Flink

over important state-of-the-art Big Data analytics engines working in cloud environments. Additionally to one-time queries, Apache Spark and Apache Flink offer to process data streams and continuous queries, such that they also belong to the type of **stream databases**. There exists various examples of Semantic Web databases on top of the different Cloud technologies like [31] (HBase, Pig), [32] (Spark) and [33] (Flink), but also other contributions avoiding to use the well-known technologies like [34] in order to support local joining. **Web Cloud Databases** rely on a new form of cloud: the web cloud [35]: One just visits with his/her web browser a certain webpage in order to connect his/her computer to the web cloud. In this way the setup of the web cloud is much easier than those of traditional clouds. Furthermore, the web cloud has a much larger number of potential nodes, as any computer running a browser may connect to and be integrated in the web cloud. New challenges arise when setting up a cloud by web browsers: The nodes may be more often disconnected. Data is processed within the browser and hence we must use the tech-

nologies offered by the browser for data management purposes. New technologies like WebAssembly [36] introducing a virtual machine for the browser may help to speed up processing in the browser. There exist first approaches to distribute SPARQL queries in some kind of web clouds [37].

Mobile Databases [38] involve the technical infrastructure of mobile providers like base stations (being near-by to their connected mobile devices) in order to speed up processing, lower communication (and hence also energy) costs, increase availability and durability (by logging at the base stations instead on mobile devices) in order to overcome limitations of the mobile devices. Some RDF stores like [39] are especially designed to run on mobile devices, but they do not consider the backend of mobile providers so far.

P2P Databases [40, 41] use peer-to-peer (P2P) networks as underlying backend technology to master a frequent joining and leaving of nodes for data storing and processing. In comparison to clouds, they are designed for a much more frequent change in their topology and for an equal distribution of functionality without distinction of master and slave nodes. P2P databases have to introduce more redundancy in data storing as well as even in processing in order to overcome the frequent disconnections to their nodes. Furthermore, P2P databases must consider heterogeneity in the connected nodes much more than other types of databases. There exist already quite many approaches for semantic data processing in P2P networks like [41], but ontology inference is considered only on a rudimentary basis and for trigger and continuous queries not at all [16].

IoT Databases [42] are especially developed to serve as data store for large-scale installations of the Internet-of-Things (IoT). IoT databases often operate in the cloud, but the communication bootleneck from the IoT devices to the cloud doesn't scale especially for IoT devices with high velocity and large-scale installations.

In companion with the cloud, fog computing [43] stores and processes data and application logic on near-things edge devices with higher capabilities (rather than primarily in cloud data centers), which saves communication avoiding the route over the internet backbone. However, fog computing is not really scalable in the number of connected things, as the near-things edge devices do not increase in number and capabilities in the same way.

The scalability issue is solved in a better way by edge computing [44], which utilizes additionally all IoT devices for data storage and processing, and executing application logic: As more IoT devices are deployed, as more data needs to be stored and processed, but as more IoT devices are also available.

Dew computing [45, 46] overcomes availability problems, where the communication between cloud and IoT devices is disturbed, by placing an additional local server near to the IoT devices taking over the tasks of the cloud during downtimes and synchronizing with the cloud at uptimes.

Besides many approaches to semantic IoT like corresponding ontologies [47] and interoperability issues [48], there are not so many contributions to semantic IoT databases. IoT databases are often organized as P2P database, especially if they work on the fog or edge, or follow the dew computing concept. Hence contributions to P2P networks processing Semantic Web data like [41] are relevant for semantic IoT databases as well. One of the big challenges here is the distribution of data and processing tasks between cloud and IoT infrastructure including the devices themselves. Furthermore, IoT devices often generate data streams, such that organizing the IoT database as stream database is a reasonable choice: The IoT application design may especially consider to reduce data by aggregation and focusing on only relevant data, which should be done nearby the things. One research direction may consider how to use Semantic Web technologies for defining such aggregation tasks. Reasoning at data sources or nearby, or in clouds is another difficult question and not so easy to answer in comparison to query processing on the fog or edge, as reasoning consumes much more processing resources.

3.2. (S)HM3P Databases and their Challenges

HM3P databases are single installations of a M3P DBMS, which are not only able to run on multiple platforms, but runs and tightly integrates different types of DBMSs for ease of use and optimization purposes *at runtime*. SHM3P databases integrate the different types of DBMSs in an additional semantic layer and supports global reasoning over all integrated DBMSs.

IoT databases operating at the same time in clouds and on fog, edge or dew computing are reasonable examples for H3MP DBMSs: They span over different platforms, the edge of the IoT network and the cloud data centers, and have to distribute functionality like data aggregation at or near to the things and complex operations, e.g., natural language processing and reasoning, at the cloud data centers. Furthermore, IoT databases have to consider different types of query processing by dealing with traditional (one-time) queries on static data, continuous queries on data streams and spatial-temporal queries on archived data of data streams.

IoT devices are often heterogeneous because they are e.g. developed by different manufacturers: the use of ontologies and hence of semantic databases simplifies the integration of these devices. Semantic IoT databases sometimes manage data at the IoT devices in the traditional way for performance reasons and only support reasoning and semantic querying at the cloud centers after transforming the data of IoT devices to semantic data [16]. Other approaches support even reasoning on streams [16].

Multi-platform DBMSs are already highly ambitious even for large, established database companies since it requires data management skills in an extremely wide spectrum (i.e., data management issues in sensors and smart objects for IoT databases are completely different from the challenges of in-memory databases of P2P data oriented systems and semantic querying and reasoning of Semantic Web databases). Hence current approaches are more on interoperability between the variety of DBMSs, each one focusing on its specific issues related to its specific functionalities. However, we propose to support a global approach to integrate all these specific functionalities in order to use their different benefits in an uniform way and to increase the overall benefits of the global approach.

New **challenges of M3P and HM3P DBMSs** in comparison to traditional DBMSs and MM-DBMSs are

- developing only one code base for the different platforms, but not introducing performance overhead in comparison to single platform databases[13]
- identifying common properties of several platforms and reusing those approaches (like fault tolerance mechanisms) in different combinations, which are best suitable for these considered platforms
- data distribution among different platforms (applying different data distribution approaches as well)
- efficient binary serialization and communication protocols for integrating the different platforms
- data distribution strategies considering overall the different properties of used platforms and models (like fast reads in relational databases on parallel servers and fast updates in cloud databases)
- query optimization and other database tasks across different platforms, which apply different database approaches
- dealing with and integrating different privacy and security mechanisms supporting different privacy and security levels in the different platforms (with research e.g. on querying heterogeneous encrypted data)
- developing multi-platform transaction synchronization approaches and supporting global transaction synchronization approaches over distributed different transaction synchronization approaches running on different platforms
- combining different types of databases (on different platforms) to offer the best of these databases and platforms *under one hood* to applications and users transparently or via intelligent integration into query language and API, e.g., guaranteeing atomicity and isolation in transactions for the data stored on a parallel server, but not for those data in the cloud supporting fast updates

Specific **challenges of SHM3P DBMSs** are

- integrating different data models in a semantic layer on top of the underlying data models
- efficient transformations from and to the semantic model in an operational system
- developing efficient semantic querying and reasoning over the integrated data of different models
- global reasoning over reasoners running on different platforms supporting some kind of distributed heterogeneous reasoning
- developing a combination of stream reasoning over streaming data (e.g. of IoT devices) with static reasoning over large-scale data sets (stored e.g. in clouds)
- supporting transactions over semantic data by integrating the reasoner in transaction synchronization

We are sure that this is not an exhaustive list of new challenges. Many further challenges will arise during developing the (S)(H)M3P DBMSs and considering especially combinations of different platforms and models at runtime.

4. Summary and Conclusions

Multi-model databases provide the infrastructure to handle the zoo of data models managed in today's companies. Multi-model databases that are able to run on a variety of platforms, which are typically deployed and in use in parallel in today's companies, are called multi-model multi-platform database management systems (M3P DBMSs). Hybrid M3P (HM3P) DBMSs span over different platforms at run-time. Our focus is on its semantic counterpart: Semantic HM3P (SHM3P) DBMSs offer its additional semantic layer for simple integration of the DBMS technologies of its operational platforms. Furthermore, we describe and analyze different types of DBMSs and platforms concerning their properties, chances and challenges for DBMSs with spe-

[13] We are of the opinion that this is possible by applying Kotlin features like expected and actual declarations for classes and types, and inline functions and classes.

cial focus on Semantic DBMSs. Current state-of-the-art (S)M3P DBMSs don't exploit the multiple platform idea to its full potential, because they typically only tightly integrate one type of platform and database. We see great further optimization possibilities in data and functionality distribution like query processing, reasoning and transaction handling, and ease of usage when different types of platforms and databases are supported in one single installation of a M3P DBMS by tightly integrating them based on a semantic layer.

References

[1] J. Lu, I. Holubová, Multi-model databases: A new journey to handle the variety of data, ACM Computing Surveys (CSUR) 52 (2019).

[2] R. Kotorov, Customer relationship management: strategic lessons and future directions, Business Process Management Journal 9 (2003) 566–571.

[3] S. Leberknight, Polyglot persistence, Scott Leberknight's Weblog, http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence, 2008.

[4] S. Groppe, J. Groppe, Hybrid multi-model multi-platform (hm3p) databases, in: Proceedings of the 9th International Conference on Data Science, Technology and Applications (DATA), 2020.

[5] JetBrains s.r.o., FAQ - Kotlin Programming Language, 2020. URL: https://kotlinlang.org/docs/reference/faq.html.

[6] B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, J. Pereira, Cloudmdsql: querying heterogeneous cloud data stores with a common language, Distributed and Parallel Databases 34 (2016) 463–503.

[7] M. Zhu, T. Risch, Querying combined cloud-based and relational databases, in: International Conference on Cloud and Service Computing, 2011, pp. 330–335.

[8] R. Bonaque, T. D. Cao, B. Cautis, F. Goasdoué, J. Letelier, I. Manolescu, O. Mendoza, S. Ribeiro, X. Tannier, M. Thomazo, Mixed-instance querying: a lightweight integration architecture for data journalism, PVLDB 9 (2016) 1513–1516.

[9] M. Hammer, D. McLeod, On Database Management System Architecture., Technical Report, MIT, Cambridge Laboratory for Computer Science, 1979.

[10] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, E. Wong, Multibase: Integrating heterogeneous distributed database systems, in: AFIPS National Computer Conference, 1981, pp. 487–499.

[11] H. Lim, Y. Han, S. Babu, How to fit when no one size fits., in: CIDR, 2013.

[12] A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska, S. Madden, D. Maier, T. Mattson, S. Papadopoulos, J. Parkhurst, N. Tatbul, M. Vartak, S. Zdonik, A demonstration of the bigdawg polystore system, Proc. VLDB Endow. 8 (2015) 1908–1911.

[13] J. Lu, Z. H. Liu, P. Xu, C. Zhang, UDBMS: road to unification for multi-model data management, in: ER Workshops, 2018, pp. 285–294.

[14] I. Holubova, S. Scherzinger, Nextgen multi-model databases in semantic big data architectures, Open Journal of Semantic Web (OJSW) 7 (2020) 1–16.

[15] W3C, Semantic Web Development Tools, accessed on 23/4/2020. https://www.w3.org/2001/sw/wiki/Tools.

[16] S. Groppe, Emergent models, frameworks, and hardware technologies for big data analytics, The Journal of Supercomputing 76 (2020) 1800–1827.

[17] L. Feigenbaum, G. T. Williams, K. G. Clark, E. Torres (editors), SPARQL 1.1 Protocol, 2013. W3C Recommendation, https://www.w3.org/TR/sparql11-protocol/.

[18] D. Weisenberger, How many atoms are there in the world?, accessed on 17.12.2020. http://education.jlab.org/qa/mathatom_05.html.

[19] E. A. Brewer, Pushing the CAP: strategies for consistency and availability, Computer 45 (2012) 23–29.

[20] D. Abadi, Consistency tradeoffs in modern distributed database system design: CAP is only part of the story, Computer 45 (2012) 37–42.

[21] R. Mutharaju, S. Sakr, A. Sala, P. Hitzler, D-sparq: Distributed, scalable and efficient rdf query engine, in: Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track), Sydney, Australia, 2013, p. 261–264.

[22] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, P. Castagna, Jena-hbase: A distributed, scalable and efficient rdf triple store, in: Proceedings of the 2012th International Conference on Posters & Demonstrations Track, Boston, USA, 2012, p. 85–88.

[23] N. Papailiou, I. Konstantinou, D. Tsoumakos, N. Koziris, H2RDF: Adaptive query processing on rdf data in the cloud, in: Proceedings of the 21st International Conference on World Wide Web,

Lyon, France, 2012, p. 397–400.

[24] N. Papailiou, I. Konstantinou, D. Tsoumakos, P. Karras, N. Koziris, H2RDF+: high-performance distributed joins over large-scale RDF graphs, in: Proceedings of the 2013 IEEE International Conference on Big Data, Santa Clara, USA, 2013, pp. 255–263.

[25] X. Zhang, M. Zhang, P. Peng, J. Song, Z. Feng, L. Zou, A scalable sparse matrix-based join for sparql query processing, in: International Conference on Database Systems for Advanced Applications, Springer, 2019, pp. 510–514.

[26] S. Werner, D. Heinrich, S. Groppe, C. Blochwitz, T. Pionteck, Runtime adaptive hybrid query engine based on fpgas, Open Journal of Databases (OJDB) 3 (2016) 21–41.

[27] I. Trummer, C. Koch, Multiple query optimization on the d-wave 2x adiabatic quantum computer, Proc. VLDB Endow. 9 (2016).

[28] S. Roy, L. Kot, C. Koch, Quantum databases, in: CIDR, 2013.

[29] T. Bittner, S. Groppe, Avoiding blocking by scheduling transactions using quantum annealing, in: 24th International Database Engineering & Applications Symposium (IDEAS), Seoul, Republic of Korea, 2020.

[30] T. Bittner, S. Groppe, Hardware accelerating the optimization of transaction schedules via quantum annealing by avoiding blocking, Open Journal of Cloud Computing (OJCC) 7 (2020) 1–21. URL: http://nbn-resolving.de/urn:nbn:de:101:1-202011221833201534395 7.

[31] S. Groppe, T. Kiencke, S. Werner, D. Heinrich, M. Stelzner, L. Gruenwald, P-luposdate: Using precomputed bloom filters to speed up sparql processing in the cloud, Open Journal of Semantic Web (OJSW) 1 (2014) 25–55.

[32] D. Graux, L. Jachiet, P. Geneves, N. Layaïda, Sparqlgx: Efficient distributed evaluation of sparql with apache spark, in: ISWC, 2016.

[33] A. Azzam, S. Kirrane, A. Polleres, Towards making distributed rdf processing flinker, in: Innovate-Data, IEEE, 2018, pp. 9–16.

[34] S. Groppe, J. Blume, D. Heinrich, S. Werner, A self-optimizing cloud computing system for distributed storage and processing of semantic web data, Open Journal of Cloud Computing (OJCC) 1 (2014) 1–14.

[35] S. Groppe, N. Reimer, Code generation for big data processing in the web using webassembly, Open Journal of Cloud Computing (OJCC) 6 (2019) 1–15.

[36] A. Rossberg (editor), WebAssembly Core Specification, W3C Proposed Recommendation, https://www.w3.org/TR/wasm-core-1/, 2019.

[37] A. Grall, P. Folz, G. Montoya, H. Skaf-Molli, P. Molli, M. Vander Sande, R. Verborgh, Ladda: Sparql queries in the fog of browsers, in: European Semantic Web Conference, Springer, 2017, pp. 126–131.

[38] V. Kumar, Mobile database systems, Wiley Online Library, 2006.

[39] D. Le-Phuoc, J. X. Parreira, V. Reynolds, M. Hauswirth, Rdf on the go: An rdf storage and query processor for mobile devices, in: ISWC, Citeseer, 2010.

[40] K. Graffi, D. Stingl, C. Gross, H. Nguyen, A. Kovacevic, R. Steinmetz, Towards a p2p cloud: Reliable resource reservations in unreliable p2p systems, in: International Conference on Parallel and Distributed Systems, 2010, pp. 27–34.

[41] R. Mietz, S. Groppe, O. Kleine, D. Bimschas, S. Fischer, K. Römer, D. Pfisterer, A p2p semantic query framework for the internet of things, PIK-Praxis der Informationsverarbeitung und Kommunikation 36 (2013) 73–79.

[42] ObjectBox Limited, The best IoT Databases for the Edge – an overview and compact guide, https://objectbox.io/the-best-iot-databases-for-the-edge-an-overview-and-compact-guide/, 2019.

[43] M. Abdelshkour, Iot, from cloud to fog computing, Cisco Blogs, http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing, 2015.

[44] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (2015) 37–42.

[45] K. Skala, D. Davidovic, E. Afgan, I. Sovic, Z. Sojat, Scalable distributed computing hierarchy: Cloud, fog and dew computing, Open Journal of Cloud Computing (OJCC) 2 (2015) 16–24.

[46] Y. Wang, Definition and categorization of dew computing, Open Journal of Cloud Computing (OJCC) 3 (2016) 1–7.

[47] S. Mishra, S. Jain, Ontologies as a semantic model in iot, International Journal of Computers and Applications 42 (2020) 233–243.

[48] A. Cimmino, M. Poveda-Villalón, R. García-Castro, ewot: A semantic interoperability approach for heterogeneous iot ecosystems based on the web of things, Sensors 20 (2020) 822.