

# Hybrid Multi-model *Multi-platform* (HM3P) Databases

Sven Groppe<sup>a</sup> and Jinghua Groppe

*Institute of Information Systems (IFIS), University of Lübeck, Ratzeburger Allee 160, D-23562 Lübeck, Germany*  
{groppe, groppej}@ifis.uni-luebeck.de

**Keywords:** Databases, Multi-platform, Multi-model, Cloud, Post-cloud, Edge/fog/dew Computing, Hardware Acceleration, Internet-of-Things (IoT), Mobile Database, Parallel Database, Main-memory Database.

**Abstract:** There exist various standards for different models of data, and hence users often must handle a zoo of data models. Storing and processing data in their native models, but spanning optimizations and processing across these models seem to be the most efficient way, such that we recently observe an advent of multi-model databases for this purpose. Companies, end users and developers typically run different platforms like mobile devices, web, desktops, servers, clouds and post-clouds (e.g., fog and edge computing) as execution environments for their applications at the same time. In this paper, we propose to utilize the different platforms according to their advantages and benefits for data distribution, query processing and transaction handling in an overall integrated hybrid multi-model multi-platform (HM3P) database. We analyze current state-of-the-art multi-model databases according to the support of multiple platforms. Furthermore, we analyze the properties of databases running on different types of platforms. We detail new challenges for the novel concept of HM3P databases concerning a global optimization of data distribution, query processing and transaction handling across multiple platforms.

## 1 INTRODUCTION

Today companies have to deal with and process data in various data formats like relational data (in relational databases), XML (for exchange), JSON (as web data), RDF (of the Semantic Web), graph data (from social networks) and unstructured data (of social media like wikis). The data is hence stored according to and processed using different models (*multi-model data* (Lu and Holubová, 2019)). 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 (Kotorov, 2003).

There are two main approaches known in literature to handle multi-model data in databases:

- **Polyglot Persistence** describes the approach of applications to use several databases at the same time to handle multi-model data (Leberknight, 2008). While very flexible in integrating data sources, the disadvantage of polyglot persistence is that data sources have to be integrated at application level without any further support of the databases. Furthermore, the performance of data processing cannot be fully optimized and fault-tolerance cannot be transparently offered across the different databases,

even if additional application logic is added.

- **Multi-Model Database Management Systems (MM-DBMSs)** offer the management of different data models in one single database (in order to overcome the disadvantages of polyglot persistence) (Lu and Holubová, 2019).

While in the past database management systems (DBMSs) run mainly on parallel servers, there are today various different platforms offering execution environments for running a DBMS<sup>1</sup>: Besides parallel servers mainly used for relational databases (of small to medium-sized companies), DBMSs run in clusters like NoSQL databases (e.g., Cassandra and MongoDB) and cloud databases (e.g., HBase). Modern databases also use hardware accelerators like GPUs, FPGAs and SmartSSDs to further increasing transaction rates. The availability of servers with huge main memory (up to several terabytes) lead to the advent of main-memory databases (e.g., SAP HANA). The Internet-of-Things (IoT) with huge amount of devices generating much data with high velocity requires data management tasks to be processed close to the sources (i.e., close to the IoT devices) reducing communication and energy costs as well as latency

---

<sup>1</sup>Note that clients of DBMSs typically run on different platforms, but we are considering the database server here.

<sup>a</sup> <https://orcid.org/0000-0001-5196-1117>

and increasing privacy. These issues are addressed by fog computing (for processing the data on close-by devices with higher resources like routers) (Abdelshkour, 2015), edge computing (for processing the data also on the IoT devices themselves) (Garcia Lopez et al., 2015) and dew computing (offering services close to the IoT devices independent of, but collaborative with cloud services for the purpose of high availability) (Wang, 2016). Mobile databases (Kumar, 2006) involve the base stations and other infrastructure of mobile providers for database tasks spanning over mobile devices. Other computing environments include processing in peer-to-peer networks (Graffi et al., 2010; Mietz et al., 2013) for fast and flexible deployment of new computing nodes. There are also first attempts to utilize quantum computers for database optimizations (Trummer and Koch, 2016; Roy et al., 2013).

Some programming languages like Kotlin (JetBrains s.r.o., 2016) support multi-platform development sharing common code between different platforms like desktop, server, web, mobile and IoT, such that the development costs for a DBMS running on multiple platforms are drastically reduced.

Puzzling all pieces together we propose the following definition:

**Definition** (Multi-Model Multi-Platform Database Management System (M3P DBMS) and Hybrid M3P (HM3P) DBMS). *A M3P DBMS is a MM-DBMS that can be executed on different platforms. An HM3P DBMS spans over different platforms in operation.*

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*<sup>2</sup>). In contrast, we envision HM3P 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). For an example installation, see Figure 1.

In contrast to polyglot persistence, advantages of (H)M3P DBMSs are e.g.

- developing only one code base for the different platforms (promising faster development cycles and less development costs per platform)
- providing one application programming interface (api) for multiple platforms, such that applications can access their database at any platform without

<sup>2</sup>Please note that private and public clouds are platforms of the same type.

changes in their code (increased interoperability and less development costs)

- support of any model at any platform
- data distribution, query optimization, transaction handling and other database tasks across different platforms
- re-use of approaches in very different platforms with similar properties (e.g., applying the same fault-tolerance methods for dealing with many disconnections in IoT databases as well as in mobile databases)
- integrating the features of different types of databases

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 HM3P 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**, where different databases are used within one application (Leberknight, 2008), offers a practical approach to use different data models and also different types of databases running on different platforms. The landscape of used databases for Big Data is large and spans over relational databases, cloud databases and NoSQL data stores like key-value, columnar, document and graph stores. 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 (Kolev et al., 2016), 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 (Zhu and Risch, 2011) offering to query cloud-based NoSQL like Google’s Bigtable and relational databases with the Google Bigtable query language GQL. Commercial multi-store products like IBM BigInsights, Microsoft HDInsight and Oracle Bigdata Appliance integrate diverse data sources by using database connectors (like JDBC drivers). However, they also don’t support to fully optimize queries

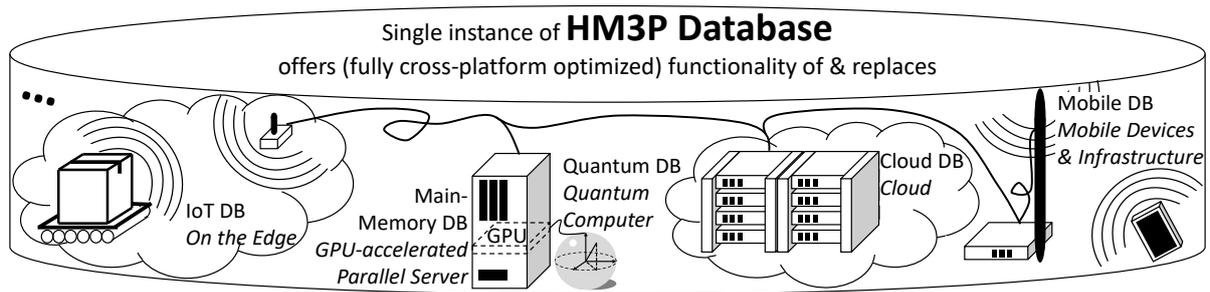


Figure 1: HM3P Database spanning over multiple platforms. Here, an HM3P database replaces an IoT database in an Industry 4.0 scenario (*using edge-computing*), a GPU-accelerated parallel database (*on a parallel server*) for archiving and generating long-term statistics of the IoT data, which is further supported by a *quantum computer* for query 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*.

across the integrated, but independent data sources, which limit data processing.

**Federation Databases** (Hammer and McLeod, 1979) and multidatabases (Smith et al., 1981) provide similar ideas to polyglot persistence but having a longer history. These types of databases have been studied during the 1980s and 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+ (Lim et al., 2013) provides unified declarative processing for the integration of several processing and database platforms. BigDAWG (Elmore et al., 2015) offers location transparency while running queries against the three different integrated systems PostgreSQL, SciDB and Accumulo.

Polyglot persistence as well as federation databases already integrate databases using different data models and running on different (types of) platforms. However, the full potential of query optimization and processing, and other database tasks has not been achieved for both approaches, as the single integrated databases are like black boxes, which are accessed via their supported query languages or their native application programming interfaces, such that cross-database optimizations are hindered.

**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 (Lu et al., 2018). 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).

Table 1 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. 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.

Table 1: Summary of key features of multi-model databases. This table is based on (Holubová and Scherzinger, 2020; Lu and Holubová, 2019) and extended by the feature “Platforms”.

Type	DBMS	Ext.	Models RCKJXGDO	Query languages	Platforms NJWLUMSZCH
Relational	PostgreSQL <sup>a</sup>	I	R-KJX--0	extended SQL	N-WLUMS-CH
	Microsoft SQL Server <sup>b</sup>	I	R--JXG-0	extended SQL	N-WL----CH
	IBM DB2 LUW <sup>c</sup>	I	R---XGDO	extended SQL/XML	N-WLU-S-C-
	IBM DB2 z/OS <sup>d</sup>	I	R---XGDO	extended SQL/XML	N-----Z--
	Oracle DB <sup>e</sup>	I	R--JX-DO	SQL/XML, SQL/JSON	N-WLUMS*CH
	MySQL <sup>f</sup>	II	R-K----0	SQL, memcached API	N-WLUMS-C-
	Sinew (Tahara et al., 2014)	III	R-K-----	SQL	N-WLUMS-CH
Column	Cassandra <sup>g</sup>	I	-C---G-0	SQL-like CQL	-JWLUMS-CH
	CrateDB <sup>h</sup>	I	RC-J-G--	SQL	-JWL-M--C-
	DynamoDB <sup>i</sup>	I	-CKJ-G-0	simple API (get/put/update) + simple queries over indices	-JWLUM--C-
	Vertica <sup>j</sup>	II	-C-J-G--	SQL-like	N--LU---CH
Key/value	Riak KV <sup>k</sup>	I	--KJXG--	Solr	N--LUM--CH
	c-treeACE <sup>l</sup>	III	R-K--G--	SQL	N-WLUMS-C-
	Oracle NoSQL DB <sup>m</sup>	III	R-K--GD-	SQL	-JWLUMS-C-
Document	Cosmos DB <sup>n</sup>	I	-CKJ----	SQL-like	N-----C-
	ArangoDB <sup>o</sup>	II	--KJ-G--	SQL-like AQL	N-WL-M--C-
	MongoDB <sup>p</sup>	II	--KJ---0	JSON-based query language	N-WL-M--C-
	Couchbase <sup>q</sup>	III	--KJ----	SQL-based N <sub>1</sub> QL	N-WL-M--CH
	MarkLogic <sup>r</sup>	III	---JX-DO	XPath, XQuery, SQL-like	N-WL-M--CH
Graph	OrientDB <sup>s</sup>	II	--KJ-G--	Gremlin, extended SQL, SPARQL	N-WLUM--CH
Object	InterSystems Caché <sup>t</sup>	III	R--JX--0	SQL with object extensions	N-WLUMS-CH

**Legend:** Ext.: I = adoption of a new storage strategy, II = extension of the original storage strategy, III = creation of a new interface, IV = no change;

**Models:** R = relational, C = column, K = key/value, J = JSON, X = XML, G = graph, D = RDF, O = object, - = no support;

**Platforms:** N = Native Machine Code, J = Java/JVM, W = Win, L = Linux, U = Unix (e.g. BSD), M = macOS, S = Solaris, Z = z/OS, C = Cloud, H = Hybrid Cloud, - = no support, \* = support for old versions.

#### References of databases:

<sup>a</sup><https://www.postgresql.org/>

<sup>b</sup><https://www.microsoft.com/de-de/sql-server/sql-server-2017-editions>

<sup>c</sup><https://www.ibm.com/analytics/db2>

<sup>d</sup><https://www.ibm.com/analytics/db2/zos>

<sup>e</sup><https://www.oracle.com/database/index.html>

<sup>f</sup><https://www.oracle.com/mysql/index.html>

<sup>g</sup><http://cassandra.apache.org/>

<sup>h</sup><https://crate.io/>

<sup>i</sup><https://aws.amazon.com/dynamodb/>

<sup>j</sup><https://www.vertica.com/>

<sup>k</sup><https://riak.com/products/riak-kv/>

<sup>l</sup><https://www.faircom.com/products/c-treeace>

<sup>m</sup><https://www.oracle.com/database/technologies/related/nosql.html>

<sup>n</sup><http://www.cosmosdb.com>

<sup>o</sup><https://www.arangodb.com/>

<sup>p</sup><https://www.mongodb.com/>

<sup>q</sup><http://www.couchbase.com/>

<sup>r</sup><https://www.marklogic.com/>

<sup>s</sup><https://orientdb.com/>

<sup>t</sup><https://www.intersystems.com/products/cache/>

## 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 running 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 plat-

forms (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 (JetBrains s.r.o., 2016). 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 Table 1). Some modern DBMSs are implemented in Java further decreasing development costs, but still running on clusters and servers operating Windows, Linux, Unix and MacOS.

### 2.3 Databases for Different Platforms

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 Table 1). 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).

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.

## 3 MULTI-PLATFORM MULTI-MODEL DATABASES

Different types of data are stored on and processed at different platforms dependent on their size, the devices they are generate 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. 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. This would also allow to offer the best features 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 HM3P 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), but to have e.g. a global optimization of data distribution, transaction handling and global queries with full potential by having freedom of processing down to the physical layer (e.g., index accesses)<sup>3</sup>. One single HM3P DBMS would also reduce development costs of applications and periods of vocational adjustment of developers by offering one API and query language for all different platforms.

### 3.1 Platforms

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

---

<sup>3</sup>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.

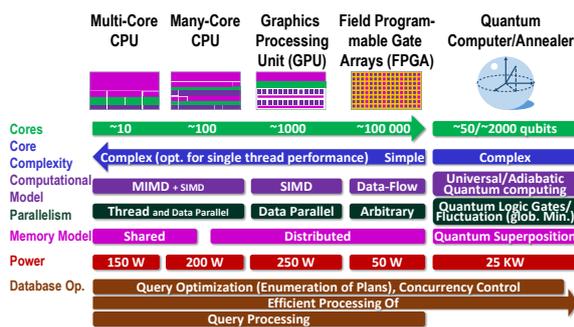


Figure 2: Hardware Architectures and Properties. Figure is based on (Plessl, 2012) and extended by the column “Quantum Computer/Annealer” and row “Database Op.”.

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 are typical DBMSs running on server platforms, but 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. Figure 2 contains an overview over different types of hardware accelerators and their properties<sup>4</sup>: The multi-core CPU offers shared memory to all cores, but each core manages also caches for faster accesses. Multi-core CPUs have a high single-core performance and run threads with different functionality (according to the multiple-instruction multiple-data (MIMD) paradigm). Many-core CPUs have a similar architecture compared to multi-core CPUs, but manage much more cores providing a higher throughput with an increase of latency and lower single thread performance.

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. Hence, neither all parallel algorithms are suitable for nor benefit from GPUs. Hence, the massive parallel processing of execution plans are ideal for many-core CPUs and GPUs

<sup>4</sup>The energy consumption is according to D-Wave’s quantum computing hardware, which is based on metal niobium loops acting as superconductors when cooled down to 15 millikelvin (-273°C). Most power is consumed by the refrigerator, which is slightly less than 25 kilowatts, see <https://spectrum.ieee.org/tech-talk/computing/hardware/how-much-power-will-quantum-computing-need>.

as well as whenever the best possibilities among enumerated ones must be found (like in query optimization and multi-version concurrency control (MVCC)).

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.

Universal quantum computers try to combine the full power of classical computers with quantum computers that manipulate (some few) qubits in superposition 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<sup>5</sup>, selecting the execution plan with the best estimated costs (from a set of enumerated plans) (Trummer and Koch, 2016) and concurrency control between transactions (Roy et al., 2013).

**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. 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**.

**Mobile Databases** (Kumar, 2006) 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.

**P2P Databases** (Graffi et al., 2010; Mietz et al.,

<sup>5</sup>investigated by Volkswagen, see <https://www.volkswagenag.com/en/news/stories/2018/11/intelligent-traffic-control-with-quantum-computers.html>

2013) 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.

**IoT Databases** (ObjectBox Limited, 2019) 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 bottleneck 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 (Abdelshkour, 2015) 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 (Garcia Lopez et al., 2015), 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 (Wang, 2016) 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.

IoT Databases are often organized as P2P database, especially if they work on the fog or edge, or follow the dew computing concept. 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.

### 3.2 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*.

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, 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.

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<sup>6</sup>
- 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)
- 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
- 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. For example, the smooth integration should guarantee atomicity and isolation in transactions for the data stored on a parallel server, but not for those data in the cloud supporting fast updates.

We are sure that this is not an exhaustive list of new challenges. Many further challenges will arise during developing the M3P and HM3P DBMSs and

---

<sup>6</sup>We are of the opinion that this is possible by applying Kotlin features like expected and actual declaration for classes and types, and inline functions and classes.

considering especially combinations of different platforms and models at runtime.

## 4 SUMMARY AND CONCLUSIONS

In this paper we analyze the landscape of multi-model databases running on multiple platforms. We call this type of database multi-model multi-platform database management system (M3P DBMS). Hybrid M3P (HM3P) DBMSs span over different platforms at run-time. Furthermore, we describe and analyze different types of DBMSs and platforms concerning their properties, chances and challenges for DBMSs. Current state-of-the-art 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 and transaction handling, and ease of usage when different types of platforms and databases are supported in one single installation of a M3P DBMS.

## ACKNOWLEDGMENTS

I want to thank Stefanie Scherzinger and Irena Holubová for their valuable comments about improving this contribution.

## REFERENCES

- Abdelshkour, M. (2015). Iot, from cloud to fog computing. Cisco Blogs, <http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing>.
- Elmore, A., Duggan, J., Stonebraker, M., Balazinska, M., Cetintemel, U., Gadepally, V., Heer, J., Howe, B., Kepner, J., Kraska, T., Madden, S., Maier, D., Mattson, T., Papadopoulos, S., Parkhurst, J., Tatbul, N., Vartak, M., and Zdonik, S. (2015). A demonstration of the bigdawg polystore system. *Proc. VLDB Endow.*, 8(12):1908–1911.
- Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., and Riviere, E. (2015). Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42.
- Graffi, K., Stingl, D., Gross, C., Nguyen, H., Kovacevic, A., and Steinmetz, R. (2010). Towards a p2p cloud: Reliable resource reservations in unreliable p2p systems. In *International Conference on Parallel and Distributed Systems*, pages 27–34.
- Hammer, M. and McLeod, D. (1979). On database management system architecture. Technical report, MIT, Cambridge Laboratory for Computer Science.
- Holubová, I. and Scherzinger, S. (2020). Multi-model databases: A new journey to handle the variety of data. *OJSW*, 7(1).
- JetBrains s.r.o. (2016). FAQ - Kotlin Programming Language. <https://kotlinlang.org/docs/reference/faq.html>.
- Kolev, B., Valduriez, P., Bondiombouy, C., Jiménez-Peris, R., Pau, R., and Pereira, J. (2016). Cloudmssql: querying heterogeneous cloud data stores with a common language. *Distributed and Parallel Databases*, 34(4):463–503.
- Kotorov, R. (2003). Customer relationship management: strategic lessons and future directions. *Business Process Management Journal*, 9(5):566–571.
- Kumar, V. (2006). *Mobile database systems*. Wiley.
- Leberknight, S. (2008). Polyglot persistence. Scott Leberknight's Weblog, [http://www.sleberknight.com/blog/sleberkn/entry/polyglot\\_persistence](http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence).
- Lim, H., Han, Y., and Babu, S. (2013). How to fit when no one size fits. In *CIDR*.
- Lu, J. and Holubová, I. (2019). Multi-model databases: A new journey to handle the variety of data. *ACM Computing Surveys (CSUR)*, 52(3).
- Lu, J., Liu, Z. H., Xu, P., and Zhang, C. (2018). UDBMS: road to unification for multi-model data management. In *ER Workshops*, pages 285–294.
- Mietz, R., Groppe, S., Kleine, O., Bimschas, D., Fischer, S., Römer, K., and Pfisterer, D. (2013). A p2p semantic query framework for the internet of things. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 36(2):73–79.
- ObjectBox Limited (2019). 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/>.
- Plessl, C. (2012). Accelerating Scientific Computing with Massively Parallel Computer Architectures. IM-PRS Winter School, Wrocław. <http://www.imprsdynamics.mpg.de/pdfs/Plessl.talk.pdf>.
- Roy, S., Kot, L., and Koch, C. (2013). Quantum databases. In *CIDR*.
- Smith, J. M., Bernstein, P. A., Dayal, U., Goodman, N., Landers, T., Lin, K. W. T., and Wong, E. (1981). Multibase: Integrating heterogeneous distributed database systems. In *AFIPS National Computer Conference*, pages 487–499.
- Tahara, D., Diamond, T., and Abadi, D. J. (2014). Sinew: A SQL System for Multi-structured Data. In *SIGMOD*.
- Trummer, I. and Koch, C. (2016). Multiple query optimization on the d-wave 2x adiabatic quantum computer. *Proc. VLDB Endow.*, 9(9).
- Wang, Y. (2016). Definition and categorization of dew computing. *OJCC*, 3(1):1–7.
- Zhu, M. and Risch, T. (2011). Querying combined cloud-based and relational databases. In *International Conference on Cloud and Service Computing*, pages 330–335.