Data Sci. Eng. (2017) 2:328–345
https://doi.org/10.1007/s41019-017-0043-3

# Big Data Management: What to Keep from the Past to Face Future Challenges?

G. Vargas-Solar[1] · J. L. Zechinelli-Martini[2] · J. A. Espinosa-Oviedo[3]

Received: 19 August 2016 / Revised: 24 April 2017 / Accepted: 1 August 2017 / Published online: 10 August 2017

**Abstract** The emergence of new hardware architectures, and the continuous production of data open new challenges for data management. It is no longer pertinent to reason with respect to a predefined set of resources (i.e., computing, storage and main memory). Instead, it is necessary to design data processing algorithms and processes considering unlimited resources via the "pay-as-you-go" model. According to this model, resources provision must consider the economic cost of the processes versus the use and parallel exploitation of available computing resources. In consequence, new methodologies, algorithms and tools for querying, deploying and programming data management functions have to be provided in scalable and elastic architectures that can cope with the characteristics of Big Data aware systems (intelligent systems, decision making, virtual environments, smart cities, drug personalization). These functions, must respect QoS properties (e.g., security, reliability, fault tolerance, dynamic evolution and adaptability) and behavior properties (e.g., transactional execution) according to application requirements. Mature and novel system architectures propose models and mechanisms for adding these properties to new efficient data management and processing functions delivered as services. This paper gives an overview of the different architectures in which efficient data management functions can be delivered for addressing Big Data processing challenges.

**Keywords** Big data · Service-based data management systems · NoSQL · Data analytics stack

This work has been partially granted by the CONCERNS project granted by Microsoft, the AMBED and Multicloud projects of the ARC6 and ARC7 programs and by the CONACyT of the Mexican government.

✉ J. A. Espinosa-Oviedo
javier.espinosa@bsc.es

G. Vargas-Solar
genoveva.vargas@imag.fr

J. L. Zechinelli-Martini
joseluis.zechinelli@udlap.mx

1 Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, LAFMIA, 38000 Grenoble, France

2 Fundación Universidad de las Américas, Puebla, Puebla, Mexico

3 Barcelona Supercomputing Center, LAFMIA, Barcelona, Spain

## 1 Introduction

Database management systems (DBMS) emerged as a flexible and cost-effective solution to information organization, maintenance and access problems found in organizations (e.g., business, academia and government). DBMS addressed these problems under the following conditions: (i) with models and long-term reliable data storage capabilities; (ii) providing retrieval and manipulation facilities for stored data for multiple concurrent users or transactions [40]. The concept of data model (most notably the relational models like the one proposed by Codd [29]; and the object-oriented data models [8]; Cattell and Barry [25]), the Structured Query Language (SQL, Melton and Simon [81]), and the concept of transaction (Gray and Reuter [57]) are crucial ingredients of successful data management in current enterprises.

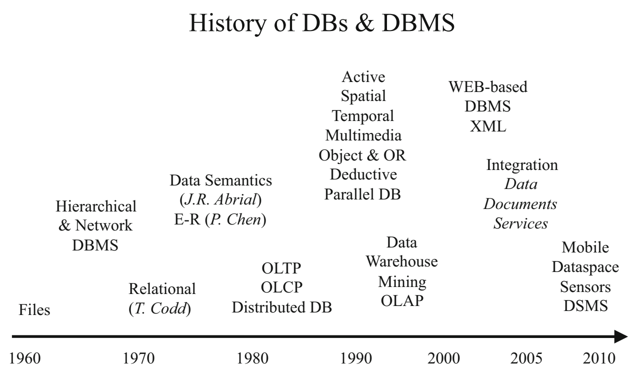
History of DBs & DBMS
Files
Hierarchical & Network DBMS
Relational (T. Codd)
Data Semantics (J.R. Abrial) E-R (P. Chen)
OLTP OLCP Distributed DB
Active Spatial Temporal Multimedia Object & OR Deductive Parallel DB
Data Warehouse Mining OLAP
WEB-based DBMS XML
Integration Data Documents Services
Mobile Dataspace Sensors DSMS
1960
1970
1980
1990
2000
2005
2010

**Fig. 1** Historical outline of DBMS [1]

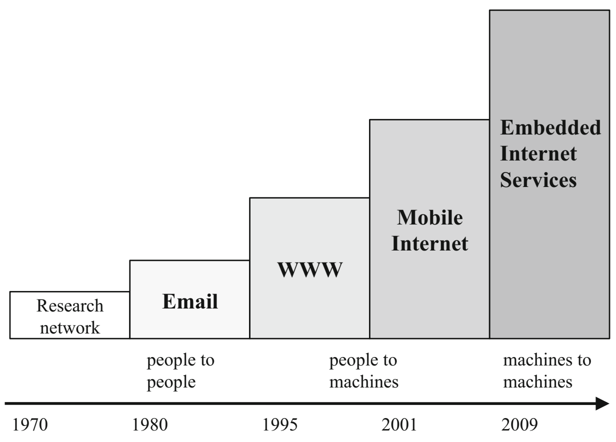

**Fig. 2** Shifts in the use of the Web [79]

Today, the data management market is dominated by major object-relational database management systems (OR-DBMS) like Oracle,[1] DB2,[2] or SQLServer.[3] These systems arose from decades of corporate and academic research pioneered by the creators of System R [6] and Ingres [97].

Since their emergence, innovations and extensions have been proposed to enhance DBMS in power, usability, and spectrum of applications (see Fig. 1).

The introduction of the relational model [29], prevalent today, addressed the shortcomings of earlier data models. Subsequent data models, in turn, were relegated or became complementary to the relational model. Further developments focused on transaction processing and on extending DBMS to support new types of data (e.g., spatial, multimedia, etc.), data analysis techniques and systems (e.g., data warehouses, OLAP systems, data mining). The evolution of data models and the consolidation of distributed systems made it possible to develop mediation infrastructures [109] that enable transparent access to multiple data sources through querying, navigation and management facilities. Examples of such systems are multi-databases, data warehouses, Web portals deployed on Internet/Intranets, polyglot persistence solutions [78]. Common issues tackled by such systems are (i) how to handle diversity of data representations and semantics? (ii) how to provide a global view of the structure of the information system while respecting access and management constraints? (iii) how to ensure data quality (i.e., freshness, consistency, completeness, correctness)?

Besides, the Web of data has led to Web-based DBMS and XML data management systems serving as pivot models for integrating data and documents (e.g., active XML). The emergence of the Web marked a turning point, since the attention turned to vast amounts of new data outside of the control of a single DBMS. This resulted in an increased use of data integration techniques and exchange data formats such as XML. However, despite its recent development, the Web itself has experienced significant evolution, resulting in a feedback loop between database and Web technologies, whereby both depart from their traditional dwellings into new application domains.

Figure 2 depicts the major shifts in the use of the Web. A first phase saw the Web as a mean to facilitate communication between people, in the spirit of traditional media and mainly through email. Afterward, the WWW made available vast amounts of information in the form of HTML documents, which can be regarded as people-to-machines communication. Advances in mobile communication technologies extended this notion to mobile devices and a much larger number of users, the emergence of IoT environments increases the number of data producers to thousands of devices. A more recent trend exemplified by Web Services, and later Semantic Web Services, as well as Cloud computing,[4] consists of machine-to-machine communication. Thus, the Web has also become a platform for applications and a plethora of devices to interoperate, share data and resources.

Most recent milestones in data management (cf. Fig. 1) have addressed data streams leading to data stream management systems (DSMS). Besides, mobile data providers and consumers have also led to data management systems dealing with mobile queries and mobile objects. Finally, the last 7 years concern challenges introduced by the XXL phenomenon including the volume of data to be managed (i.e., Big Data volume) that makes research turn back the eyes toward DBMS architectures (cloud, in-memory, GPU DBMSs), data collection construction[5] and parallel data

[1] http://www.oracle.com.

[2] http://www-01.ibm.com/software/data/db2/.

[3] http://www.microsoft.com/sqlserver/2008/en/us/.

[4] Cloud computing enables on-demand network access to computing resources managed by external parties.

[5] See http://www.datascienceinstitute.org.

processing. In this context, it seems that Big Data processing must profit from available computing resources by applying parallel execution models, thereby achieving results in "acceptable" times.

Relational queries are ideally suited to parallel execution because they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series giving pipelined parallelism [39]. By partitioning the input data among multiple processors and memories, an operator can often be split into many independent operators, each working on a part of the data. This partitioned data and execution leads to partitioned parallelism that can exploit available computing resources.

In this context, we can identify three major aspects that involve database and Web technologies, and that are crucial for satisfying the new information access requirements of users: (i) a large number of heterogeneous data sources accessible via standardized interfaces, which we refer to as *data services* (e.g., Facebook, Twitter); (ii) computational resources supported by various platforms that are also publicly available through standardized interfaces, which we call *computation services* (e.g., hash Amazon E3C service); (iii) mobile devices that can both generate data and be used to process and display data on behalf of the user.

The new DBMS aims at fulfilling ambient applications requirements, data curation and warehousing, scientific applications, on-line games, among others [67]. Therefore, future DBMS must address the following data management issues:

- Data persistence for managing distributed storage spaces delivered by different providers (e.g., dropbox, onedrive, and googledrive); efficiently and continuously ensuring data availability using data sharing and duplication; ensuring data usability and their migration into new hardware storage supports.
- Efficient and continuous querying, and mining of data flows. These are complex processes requiring huge amounts of computing resources. They must be designed,[6] implemented and deployed on well adapted architectures such as the grid, the cloud but also sensor networks, mobile devices with different physic capacities (i.e., computing and storage capacity).
- Querying services that can implement evaluation strategies able to:
  - (i) process continuous and one-shot queries that include spatiotemporal elements and nomad sources;
  - (ii) deal with exhaustive, partial and approximate answers;
  - (iii) use execution models that consider accessing services as data providers and that include as a source the wisdom of the crowd;
  - (iv) integrate "cross-layer" optimization that includes the network and the enabling infrastructure as part of the evaluation process, and that can use dynamic cost models based on execution, economic, and energy costs.

The DBMS of the future must also enable the execution of algorithms and of complex processes (scientific experiments) that use huge data collections (e.g., multimedia documents, complex graphs with thousands of nodes). This calls for a thorough revision of the hypotheses underlying the algorithms, protocols and architectures developed for classic data management approaches [31]. In the following sections we discuss some of these issues focusing mainly on the way data management services of different granularities are delivered by different DBMS architectures. Section 2 analyses DBMS architectures evolution, from monolithic to customizable systems. Section 3 discusses how scalability and extensibility properties can have associated implications on systems performance. Section 4 presents important issues and challenges concerning management systems through the description of Big Data stacks, Big Data management systems, environments and data cloud services. Section 5 discusses open issues on DBMS architectures and how they deliver their functions for fulfilling application requirements. Section 6 describes some perspectives.

## 2 From Monolithic to Customizable DBMS Architectures

Different kinds of architectures serve different purposes. The ANSI/SPARC [1] architecture (Fig. 3) that characterizes classic database management systems (relational, object oriented, XML) deployed on client-server architectures has evolved in parallel to the advances resulting from new application requirements, data volumes, and data models. The 3-level-schema architecture reflects the different levels of abstraction of data in a database system distinguishing: (i) the external schemata that users work with, (ii) the internal integrated schema of the entire database; (iii) the physical schema determining the storage and the organization of databases on secondary storage.

The structure of a monolithic DBMS shown in Fig. 4 shows three key components of the system: the storage manager, the transaction manager, and the schema manager.

[6] See MapReduce models for implementing relational operators [2].

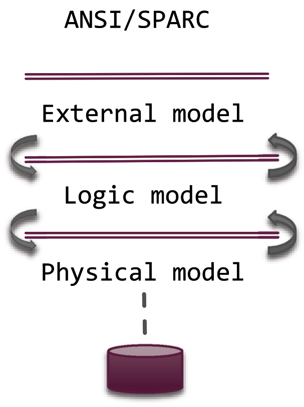

**Fig. 3** ANSI-SPARC DBMS architecture

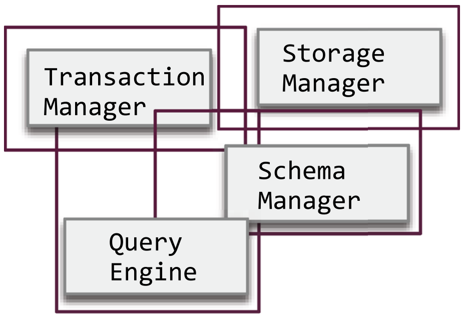

**Fig. 4** Classic DBMS functions [40]

The evolution of devices with different physical capacities (i.e., storage, computing, memory), and systems requiring data management functions started to show that adding more and more functions to monolithic DBMS does not work. Instead, it seems attractive to consider the alternative of extending DBMS allowing functionality to be added or replaced in a modular manner, as needed.

While such a closely woven implementation provides good performance/efficiency, customization is an expensive and difficult task because of the dependencies among the different components. For example, changing the indexing or clustering technique employed by the storage manager, changing the instance adaptation approach employed by the schema manager or the transaction model can have a large ripple effect on the whole system [40].

During the last twenty years, in order to better match the evolution of user and application needs, many extensions have been proposed to enhance the DBMS functions. In order to meet all new requirements, DBMS were extended to include new functionalities. Extensible and personalizable database systems [22] were an attempt to ease the construction of DBMS by exploiting software reusability [51], and proposing a general core that can be customized or extended, or even used to generate some DBMS parts. Trade-offs between modularity and efficiency, granularity of services, and the number of inter-service relationships result in DBMS designs which lack customizability.[7] A study of the standard task-oriented architecture of DBMS can be useful to determine their viability in new environments and for new applications. The following paragraphs give an overview of the main elements for showing how DBMS have and should evolve in order to address the scalability and performance requirements for data management.

## 2.1 Classic Functional Architecture

The classic DBMS architecture consists of a number of layers [36, 63, 64] each supporting a set of data types and operations at its interface. It consists of several components (modules or managers of concrete or abstract resources). The data types and operations defined for the modules of one layer are implemented using the concepts (data types and operations) of the next-lower level. Therefore, the layered architecture can also be considered as a stack of abstract machines. The layered architecture model as introduced by Härder and Reuter (1983) is composed of five layers described in [40]:

1. The uppermost layer supports logical data structures such as relations, tuples, and views. Typical tasks of this layer include query processing and optimization, access control, and integrity enforcement.
2. The next layer implements a record-oriented interface. Typical entities are records and sets[8] as well as logical access paths. Typical components are the data dictionary, transaction management, and cursor management.
3. The middle layer manages storage structures (internal records), physical access paths, locking, logging, and recovery. Therefore, relevant modules include the record manager, physical access path managers (e.g., a hash table manager), and modules for lock management, logging, and recovery.
4. The next layer implements (page) buffer management and implements the page replacement strategy. Typical entities are pages and segments.
5. The lowest layer implements the management of secondary storage (i.e., maps segments, and pages to blocks and files).

Due to performance considerations, no concrete DBMS has fully obeyed the layered architecture [40]. Note that

[7] "Although a task-oriented architecture is much more suitable for reasoning about extensibility and DBMS construction, reference architectures rarely exist (with the straw-man architecture developed by the Computer Corporation of America, CCA 1982, as a notable exception)" [40].

[8] As found in the Committee on Data Systems Languages, CODASYL data model.

different layered architectures and different numbers of layers are proposed, depending on the desired interfaces at the top layer. If, for instance, only a set-oriented interface is needed, it is useful to merge the upper two layers. In practice, most DBMS architectures have been influenced by System R [6], which consists of two layers:

- The relational data system (RDS), providing the relational data interface (RDI). It implements SQL (including query optimization, access control, triggers, etc.);
- The relational storage system (RSS), supporting the relational storage interface (RSI). It provides access to single tuples of base relations at its interface.

Layered architectures [64] were designed to address customizability, but they provide partial solutions at a coarse granularity. In the layered architecture proposed by [64], for example, the concurrency control components are spread across two different layers. Customization of the lock management or recovery mechanisms (residing in the lower layer) have a knock-on effect on the transaction management component (residing in the higher layer) [40].

As we will discuss in the next sections, layered architectures are used by existing DBMS, and they remain used despite the different generations of these systems. In each generation, layers and modules were implemented according to different paradigms (e.g., object, component, and service oriented) changing their granularity and the transparency degree adopted for encapsulating the functions implemented by each layer.

## 2.2 OODBMS: Relaxing Data Management and Program Independence

The first evolution of DBMS is when the object-oriented (OO) paradigm emerged, and the logic and physical levels started to approach for providing efficient ways of dealing with persistent objects. Together with the OO paradigm emergence, it was possible to develop applications requiring databases that could handle very complex data, that could evolve gracefully, and that could provide the high-performance dictated by interactive systems. Database applications could be programmed with an OO language, and then object persistence was managed by an Object-Oriented DBMS (OODBMS). The OODBMS manifesto [8] stated that persistence should be orthogonal, i.e., each object, independent of its type, is allowed to become persistent as such (i.e., without explicit translation). Persistence should also be implicit: the user should not have to explicitly move or copy data to make it persistent. This implied also that transparency was enforced regarding secondary storage management (index management, data clustering, data buffering, access path selection and query optimization).

Extensible database systems [22] allowed new parts such as abstract data types or index structures to be added to the system. Enhancing DBMS with new Abstract Data Type (ADT) or index structures was pioneered in the Ingres/Postgres systems [77, 99, 100]. Ingres supports the definition of new ADTs, including operators. References to other tuples can be expressed through queries (i.e., the data type postquel), but otherwise ADTs, and their associated relations, still had to be in first normal form. This restriction was relaxed in systems that have a more powerful type system (e.g., an OO data model) [10, 35, 41, 74, 93]. Another area in which extensions have been extensively considered are index structures. In Ingres/Postgres, existing indexes (such as B-trees) can be extended to also support new types (or support existing types in a better way). To extend an index mechanism, new implementations of type-specific operators of indexes have to be provided by the user. In this way, existing index structures were tailored to fit new purposes, and thus have been called extended secondary indexes (see DB2 UDB object-relational DBMS).[9]

This evolution responded to the need of providing flexibility to the logic level adapting the physical level in consequence. The idea was to approach the three levels by offering ad hoc query facilities, and let applications define the way they could navigate the objects collections, for instance, a graphical browser could be sufficient to fulfill this functionality [8]. This facility could be supported by the data manipulation language or a subset of it.

As the architecture of the DBMS evolved according to the emergence of new programming paradigms like components and services, and to "new" data models like documents (XML), the frontiers among the three levels started to be thiner, and transparency concerning persistence and transaction management was less important. Component-oriented middleware started to provide persistence services and transaction monitors as services that required programmers to configure and integrate these properties within the applications. Data and program independence was broken but the ad hoc configuration of data management components or services seemed to be easier to configure since it was more efficient to personalize functions according to application needs.

## 2.3 Component-Oriented DBMS: Personalizing Data Management

Component aware [43, 47, 72, 84, 85, 88, 102] was a paradigm to address reusability, separation of concerns (i.e., separation of functional from non-functional

[9] http://www-01.ibm.com/software/data/db2/.

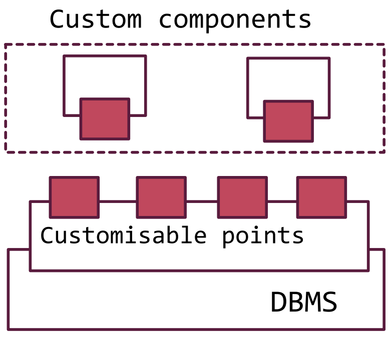

**Fig. 5** Component DBMS [40]

concerns) and ease of construction.[10] Component-based systems are built by putting components together to form new software systems. Systems constructed by composition can be modified or extended by replacing or adding new components (Fig. 5).

Approaches to extend and customize DBMS adopted the component-oriented paradigm for designing at least the customizable modules of the architecture, as components. Plug-in components are added to functionally complete DBMS and fulfill specialized needs. The components of component database management systems (CDBMS) are families of base and abstract data types or implementations of some DBMS function, such as new index structures. To date, all systems in this category are based on the relational data model and existing relational DBMS, and all of them offer some OO extensions. Example systems include IBM, DB2 UDB (IBM 1995), Informix Universal Server (Informix 1998), Oracle8 (Oracle 1999), and Predator [94]. Descriptions of sample component developments can be found in [18, 38].

Furthermore, the customization approach employed by most commercial DBMS are still largely monolithic (to improve performance). Special points similar to hot spots in OO frameworks [44] allow to custom components to be incorporated into the DBMS. Examples of such components include Informix DataBlades,[11] Oracle Data Cartridges [11] and DB2 Relational Extenders [37]. However, customization in these systems is limited to the introduction of user-defined types, functions, triggers, constraints, indexing mechanisms, and predicates, etc.

### 2.3.1 *Database Middleware*

Another way of addressing DBMS "componentization" was to provide database middlewares. Such middlewares leave data items under the control of their original (external) management systems while integrating them into a common DBMS-style framework. External systems exhibit, in many cases, different capabilities, such as query languages with varying power or no querying facilities at all. The different data stores might also have different data models (i.e., different data definition and structuring means), or no explicit data model at all. The goal of graceful integration is achieved through componentization.

The architecture introduces a common (intermediate) format into which the local data formats can be translated. Specific components perform this kind of translation. Besides, common interfaces and protocols define how the database middleware system and the components should interact (e.g., in order to retrieve data from a data store). These components (called wrappers) are also able to transform requests issued via these interfaces (e.g., queries) into requests understandable by the external system. In other words, these components implement the functionality needed to access data managed by the external data store. Examples of this approach include Disco [104], Garlic [92], OLE DB [15–17], Tsimmis [49], Harmony [91] (which implemented the CORBA query service), and Sybase Adaptive Server Enterprise [86]. Sybase allows access to external data stores, in Sybase called specialty data stores, and other types of database systems. ADEMS [21, 32] proposes mediation cooperative components or services that can broker and integrate data coming from heterogeneous sources. The cooperative brokers allow to build an extensible data mediation system.

### 2.3.2 *Configuring and Unbundling Data Management*

Configurable DBMS rely on unbundled DBMS tasks that can be mixed and matched to obtain database support (see Fig. 6). The difference lies in the possibility of adapting functions (called services) implementations to new requirements or in defining new services whenever needed. Configurable DBMS also consider services as unbundled representations of DBMS tasks. However, the models underlying the various services, and defining the semantics of the corresponding DBMS parts can now, in addition, be customized. Components for the same DBMS task can vary not only in their implementations for the same standardized interface, but also in their interfaces for the same task. DBMS implementors select (or construct new) components implementing the desired functionality, and obtain a DBMS by assembling the selected components. There are different approaches for configuring and composing

[10] A (software) component is a software artifact modeling and implementing a coherent and well-defined set of functions. It consists of a component interface and a component implementation. Components are black boxes, which means that clients can use them properly without knowing their implementation. Component interface and implementation should be separated such that multiple implementations can exist for one interface and implementations can be exchanged.

[11] http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.dbdk.doc/dbdk26.htm.

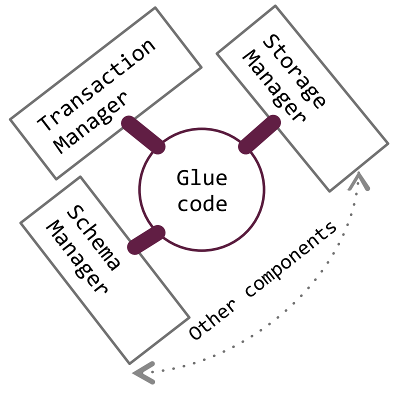

**Fig. 6** Extensible DBMS [40]

unbundled DBMS services: kernel systems, customizable systems, transformational systems, toolkits, generators and frameworks [40].

In principle, (internal) DBMS components are programmed and exchanged to achieve specific functionality in a different way than in the original system. A crucial element is the underlying architecture of the kernel, and the proper definition of points where exchanges can be performed. An example of this kind of DBMS is Starburst [60, 73]: its query language can be extended by new operators on relations [61] and various phases of the query processing are also customizable (e.g., functions are implemented using the interfaces of a lower layer (kernel) sometimes using a dedicated language). GENESIS [12, 13] is a transformational approach that supports the implementation of data models as a sequence of layers. The interface of each layer defines its notions of files, records, and links between files. Transformations themselves are collected in libraries, so that they can be reused for future layer implementations. Another transformational approach that uses specification constructs similar to those of Acta [28] has been described by [50]. EXODUS [23] applies the idea of a toolkit for specific parts of the DBMS. A library is provided for access methods. While the library initially contains type-independent access methods such as B-trees, grid files, and linear hashing, it can also be extended with new methods. Other examples are the Open OODB (Open Object-Oriented Database) approach [14, 108], Trent [70] for the construction of transaction managers (mainly, transaction structures and concurrency control) and *“ à la carte”* [42] for the construction of heterogeneous DBMS.

One problem in any toolkit approach is the consistency (or compatibility) of reused components. Generation approaches instead support the specification of (parts of) a DBMS functionality and the generation of DBMS components based on those specifications. A programmer defines a model (e.g., an optimizer, a data model, or a transaction model), which is given as input to a generator. The generator then automatically creates a software component that implements the specified model based on some implementation base (e.g., a storage manager or kernel in the case of data model software generation). An example of a generator system is the EXODUS query-optimizer generator [55]. Volcano [56], the successor of the EXODUS optimizer generator, also falls into the group of generator systems. Volcano has been used to build the optimizer for Open OODB [14].

Systems like KIDS [52], Navajo, and Objectivity [59] provide a modular, component-based implementation. For example, the transaction manager (or any other component) can be exchanged with the ripple effect mainly limited to the glue code. However, to strike the right balance between modularity and efficiency the design of the individual components is not highly modular. In fact, the modularity of the individual components is compromised to preserve both modularity and efficiency of the DBMS. Approaches like NODS [30] proposed service-oriented networked systems at various granularities that cooperated at the middleware level. The NODS services could be customized on a per-application basis at a fine grained level. For example, persistence could be configured at different levels [48] memory, cache or disk and it could cooperate with fault tolerance protocols for providing, for example, different levels of atomic persistent data management. Other frameworks for building query optimizers are the ones described in [89], Cascades [54], and EROC (Extensible Reusable Optimization Components) [80] and [34, 107]. Framboise [46] and ODAS [33, 105] are frameworks for layering active database functionality on top of passive DBMS.

However, customization at a finer granularity (i.e., the components forming the DBMS) is expensive. Such customization is cost-effective if changes were localized without compromising the system performance. Such performance can be ensured through closely woven components, i.e., both modularity and efficiency need to be preserved.

### 2.3.3 Summarizing Componentization of DBMS

CDBMS were successful because of the adoption of the notion of cartridge or blade by commercial DBMS. Other academic solutions were applied in some concrete validations. It is true that they enabled the configuration of the DBMS, but they still provided monolithic, complex, resources consuming systems (“kernels”) that need to be tuned and carefully managed for fulfilling the management of huge data volumes. These systems continued to encourage classic conception of information systems, with clear and complete knowledge of the data they manage, with global constraints, and homogeneous management with well identified needs. Yet, the evolution of technology, and the production of data stemming from different devices and services, the access to non-curated continuous data collections, the democratized access to continuous information (for example in social networks) calls for light weight data management services delivered in ad hoc personalized manners and not only in full-fledged one fits all systems like the (C)DBMS. Together with this evolution, emerged the notion of service aiming to ease the construction of loosely coupled systems. DBMS then started to move toward this new paradigm and were redefined as data management service providers.

## 2.4 Service-Oriented DBMS

Today the DBMS architecture has evolved to the notion of service-based infrastructure where services[12] are adapted and coordinated for implementing ad hoc data management functions (storage, fragmentation, replication, analysis, decision making, data mining). These functions are adapted and tuned for managing huge distributed multiform multimedia data collections. Applications can extend the functionality of DBMS through specific tasks that have to be provided by the data management systems, these tasks are called services, and allow interoperability between DBMS and other applications [52].

Subasu et al. [101] proposes a database architecture on the principles of service-oriented architecture (SOA) as a system capable of handling different data types, being able to provide methods for adding new database features (see Fig. 7). The service-based data management system (SBDMS) architecture borrows the architectural levels from Härder [62], and includes new features and advantages introduced by SOA into the field of database architecture. It is organized into functional layers that each with specialized services for specific tasks.

*Storage services* work on the byte level, in very close collaboration with file management functions of the operating system. These services have to handle the physical specifications of each non-volatile device. In addition, they provide services for updating existing data and finding stored data, propagating information from the Access Services Layer to the physical level. Since different data types require different storage optimizations, special services are created to supply their particular functional needs. This layer is equivalent to the first and second layer of the

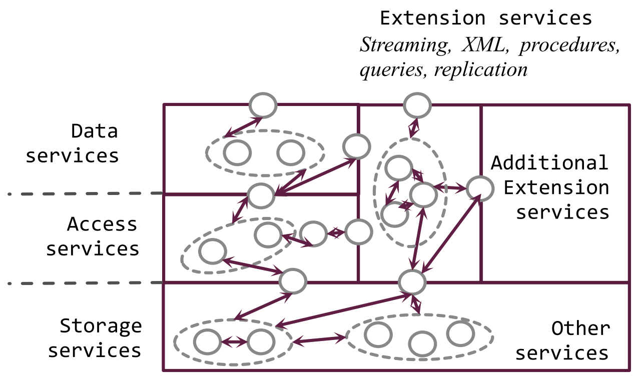
Extension services
Streaming, XML, procedures, queries, replication
Data services
Access services
Storage services
Additional Extension services
Other services

**Fig. 7** Service-oriented DBMS [101]

five layer architecture presented by Härder and Reuter [62, 65].

*Access services* is in charge of the physical data representations of data records and provides access path structures like B-trees. It provides more complex access paths, mappings, particular extensions for special data models, that are represented in the Storage Services Layer. Moreover, it is responsible for sorting record sets, navigating through logical record structures, making joins, and similar higher-level operations. This layer represents a key factor to database performance. The Access Services Layer has functions that are comparable to those in the third and fourth layer as presented by Härder and Reuter [62, 65].

*Data services* provide data represented in logical structures like tables or views. These are data structures without any procedural interface to the underlying database. The Data Service Layer can be mapped to the Non-Procedural and Algebraic Access level in the architecture by Härder and Reuter [62, 65].

*Extension services* users can design tailored extensions for example, creating new services or reusing existing ones from any available service from the other layers. These extensions help to manage different data types like XML files or streaming data. In this layer, users can integrate application-specific services in order to provide specific data types or specific functionalities needed by their applications (e.g., for optimization purposes).

A service-based DBMS externalizes the functions of the different systems layers, and enables the programming of personalized data management as a service systems. They make it possible to couple the data model characteristics with well adapted management functions that can themselves be programmed in an ad hoc manner. The DBMS remains a general purpose system that can be personalized, thanks to service composition, to provide ad hoc data management. It is then possible to have services deployed in architectures that make them available to applications in a simple way (e.g., cluster, cloud).

[12] Services in the SOA approach are software elements accessible through a well defined interface without requiring any knowledge on their implementation. SOAs can be implemented through a wide range of technologies like RPC, RMI, CORBA, COM, and Web services, not making any restrictions on the implementation protocols. In general, services can communicate using an arbitrary protocol, for example, they can use a file system to send data between their interfaces.

As discussed before the evolution of the DBMS architecture responds to the evolution of applications requirements in regard to efficient management. With the emergence of the notion of service, the DBMS architecture has been "fragmented" into components and services that are deployed in distributed platforms such as the Web 2.0. Applications use different kinds of data that must be managed according to different purposes: some data collections are read oriented with few writes; other data is modified continuously, and it is exploited by non-concurrent read operations. Some collections are shared, and they can support low consistency levels as long as they are available. Furthermore, such data is multiform, and more and more multimedia, they are modeled or at least exchanged as documents, particularly if they stem from the Web.

Requirements concerning data management performance vs. volume, and the effort of constructing data collections themselves has determined the evolution of DBMS toward efficiency. The three level architecture that encouraged program-data independence based on series of transformations among layers seems inappropriate to fulfill performance requirements. The architectures are making levels separations thin. The principle being that the less transformations among data are required the more efficient are data management functions, particularly querying, accessing, and processing. It seems that the very principle of independence between programs and data management is a very expensive quality that is not worth paying in certain situations.

## 3 Intensive Big Data Management: Bringing Scalability to Data Management

According to [19] a consistent challenge is that real Big Data clusters involve multi-user, multi-class (i.e., mixed job size) workloads not just one analysis job at a time or a few analysis jobs at a time, but a mix of jobs and queries of widely varying importance and sizes. The next generation of Big Data management services will have to propose approaches for dealing with such workloads effectively: *a long-standing open problem in the parallel database world and in the Hadoop world* as stated in [19]. We believe that three aspects are key for dealing with intensive Big Data management: (i) dealing with efficient simple storage systems that focus efficient read and write operations; (ii) multiple storage spaces given the volume and variety properties of data collections; (iii) data analytics workflows supported by efficient underlying data management infrastructures; (iv) parallel programming models to deal with the execution of greedy data analytic tasks that might require important computing and memory resources; (v) cloud providers that can provide storage, computing and memory resources in an elastic and "transparent" way, and that can enable the execution of greedy processes running on top of data collections hosted on their storage services.

### 3.1 NoSQL Data Store Managers

New kinds of data with specific structures (e.g., documents, graphs) produced by sensors, Global Positioning Systems (GPS), automated trackers and monitoring systems has to be manipulated, analyzed, and archived [103]. These large volumes of data sets impose new challenges and opportunities around storage, analysis, and archival. NoSQL stores seem to be appropriate systems that claim to be simpler, faster, and more reliable. This means that traditional data management techniques around upfront schema definition and relational references are being questioned.

Even if there is no standard definition of what NoSQL means,[13] there are common characteristics of these systems: (i) they do not rely on the relational model and do not use the SQL language; (ii) they tend to run on cluster architectures; (iii) they do not have a fixed schema, allowing to store data in any record. The systems that fall under the NoSQL umbrella are quite varied, each with their unique sets of features and value propositions. Examples include MongoDB, CouchDB, Cassandra, Hbase, and also BigTable and SimpleDB, which fit in general operating characteristics.

Despite the profound differences among the different NoSQL systems, the common characteristic with respect to the architecture is that the external and logic levels of RDBMS disappear. This means that the applications are close to the physical level with very few independence program and data. Data processing functions like querying, aggregating, analyzing are conceived for ensuring efficiency. For example, Google's Bigtable adopts a column-oriented data model avoiding consuming space when storing nulls by simply not storing a column when a value does not exist for that column. Columns are capable of storing any data types as far as the data can be persisted in the form of an array of bytes. The sorted ordered structure makes data seek by row-key extremely efficient. Data access is less random and ad hoc, and lookup is as simple as finding the node in the sequence that holds the data. Data are inserted at the end of the list.

Another evidence of the proximity to the physical model exploited by NoSQL systems are key-value stores that exploit hash-map (associative array) for holding key-value pairs. The structure is popular because thereby stores

[13] The notion was introduced in a workshop in 2009 according to [78].

provide a very efficient O(1) average algorithm running time for accessing data. The key of a key-value pair is a unique value in the set and can be easily looked up to access the data. Key-value pairs are of varied types: some keep the data in memory and some provide the capability to persist the data to disk. The underlying data storage architecture is in general a cluster, and the execution model of data processing functions is Map-Reduce. Thus data are, in general, managed in cache using for example the memcached protocol[14] particularly popular in key-value stores. A cache provides an in-memory snapshot of the most-used data in an application. The purpose of cache is to reduce disk I/O.

In some situations, availability cannot be compromised, and the system is so distributed that partition tolerance is required. In such cases, it may be possible to compromise strong consistency. The counterpart of strong consistency is weak consistency. Inconsistent data are probably not a choice for any serious system that allows any form of data updates but eventual consistency could be an option. Eventual consistency alludes to the fact that after an update all nodes in the cluster see the same state eventually. If the eventuality can be defined within certain limits, then the eventual consistency model could work. The term BASE (Basically Available Soft-state Eventually) [106] denotes the case of eventual consistency.[15]

NoSQL systems promote performance, scalable, clustered oriented data management and schema-less data design focusing on data distribution, duplication and on demand persistence. The logic and external levels of the classic DBMS architecture are erased exposing the physical level to applications with certain transparency. The application describes its data structures that can be persistent, and that can be retrieved using indexing mechanisms well adapted to these structures. Assuming good amounts of available memory resources, they promote parallel data querying including data processing tasks. Data availability is ensured through replication techniques and persistence on second memory is done on demand. For the time being, given that data management is done at the physical level, and that there is few data - program independence, there is a lot of programming burden to be undertaken by application programmers. Most NoSQL systems do not support high-level query languages with built-in query optimization. Instead, they expect the application programmer to worry about optimizing the execution of their data access calls with joins or similar operations having to be implemented in the application [82]. Another drawback mainly associated to the historical moment is that application programming interfaces (APIs) are not yet standardized, thus standardized bindings are missing, and they have to be programmed and maintained.

NoSQL systems overcome some of the shortcomings of the relational systems but leave aside good principles of the RDBMS, which go beyond the relational model and the SQL language. The schema-less approach seems to respond to a schema evolution requirement stemming from applications dealing with data in a very simple way (read/write operations). As discussed in [82], Web data like logs of activities in an e-commerce site or data managed by social media applications like Facebook are examples of cases needing schema evolutions because data are not very structured and, even when it is structured, the structure changes a lot over time.

### 3.2 Dealing with Multiple Storage Spaces

The use of heterogeneous data stores within a single system is gradually becoming a common practice in application development. Modern applications tend to rely on a polyglot approach to persistence, where traditional databases, non-relational data stores, and scalable systems associated with the emerging NewSQL movement, are used simultaneously.

As part of the emerging polyglot persistence movement [78], the simultaneous use of multiple scalable SQL, NoSQL, and NewSQL data stores within a single system is gradually becoming a common practice in modern application development [26, 66, 83]. Nonetheless, the combination of these heterogeneous databases, flexible schemata, and non-standard APIs represents an added complexity for application developers. For example, considering that the schemata used by these applications are spread across multiple data stores, each of which possibly relies on distinct data models (such as key-value, document, graph, etc.), developers must be familiar with a high number of implementation details, in order to effectively work with, and maintain the overall database model.

Due to the nature of schema-less data stores, developers also need to provide an adequate maintenance of the implicit schemata that these applications rely upon. This is due to the fact that the source code generally contains assumptions about the data structures used by the application (such as field names, types, etc.), even if the data

[14] According to Wikipedia, memcached is a general purpose distributed memory caching approach that was originally developed by Danga Interactive http://www.memcached.org. It is often used to speed up dynamic database-driven Websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.

[15] Eventual consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value. Eventual consistency is purely a liveness guarantee (reads eventually return the same value) and does not make safety guarantees: an eventually consistent system can return any value before it converges (Wikipedia).

stores themselves do not enforce any particular schema [78]. Essentially, we consider that the schemata are shifted from the database to the application source code. This is because many Web applications need to deal with persistent objects that maintain data about their execution state. However, having the data schemata as part of the application code can lead to maintenance and performance issues. For instance, developers have to manually analyze the full source code in order to effectively understand the data structures used by these applications. This can be an error-prone activity, due to the combination of different programming styles, APIs, and development environments.

Together with other approaches in the domain and in industry, we propose an approach and tool named ExSchema[16] that enables the automatic discovery of schemata from polyglot persistence applications. The discovery mechanism is based on source code analysis techniques, particularly on API usage and on the analysis of standard data layer patterns. The tool receives application source code as input, containing invocations to the APIs of one or more data stores (graph, key-value, relational, and column). The ExSchema analyzers examine this application source code and share their analysis results between each other. For example, in order to identify schema update methods, we first identify the declaration of variables. The schema fragments recovered by the code analyzers are grouped together, according to the data models supported by our tool, and by extending the translation mechanisms detailed in [9], with the identification of relationships between graph entities. The discovered schemata are represented using a set of meta-layer constructs, and finally, this meta-layer representation is transformed into a PDF image, and a set of Spring Roo scripts [75]. This means that if, for example, the analyzed application relies on graph and document data stores, our tool generates two schemata, one for each data model. Both schemata are depicted in a unique PDF image and two Spring Roo scripts are generated, one for each schema. Another example of data collection schema inference is present in current SPARK SQL functions. In this platform schemata of DataFrames and Data Sets[17] can be inferred, extracted or explicitly defined using built-in operations.

## 3.3 Data Analytics

Methods for querying and mining Big Data are fundamentally different from traditional statistical analysis on small samples. Big Data are often noisy, dynamic, heterogeneous, interrelated and untrustworthy. Nevertheless, even noisy Big Data could be more valuable than tiny samples. Indeed, general statistics obtained from frequent patterns and correlation analysis usually overpower individual fluctuations and often disclose more reliable hidden patterns and knowledge.

Big Data forces to view data mathematically (e.g., measures, values distribution) first and establish a context for it later. For instance, how can researchers use statistical tools and computer technologies to identify meaningful patterns of information? How shall significant data correlations be interpreted? What is the role of traditional forms of scientific theorizing and analytic models in assessing data? What you really want to be doing is looking at the whole data set in ways that tell you things and answer questions that you are not asking. All these questions call for well-adapted infrastructures that can efficiently organize data, evaluate and optimize queries, and execute algorithms that require important computing and memory resources.

Big Data are enabling the next generation of interactive data analysis with real-time answers. In the future, queries toward Big Data will be automatically generated for content creation on Web sites, to populate hot-lists or recommendations, and to provide an ad hoc analysis of data sets to decide whether to keep or to discard them [68]. Scaling complex query processing techniques to terabytes while enabling interactive response times is a major open research problem today.

Analytical pipelines can often involve multiple steps, with built-in assumptions. By studying how best to capture, store, and query provenance, it is possible to create an infrastructure to interpret analytical results and to repeat the analysis with different assumptions, parameters, or data sets. Frequently, it is data exploration and visualization that allow Big Data to unleash its true impact. Visualization can help to produce and comprehend insights from Big Data. Visually, Tableau, Vizify, D3.js, R, are simple and powerful tools for quickly discovering new things in increasingly large datasets.

## 3.4 Parallel Model for Implementing Data Processing Functions

A consensus on parallel and distributed database system architecture emerged in the 1990's. This architecture was based on a shared-nothing hardware design [98] in which processors communicate with one another only by sending messages via a network. In such systems, tuples of each relation in the database were partitioned (declustered) across disk storage units attached directly to each processor. Partitioning allowed multiple processors to scan large relations in parallel without the need for any exotic I/O devices. Such architectures were pioneered by Teradata in the late seventies, and by several research projects. This

[16] http://code.google.com/p/exschema/.

[17] http://spark.apache.org/docs/latest/sql-programming-guide.html.

design is used by Teradata, Tandem, NCR, Oracle-nCUBE, and several other products. The research community adopted this shared-nothing dataflow architecture in systems like Arbre, Bubba, and Gamma.

The share-nothing design moves only questions and answers through the network. Raw memory accesses and raw disk accesses are performed locally in a processor, and only the filtered (reduced) data are passed to the client program. This allows a more scalable design by minimizing traffic on the network. The main advantage of shared-nothing multi-processors is that they can be scaled up to hundreds and probably thousands of processors that do not interfere with one another [39]. Twenty years later, Google's technical response to the challenges of Web-scale data management and analysis was the Google File System (GFS) [19]. To handle the challenge of processing the data in such large files, Google pioneered its Map-Reduce programming model and platform [53]. This model enabled Google's developers to process large collections of data by writing two user-defined functions, map and reduce, that the Map-Reduce framework applies to the instances (map) and sorted groups of instances that share a common key (reduce) similar to the sort of partitioned parallelism utilized in shared-nothing parallel query processing [19].

Yahoo!, Facebook, and other large Web companies followed. Taking Google's GFS and Map-Reduce papers as rough technical specifications, open-source equivalents were developed, and the Apache Hadoop Map-Reduce platform, and its underlying file system HDFS emerged.[18] Microsoft's technologies include a parallel runtime system called Dryad [69], and two higher-level programming models, DryadLINQ [110] and the SQL-like SCOPE language [27]. The Hadoop community developed a set of higher-level declarative languages for writing queries and data analysis pipelines that are compiled into Map-Reduce jobs, and then executed on the Hadoop MapReduce platform. Popular languages include Pig from Yahoo! [87], Jaql from IBM,[19] and Hive from Facebook.[20] Pig is relational-algebra-like in nature, and is reportedly used for over 60% of Yahoo!'s Map-Reduce use cases; Hive is SQL-inspired and reported to be used for over 90% of the Facebook Map-Reduce use cases [19].

Recent works agree on the need to study the Map-Reduce model for identifying its limitations and pertinence for implementing data processing algorithms like relational operators (i.e., join). Other platforms oriented to dataflows like Spark propose alternatives to data processing requiring computing resources and also Storm and Flink for dealing with streams (i.e., Big data velocity). New research opportunities are open in the database domain for studying different Map-Reduce models and proposing parallel programming strategies for accessing data that will consider the characteristics of different architectures like the cloud and its economic model and the QoS requirements of applications, and other architectures like clusters, HPC, grid and GPU.

[18] http://hadoop.apache.org.

[19] http://code.google.com/p/jaql/,

[20] http://hive.apache.org.

## 4 Big Data Analytics Systems

The emergence of Big Data some years ago denoted the challenge of dealing with huge collections of heterogeneous data continuously produced and to be exploited through data analytics processes. First approaches have addressed data volume and processing scalability challenges. Solutions can be described as balancing delivery of physical services such as: (i) hardware (computing, storage and memory); (ii) communication (bandwidth and reliability) and scheduling; (iii) greedy analytics and mining processes with high in-memory and computing cycles requirements. The next sections describe different systems approaches that provide solutions for dealing with Big Data: analytics stacks, distributed data persistence solutions, cloud data management services and parallel runtime environments.

### 4.1 Big Data Analytics Stacks

Due to their democratization, Big Data management and processing are no longer only associated to scientific applications with prediction, analytics requirements. Artificial intelligence algorithms requirements also call for Big Data aware management related to the understanding and automatic control of complex systems, to decision making in critical and non-critical situations. Therefore, new data analytics stacks have emerged as environments that provide the necessary underlying infrastructure for giving access to data collections and implementing data processing workflows to transform them and execute data analytics operations (statistics, data mining, knowledge discovery, computational science processes) on top of them.

One of the most prominent ones are Berkeley Data Analytics Stack (BDAS) from the AMPLAb project in Berkeley. BDAS is a multi-layered architecture that provides tools for virtualizing resources, addressing storage, data processing and querying as underlying tools for Big Data aware applications. Another important Big Data stack system is AsterixDB[21] from the Asterix project. AsterixDB is a scalable, open source Big Data Management System (BDMS).

[21] https://asterixdb.apache.org.

Data lake environments also deal with Big Data management and analytics through integrated environments designed as toolkits. A data lake is a shared data environment consisting of multiple repositories. It provides data to an organization for a variety of analytics processing including discovery and exploration of data, simple ad hoc analytics, complex analysis for business decisions, reporting, real-time analytics. Industrial solutions are in the market today, such as Microsoft Azure Data Lake, IBM, and Teradata.

## 4.2 Distributed Data Persistence Solutions

Data reads and writes in many data analytics workflows are guided by the RUM conjecture (Read, Update, Memory (or storage) overhead) [7] that characterizes the challenge of reducing overhead data being read, updated and stored (in memory, cache or disk). Several platforms address some aspect of the problem like Big Data stacks [3, 45]; data processing environments (*e.g.,* Hadoop, Spark, CaffeonSpark); data stores dealing with the CAP (consistency, atomicity and partition tolerance) theorem (*e.g.,* NoSQL's); and distributed file systems (*e.g.,* HDFS). The principle is to define API's (application programming interface) to be used by programs to interact with distributed data storage layers that can cope with distributed and parallel architectures.

In the distributed systems domain objects persistence has been an important issue addressed already by consolidated middleware such as JBOSS and PiJAMA. The new exascale requirements introduced by greedy processes often related to Big Data processing has introduced objects persistence again. In order for exascale and/or Big Data systems to deliver the needed I/O performance, new storage devices such as NVRAM or Storage Class Memories (SCM) need to be included into the storage/memory hierarchy. Given that the nature of these new devices will be closer to memory than to storage (low latencies, high bandwidth, and byte-addressable interface) using them as block devices for a file system does not seem to be the best option. DataClay,[22] proposes object storage to enable both the programmer, and DataClay, to take full advantage of the coming high-performance and byte-addressable storage devices. Today, given the lack of such devices, DataClay performs a mapping of such abstractions to key-value stores such as Kinetic drives from Seagate.[23]

Data structures and associated functions are sometimes more important for some requirements rather than non-functional properties like RUM or CAP. Non-relational databases have emerged as solutions when dealing with huge data sets and massive query work load. These systems have been redesigned to achieve scalability and availability at the cost of providing only a reduced set of low-level data management functions, thus forcing the client application to take care of complex logic. Existing approaches like Hecuba [4], Model2Roo [24] provide tools and interfaces, to ensure an efficient and global interaction with non-relational technologies.

The large spectrum of data persistence and management solutions are adapted for addressing workloads associated with Big Data volumes; and either simple read write operations or with more complex data processing tasks. The challenge today is choosing the right data management combination of tools for variable application requirements and architecture characteristics. Plasticity of solutions is from our point of view the most important property of such tools combination.

## 4.3 Cloud Data Management Services

Cloud computing is emerging as a relatively new approach for dealing with and facilitating unlimited access to computing and storage resources for building applications. The underlying infrastructure manages such resources transparently without including code in the application for managing and reserving more resources than those really required. The difference with classic approaches is that the application can have an ad hoc execution context, and that the resources it consumes are not necessarily located in one machine. Thanks to the cloud properties, applications can have ad hoc execution contexts. Following the same approach, database management systems functions can be delivered as services that must be tuned and composed for efficiently and inexpensively managing, querying and exploiting huge data sets.

Cloud architectures provide services at different scales and add constraints for accessing data for instance, access control, resources reservation, and assignment using priorities (e.g., in grid architectures) and economic cost (e.g., in the cloud). Applications deployed in these architectures specify QoS preferences (SLA contracts) that include execution and processing time, data pertinence and provenance, economic cost, and data processing energy consumption cost.

Thus data management must be revisited for designing strategies that couple the characteristics of novel architectures with users' preferences. In this context we identify three key scientific challenges: (i) data (flows) access and processing guided by SLA contracts, where data are produced by services and devices connected on heterogeneous networks; (ii) estimation and reduction in temporal,

[22] Toni Cortes, Anna Queralt, Jonathan Martí, and Jesus Labarta, DataClay: toward usable and shareable storage, http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/.

[23] http://www.seagate.com.

economic and energy consumption cost for accessing and processing data; (iii) optimization of data processing guided by SLA contracts expressed using cost models as reference.

### 4.4 ParAllel Runtime Environments

Today maybe because of the emergence of Big Data and greedy algorithms and applications requiring computing resources, parallel architectures have come back in the arena. There are different kinds of computing, memory and storage resources providers that adopt their own method for delivering such resources for executing programs. According to [76] there are three categories of resources provision: (i) Platform-as-a-Service (PaaS) frameworks, (ii) programming models for computing intensive workloads and (iii) programming models for Big Data.

PaaS offer APIs to write applications. For example, in the Microsoft Azure Cloud programming model applications are structured according to roles, which use APIs to communicate (queues) and to access persistent storage (blobs and tables). Microsoft Generic Worker proposes a mechanism to develop a Worker Role that eases the porting of legacy code in the Azure platform [95]. Google App Engine provides libraries to invoke external services and queue units of work (tasks) for execution; furthermore, it allows to run applications programmed in the Map-Reduce model. Data transfer and synchronization are handled automatically by the runtime. Environments for computing workload intensive applications use in general the bag of tasks execution model conceiving an application as composed of independent parallel tasks. For example, the Cloud BigJob, Amazon EC2, Eucalyptus and Nimbus Clouds and ProActive that offers a resource manager developed to mix Cloud and Grid resources [5, 90]. Map-Reduce programming is maybe the most prominent programming model for data intensive applications. Map-Reduce-based runtime environments provide good performance on cloud architectures above all on data analytics tasks working on large data collections. Microsoft Daytona[24] proposes an iterative Map-Reduce runtime for Windows Azure to support data analytics and machine learning algorithms. Twister [58] is an enhanced Map-Reduce runtime with an extended programming model for iterative Map-Reduce computations. Hadoop [20] is the most popular open source implementation of Map-Reduce on top of HDFS, as said in the previous section. The use of Hadoop avoids the lock into a specific platform allowing to execute the same Map-Reduce application on any Hadoop compliant service, as the Amazon Elastic Map-Reduce.[25]

## 5 Discussion

In order to face challenges introduced by novel applications requirements, the database community came up with new ways of delivering the system's internal and external functions to applications: as components (by the end of the 90's), as peer-to-peer networks (in the beginning of the 2000's), and as service-based database management systems the last 10 or 15 years. These new ways of delivering data management functions have been done under different architectures: centralized, distributed, parallel and on-cloud. Data management services are deployed on different system/hardware architectures: client-server (on classic and embedded systems), distributed on the grid, on P2P networks, on the W2.0, and recently on the clouds.

From our high-level view of the architecture of DBMS, we can conclude that although they make efficient use of the resources of their underlying environment, once configured they are fixed to that environment as well. Of course, there are settings with DBMS deployed on top of different target architectures but then it is up to the database manager and the application to give an integrated view of such multi-database solution. Designing and maintaining multi-databases requires important effort and know how. Such solutions tend to require also effort when they evolve due to new/changing data storage, data processing and application requirements. In a service-oriented environment where various hardware and software platforms are hidden behind service interfaces, deciding where to store data, is up to the data provider and the data consumer has few or no control on these issues, given services autonomy. Furthermore, for several applications in dynamic environments ACID transactions may not be feasible or required. In addition, DBMS are not easily portable and often impose a large footprint. A related problem is that they are difficult to evolve and maintain. Indeed, adding new functions, supporting evolutions of the data model, extending the query language and then supporting well adapted optimization strategies, can require important coding effort. For example, extending DBMS for dealing with streams, multimedia, geographical data implied important effort some years ago. Having new transactional models has required big implementation effort to materialize theory into efficient transaction managers. Changes should not penalize previous applications and they must efficiently support new uses. For these reasons, several researchers have concluded that they in fact exhibit under performance [96] or are even inappropriate [71] for a variety of new applications.

[24] http://research.microsoft.com/en-us/projects/daytona/ Last time visited 29.03.2016.

[25] Amazon elastic map reduce. http://aws.amazon.com/documentation/elasticmapreduce/. Last visited on 30.04.2016.

Consequently, the core functionality of the DBMS must be adapted for new settings and applications, particularly for dynamic and service-oriented environments. Services allow dynamic binding to accomplish complex tasks for a given client. Moreover, services are reusable, and have high autonomy because they are accessed through well defined interfaces. Organizing services on layers is a solution to compose a large numbers of services, and will help in making decisions about their granularity (e.g., micro-services). This allows to reuse optimized functionality that is shared by several applications instead of having to invest effort on the implementation of the same functionality again. Another way of extending the system is by invoking internal services through calls from external Web services or Web servers. Users can thereby have their own tailored services on their personal computers to replace or extend existing SBDBM services according to their needs.

Developers of new applications can benefit from service reuse by integrating one or more services that run on the SBDBMS, from any available layer, in an application. This can provide optimized access to their application-specific data. For example, assume that an application needs access to the storage level of the DBMS in order to obtain statistical information, such as available storage space or data fragmentation. In this case, the developer can add the necessary information services to the storage level. Thus, she provides the information source required for her application. Then, the application has to just invoke these services to retrieve the data. Furthermore, other services from other layers can be used together with this kind of extension services if required. Services can be distributed, and be made redundant by using several computers connected through a network. Therefore, a SBDBMS can be customized to use services from other specific locations to optimize particular tasks. This approach introduces a high degree of adaptability into the database system. Priorities can be assigned to services that demand a considerable amount of resources, thereby enabling Quality of Service agreements (QoS) for special data types like multimedia and streaming data.

These challenges imply the construction of service-based middleware with two open problems:

1. Exploit available resources making a compromise between QoS properties and Service Level Agreements (SLA) considering all the levels of the stack (i.e., from the network (infrastructure) to the application level).
2. Optimally coordinate services considering applications' and users' characteristics for fulfilling their requirements.

## 6 Perspectives

An important observation to make is that in today's perspectives introduced by architectures like the cloud, and by movements like Big Data, there is an underlying economic model that guides directly the way they are addressed. This has not been a common practice in previous eras, but today the "pay-as-you-go" economic models have become an important variable of (big) data production, consumption, and processing.

Which is the value to obtain from Big Data? Big Data is a dynamic/activity that crosses many IT borders. Big Data is not only about the original content stored or being consumed but also about the information around its consumption. Big Data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis.[26] Even if technology has helped by driving the cost of creating, capturing, managing, and storing information the prime interest is economic: the trick is to generate value by extracting the right information from the digital universe.

## References

1. Adiba M (2007) Ambient, continuous and mobile data, Presentation
2. Afrati FN, Sarma AD, Menestrina D, Parameswaran AG, Ullman JD (2012) Fuzzy joins using mapreduce. In: ICDE, pp 498–509
3. Alexandrov A, Bergmann R, Ewen S, Freytag JC, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V et al (2014) The stratosphere platform for big data analytics. VLDB J 23(6):939–964
4. Alomar G, Fontal YB, Torres Viñals J (2015) Hecuba: Nosql made easy. In: Montserrat GF (ed) BSC doctoral symposium, 2nd edn. Barcelona Supercomputing Center, Barcelona, pp 136–137
5. Amedro B, Baude F, Caromel D, Delbé C, Filali I, Huet F, Mathias E, Smirnov O (2010) An efficient framework for running applications on clusters, grids, and clouds. In: Cloud computing. Springer, New York, pp 163–178
6. Astrahan MM, Blasgen MW, Chamberlin DD, Eswaran KP, Gray JN, Griffiths PP, Frank King W, Lorie RA, McJones PR, Mehl JW et al (1976) System R: relational approach to database management. ACM Trans Database Syst 1(2):97–137

[26] http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

7. Athanassoulis M, Kester M, Maas L, Stoica R, Idreos S, Ailamaki A, Callaghan M (2016) Designing access methods: the rum conjecture. In: International conference on extending database technology (EDBT)
8. Atkinson MP, Bancilhon F, DeWitt DJ, Dittrich KR, Maier D, Zdonik SB (1989) The object-oriented database system manifesto. In: DOOD, vol 89, pp 40–57
9. Atzeni P, Bugiotti F, Rossi L (2012) Uniform access to non-relational database systems: the SOS platform. In: Advanced information systems engineering. Springer, New York, pp 160–174
10. Bancilhon F, Delobel C, Kanellakis PC (eds) (1992) Building an object-oriented database system, the story of O2. Morgan Kaufmann, San Francisco
11. Banerjee S, Krishnamurthy V, Krishnaprasad M, Murthy R (2000) Oracle8i-the XML enabled data management system. In: Proceedings of the 16th international conference on data engineering, pp 561–568
12. Batoory DS, Barnett JR, Garza JF, Smith KP, Tsukuda K, Twichell BC, Wise TE (1988) GENESIS: an extensible database management system. IEEE Trans Softw Eng 14(11):1711–1730
13. Batory DS (1988) Concepts for a database system compiler. In: Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. ACM, pp 184–192
14. Blakeley JA (1994) Open object database management systems. In: SIGMOD conference, p 520
15. Blakeley JA (1996) Data access for the masses through ole db. In: ACM SIGMOD record, vol 25. ACM, New York, pp 161–172
16. Blakeley JA (1996) OLE DB: a component DBMS architecture. In: Proceedings of the twelfth international conference on data engineering, pp 203–204
17. Blakeley JA, Pizzo MJ (1998) Microsoft universal data access platform. In: ACM SIGMOD record, vol 27. ACM, pp 502–503
18. Bliujute R, Saltenis S, Slivinskas G, Jensen CS (1999) Developing a datablade for a new index. In: Proceedings of the 15th international conference on data engineering, pp 314–323
19. Borkar V, Carey MJ, Li C (2012) Inside big data management: ogres, onions, or parfaits? In: Proceedings of the 15th international conference on extending database technology. ACM, pp 3–14
20. Borthakur D (2007) The hadoop distributed file system: architecture and design. Hadoop Proj Website 11(2007):21
21. Bruno G, Collet C, Vargas-Solar G (2006) Configuring intelligent mediators using ontologies. In: EDBT workshops, pp 554–572
22. Carey M, Haas L (1990) Extensible database management systems. ACM SIGMOD Rec 19(4):54–60
23. Carey MJ, DeWitt DJ, Frank D, Graefe G, Richardson JE, Shekita EJ, Muralikrishna M (1991) The architecture of the EXODUS extensible dbms. In: On object-oriented database, system, pp 231–256
24. Castrejón JC, López-Landa R, Lozano R (2011) Model2roo: a model driven approach for web application development based on the eclipse modeling framework and spring roo. In: 21st international conference on electrical communications and computers (CONIELECOMP), pp 82–87
25. Cattell RGG, Barry D (eds) (1997) The object database standard: ODMG 2.0. Morgan Kaufmann, San Francisco
26. Cattell R (2010) Scalable SQL and NoSQL data stores. SIGMOD Record 39(4):12–27
27. Chaiken R, Jenkins B, Larson PÅ, Ramsey B, Shakib D, Weaver S, Zhou J (2008) Scope: easy and efficient parallel processing of massive data sets. Proc VLDB Endow 1(2):1265–1276
28. Chrysanthis PK, Ramamritham K (1994) Synthesis of extended transaction models using acta. ACM Trans Database Syst 19(3):450–491
29. Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13:377–387
30. Collet C (2000) The NODS project: networked open database services. In: 14th European conference on object-oriented programming (ECOOP-2000), June 2000
31. Collet C, Amann B, Bidoit N, Boughanem M, Bouzeghoub M, Doucet A, Gross-Amblard D, Petit J-M, Hacid M-S, Vargas-Solar G (2013) De la gestion de bases de données à la gestion de grands espaces de données. Ingénierie des Systèmes d'Information 18(4):11–31
32. Collet C, Belhajjame K, Bernot G, Bobineau C, Bruno G, Finance B, Jouanot F, Kedad Z, Laurent D, Tahi F, Vargas-Solar G, Vu T-T, Xue X (2004) Towards a mediation system framework for transparent access to largely distributed sources, the mediagrid project. In: ICSNW, pp 65–78
33. Collet C, Vargas-Solar G, Grazziotin-Ribeiro H (2000) Open active services for data-intensive distributed applications. In: IDEAS, pp 349–359
34. Collet C, Vu T-T (2004) QBF: a query broker framework for adaptable query evaluation. In: FQAS, pp 362–375
35. Dadam P, Kuespert K, Andersen F, Blanken HM, Erbe R, Guenauer J, Lum VY, Pistor P, Walch G (1986) A DBMS prototype to support extended NF2 relations: an integrated view on flat tables and hierarchies. In: SIGMOD conference, pp 356–367
36. Davulcu H, Freire J, Kifer M, Ramakrishnan IV (1999) A layered architecture for querying dynamic web content. In: ACM SIGMOD record, vol 28. ACM, pp 491–502
37. Dessloch S, Chen W, Chow J-H, Fuh Y-C, Grandbois J, Jou M, Mattos NM, Nitzsche R, Tran BT, Wang Y(2001) Extensible indexing support in db2 universal database. In: Compontent database systems, pp 105–138
38. Dessloch S, Mattos N (1997) Integrating SQL databases with content-specific search engines. VLDB 97:528–537
39. DeWitt D, Gray J (1992) Parallel database systems: the future of high performance database systems. Commun ACM 35(6):85–98
40. Dittrich KR, Geppert A (2000) Component database systems. Morgan Kaufmann, San Francisco
41. Dittrich KR, Gotthard W, Lockemann PC (1986) DAMOKLES–a database system for software engineering environments. In: Advanced programming environments, pp 353–371
42. Drew P, King R, Heimbigner D (1992) A toolkit for the incremental implementation of heterogeneous database management systems. VLDB J Int J Very Large Data Bases 1(2):241–284
43. D'souza DF, Wills AC (1998) Objects, components, and frameworks with UML: the catalysis approach, vol 1. Addison-Wesley, Reading
44. Fayad M, Schmidt DC (1997) Object-oriented application frameworks. Commun ACM 40(10):32–38
45. Franklin M (2013) The berkeley data analytics stack: present and future. In: IEEE international conference on big data, pp 2–3
46. Fritschi H, Gatziu S, Dittrich KR (1998) FRAMBOISE: an approach to framework-based active database management system construction. In: Proceedings of the seventh international conference on information and knowledge management. ACM, pp 364–370
47. Frost S (1998) Component-based development for enterprise systems: applying the SELECT perspective. Cambridge University Press, Cambridge
48. García-Bañuelos L, Duong P-Q, Collet C (2003) A component-based infrastructure for customized persistent object management. In: DEXA workshops, pp 536–541
49. Garcia-Molina H, Papakonstantinou Y, Quass D, Rajaraman A, Sagiv Y, Ullman J, Vassalos V, Widom J (1997) The TSIMMIS approach to mediation: data models and languages. J Intell Inform Syst 8(2):117–132

50. Georgakopoulos D, Hornick M, Krychniak P, Manola F (1994) Specification and management of extended transactions in a programmable transaction environment. In: Proceedings of 10th international conference data engineering, 1994, pp 462–473
51. Geppert A, Dittrich KR (1994) Constructing the next 100 database management systems: like the handyman or like the engineer? ACM SIGMOD Rec 23(1):27–33
52. Geppert A, Scherrer S, Dittrich KR (1997) Construction of database management systems based on reuse. University of Zurich, KIDS
53. Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. In: ACM SIGOPS operating systems review, vol 37. ACM, pp 29–43
54. Graefe G (1995) The Cascades framework for query optimization. Data Eng Bull 18(3):19–29
55. Graefe G, DeWitt DJ (1987) The EXODUS optimizer generator, vol 16. ACM, New York
56. Graefe G, McKenna WJ (1993) The volcano optimizer generator: extensibility and efficient search. In: Proceedings of the ninth international conference on data engineering, 1993, pp 209–218
57. Gray J, Reuter A (1993) Transaction processing: concepts and techniques. Morgan Kaufmann Publishers, Burlington
58. Gunarathne T, Zhang B, Tak-Lon W, Qiu J (2013) Scalable parallel computing on clouds using twister4azure iterative mapreduce. Future Gener Comput Syst 29(4):1035–1048
59. Guzenda L (2000) Objectivity/DB-a high performance object database architecture. In: Workshop on high performance object databases
60. Haas LM, Chang W, Lohman GM, McPherson J, Wilms PF, Lapis G, Lindsay B, Pirahesh H, Carey MJ, Shekita E (1990) Starburst mid-flight: as the dust clears [database project]. IEEE Trans Knowl Data Eng 2(1):143–160
61. Haas LM, Freytag JC, Lohman GM, Pirahesh H (1989) Extensible query processing in starburst. In: SIGMOD conference, pp 377–388
62. Haerder T (2005) DBMS architecture-still an open problem. BTW 65:2–28
63. Haerder T, Rahm E (2001) Datenbanksysteme: Konzepte und Techniken der Implementierung; mit 14 Tabellen. Springer, Berlin
64. Haerder T, Reuter A (1983) Principles of transaction-oriented database recovery. ACM Comput Surv 15(4):287–317
65. Haerder T, Reuter A (1985) Architektur von datenbanksystemen fuer non-standard-anwendungen. In: BTW, pp 253–286
66. Hainaut JL, Henrard J, Englebert V, Roland D, Hick JM (2009) Database reverse engineering. In: Encyclopedia of database systems, pp 723–728
67. Hey T, Tansley S, Tolle KM (eds) (2009) The fourth paradigm: data-intensive scientific discovery. Microsoft Research
68. Idreos S, Alagiannis I, Johnson R, Ailamaki A (2011) Here are my data files. Here are my queries. Where are my results? In: Proceedings of 5th Biennial conference on innovative data systems research, number EPFL-CONF-161489
69. Isard M, Budiu M, Yuan Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Oper Syst Rev 41(3):59–72
70. Knolle H, Schlageter G, Welker E, Unland R (1992) TOPAZ: a tool kit for the construction of application-specific transaction managers. In: Objektbanken fuer experten. Springer, pp 254–280
71. Kossmann D (2008) Building web applications without a database system. In: Proceedings of the 11th international conference on extending database technology: advances in database technology, EDBT '08. ACM, New York, pp 3
72. Krieger D, Adler RM (1998) The emergence of distributed component platforms. Computer 31(3):43–53
73. Lindsay B, McPherson J, Pirahesh H (1987) A data management extension architecture, vol 16. ACM, New York
74. Linnemann V, Kuespert K, Dadam P, Pistor P, Erbe R, Kemper A, Suedkamp N, Walch G, Wallrath M (1988) Design and implementation of an extensible database management system supporting user defined data types and functions. In: VLDB, pp 294–305
75. Long J, Mayzak S (2011) Getting started with Roo. O'Reilly, Sebastopol
76. Lordan F, Tejedor E, Ejarque J, Rafanell R, Alvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia RM (2014) Servicess: An interoperable programming framework for the cloud. J Grid Comput 12(1):67–91
77. Lynch CA, Stonebraker M (1988) Extended user-defined indexing with application to textual databases. In: VLDB, pp 306–317
78. Fowler M, Sadalage P (2012) A brief guide to the emerging world of polyglot persistence, NoSQL Distilled
79. Mattern F (2001) Ubiquitous computing. Presentation
80. McKenna WJ, Burger L, Hoang C, Truong M (1996) EROC: a toolkit for building neato query optimizers. In: VLDB. Citeseer, pp111–121
81. Melton J, Simon AR (1993) Understanding the new SQL: a complete guide. Morgan Kaufmann, Burlington
82. Mohan C (2013) History repeats itself: sensible and nonsensql aspects of the NoSQL hoopla. In: Proceedings of the 16th international conference on extending database technology. ACM, pp 11–16
83. Mullins C (2012) Database administration: the complete guide to DBA practices and procedures, 2nd edn. Addison-Wesley (**ISBN 0201741296**)
84. Nierstrasz O, Dami L (1995) Component-oriented software technology. Object-Oriented Softw Compos 1:3–28
85. Nierstrasz O, Dami L (1995) Research directions in software composition. ACM Comput Surv 27(2):262–264
86. Olson S, Pledereder R, Shaw P, Yach D (1998) The sybase architecture for extensible data management. IEEE Data Eng Bull 21(3):12–24
87. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A (2008) Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, pp 1099–1110
88. Orfali R, Harkey D, Edwards J (1996) The essential distributed objects survival guide. Wile (**ISBN 0471129933**)
89. Tamer Özsu M, Muñoz A, Szafron D (1995) An extensible query optimizer for an objectbase management system. In: CIKM, pp 188–196
90. Peng J, Zhang X, Lei Z, Zhang B, Zhang W, Li Q (2009) Comparison of several cloud computing platforms. In: IEEE of second international symposium on information science and engineering (ISISE), 2009, pp 23–27
91. Rohm U, Bohm K (1999) Working together in harmony-an implementation of the corba object query service and its evaluation. In: Proceedings of the IEEE 15th international conference on data engineering, pp 238–247
92. Roth MT, Schwarz PM (1997) Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In: VLDB, vol 97. DTIC Document, pp 25–29
93. Schek H-J, Paul H-B, Scholl MH, Weikum G (1990) The DASDBS project: objectives, experiences, and future prospects. IEEE Trans Knowl Data Eng 2(1):25–43
94. Seshadri P (1998) Predator: a resource for database research. ACM SIGMOD Rec 27(1):16–20
95. Simmhan Y, Van Ingen C, Subramanian G, Li J (2010) Bridging the gap between desktop and the cloud for escience applications. In: 2010 IEEE 3rd international conference on cloud computing (CLOUD), pp 474–481

96. Stonebraker M, Cetintemel U (2005) One size fits all: an idea whose time has come and gone. In: Proceedings of the 21st international conference on data engineering, ICDE '05. IEEE Computer Society, Washington, pp 2–11
97. Stonebraker M, Held G, Wong E, Kreps P (1976) The design and implementation of INGRES. ACM Trans Database Syst 1:189–222
98. Stonebraker M, Katz RH, Patterson DA, Ousterhout JK (1988) The design of XPRS. In: VLDB, pp 318–330
99. Stonebraker M, Rowe LA (1986) The design of postgres. In: SIGMOD conference, pp 340–355
100. Stonebraker M, Rubenstein WB, Guttman A (1983) Application of abstract data types and abstract indices to CAD data bases. In: Engineering design applications, pp 107–113
101. Subasu I, Ziegler P, Dittrich KR (2007) Towards service-based data management systems. In: Workshop proceedings of datenbanksysteme in business, technologie und Web (BTW 2007), pp 3–86130
102. Szyperski CA (2002) Component software: beyond OO programming, 2nd edn. Addison-Wesley (**ISBN 0201745720**)
103. Tiwari S (2011) Professional NoSQL. Wiley, Hoboken
104. Tomasic A, Raschid L, Valduriez P (1998) Scaling access to heterogeneous data sources with disco. IEEE Trans Knowl Data Eng 10(5):808–823
105. Vargas-Solar G, Collet C, Grazziotin-Ribeiro H (2000) Active services for federated databases. SAC 1:356–360
106. Vogels W (2009) Eventually consistent. Commun ACM 52(1):40–44
107. Vu TT, Collet C (2004) Adaptable query evaluation using qbf. In: IDEAS, pp 265–270
108. Wells DL, Blakeley JA, Thompson CW (1992) Architecture of an open object-oriented database management system. IEEE Comput 25(10):74–82
109. Wiederhold G (1992) Mediators in the architecture of future information systems. Computer 25(3):38–49
110. Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson Ú, Gunda PK, Currey J (2008) DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language. In: OSDI, vol 8, pp 1–14