

Big Data Management in the Cloud: Evolution or Crossroad?

Abdelkader Hameurlain^(✉) and Franck Morvan

IRIT Institut de Recherche en Informatique de Toulouse IRIT,
Paul Sabatier University, 118, Route de Narbonne, 31062 Toulouse Cedex, France
{[abdelkader.hameurlain](mailto:abdelkader.hameurlain@irit.fr),[franck.morvan](mailto:franck.morvan@irit.fr)}@irit.fr

Abstract. In this paper, we try to provide a synthetic and comprehensive state of the art concerning big data management in cloud environments. In this perspective, data management based on parallel and cloud (e.g. MapReduce) systems are overviewed, and compared by relying on meeting software requirements (e.g. data independence, software reuse), high performance, scalability, elasticity, and data availability. With respect to proposed cloud systems, we discuss evolution of their data manipulation languages and we try to learn some lessons should be exploited to ensure the viability of the next generation of large-scale data management systems for big data applications.

Keywords: Big data management · Data partitioning · Query processing and optimization · Parallel Relational Database Systems · High performance · Scalability · Cloud systems · Hadoop · Mapreduce · Spark · Elasticity

1 Introduction

Data management process dates from long ago (i.e. since 4000 BC). Between 4000 BC and 2000 AD, five generations of data management have been distinguished by [28]: “Manual processing paper and pencil; Mechanical punched card; Stored program sequential record processing; On-line navigational set processing; Nonprocedural- relational; Multimedia internetwork”. In this paper, we will be interested in the penultimate generation.

A large number of datasets (structured, unstructured and semi-structured data) are produced by different sources (e.g. scientific observation, simulation, sensors, logs, social networks, finance). This large number of datasets, often referred to as Big Data and characterized by 4Vs (Volume, Variety, Velocity, and Value) [57], are distributed in large scale, heterogeneous, and produced continuously. The management of such data raises new problems and presents a real challenge such as: data modeling and storage, query processing and optimization, data replication and caching, cost models, concurrency control and transaction, data privacy and security, data streaming, monitoring services and tuning, autonomic data management (e.g. self tuning, self repairing).

In the landscape of database management systems, data analysis (OLAP) and transaction processing systems (OLTP) are separately managed. The reasons [2] for this dichotomy are that both systems have very different functionalities, characteristics and requirements. This paper will focus only on the first class (OLAP systems).

Very recently (with respect to reference period 4000 BC), we have seen an explosion in the volume of data manipulated by applications. This phenomenon results from, on the one hand, that it is easier to collect information (e.g. log information) and, on the other hand, lower cost of storage devices. Querying and analyzing of collected data, with acceptable response time, have become essential for many companies such as web societies. Furthermore, some applications with a small amount of data need high availability of data. This is the case, for example, with data from an online game with great success. In both contexts, a uniprocessor database server can quickly become a bottleneck or result in prohibitive response times for certain queries.

To manage a huge amount of data and meet the requirements in terms of high performance (e.g. minimizing of response time) and resource availability (e.g. data source), there are two approaches: parallel database systems and cloud systems. The parallel database systems [20, 47, 62] have been an important success, both in research in the early 90s and now in industry. They have enabled many applications handling large data volumes to meet their requirements in terms of high performance and resource availability. It is recognized that parallel database systems are very expensive and require having high level skills within the company to administer the systems and databases. As for cloud systems, developed by using data processing frameworks (e.g. Hadoop MapReduce, Apache Spark) [4, 18, 29], they allow a company to reduce these costs in terms of infrastructures either by purchasing a server comprised of low-cost commodity machines or by renting services (Infrastructure-as-a Service IaaS, Platform-as-a Service PaaS, Software-as-a Service SaaS) in pay-per-use.

Currently, new tools [21, 49, 54, 65] allow to make the bridge between both approaches. These tools allow either to a MapReduce program to load data from a relational database, either to convert, through a wrapper, a file stored in an HDFS format into a relational format. This class of systems is called multistore systems [7].

In this paper we propose a synthetic and comprehensive state of the art concerning: (i) big data management in parallel database systems and cloud systems, and (ii) evolution of data manipulation languages in cloud systems. There are many synthesis papers about data management in cloud systems [2, 12, 25, 45, 55]. Agrawal et al. [2] and Chaudhuri [12] focus on the future challenges of clouds systems to meet the needs of applications. As far as the contributions of Floratou et al. [25] and Stonebraker et al. [55] they propose a performance comparison of applications with cloud systems and parallel database systems. They point out the advantages and drawbacks of each system depending on the application type.

The rest of this paper is structured as follows. In Sect. 2, data management based on parallel and cloud (e.g. MapReduce) systems are over-viewed and compared by relying on meeting software requirements (e.g. data independence, software reuse), high performance, scalability, elasticity, and data availability. Section 3 presents an overview of data manipulation languages proposed in cloud systems: (i) without relational operators, and (ii) including relational operators in data manipulation languages. In Sect. 4, with respect to proposed cloud systems, we try to learn some lessons and we discuss the evolution of their data manipulation languages. We conclude in Sect. 5.

2 Parallel Database Systems Versus Cloud Systems

2.1 Parallel Relational Database Systems

This sub-section presents an extended abstract of the paper [33]. Parallel database systems have been developed for applications processing a large volume of data. Their main objectives are to obtain high performance and resource availability. High performance can be obtained by integrating and efficiently exploiting different types of parallelism (partitioned parallelism, independent parallelism and pipeline parallelism) in relational database systems on parallel architecture models. More precisely, the objectives of parallel databases are: (i) ensuring the best cost/performance ratio compared with a mainframe solution, (ii) minimizing query response times by efficiently exploiting the different forms of parallelism and data placement approaches, (iii) improving the parallel system throughput by efficiently managing resources, and (iv) insuring scalability, which consists in holding the same performance after adding new resources and applications. A parallel relational database system is a standard relational DBMS implemented on a MIMD (Multiple Instruction Multiple Data Stream) parallel architecture. Editors of parallel servers offer three main categories of parallel RDBMS based on: (i) shared memory multiprocessor architecture, (ii) shared disks multiprocessor architecture, and (iii) shared nothing multiprocessor architecture (e.g. DBC 1012 Teradata [8,64], Tandem NonStop SQL [23], DB2 Edition [5], ORACLE Parallel Query).

Partitioning a relation is defined as distributing the tuples of the relation among several nodes (attached disks) [20]. In a parallel RDBMS, it becomes possible to: (i) improve I/O bandwidth by fully exploiting the parallelism of read operations of one or more relations (ii) apply data locality principle (operators are performed where/or very close to the data are located), and (iii) facilitate load balancing to maximize throughput. The key problem with data partitioning, also called data placement, consists in reaching and holding the best tradeoff between processing and communication [17]. Two approaches make it possible to solve the data placement problem of a set of relations in a parallel RDBMS. The first approach, called full desclustering [46], consists in distributing horizontally each base relation over all the nodes of the system. It is applied to shared memory parallel RDBMS. The second approach, called partial desclustering, consists in distributing each base relation over a subset of nodes. It is mainly found in a

shared-nothing parallel RDBMS which own a large number of nodes. However, whatever the advocated data placement approach used, there are many data partitioning methods. In parallel systems, three methods are generally offered to the administrator to distribute data over nodes [20, 46]: round robin, use of a hashing function which associates a node number to one or more attribute values of a relation, and use of range partitioning given by the administrator thanks to fragmentation predicates. These partitioning methods have great influence and impact on load balancing.

The optimization phase of SQL queries consists basically of three phases: logical optimization, physical optimization and parallelization. The problem raised at physical optimization, dealing with the choice of a scheduling search strategy for join operators among enumerative strategies or randomized ones. Each one of these strategies is more or less adapted depending on the query characteristics. A query is characterized by the number of relations it refers to (i.e. size), the number of join predicates and the way they are arranged (i.e. query shape) and its nature (i.e. ad-hoc or repetitive). As far as parallelization strategies, they have been introduced concerning inter-operation parallelization phase, the key problem of optimization [13, 14, 31, 32, 34, 36, 52, 58, 68]. Indeed, two inter-operation parallelization approaches have been described in the literature [47]: the one-phase and the two-phase approaches. In the two-phase approach [13, 31, 32, 34, 36], the first phase consists in generating a query execution plan (without considering run time resources). The second phase ensures an optimal allocation of resources (memory and processors) for the previously generated plan. As for the one-phase approach [14, 52, 68], plan generation and resource allocation are packed into one single phase.

Finally, with regard to minimization of communication costs, which represents the plague of parallelism, the parallel processing of SQL queries requires the initialization of several processes on different processors with underlying data communications. The main problem to be solved by parallel execution models is to find the best processing-communication trade-off in order to maximize the system throughput and minimize the response time, while maintaining an acceptable cost optimization. In this objective, several efforts have been conducted to reduce inter-processor communication costs, and in particular to avoid the redistribution of data and minimize message transfers [23, 30, 35].

2.2 Cloud Systems

The rapid development of information technology and the popularity of the Internet allow the emergence of new Web applications (e.g., social networks, log and profile analysis, online document indexing). These new applications produce data that are often under the form of continuous streaming (e.g., data from sensors), with very large volume, in heterogeneous formats and distributed in a large scale. To address these data characteristics, data management community has recently proposed, with respect to traditional RDBMS, new data management systems that are more flexible in terms of data models (compared to relational model

which is operational since 30 years), more cost-efficient in terms of investment (as most of these systems are open-source) and provide better availability of resources (i.e. data sources and computing resources (CPU, RAM, I/O and network bandwidth)) in terms of fault-tolerance. From this perspective, four classes of data management systems [37, 56], depending on the adopted data structure type, have been designed and developed, mainly, by Google, Amazon, Microsoft, Yahoo, Facebook, IBM, and Oracle: (i) Key-Value based systems, (ii) Document based systems, (iii) Column based systems and (iv) Graph based systems for social networks.

These systems, based on a shared-nothing architecture, have been developed in Hadoop environment, using the functional programming model MapReduce and HDFS/GFS (Hadoop Distributed/Google File System). In a cloud environment, these systems are aimed to achieve high performance, maintain scalability (because they are based on massively parallel architectures), ensure elasticity (on-demand service and pay-per-use) and guarantee the fault-tolerance. High performance is based on the intensive exploitation of intra-operation parallelism of an operation (in a *map* or *reduce* operation) and independent parallelism (between multiple *map* or *reduce* programs). The elasticity paradigm [50] consists in allocating resources dynamically on demand. It extends the objective function, combined with dynamic resource allocation models, by integrating the economic model meeting the “pay-per-use” principle (tenant side), and guaranteeing a minimum profitability (provider side). Finally, in most of the proposed solutions in cloud systems fault-tolerance is managed. In fact, when a processor or process fails, only a part of the query is executed again. This is very attractive for applications that have queries that can take up to several hours as, for example, the analysis of log files.

In addition, to insure a uniform access to heterogeneous, autonomous and distributed data sources (e.g. RDBMS, NoSQL, HDFS) several multistore systems have been, recently, proposed. These systems, based on Mediator-Wrapper architecture [63] can be classified in three categories [7]: (i) loosely-coupled multistore systems, (ii) tightly-coupled multistore systems, based on a shared-nothing architecture, whose objective is the high performance. This approach consists in modifying the SQL engine to make the data access to HDFS transparent. Polybase [21] illustrates this approach. And (iii) hybrid/integrated multistore systems, whose objective is to query indifferently structured and unstructured data using a SQL-like declarative language. SCOPE system [67] and CoherentPaaS project [7], illustrate this approach.

2.3 Comparison

In this sub-section, we propose a qualitative comparison between Parallel Relational Database Systems PRDBMS and cloud systems/MapReduce by pointing out their advantages and weakness. For a quantitative comparison, we strongly suggest to authors reading the very good papers [25, 38, 51, 55].

Advantages and Weakness of PRDBMS. With respect to compilation of recently published studies and experiences, the advantages of PRDBMS can be, mainly, summarized as follows: (i) logical data independence, meaning that any modification of a schema (data structure) have not any impact on application programs, (ii) regular data structure (relational schema), homogeneity and stability of parallel infrastructure (shared-nothing architecture) enable to estimate and deduce relevant annotations (Metadata, and Cost Models are used by an Optimizer-Parallelizer to generate an efficient parallel execution plan), (iii) partitioning degree for each base relation and parallelism degree estimation for each relational operator can be estimated by analytical models, (iv) declarative languages and sophisticated query Optimizer-Parallelizer (physical optimization, exploiting and integrating of partitioned, independent and dependent (pipeline) parallelisms), and (v) minimizing of communication costs by avoiding the data redistribution in some favorable cases.

However, their main weakness, in massive parallel environments, can be summarized as follows: (i) PRDBS run only on expensive servers, (ii) require very high level of skills to manage and administrate these systems, (iii) weak fault-tolerance in massive PRDBS, and (iv) hard management of Web applications (Web datasets are unstructured).

Advantages and Weakness of Cloud Systems/MapReduce. The main advantages of cloud systems/MapReduce are: (i) scaling very well to manage massive datasets, (ii) support the partitioned and independent parallelisms, (iii) mechanism to achieve load-balancing, and (iv) strong fault-tolerance because of HDFS characteristics and used mechanisms to data replication (a file is partitioned into fixed size chunks of 64 MB and each chunk is replicated, by default, three times).

As far as the weakness of MapReduce (initial version) two levels can be distinguished: application level and software level. In fact, application side, the developers: (i) are forced to translate their business logic to MapReduce model, (ii) have to provide efficient implementation for the *map* and *reduce* functions, and to determine the best scheduling of *map* and *reduce* operations. With regard to software side: (i) data-dependence: so, we lost the propriety of logical data independence which is a qualitative requirement of software engineering, (ii) extensive materialization of I/O (Input/Output), because each result of a *map* instance is written on the disk.

To avoid this weakness, recently, pipeline parallelism has been implemented in Tez framework which is used, recently, to improve Hive performance [24]. Also, Cloudera Impala [15] implements pipeline parallelism for all queries. However, the consequence is that the fault-tolerance or resource availability will be seriously weakened. For detailed and complete analysis of weakness of MapReduce, the most relevant work can be found in the recent survey papers [22, 44].

3 Evolution of Data Manipulation Languages in Cloud Systems

3.1 Introduction

First tools, such as MapReduce [18], Bigtable [10] and PNUTS [16], were proposed to develop cloud applications. These tools allow querying data using procedural languages without relational operators. Programs written using these languages introduce a dependency between data structure and programs. Thus, when you have some modifications on the data structure it is also necessary to adapt the programs. In addition, these programs are more difficult to optimize than program writing with a declarative language. Indeed, optimization of functions defined by “*users have never been a central data management challenge in researches*” [12]. Program optimizations and their maintenance due to data structure evolution lead to important human costs. This is why the first tools were mainly used for queries which are performed only once, such as applications on logs and paper collections [26]. Recently, new tools have emerged in order to avoid the dependency between data structure and programs. Their common goal is to use the benefits of data independence and implicit (automatic) optimization programs of parallel database approaches and the advantages of scalability, fault tolerance and elasticity of cloud systems. Thus, high level of declarative languages have emerged HiveQL [60], SCOPE language [67], and CloudMdsQL [7]. This allows an automatic optimization-parallelization of queries. Moreover, new tools [21, 49, 54, 65], based on integrated approach, allow to make the bridge between the both approaches. These tools allow either to a MapReduce program to load data from a relational database, either to convert, through a wrapper, a file stored in an HDFS format into a relational format.

In this section we propose a state of the art concerning the evolution of data manipulation languages in cloud systems, and we point out why the proposed languages have evolved. More precisely, we present an overview of data manipulation languages, first, without relational operators, and next, with relational operators in data manipulation languages.

3.2 Data Manipulation Languages Without Relational Operators

The first tools [10, 16, 18] proposed in the literature allow to manipulate massive datasets by using procedural languages. Generally, they propose relatively simple languages which permit only filter or project operation on massive datasets. These tools were, mainly, designed to serve Web applications which do not need complex queries. For example, they want to query massive datasets such as logs and click streams. However, Web applications require scalability, high performance (e.g. minimization of response time) and high availability of data.

A very popular framework for processing massive datasets is MapReduce [18]. It allows the programmer to write *map* and *reduce* functions which correspond respectively to perform grouping and reduce functions. The programming model provides a good level of abstraction. However, for some applications

(e.g. applications querying a relational model) this programming model is not suitable. It can make it complex to write some programs. For example, it is not easy for a programmer to write projection, selection, or join operators over datasets with *map* and *reduce* functions [9]. It also requires valuable expertise from the programmer since users specify the physical execution plan (i.e. users implement the operators and the scheduling of operators). This physical execution plan has many chances to be sub-optimal and there does not exist an automatic optimization process for a user program. We find a first solution to these problems in Bigtable [10] and Pnuts systems [16].

Bigtable [10] is a distributed storage system which supports a simple data model that look likes a relational model. For that it relies on the file management system GFS (Google File System) [27] which provides fault-tolerance and data availability [50]. A table is a sparse, distributed, persistent multidimensional sorted *map* where data is organized into three dimensions: rows, columns and timestamps. Rows are grouped together to form the unit of distribution and load balancing. Columns are grouped to form the unit of access control. As for timestamps, they allow to differentiate different versions of the same data. Bigtable provides a basic API for creating, deleting and querying a table in a procedural language such as C++. This API provides only simple operators for iterates over subsets of data produced by a scan operator. There is no implementation of complex operators like join and minus operations. In the same way, PNUITS [16] supports only selection and projection. It presents a simplified relational data model where data are stored in hashed or ordered relation. Pnuts tables are horizontally partitioned and each partition named tablets is no bigger than 1Gbyte. Other systems like Dynamo [19] and Cassandra [42] use an even simpler data access. Dynamo is used only by Amazon's internal services. Dynamo has a simple key/value interface which offers read and write operations to a data item that is uniquely identified by a key. As far as Cassandra, the data are also partitioned using a hash function and data are accessed by a key using an API composed of three methods: insert, get and delete.

3.3 Data Manipulation Languages Using Relational Operators

The use of low level languages, like MapReduce, forces the users to write repetitively the same code for standard operations, like relational operators (e.g. join operator), for all new datasets. This is expensive in terms of development. Furthermore, the programs are complex to read. The bug probability is increased and an optimization process is complex. Based on these observations, the Pig Latin [26, 48] and Jaql [6] languages have been proposed. These languages allow developers to work at a higher level of abstraction than MapReduce language. With Pig Latin a user can write without knowing the physical organization of the data and it introduces new operators like join. Pig Latin programs encode explicit dataflow graphs which interleave relational-style operators like join and filter with user-provided executables. A Pig Latin program is compiled in MapReduce program after four steps of transformation. Two of these steps concern the optimization process. A classical logical optimization step, where filter and

project operators pushdown on the graph in order to reduce the processed data volume. With regard to the second optimization step, it optimizes the MapReduce program generated by a MapReduce compiler step. This optimization step consists in break distributive and algebraic aggregation functions into a series of *map*, *combine* and *reduce* operators. The authors [3, 26] use the Combiner/Intermediate Reduce as often as possible in order to (i) reduces the volume of data handled by the shuffle and merge operators and (ii) balance the amount of data associated for each key in order to limit the data skew for the Reduce operator. Jaql is also a procedural language where the functions are combined using ‘->’ operator inspired from Unix pipes. A Jaql script [6] is transformed in a MapReduce program by a compiler following approximately the same optimization step as in Pig system. A difference with Pig Latin language is that the users have access to the internal system. This allows to users to develop a specific feature to solve performance problems.

A program written with Pig Latin or Jaql is complex to optimize. Indeed, the user determines a scheduling of relational operators. For an optimizer, it is difficult to determine a new optimal scheduling like in physical optimization of relational systems [43]. Furthermore, the alternative which suggests writing a program with low level languages requires a high expertise level from the user. For these reasons, Hive [59, 60], SCOPE [67] systems, and CoherentPaaS project [7] propose to use declarative languages close to SQL language. These languages allow a user to define a relation compounded of several typed columns. Each of these languages can load data from external data sources and insert query results in formats defined by users. For example, SCOPE language has been enriched by *extractor* operators in order to parse and construct rows from any kind of data sources and *outputter* operators to format the final result of a query. As for HiveQL language, the formats of an external data source or result is defined in data definition language (i.e. during create table order).

With regard to the optimization process, in Hive system, it comprises four steps: (i) a logical optimization step, (ii) a simplification step which prunes partitions and buckets that are not needed by the query, (iii) a combiner step which groups multiple joins sharing the same join attribute in order to be executed in a single MapReduce join, and (iv) a step which adds repartition operators for join, group-by and custom MapReduce operators. As for SCOPE system [67], the optimization process includes a logical optimization and chooses for each operator the best algorithm to process it (e.g. hash join or sort-merge join) and determines the scheduling of operators. With regard to avoid data reshuffling, which deteriorates the performance, a top-down approach is proposed in order that a parent operator imposes its requirements to the child operators [3, 66, 67].

4 Discussion

To manage a huge amount of data there are two approaches: parallel database servers and tools proposed by cloud servers. The parallel database servers have been an important success, whether in research in the early 90s and now in

industry. They have enabled many applications handling large data volumes to meet their requirements in terms of high performance and resource availability. However, the use of a parallel database server is expensive for a company. Indeed, it requires the purchase of an expensive server and requires having high level skills within the company to administer servers and databases.

An alternative approach is to use tools proposed by cloud servers to manipulate massive datasets. This allows a company to reduce these costs in terms of infrastructures (IaaS) either by purchasing a server comprised of low-cost commodity machines or by renting services (PaaS and/or SaaS) in pay-per-use. An important characteristic of cloud systems is to provide a mechanism for integrated fault tolerance. This feature is important because it avoids restarting, from the beginning, a program in case of processor or process failure. As data volumes grow every day, this feature has become a critical requirement for applications. In addition, more and more applications querying only one time a dataset. For this kind of application, the loading cost of the dataset in a database server becomes prohibitive for a single query. Hence, this kind of application is not suitable for a database server, and many applications have turned to use cloud systems.

Regarding to the state of the art and previous quantitative and qualitative studies and comparisons [25,38,55] between PRDBMS and cloud systems, we can point out the following statements:

1. *Functional Complementarity* between cloud systems and parallel DB systems: in fact, the cloud systems are not intended to replace traditional RDBMS but rather to provide them with the missing features, particularly, for Web applications and Internet services. Moreover, these systems provide scalability in terms of loads adapted for Web applications. In [55], the authors have conducted a benchmark study by comparing Hadoop/MapReduce and two parallel RDBMSs. “*The results show that the DBMSs are substantially faster than the MapReduce system once the data is loaded*”. Their main conclusion of [55] is that: “*MapReduce complement DBMSs since databases are not designed for ETL (Extract-Transform-Load) tasks, a MapReduce specialty*”.
2. *Maturity*: Compared to traditional RDBMS, the cloud systems lack maturity and standardization/normalization (e.g., query languages). These systems require more experimentation and benchmarking (e.g. TPC - H, and TPC - DS) [24] with full-scale big data applications while taking into maximum consideration simultaneously their Vs which characterize them.

As far as the evolution of data manipulation languages in cloud systems, the first proposed languages allow manipulation of data stored in a cloud system, either with functional languages like MapReduce or with imperative languages like C++. These languages force the developer to: (i) modify the program if the data structure changes, and (ii) often rewrite the very similar code on different datasets. In addition, in case of performance problems, these programs are very difficult to optimize due, in particular, to their understanding which is complex. Thus, new languages like Pig Latin [26] and Jaql [6] have been proposed using relational operators such as join. As a result, user programs are more readable. Using these high

level languages, the automatic optimization process has been introduced as the logic optimization, conventionally used by a RDBMS. However, these languages are still procedural and classical physical optimization process cannot be applied like in RDBMS. Therefore, Hive [60], Clydesdale [40] and SCOPE [67] systems and CoherentPaas project [7] use non-procedural (i.e. declarative) languages close to the SQL language. This helped to adapt the automatic relational optimization process proposed in the context of RDBMS (e.g. parallel RDBMS). The major drawback of this optimization process, compared to those used in RDBMS, is the scarcity of statistics stored in the meta-base. Indeed, in a cloud system, at compile time, there is no statistics on datasets (e.g. cardinality). This blinds the optimization process and impacts the choice of the optimal execution plan. Hence, a dynamic optimization process becomes necessary in order to react to sub-optimal execution plans. In this objective, [1] proposes to collect statistics at runtime, and adapts the execution plan at runtime by interfacing with a query optimizer. This proposal can be seen as an elegant adaptation of [39], proposed in a parallel database system, to a cloud system. More generally, with respect to the issue of query optimization in cloud environments, the most recent and relevant proposals are described in [11, 41, 53, 61].

5 Conclusion

In this paper, we provided a synthetic and highlight state of the art concerning big data management in cloud environments. In this objective, we have tackled two major issues: (i) data management based on parallel and cloud (e.g. MapReduce) systems are over-viewed and compared by relying on meeting software requirements (e.g. data independence, software reuse), high performance, scalability, elasticity, and data availability and (ii) we mainly focused on the evolution of data manipulation languages in cloud systems. Initially, these languages were low level procedural languages as for example MapReduce. For software engineering requirements these languages evolved by introducing relational algebra operators while remaining procedural. It also allowed introducing some optimization processes used classically in RDBMS. Then, they continued their evolution for optimization needs. They became declarative to increase the opportunities of automatic optimization for user queries. The various optimization steps are very close to those used in parallel RDBMS. The main difference is due to the quasi-absence of statistics used by cost models. This blinds the optimization process and an efficient dynamic optimization becomes essential and necessary, to correct sub-optimality of sub-execution plans.

References

1. Agarwal, S., Kandula, S., Bruno, N., Wu, M., Stoica, I., Zhou, J.: Reoptimizing data parallel computing. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, 25–27 April 2012, pp. 281–294 (2012). <https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/agarwal>

2. Agrawal, D., El Abbadi, A., Ooi, B.C., Das, S., Elmore, A.J.: The evolving landscape of data management in the cloud. *IJCSE* **7**(1), 2–16 (2012). <http://dx.doi.org/10.1504/IJCSE.2012.046177>
3. Akbarinia, R., Liroz-Gistau, M., Agrawal, D., Valduriez, P.: An efficient solution for processing skewed mapreduce jobs. In: Chen, Q., Hameurlain, A., Toumani, F., Wagner, R., Decker, H. (eds.) *DEXA 2015. LNCS*, vol. 9262, pp. 417–429. Springer, Heidelberg (2015)
4. Apache Spark. <https://spark.incubator.apache.org/>
5. Baru, C.K., Fecteau, G., Goyal, A., Hsiao, H., Jhingran, A., Padmanabhan, S., Wilson, W.G.: An overview of DB2 parallel edition. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, San Jose, California, 22–25 May 1995, pp. 460–462 (1995). <http://doi.acm.org/10.1145/223784.223876>
6. Beyer, K.S., Ercegovac, V., Gemulla, R., Balmin, A., Eltabakh, M.Y., Kanne, C., Özcan, F., Shekita, E.J.: JAQL: a scripting language for large scale semistructured data analysis. *PVLDB* **4**(12), 1272–1283 (2011). <http://www.vldb.org/pvldb/vol4/p1272-beyer.pdf>
7. Bondiombouy, C., Kolev, B., Levchenko, O., Valduriez, P.: Integrating big data and relational data with a functional SQL-like query language. In: Chen, Q., Hameurlain, A., Toumani, F., Wagner, R., Decker, H. (eds.) *DEXA 2015. LNCS*, vol. 9261, pp. 170–185. Springer, Heidelberg (2015)
8. Cariño, F., Kostamaa, P.: Exegesis of DBC/1012 and P-90 - industrial supercomputer database machines. In: Etienneble, D., Syre, J.-C. (eds.) *PARLE 1992. LNCS*, vol. 605, pp. 877–892. Springer, Heidelberg (1992). http://dx.doi.org/10.1007/3-540-55599-4_130
9. Chaiken, R., Jenkins, B., Larson, P., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB* **1**(2), 1265–1276 (2008). <http://www.vldb.org/pvldb/1/1454166.pdf>
10. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. *ACM Trans. Comput. Syst.* **26**(2), 4 (2008). <http://doi.acm.org/10.1145/1365815.1365816>
11. Chang, L., Wang, Z., Ma, T., Jian, L., Ma, L., Goldshuv, A., Loneragan, L., Cohen, J., Welton, C., Sherry, G., Bhandarkar, M.: HAWQ: a massively parallel processing SQL engine in hadoop. In: *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, 22–27 June 2014*, pp. 1223–1234 (2014). <http://doi.acm.org/10.1145/2588555.2595636>
12. Chaudhuri, S.: What next?: a half-dozen data management research goals for big data and the cloud. In: *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, 20–24 May 2012*, pp. 1–4 (2012). <http://doi.acm.org/10.1145/2213556.2213558>
13. Chekuri, C., Hasan, W., Motwani, R.: Scheduling problems in parallel query optimization. In: *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, California, USA, 22–25 May 1995*, pp. 255–265 (1995). <http://doi.acm.org/10.1145/212433.212471>
14. Chen, M., Lo, M., Yu, P.S., Young, H.C.: Using segmented right-deep trees for the execution of pipelined hash joins. In: *Proceedings of 18th International Conference on Very Large Data Bases, Vancouver, Canada, 23–27 August 1992*, pp. 15–26 (1992). <http://www.vldb.org/conf/1992/P015.PDF>
15. Cloudera Impala. <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html>

16. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D., Yerneni, R.: PNUTS: Yahoo!’s hosted data serving platform. *PVLDB* **1**(2), 1277–1288 (2008). <http://www.vldb.org/pvldb/1/1454167.pdf>
17. Copeland, G.P., Alexander, W., Boughter, E.E., Keller, T.W.: Data placement in bubba. In: *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, 1–3 June 1988, pp. 99–108 (1988). <http://doi.acm.org/10.1145/50202.50213>
18. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: *6th Symposium on Operating System Design and Implementation (OSDI 2004)*, San Francisco, California, USA, 6–8 December 2004, pp. 137–150 (2004). <http://www.usenix.org/events/osdi04/tech/dean.html>
19. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., Vogels, W.: Dynamo: amazon’s highly available key-value store. In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007*, Stevenson, Washington, USA, 14–17 October 2007, pp. 205–220 (2007). <http://doi.acm.org/10.1145/1294261.1294281>
20. DeWitt, D.J., Gray, J.: Parallel database systems: The future of high performance database systems. *Commun. ACM* **35**(6), 85–98 (1992). <http://doi.acm.org/10.1145/129888.129894>
21. DeWitt, D.J., Halverson, A., Nehme, R.V., Shankar, S., Aguilar-Saborit, J., Avanes, A., Flasz, M., Gramling, J.: Split query processing in polybase. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, New York, NY, USA, 22–27 June 2013, pp. 1255–1266 (2013). <http://doi.acm.org/10.1145/2463676.2463709>
22. Doukeridis, C., Nøravåg, K.: A survey of large-scale analytical query processing in mapreduce. *Vldb J.* **23**(3), 355–380 (2014). <http://dx.doi.org/10.1007/s00778-013-0319-9>
23. Englert, S., Glasstone, R., Hasan, W.: Parallelism and its price: a case study of nonstop SQL/MP. *SIGMOD Rec.* **24**(4), 61–71 (1995). <http://dx.doi.org/10.1145/219713.219760>
24. Floratou, A., Minhas, U.F., Özcan, F.: SQL-on-hadoop: full circle back to shared-nothing database architectures. *PVLDB* **7**(12), 1295–1306 (2014). <http://www.vldb.org/pvldb/vol7/p1295-floratou.pdf>
25. Floratou, A., Teletia, N., DeWitt, D.J., Patel, J.M., Zhang, D.: Can the elephants handle the NoSQL onslaught? *PVLDB* **5**(12), 1712–1723 (2012). <http://vldb.org/pvldb/vol5/p1712-avriliafloratou-vldb2012.pdf>
26. Gates, A., Natkovich, O., Chopra, S., Kamath, P., Narayanam, S., Olston, C., Reed, B., Srinivasan, S., Srivastava, U.: Building a highlevel dataflow system on top of mapreduce: the pig experience. *PVLDB* **2**(2), 1414–1425 (2009). <http://www.vldb.org/pvldb/2/vldb09-1074.pdf>
27. Ghemawat, S., Gobioff, H., Leung, S.: The Google file system. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003*, Bolton Landing, NY, USA, 19–22 October 2003, pp. 29–43 (2003). <http://doi.acm.org/10.1145/945445.945450>
28. Gray, J.: Evolution of data management. *IEEE Comput.* **29**(10), 38–46 (1996). <http://dx.doi.org/10.1109/2.539719>
29. Hadoop. <http://hadoop.apache.org>

30. Hameurlain, A., Morvan, F.: An optimization method of data communication and control for parallel execution of SQL queries. In: Proceedings of 4th International Conference on Database and Expert Systems Applications, DEXA 1993, Prague, Czech Republic, 6–8 September 1993, pp. 301–312 (1993). http://dx.doi.org/10.1007/3-540-57234-1_27
31. Hameurlain, A., Morvan, F.: A parallel scheduling method for efficient query processing. In: Proceedings of the 1993 International Conference on Parallel Processing. Algorithms & Applications, Syracuse University, NY, USA, 16–20 August 1993, vol. III, pp. 258–262 (1993). <http://dx.doi.org/10.1109/ICPP.1993.31>
32. Hameurlain, A., Morvan, F.: Scheduling and mapping for parallel execution of extended SQL queries. In: CIKM 1995, Proceedings of the 1995 International Conference on Information and Knowledge Management, Baltimore, Maryland, USA, 28 November–2 December 1995, pp. 197–204 (1995). <http://doi.acm.org/10.1145/221270.221567>
33. Hameurlain, A., Morvan, F.: Parallel relational database systems: Why, how and beyond. In: Proceedings of 7th International Conference on Database and Expert Systems Applications, DEXA 1996, Zurich, Switzerland, 9–13 September 1996, pp. 302–312 (1996). <http://dx.doi.org/10.1007/BFb0034690>
34. Hasan, W., Motwani, R.: Optimization algorithms for exploiting the parallelism-communication tradeoff in pipelined parallelism. In: VLDB 1994, Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 12–15 September 1994, pp. 36–47 (1994), <http://www.vldb.org/conf/1994/P036.PDF>
35. Hasan, W., Motwani, R.: Coloring away communication in parallel query optimization. In: VLDB 1995, Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland, 11–15 September 1995, pp. 239–250 (1995). <http://www.vldb.org/conf/1995/P239.PDF>
36. Hong, W.: Exploiting inter-operation parallelism in XPRS. In: Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, 2–5 June 1992, pp. 19–28 (1992). <http://doi.acm.org/10.1145/130283.130292>
37. Indrawan-Santiago, M.: Database research: Are we at a crossroad? reflection on nosql. In: 15th International Conference on Network-Based Information Systems, NBIS 2012, Melbourne, Australia, 26–28 September 2012, pp. 45–51 (2012). <http://dx.doi.org/10.1109/NBiS.2012.95>
38. Jiang, D., Ooi, B.C., Shi, L., Wu, S.: The performance of mapreduce: an in-depth study. PVLDB **3**(1), 472–483 (2010). <http://www.comp.nus.edu.sg/vldb2010/proceedings/files/papers/E03.pdf>
39. Kabra, N., DeWitt, D.J.: Efficient mid-query re-optimization of sub-optimal query execution plans. In: SIGMOD 1998, Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, 2–4 June 1998, pp. 106–117 (1998). <http://doi.acm.org/10.1145/276304.276315>
40. Kaldewey, T., Shekita, E.J., Tata, S.: Clydesdale: structured data processing on mapreduce. In: Proceedings of 15th International Conference on Extending Database Technology, EDBT 2012, Berlin, Germany, 27–30 March 2012, pp. 15–25 (2012). <http://doi.acm.org/10.1145/2247596.2247600>
41. Karanasos, K., Balmin, A., Kutsch, M., Ozcan, F., Ercegovac, V., Xia, C., Jackson, J.: Dynamically optimizing queries over large scale data platforms. In: International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, 22–27 June 2014, pp. 943–954 (2014). <http://doi.acm.org/10.1145/2588555.2610531>

42. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. *Opera. Syst. Rev.* **44**(2), 35–40 (2010). <http://doi.acm.org/10.1145/1773912.1773922>
43. Lancelotte, R.S.G., Valduriez, P.: Extending the search strategy in a query optimizer. In: *Proceedings of 17th International Conference on Very Large Data Bases*, Barcelona, Catalonia, Spain, 3–6 September 1991, pp. 363–373 (1991). <http://www.vldb.org/conf/1991/P363.PDF>
44. Lee, K., Lee, Y., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with mapreduce: a survey. *SIGMOD Rec.* **40**(4), 11–20 (2011). <http://doi.acm.org/10.1145/2094114.2094118>
45. Li, F., Ooi, B.C., Özsu, M.T., Wu, S.: Distributed data management using mapreduce. *ACM Comput. Surv.* **46**(3), 31: 1–31: 42 (2014). <http://doi.acm.org/10.1145/2503009>
46. Livny, M., Khoshafian, S., Boral, H.: Multi-disk management algorithms. In: *SIGMETRICS*, pp. 69–77 (1987). <http://doi.acm.org/10.1145/29903.29914>
47. Lu, H., Tan, K.L., Ooi, B.C.: *Query Processing in Parallel Relational Database Systems*. IEEE CS Press, Los Alamitos (1994)
48. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: a not-so-foreign language for data processing. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD 2008, Vancouver, BC, Canada, 10–12 June 2008, pp. 1099–1110 (2008). <http://doi.acm.org/10.1145/1376616.1376726>
49. Oracle. <http://www.oracle.com/technetwork/bdc/hadoop-loader/connectors->
50. Ozsu, M.T., Valduriez, P.: *Principles of Distributed Database Systems*. Springer, New York (2011)
51. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD 2009, Providence, Rhode Island, USA, 29 June–2 July 2009, pp. 165–178 (2009). <http://doi.acm.org/10.1145/1559845.1559865>
52. Schneider, D.A., DeWitt, D.J.: Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In: *Proceedings of 16th International Conference on Very Large Data Bases*, Brisbane, Queensland, Australia, 13–16 August 1990, pp. 469–480 (1990). <http://www.vldb.org/conf/1990/P469.PDF>
53. Soliman, M.A., Antova, L., Raghavan, V., El-Helw, A., Gu, Z., Shen, E., Caragea, G.C., Garcia-Alvarado, C., Rahman, F., Petropoulos, M., Waas, F., Narayanan, S., Krikellas, K., Baldwin, R.: Orca: a modular query optimizer architecture for big data. In: *International Conference on Management of Data*, SIGMOD 2014, Snowbird, UT, USA, 22–27 June 2014, pp. 337–348 (2014). <http://doi.acm.org/10.1145/2588555.2595637>
54. Sqoop. <http://sqoop.apache.org/>
55. Stonebraker, M., Abadi, D.J., DeWitt, D.J., Madden, S., Paulson, E., Pavlo, A., Rasin, A.: Mapreduce and parallel DBMSs: friends or foes? *Commun. ACM* **53**(1), 64–71 (2010). <http://doi.acm.org/10.1145/1629175.1629197>
56. Stonebraker, M., Cattell, R.: 10 rules for scalable performance in ‘simple operation’ datastores. *Commun. ACM* **54**(6), 72–80 (2011). doi:10.1145/1953122.1953144. <http://doi.acm.org/10.1145/1953122.1953144>
57. Stonebraker, M., Madden, S., Dubey, P.: Intel “big data” science and technology center vision and execution plan. *SIGMOD Rec.* **42**(1), 44–49 (2013). <http://doi.acm.org/10.1145/2481528.2481537>

58. Tan, K., Lu, H.: Pipeline processing of multi-way join queries in shared-memory systems. In: Proceedings of the 1993 International Conference on Parallel Processing. Architecture, Syracuse University, NY, USA, 16–20 August 1993, vol. I, pp. 345–348 (1993). <http://dx.doi.org/10.1109/ICPP.1993.147>
59. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive - a warehousing solution over a map-reduce framework. *PVLDB* **2**(2), 1626–1629 (2009)
60. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., Murthy, R.: Hive - a petabyte scale data warehouse using hadoop. In: Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, Long Beach, California, USA, 1–6 March 2010, pp. 996–1005 (2010). <http://dx.doi.org/10.1109/ICDE.2010.5447738>
61. Trummer, I., Koch, C.: Multi-objective parametric query optimization. *PVLDB* **8**(3), 221–232 (2014). <http://www.vldb.org/pvldb/vol8/p221-trummer.pdf>
62. Valduriez, P.: Parallel database systems: open problems and new issues. *Distrib. Parallel Databases* **1**(2), 137–165 (1993). doi:10.1007/BF01264049. <http://dx.doi.org/10.1007/BF01264049>
63. Wiederhold, G.: Mediators in the architecture of future information systems. *IEEE Comput.* **25**(3), 38–49 (1992). <http://dx.doi.org/10.1109/2.121508>
64. Witkowski, A., Cariño, F., Kostamaa, P.: NCR 3700 - the next-generation industrial database computer. In: Proceedings of 19th International Conference on Very Large Data Bases, Dublin, Ireland, 24–27 August 1993, pp. 230–243 (1993). <http://www.vldb.org/conf/1993/P230.PDF>
65. Xu, Y., Kostamaa, P., Gao, L.: Integrating hadoop and parallel DBMs. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, 6–10 June 2010, pp. 969–974 (2010). <http://doi.acm.org/10.1145/1807167.1807272>
66. Zha, L., Zhang, J., Liu, W., Lin, J.: An uncoupled data process and transfer model for mapreduce. In: Hameurlain, A., Küng, J., Wagner, R., Bellatreche, L., Mohania, M. (eds.) TLDKS XVII. LNCS, vol. 8970, pp. 24–44. Springer, Heidelberg (2015). http://dx.doi.org/10.1007/978-3-662-46335-2_2
67. Zhou, J., Bruno, N., Wu, M., Larson, P., Chaiken, R., Shakib, D.: SCOPE: parallel databases meet mapreduce. *VLDB J.* **21**(5), 611–636 (2012). <http://dx.doi.org/10.1109/PDIS.1993.253066>
68. Ziane, M., Zait, M., Borla-Salamet, P.: Parallel query processing in DBS3. In: Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS 1993), Issues, Architectures, and Algorithms, San Diego, CA, USA, 20–23 January 1993, pp. 93–102 (1993). <http://dx.doi.org/10.1109/PDIS.1993.253066>

Beyond Databases, Architectures and Structures.
Advanced Technologies for Data Mining and Knowledge
Discovery

12th International Conference, BDAS 2016, Ustroń,
Poland, May 31 - June 3, 2016, Proceedings

Kozielski, S.; Mrozek, D.; Kasprowski, P.;
Małysiak-Mrozek, B.; Kostrzewa, D. (Eds.)

2016, XVII, 738 p. 250 illus., Softcover

ISBN: 978-3-319-34098-2