Big Data Analytics Using SQL: Quo Vadis?

K. T. Sridhar

Big Data Analytics using SQL: Quo Vadis?

K.T.Sridhar

XtremeData Technologies/XtremeData, Inc.
Bangalore, India/Schaumburg, USA
sridhar@xtremedata.com

Abstract. Big Data processing and analytics are dominated by tools other than SQL based relational databases, which have lost their *numero uno* status. In a world deluged by data, the general perception is that SQL databases play a marginal role even for analyzing structured Big Data despite their inherent strengths in processing such data. Focusing on the most important aspect of Big Data processing, namely analytics for data mining, we examine the validity of this perception through a study of competing technologies, published results on SQL implementations of data mining algorithms, the impact of cloud platforms and the raging debate on SQL vs NoSQL vs NewSQL. Contrary to the general belief, it appears that SQL databases in their parallel, columnar deployments on cloud with UDF support do solve some, if not all, Big Data problems and are not likely to become dinosaurs in Big Data era.

Key words: data mining, Big Data, parallel DBMS, SQL, NoSQL

1 Introduction

There is a churn in the data processing world leading to a metamorphosis of database technology as understood in latter years of 20th century and early 21st century. The deluge of data in a variety of forms, from a connected world driven by internet and mobile technologies, has ushered in Big Data [1], [2] and new paradigms of query processing that appear to be beyond the realm of relational model and SQL, the *lingua franca* of database systems. If the data wave from an internet used by humans dislodged DBMSs from *numero uno* status, how would they fare with *data tsunamis* likely to arise from sensor driven Internet of Things and its use in oncoming Industry 4.0 [3], the 4th industrial revolution?

Big Data, with its goal of deriving value through analytics for informed decision making and its envisaged role in Industry 4.0, brings to fore the question of *quo vadis (whither going)?* on relational, SQL databases. Are they relevant anymore for processing Big Data using techniques [4], [5] of data mining? If yes, on what type of data and at what scale? If no, what are the alternatives?

To understand and answer these questions, this paper presents a state-of-the-art survey of SQL databases and contending technologies for Big Data processing, focusing more closely on Big Data analytics and its solutions realized within a relational database emphasizing algorithms, SQL techniques, platforms

and products. Unlike other Big Data analytics studies [6], [7], which adopt only a NoSQL perspective, we examine the role of relational SQL for such analytics.

This paper is organized as follows. Section 2 summarizes evolution and status of contending technologies: SQL, NoSQL and NewSQL. Section 3 addresses issues in Big Data analytics implementation within a SQL relational database, presenting results obtained until now. Section 4 touches upon related topic of platforms and products, in context of cloud. Section 5 summarizes criticisms and limitations of competing Big Data technologies; section 6 concludes the paper.

2 Data Processing Systems

Since the late seventies of 20th century, relational model and products based on it using row stores have dominated data processing applications. The rise of the internet, Big Data, IoT and novel applications have questioned this vice-like grip and given birth to a host of newer data processing technologies. The contending technologies, SQL databases, NoSQL systems and the most recent entrant NewSQL systems are discussed in subsections 2.1 to 2.3.

2.1 SQL Databases

SQL DBMSs targeted business data processing for enterprise systems and were very successful in providing OLTP solutions for ERP, SCM, CRM, banking, etc. ERP, the back bone of EIS has been somewhat immune [8] to Big Data, and advances in it, but is likely to be shaken up by Industry 4.0.

Applications used a row store with SQL and relational model supported by indexes, ACID (*atomicity, consistency, isolation and durability*) transactions and cost-based query planners. Based on business intelligence requirements, SQL DBMSs evolved to tackle analytic query processing of data warehouses optimized for dimensional modeling. For performance gains, they adopted parallel programming techniques to run on a shared nothing cluster of nodes as MPP SQL systems partitioning data across nodes. SQL appliances were next, led by Netezza: MPP row store with custom FPGA hardware for query processing.

Despite significant progress in SQL row store DBMSs, over a decade ago, 2014 Turing Award winner, Stonebraker argued [9] that requirements and characteristics of data centric systems vary widely and the then prevalent architecture of databases as all-encompassing, monolithic, "*one size fits all*" systems was no longer relevant or applicable. They categorize extant DBMSs as "*outbound*" systems that must write before processing, and illustrate their unsuitability for

- low latency systems for algorithmic trading, essentially "*inbound*" systems
- data warehousing and OLAP, better served by column stores
- scientific databases that require native support for arrays
- text engines: custom solutions for web (inbound), medical/legal/library data
- semi-structured data such as XML, JSON, etc., common in Web 2.0
- IoT sensor network processing systems, akin to low latency systems

Advocating use of domain specific DB engines, they show performance advantage for such engines over row databases for the first four applications in stream processing, data warehousing, scientific databases and text management. Authors of [9] conclude with the prescient observation "*'one size fits all' theme is unlikely to successfully continue under these circumstances*".

Today, we have a variety of special purpose data processing systems that neither use relational model nor SQL but adopt newer programming paradigms. In data warehousing market, SQL databases changed their underlying store model to columnar and continue to retain their OLAP market share.

Based on an eighties proposal for database page storage by vertical partitioning [10] of columns, research prototypes MonetDB [11], [12] and C-Store [13] pioneered column oriented databases that were followed up by successful commercial products. Today, almost all industrial DBMSs support some form of column store, simulated or modern, [14] with performance gains over only-row counterparts due to IO reduction, compression, late materialization, etc.

2.2 NoSQL Systems

The difficulties in scaling up SQL database systems for on-line, web scale processing, and in tune with the thinking against one-size-fits-all, NoSQL [1], [15] systems were born. They were built on non-relational models abandoning ACID conformance of databases. The underlying models of NoSQL (*Not only SQL*; not *No to SQL*) systems include key/value, documents, columnar, graphs and streams. For the type of applications targeted, NoSQL proponents believe that

- the rigid relational schema was inflexible, particularly for unstructured data
- guaranteeing ACID properties of transactions reduced performance
- emphasis on high availability is paramount
- horizontal scalability is preferable to expensive vertical scalability of DBMS
- non-procedural SQL was not the most suitable programming language

The most significant impetus for NoSQL systems came from CAP theorem [16] that states an impossibility result for trade-offs in implementing distributed systems: a network shared-data system can have only two of three desirable properties *consistency* (C), high *availability* (A) and *partition* (P) tolerance. The CAP theorem led to an alternative view of transactions favoring "*availability, graceful degradation and performance*" [16] over consistency: BASE standing for *basically available* (BA), *soft-state* (S) and *eventual consistency* (E).

Several NoSQL systems choosing availability and partition tolerance over consistency were built for a diverse set of data processing applications. Table 1 summarizes features of some major NoSQL products. Two other contemporaneous developments contributed to growth and popularization of NoSQL systems:

- **MapReduce** [17], Google's divide-and-conquer software framework for distributed systems: Inspired by LISP like functional programming style, it advocates programming of distributed applications using two functions *Map* and

Table 1. Major NoSQL Products

Product	*Model*	*Transn*	*MapR*	*Program API*	*License*
Bigtable	columnar	C+A	Yes	Java, HBase API	Google SaaS
HBase	columnar	BASE	Yes	Java	open source
Cassandra	columnar	BASE	Yes	CQL	open source
Dynamo	key/value	BASE	by EMR	Java,Python,Ruby,..	Amazon SaaS
MongoDB	document	BASE	Yes	C/C++,JavaScript	open source
Neo4j	graph	ACID	No	Cypher	open source

Reduce that work on key/value pairs with their execution, including failures, handled by the framework.

– **Hadoop** a distributed file system from Apache: An open source system with the goals of performance, availability and scalability, modeled on the proprietary Google File System underlying MapReduce, it made MapReduce style application development popular in the community.

Relieving the programmer of managing a parallel application running in a distributed environment of commodity clusters, with fault tolerance support, was a major step. Hadoop contributed to the meteoric rise and adoption of MapReduce for solving web scale data processing problems, with several NoSQL products incorporating the MapReduce model as shown in Table 1.

2.3 NewSQL Products

Disputing the importance of BASE over ACID, Stonebraker et al [18] investigate reasons for under performance of SQL databases in OLTP applications of Big Data era and cite locking, latching, recovery and buffer pool management as the reasons. Eliminating these bottlenecks in their prototype H-Store database they claim 82x performance gain in TPC-C benchmark compared to row DBMSs.

The next few years heralded the term NewSQL [19] coined to refer to a class of products that preserve relational model, ACID transactions and SQL but offer NoSQL like performance and scalability for OLTP read-write workloads. Elaborating on applications of NewSQL systems [19] characterizes them as "*executing read-write transactions that (1) are short-lived, (2) touch a small subset of data using index lookups and (3) are repetitive*". They also observe that their characterization of NewSQL is in consonance with the more narrow definition of [18]: being lock-free and using a shared nothing distributed architecture.

As of 2016, [19] lists seventeen products as NewSQL systems including SAP HANA, Amazon's Aurora and both H-Store and VoltDB from Stonebraker et al. It is interesting to note that 15 of the 17 products listed in Table 1 of [19] use MVCC [20] for concurrency control found in several row store SQL DBs including open source PostgreSQL, and SQL column stores for analytic workloads.

3 Big Data Analytics

The term Big Data is all around us and became part of 21st century English when Oxford dictionary defined it in 2013. Most authors characterize Big Data [1], [2], [6], [7] through 3Vs (*volume*, *velocity* and *variety*) or more (*veracity*, *variability*, *value*). In the user community, as well as lexicon definition, the stress for Big Data has been on data mining, generally understood as *discovery of models for data* using statistics, machine learning and computer science [4], [5].

Though both data mining and analysis techniques predate Big Data, they are current hot topics due to technology challenges and commercial value gained by enterprises through insights gleaned from data. Technology challenges arise from the general perception of SQL databases being inadequate [1], [2], [7], [15] for Big Data processing, due to their difficulty in dealing with the 3Vs:

- **volume**: horizontal scalability, elastic or not, is an issue for DBMSs; the advocated vertical scalability is too expensive
- **velocity**: consequent to being outbound [9] systems, SQL databases do not perform well on streaming data for real-time analytics
- **variety**: heterogeneous data are anathema to SQL databases that deal well with structured data (e.g. number, boolean, varchar), partially well with semi-structured data (e.g. XML, JSON) but are inadequate with unstructured data (e.g. tweets, text, video, audio)

Does this imply the death knell of relational, SQL databases for Big Data analytics? No; there is a contrarian view that we examine in subsections of this section. Our focus is on high volume structured data and analytics on such data; we do not touch upon other aspects of Big Data processing: data collection, cleansing, loading or privacy. Section 3.1 discusses data mining algorithms, 3.2 addresses the question of mining using SQL and section 3.3 surveys published work on implementing data mining algorithms in relational DBMS using SQL.

3.1 Data Mining Algorithms

Data mining algorithms build models to classify data and the models may be used with unlabeled data for prediction or scoring. At a broad level, the learning techniques used by these algorithms may be classified [4], [5] as (a) *supervised* that uses a training set for correct classification and (b) *unsupervised* that discovers a model without any training set or *a priori* knowledge.

A large number of data mining algorithms addressing a variety of topics, clustering, classification, statistical learning, association mining, link mining, bagging/boosting, dimensionality reduction and regression have been published. The often quoted survey [21] discusses about 30 algorithms for the important unsupervised learning technique of clustering that partitions data into similar groups. Wu et al [22] conducted a survey to identify the top 10 data mining algorithms ranking them based on votes polled and citations. Table 2 summarizes details of top 10 data mining algorithms of Wu et al (three with same rank 7).

Table 2. IEEE KDD Top-10 Data Mining Algorithms

Algorithm	*Mining Topic*	*Year*	*Rank*	*Notes*
(abbrevn: s.⇒Supervised; u.⇒Unsupervised; DT⇒Decision Tree; info⇒information)				
C4.5	s.Classification	1993	1	DT: info entropy; info gain ratio
k-Means	u.Clustering	1967	2	partitioning by similarity; iterative
SVM	s.Stats Learning	1995	3	find best fit hyperplane
Apriori	u.Association	1994	4	find frequent itemsets
EM	u.Stats Learning	2000	5	maximize loglikelihood
PageRank	u.Link Mining	1998	6	network link analysis; graphs mining
AdaBoost	s.Boosting	1997	7	ensemble learning
kNN	s.Classification	1996	7	k neighbors by *nearest* criterion
Naive Bayes	s.Classification	2001	7	non-iterative, Bayesian probability
CART	s.Classification	1984	10	DT: binary recursive; gini index split

Table 2 includes year of publication of algorithm, and a very brief note on nature of algorithm; more details of the algorithms may be found in [22], the original publication references cited therein, and in data mining books [4], [5].

3.2 Why not SQL for Data Mining?

Some observations on Table 2 algorithms rated highly by mining community:

- formulated on or before 2001, most in 20th century when Big Data was unknown; nothing intrinsic to NoSQL or Big Data in analytics techniques.
- based on mathematics or statistics dealing with numbers or categorical data, both of which are essentially structured data.
- mostly iterative in nature, a programming style that is not supported by a declarative language like SQL.

Though structured data processed by mining algorithms is well handled by SQL databases, iterative nature of algorithms has been a stumbling roadblock. DBMS vendors responded by including imperative style programming with SQL; external to DBMS in C/C++, Java, Python, etc. through ODBC/JDBC interfaces; and as internal database objects: stored procedures in PL/SQL type imperative SQL, or user defined functions (UDF) in C/C++, Java, Python, etc. External programs incur data transfer cost, while code in UDF or stored procedures runs in DBMS environment close to data with performance gains.

Adopting parallel techniques through MPP shared nothing systems, for performance gains with high volume data, originated in DBMS world: first such commercial MPP system from TeraData was in 1986 [1], [23]. Evolution of SQL row DBs into column stores [14], targeting OLAP with better performance, has enhanced their suitability for structured data mining applications.

Ordonez investigates suitability of SQL databases [24] for implementing data mining algorithms and concludes that parallel columnar databases with UDFs can solve important Big Data problems. Row database vendors, Oracle, IBM,

Teradata and Microsoft, offer data mining packages tightly coupled to their products modifying internal DBMS code with SQL extensions. As data mining involves development of newer, or modifications to existing, algorithms both needing source code access, such packages are not very popular. Implementations of some of the top 10 algorithms and others exist in user developed SQL.

3.3 Data Mining Algorithms in SQL

k-Means Clustering: The importance of *sufficient statistics*, smaller in size than data, for decoupling mining algorithms from data was highlighted [25], and used in [26] to scale k-Means for large databases. Size in thousands of [26] was scaled to millions [27] on a 4-nodes, parallel row DBMS of TeraData with standard and optimized versions, evaluating performance varying dimensions, clusters and data size. Standard version runs a bunch of SQL statements computing Euclidean distance between points and cluster centroids iteratively until termination. Optimized version improves performance with SQL tricks to reduce joins/groupings, UDFs and uses sufficient statistics of [26], which is defined as a triplet for data of d dimensions and size n to be partitioned into k clusters; sufficient statistics does not eliminate multiple scans due to iterations.

$$N_j = |X_j| \quad \text{cluster } j \text{ size; vector (k x 1) with } n = \sum_{j=1}^{k} N_j \tag{1}$$

$$L_j = \sum_{i=1}^{N_j} x_i \quad \text{cluster } j \text{ sum; matrix (d x k)} \tag{2}$$

$$Q_j = \sum_{i=1}^{N_j} x_i x_i^T \quad \text{cluster } j \text{ quadratic sum; matrix (d x k)} \tag{3}$$

Using equations (1) to (3), cluster weight W_j, cluster centroid C_j and cluster variance R_j for iteration termination are computed [27] as below:

$$W_j = N_j/n \qquad C_j = L_j/N_j \qquad R_j = Q_j/N_j - L_j L_j^T/N_j^2 \tag{4}$$

Apriori Association Mining: Association mining algorithm Apriori for market basket problems was programmed in SQL with UDFs [28] on DB2 system. Several alternatives for implementing [29] Apriori in DB2 SQL have also been explored: plain SQL using joins and subqueries, cache-mine, stored procedures and UDFs. Both papers use a non-parallel row DBMS.

Expectation Maximization: EM maximizes loglikelihood and for each point x_i finds its probability for cluster j; version given in SQL [30] uses Mahalanobis distance on d dimensions and k clusters with C_j being the mean vector of size d and R_j the covariance matrix (d x d) with zeros for off-diagonal elements.

$$\delta_{ij} = (x_i - C_j)^T R_j^{-1} (x_i - C_j) \quad \text{Mahalanobis distance of } x_i \text{ to cluster } j \tag{5}$$

$$P(x_i) = \frac{e^{-\delta_{ij}/2}}{\sqrt{(2\pi)^d |R_j|}} \quad \text{probability of } x_i \text{ for cluster } j \tag{6}$$

Sufficient statistics to improve performance of EM in SQL is suggested in [31].

PageRank: The algorithm that made Google the leader in web search was implemented [32] in SQL with query optimization on 4 parallel nodes of column store Vertica on publicly available real-life data sets (Twitter, Livejournal and YouTube) of large sizes: graphs varying from 81k to 41.6 million nodes, 1.7 million to 1.4 billion edges. They report competitive performance for SQL with NoSQL products GraphLab and Giraph with less system resource utilization for memory and read I/O; extending the comparison of PageRank to mixed graph and relational analysis problems SQL Vertica outperforms Giraph by 17x.

Naive Bayes: Assumes Gaussian classes and independence across dimensions to compute [33], [34] sufficient statistics, N_g, L_{gh} and Q_{gh}, for g classes of training set across d dimensions ($h \in 1..d$) like in (1) to (3), and finds class prior π_g, class means C_{gh} and class variance R_{gh} to classify new data by probability $p(x_i)$.

$$\pi_g = N_g/n \qquad C_{gh} = L_{gh}/N_g \qquad R_{gh} = Q_{gh}/N_g - L_{gh}L_{gh}^T/N_g^2 \quad \text{h} \in 1..\text{d} \tag{7}$$

$$p(x_{ih}|g)_{h\in 1..d} = \frac{e^{-(x_{ih}-C_{gh})^2/2R_{gh}}}{\sqrt{2\pi R_{gh}}} \quad \text{probability of } x_i, \text{ dimension } h \text{ class } g \tag{8}$$

$$p(x_i|g) = \Pi p(x_{ih}|g)_{h\in 1..d} \quad \text{joint probability of } x_i, \text{ all dimensions class } g \tag{9}$$

Final scoring is to class c with maximum probability $p(x_i) = max(p(x_i|g))$. SQL and MapReduce versions are compared in [34] with better performance for SQL.

kNN: Points in a multidimensional space are mapped [35] to one dimension defining *z-value* of a point by interleaving binary representation of its coordinates from MSB to LSB. For a point $p_i = (x_i, y_i)$ in 2-d space, its z-value $z_p(x_i, y_i)$ is:

$$z_p(x_i, y_i) = bit_n(x_i) \mid bit_n(y_i) \mid bit_{n-1}(x_i) \mid bit_{n-1}(y_i) \mid ... \mid bit_0(x_i) \mid bit_0(y_i) \tag{10}$$

where $bit_k(v)$ is the kth bit of value v. The *Z-order* of points on z_p is a SQL range query, generally preserving spatial locality. But, for a theoretical guarantee they use random shifts to define a γ-*neighborhood* and propose algorithms for approximate/exact kNN, distance based θ-join and kNN-joins; analyze complexity, implement in SQL to compare with others (iDistance & Medrank in SQL). z_p is extended to real values, higher dimensions and queries with ad-hoc conditions.

Decision Trees: Two of Top-10, C4.5 and CART, are decision trees, which are greedy, recursive, memory/time intensive algorithms, but intuitive and used widely. Using sufficient statistics (counts: *CC tables*) for splitting, SQL and C++ middleware [36] shows scalable full tree construction with C4.5/CART like entropy measure for selection. Primitives in SQL based on CC tables for C4.5, CART, etc., are given in [37]; C4.5 is implemented [38] as Oracle PL/SQL stored procedure, and decision tree constructed [39] from SQL data cubes.

Graphs Mining: Graph analytics applications, like PageRank, use mining techniques to process graphs. Study in [32] also includes two other graph algorithms in SQL: single source shortest path (SSSP) and HCC to find connected components of a graph. In performance comparison of mixed graph and relational analysis for SSSP on Twitter data, Vertica SQL outperforms Giraph by 4x.

Others: Sufficient statistics is used to build other statistical models in SQL [31], [40]: linear regression/correlation for n variables; dimensionality reduction for preprocessing mining data by principal component analysis (PCA). Regression over 2 variables, multidimensional analysis (cube, rollup, grouping set) and windowing analysis (partitions, order, frames) are part of standard SQL.

4 Platforms and Products

The advent of cloud computing through pay-by-use public services democratizes grid/cluster computing: facilitates scale out, parallel applications and data storage for Big Data processing. Through a browser based GUI any data scientist with web access may harness the power of parallel computing and large stores without recourse to high capital investment or a team of system specialists.

Multiple vendors offer managed IaaS (Infrastructure as a Service) environments with choice of configurations to suit budget and application requirements: Amazon AWS, Microsoft Azure, CenturyLink, Samsung, INAP, Alibaba, etc. Several MPP SQL analytics products are available on public clouds along with competing NoSQL products. Table 3 lists some leading relational SaaS products for Big Data analytics on cloud; all products listed support horizontal scaling.

Table 3. Cloud SaaS: MPP SQL Analytics Products

Product	*DB Store*	*Cloud*	*Store Type*	*Prem*	*UDF*	*Vendor*
Redshift	column	AWS	attached	No	Yes	Amazon
SQL DW	column	Azure	blob+attach	No	Yes	Microsoft
Vertica	column	AWS,Az	attached	Yes	Yes	HP/MicroFocus
dbX	column+row	agnostic	attach/NW	Yes	Yes	XtremeData
Greenplum	row+apnd col	AWS,Az	attach+S3	Yes	Yes	EMC Pivotal
Snowflake	column	AWS	S3+attach	No	No	Snowflake

Table 3 categorizes the listed cloud products on high level criteria for Big Data analytics rather than a detailed evaluation of SQL features support:

1. **DB Store**: Column store has been shown to have better performance for analytics than row store [14]; five of the six products support a native column store with compression; Greenplum is a native row store with restricted, append-only support for column store; dbX is a hybrid store with no serious use-case restrictions: modern column store with compression and row store.

2. **Cloud**: A product available on multiple cloud platforms offers mobility across IaaS platforms and imposes no restriction on the application. Only dbX is cloud agnostic: available on AWS, Azure and other smaller public clouds; Amazon and Microsoft products are tied down to respective vendor clouds; Snowflake is only on AWS; other two on both AWS and Azure.
3. **Store Type**: Type of cloud storage impacts cost: irrespective of usage, products that run only on attached store must be 24x7 as data is lost on shutdown of compute instances. With network storage (AWS EBS or Azure premium IO), compute and storage are decoupled: shutting down compute instances preserves store for later use. dbX offers services on both store types; Redshift is preconfigured on attached; Vertica/Greenplum use or recommend attached. SQL Datawarehouse and Snowflake, targeting elastic scale-out, use a low cost store with poorer performance (Azure blob or AWS S3) as primary store caching retrieved data on attached store with attendant performance overheads; no shutdown data loss. Greenplum also uses external files on S3.
4. **On Premise**: On premise deployments are sought by users who may not want to store their data on public clouds for security/privacy reasons, or enterprise users who wish to build AaaS (Analytics as a Service) private clouds. Vertica and Greenplum are available as SQL appliances bundled on vendor hardware; dbX may be deployed on commodity clusters and even on other virtualized environments such as VMware; other DBs only on cloud.
5. **UDF**: Table 3 lists UDF support as sections 3.2 and 3.3 highlight importance of UDFs for data mining algorithms in SQL. Snowflake is the only product without UDF support; others offer it in different languages: PL/SQL type stored procedures (SQL DW, dbX, Greenplum), C/C++ (Vertica, dbX, Greenplum), Python (Redshift, dbX, Greenplum).

Some products also offer API interfaces to external open source data mining packages such as R and MADLIB, an approach similar to vendor specific mining products. Both Vertica and Greenplum offer customized version of R compatible to their products; additionally, Greenplum SQL may also be used with MADLIB.

5 Discussion

MapReduce, key-enabler of most NoSQL systems, is compared [41] with two SQL parallel databases (Vertica and row DBMS) on clusters of 100 nodes with large, synthetic web crawler type data. The benchmark used 5 tasks; grep task as in [17] and 4 DBMS analysis tasks: selection, aggregation, join and UDF aggregation. Both DBMSs outperformed MapReduce on all 5 tasks; average values: row store (3.2x); column store (7.4x). Data load was easier and faster on MapReduce.

Based on results of [41], criticisms by Stonebraker et al [23], [41] of MapReduce include (1) repetitive record parsing as data is stored in text form (2) lack of compression advantage: slowing down with block/record level compression (3) pull model of Reduce for data exchange with Map (4) absence of plan optimization (5) lack of high level interfaces and developer eco-system (6) schema

less world. They surmise that MapReduce "*is more like an extract-transform-load (ETL) system*" [23] and hence complementary, rather than competitive, to DBMSs. Basing their comparison on a technical evaluation, they perhaps understate the importance of MapReduce framework, Hadoop or GFS, which simplifies distributed application development, by managing everything including failures.

Mohan, inventor of ARIES recovery fundamental to ACID transactions, criticizes [42] NoSQL for oversimplifying complex issues with ad-hoc solutions, being expedient not rigorous, missing interactive query support and ignoring history.

Challenges for MapReduce in Big Data include [43]: (1) data storage issues without schemas (2) coding iterative analytics algorithms in MapReduce (3) performance overheads with correlated data for predictive modeling (4) harder interactive data exploration without high level interfaces like SQL (5) same issues as SQL DBs in low latency applications (6) lack of security and privacy features and its legal impact for proposed privacy regulations.

Examples of on-going work to mitigate the challenges: integration with SQL DBMSs such as Oracle and Greenplum, Spark and HaLoop to deal with iterative algorithms, Storm for low latency applications, SQL like Hive for interactive analytics, Mahout for data mining, etc. It appears that NoSQL systems are evolving like DBMSs into vertically segmented engines to address Big Data.

Revisiting CAP theorem, *raison d'êtere* of NoSQL systems, its proposer Brewer considers "*'2 of 3' formulation was misleading because it tended to oversimplify the tension among properties*" [44], and proposes alternatives to deal with partition tolerance. Stressing consistency-latency trade-off [45] suggests that CAP's 2-of-3 limitation, applicable only in the context of failures, has been misinterpreted to build limited systems. NewSQL prefers ACID to BASE.

Parallel database theory questions characterization of Big Data through 3Vs and suggests alternative dimensions: communication, iteration and failure [46]. Despite high scale-out of a few NoSQL products, the comprehensive survey [15] discusses issues and limitations in horizontal scaling of other NoSQL products.

Social factors too contribute to perception of SQL not being suitable for Big Data analytics: (1) unlike parallel DBMSs, almost all NoSQL products are open source with no cost (2) limited mathematical exposure of programmers hampers translation of complex and iterative algorithms into declarative SQL.

The 18th KDnuggets poll of 2017 (www.kdnuggets.com) lists percentages of 2900 voters for language use: Python (52.5), R (52.1), SQL (34.9), Java (13.8), C/C++ (6.3); SQL holds one third share! Big Data users [15] of MySQL: Facebook for social graph data, Wikipedia on MariaDB, open source fork of MySQL.

6 Conclusion

It is an axiomatic fact that Big Data and its analysis are decision making drivers in a 21st century world driven by web, mobile and IoT technologies. To understand why 20th century *numero uno* tool for data processing, SQL rdbms, has lost its primacy we have briefly summarized its shortcomings that led to birth of competing technologies, NoSQL and NewSQL, and traced their evolution.

Focusing on the important aspect of analytics in Big Data processing, we examined suitability of SQL relational databases for data mining, and presented published work on data mining algorithms in SQL; majority of the top ten data mining algorithms and a few others in SQL solve Big Data analytics problems on parallel, columnar DBMS with UDFs. Cloud deployments of such products makes them more accessible, at lower cost and easy scale-out, even elastic.

Comparative discussion in section 5 shows that no technology is fully ready for all challenges of Big Data, more so for IoT and Industry 4.0 that could possibly use blockchain based P2P networks for devices [3], [47] along with multiple options on cloud: NewSQL stream DBMSs, NoSQL Storm, or even parallel, columnar MPP DBMSs fed by distributed streaming platform Apache Kafka.

We observe a convergence of technologies, and note that relational model and SQL are unlikely to disappear, a view endorsed by [19]: "*all of the key systems in these groups will support some form of relational model and SQL*".

References

1. Chen, M., Mao, S., Liu, Y.: *Big Data: A Survey.* In: Mobile Network Applications, vol. 19, pp. 171-209, Springer Science (2014)
2. Khan, N., et al: *Big Data: Survey, Technologies, Opportunities, and Challenges.* In: The Scientific World Journal, vol. 2014, Article ID 712826, 18 pages, (2014)
3. English, M., Auer, S., Domingue, J.: *Block Chain Technologies & the Semantic Web: A Framework for Symbiotic Development.* In: Lehmann, J., et al (eds.), Computer Science Conf. for University of Bonn Students, pp. 47-61, (2016)
4. Han, J., Kamber, M.: *Data Mining: Concepts and Techniques.* Morgan Koffman Publishers, Elsevier, 3rd Edition, (2012)
5. Leskovec, J., Rajaraman, A., Ullman, J.: *Mining of Massive Datasets.* Cambridge University Press, 2nd Edition, (2014)
6. Elgendy, N., Elragal, A.: *Big Data Analytics: A Literature Review Paper.* In: Perner, P. (ed.), ICDM 2014, LNAI 8557, pp. 214-227, Springer (2014)
7. Che, D., Safran, M., Peng, Z.: *From Big Data to Big Data Mining: Challenges, Issues and Opportunities.* In: Hong, B., et al (eds.), DASFAA Workshops 2013, LNCS 7827, pp. 1-15, Springer (2013)
8. Elragal, A.: *ERP and Big Data: The Inept Couple.* In: CENTERIS 2014, Procedia Technology, vol. 16, pp. 242-249, Elsevier (2014)
9. Stonebraker, M., Cetintemel, U.: *"One Size Fits All": An Idea whose Time has Come and Gone.* In: ICDE'05, pp. 2-11, IEEE (2005)
10. Copeland, G.P., Khoshafian, S.N.: *A Decomposition Storage Model.* In: SIGMOD'85, pp. 268-279, ACM (1985)
11. Boncz, P., Kersten, M.L., Manegold, S.: *Breaking the Memory Wall in MonetDB.* In: Communications of the ACM, vol. 51, no. 12, pp. 77-85, ACM (2008)
12. Idreos, S., et al: *MonetDB: Two Decades of Research in Column-oriented Database Architectures.* In: IEEE Data Engg. Bulletin, vol. 35, no. 1, pp. 40-45, IEEE (2012)
13. Stonebraker, M., et al: *CStore: A Column Oriented DBMS.* In: VLDB'05, pp. 553-564, (2005)
14. Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., Madden, S.: *The Design and Implementation of Modern Column Oriented Database Systems.* In: Foundations and Trends in Database, vol. 5, no. 3, pp. 197-280, (2012)

15. Strauch, C.: *NoSQL Databases.* In: Selected Topics on Software-Technology Ultra-Large Scale Sites, `http://www.christof-strauch.de/nosqldbs.pdf`, pp. 1-149, Stuttgart Media University (2011)
16. Brewer, E.: *Towards Robust Distributed Systems.* In: 19th Symposium on Principles of Distributed Computing, PODC'00, pp. 7-10, ACM (2000)
17. Dean, J., Ghemawat, S.: *MapReduce: Simplified Data Processing on Large Clusters.* In: OSDI'04, pp. 137-149, USENIX (2004)
18. Stonebraker, M., et al: *The End of an Architectural Era: (Its time for a Complete Rewrite).* In: VLDB'07, pp. 1150-1160, (2007)
19. Pavlo, A., Aslett, M.: *What's Really New with NewSQL?* In: SIGMOD Record, vol. 45, no. 2, ACM (2016)
20. Bernstein, P.A., Goodman, N.: *Multiversion Concurrency Control: Theory and Algorithms.* In: ACM Trans. on Database Systems, vol. 8, pp. 465-483, ACM (1983)
21. Xu, R., Wunsch, D.: *Survey of Clustering Algorithms.* In: IEEE Trans. on Neural Networks, vol. 16, no. 3, pp. 645-678, IEEE (2005)
22. Wu, X., et al: *Top 10 Algorithms in Data Mining.* In: Knowledge Information Systems, vol. 14, pp. 1-37, Springer (2008)
23. StoneBraker, M., et al: *MapReduce and Parallel DBMSs: Friends or Foes?* In: Communications of the ACM, vol. 53, no. 1, ACM (2010)
24. Ordonez, C.: *Can We Analyze Big Data inside a DBMS?* In: Proc. of 16th Intl. Workshop on Data Warehousing and OLAP, DOLAP'13, pp. 85-92, ACM (2013)
25. Graefe, G., Fayyad, U., Chaudhuri, S.: *On the Efficient Gathering of Sufficient Statistics from Large SQL Databases.* In: KDD'98, pp. 100-105, AAAI (1998)
26. Bradley, P.S., Fayyad, U., Reina, C.: *Scaling Clustering Algorithms to Large Databases.* In: KDD'98, pp. 9-15, AAAI (1998)
27. Ordonez, C.: *Programming the K-means Clustering Algorithm in SQL.* In: KDD'04, pp. 823-828, AAAI (2004)
28. Agrawal, R., Shim, K.: *Developing Tightly Coupled Data Mining Applications on a Relational Database System.* In: KDD'96, pp. 287-290, AAAI (1996)
29. Sarawagi, S., Thomas, S., Agrawal, R.: *Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications.* In: SIGMOD'98, pp. 343-354, ACM (1998)
30. Ordonez, C., Cereghini, P.: *Fast Clustering in SQL using the EM Algorithm.* In: SIGMOD'00, pp. 559-570, ACM (2000)
31. Ordonez, C.: *Statistical Model Computation with UDFs.* In: IEEE Trans. on Knowledge and Engineering, vol. 22, no. 12, pp. 1752-1765, IEEE (2010)
32. Jindal, A., Madden, S., Castellanos, M., Hsu, M.: *Graph Analytics using the Vertica Relational Database.* In: IEEE Conf. on Big Data, pp. 1191-1200, IEEE (2015)
33. Ordonez, C., Pitchaimalai, S.K.: *Bayesian Classifiers Programmed in SQL.* In: IEEE Trans. on Knowledge and Engg., vol. 22, no. 1, pp. 139-144, IEEE (2010)
34. Pitchaimalai, S.K., et al: *Comparing SQL & MapReduce to Compute Naive Bayes in a Single Table Scan.* In: CloudDB'10, pp. 9-16, ACM (2010)
35. Yao, B., Li., F., Kumar, P.: *K Nearest Neighbor Queries and KNN-Joins in Large Relational Databases (Almost) for Free.* In: ICDE'10, pp. 4-15, IEEE (2010)
36. Chaudhari, S., Fayyad, U., Bernhardt, J.: *Scalable Classification over SQL Databases.* In: 15th ICDE'99, pp. 470-489, IEEE (1999)
37. Sattler, K-U., Dunemann, O.: *SQL Database Primitives for Decision Tree Classifiers.* In: CIKM'01, pp. 379-386, ACM (2001)
38. Taniar, D., D'Cruz, G., Rahayu, J.W.: *Implementation of Classification Rules using Oracle PL/SQL.* In: FSKD 2002, pp. 509-513, (2002)

39. Fu, L.: *Construction of Decision Trees Using Data Cubes.* In: 7th ICEIS, pp. 119-126, (2005)
40. Ordonez, C., Garcia-Alvardo, C.: *A Data Mining System Based on SQL Queries and UDFs for Relational Databases.* In: CIKM'11, pp. 2521-2524, ACM (2011)
41. Pavlo, A., et al : *A Comparison of Approaches to Large Scale Data Analysis.* In: SIGMOD'09, pp. 165-178, ACM (2009)
42. Mohan, C.: *History Repeats Itself: Sensible and NonsensSQL of the NoSQL Hoopla.* In: Proc. of EDBT/ICDT'13, pp. 11-16, (2013)
43. Grolinger, K., et al: *Challenges for MapReduce in Big Data.* In: SERVICES'14, pp. 182-189, IEEE (2014)
44. Brewer, E.: *CAP Twelve Years Later: How the "Rules" Have Changed.* In: IEEE Computer, vol. 45, no. 2, pp. 23-29, IEEE (2012)
45. Abadi, D.: *Consistency Tradeoffs in Modern Distributed Database System Design.* In: IEEE Computer, vol. 45, no. 2, pp. 37-42, IEEE (2012)
46. Suciu, D.: *Big Data Begets Big Database Theory.* In: Olteanau, B., Gottlob, G., Schallart, C. (eds.), BNCOD 2013, LNCS 7968, pp. 1-5, Springer (2013)
47. Brody, P., Pureswaran, V.: *Device Democracy: Saving the Future of the Internet of Things.* In: IBM Institute for Business Value Tech. Rep., 25 pages, IBM (2014)