SIGMOD Officers, Committees, and Awardees

Chair	Vice-Chair	Secretary/Treasurer
Donald Kossmann Systems Group ETH Zürich Cab F 73 8092 Zuerich SWITZERLAND +41 44 632 29 40 <donaldk AT inf.ethz.ch>	Anastasia Ailamaki School of Computer and Communication Sciences, EPFL EPFL/IC/IIF/DIAS Station 14, CH-1015 Lausanne SWITZERLAND +41 21 693 75 64 <natassa AT epfl.ch>	Magdalena Balazinska Computer Science & Engineering University of Washington Box 352350 Seattle, WA USA +1 206-616-1069 <magda AT cs.washington.edu>

SIGMOD Executive Committee:
Donald Kossmann (Chair), Anastasia Ailamaki (Vice-Chair), Magdalena Balazinska, K. Selçuk Candan, Yanlei Diao, Curtis Dyreson, Yannis Ioannidis, Christian Jensen, and Jan Van den Bussche.

Advisory Board:
Yannis Ioannidis (Chair), Rakesh Agrawal, Phil Bernstein, Stefano Ceri, Surajit Chaudhuri, AnHai Doan, Michael Franklin, Laura Haas, Joe Hellerstein, Stratos Idreos, Tim Kraska, Renee Miller, Chris Olsten, Beng Chin Ooi, Tamer Özsu, Sunita Sarawagi, Timos Sellis, Gerhard Weikum, John Wilkes

SIGMOD Information Director:
Curtis Dyreson, Utah State University <curtis.dyreson AT usu.edu>

Associate Information Directors:
Huiping Cao, Manfred Jeusfeld, Asterios Katsifodimos, Georgia Koutrika, Wim Martens

SIGMOD Record Editor-in-Chief:
Yanlei Diao, University of Massachusetts Amherst <yanlei AT cs.umass.edu>

SIGMOD Record Associate Editors:
Vanessa Braganholo, Marco Brambilla, Chee Yong Chan, Rada Chirkova, Zachary Ives, Anastasios Kementsietsidis, Jeffrey Naughton, Frank Neven, Olga Papaemmanouil, Aditya Parameswaran, Alkis Simitsis, Wang-Chiew Tan, Nesime Tatbul, Marianne Winslett, and Jun Yang

SIGMOD Conference Coordinator:
K. Selçuk Candan, Arizona State University

PODS Executive Committee:
Jan Van den Bussche (Chair), Tova Milo, Diego Calvanse, Wang-Chiew Tan, Rick Hull, Floris Geerts

Sister Society Liaisons:
Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE).

Awards Committee:
Maurizio Lenzerini (Chair), Elisa Bertino, Surajit Chadhuri, Martin Kersten, Jennifer Widom

Jim Gray Doctoral Dissertation Award Committee:
Ashraf Aboulnaga (co-Chair), Juliana Freire (co-Chair), Kian-Lee Tan, Andy Pavlo, Aditya Parameswaran, Ioana Manolescu, Lucian Popa, Chris Jermaine, Renée Miller

SIGMOD Systems Award Committee:
David DeWitt (Chair), Make Cafarella, Mike Carey, Yanlei Diao, Mike Stonebraker

SIGMOD Officers, Committees, and Awardees (continued)

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Formerly known as the "SIGMOD Innovations Award", it now honors Dr. E. F. (Ted) Codd (1923 - 2003) who invented the relational data model and was responsible for the significant development of the database field as a scientific discipline. Recipients of the award are the following:

Michael Stonebraker (1992)
David DeWitt (1995)
Serge Abiteboul (1998)
Rudolf Bayer (2001)
Ronald Fagin (2004)
Jennifer Widom (2007)
Umeshwar Dayal (2010)
Stefano Ceri (2013)
Gerhard Weikum (2016)

Jim Gray (1993)
C. Mohan (1996)
Hector Garcia-Molina (1999)
Patricia Selinger (2002)
Michael Carey (2005)
Moshe Y. Vardi (2008)
Surajit Chaudhuri (2011)
Martin Kersten (2014)

Philip Bernstein (1994)
David Maier (1997)
Rakesh Agrawal (2000)
Don Chamberlin (2003)
Jeffrey D. Ullman (2006)
Masaru Kitsuregawa (2009)
Bruce Lindsay (2012)
Laura Haas (2015)

SIGMOD Systems Award

For technical contributions that have had significant impact on the theory or practice of large-scale data management systems.

Michael Stonebraker and and Lawrence Rowe (2015)

Martin Kersten (2016)

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)
Jeffrey Ullman (1996)
Raghu Ramakrishnan (1999)
Daniel Rosenkrantz (2001)
Surajit Chaudhuri (2004)
Hans-Jörg Schek (2007)
David Lomet (2010)
H.V. Jagadish (2013)
Samuel Madden (2016)

Gio Wiederhold (1995)
Avi Silberschatz (1997)
Michael Carey (2000)
Richard Snodgrass (2002)
Hongjun Lu (2005)
Klaus R. Dittrich (2008)
Gerhard Weikum (2011)
Kyu-Young Whang (2014)

Yahiko Kambayashi (1995)
Won Kim (1998)
Laura Haas (2000)
Michael Ley (2003)
Tamer Özsu (2006)
Beng Chin Ooi (2009)
Marianne Winslett (2012)
Curtis Dyreson (2015)

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau. *Honorable Mentions*: Marcelo Arenas and Yanlei Diao.
- **2007** *Winner*: Boon Thau Loo. *Honorable Mentions*: Xifeng Yan and Martin Theobald.
- **2008** *Winner*: Ariel Fuxman. *Honorable Mentions*: Cong Yu and Nilesh Dalvi.
- **2009** *Winner*: Daniel Abadi. *Honorable Mentions*: Bee-Chung Chen and Ashwin Machanavajjhala.
- **2010** *Winner:* Christopher Ré. *Honorable Mentions*: Soumyadeb Mitra and Fabian Suchanek.
- **2011** *Winner*: Stratos Idreos. *Honorable Mentions*: Todd Green and Karl Schnaitterz.
- **2012** *Winner*: Ryan Johnson. *Honorable Mention*: Bogdan Alexe.
- **2013** *Winner*: Sudipto Das, *Honorable Mention*: Herodotos Herodotou and Wenchao Zhou.
- **2014** *Winners*: Aditya Parameswaran and Andy Pavlo.
- **2015** *Winner*: Alexander Thomson. *Honorable Mentions*: Marina Drosou and Karthik Ramachandra
- **2016** *Winner*: Paris Koutris. *Honorable Mentions*: Pinar Tozun and Alvin Cheung

A complete list of all SIGMOD Awards is available at: **http://sigmod.org/sigmod-awards/**

[Last updated : June 30, 2016]

Editor's Notes

Welcome to the June 2016 issue of the ACM SIGMOD Record!

First of all, I would like to welcome Frank Neven as the new associate editor for the Database Principles Column. This column continues with invitation-based contributions, with a goal to bring the latest results from the Database theory community to the SIGMOD Record readers.

This issue opens with a Database Principles article by Olteanu and Schleich on Factorized Databases. This article takes a fresh look at the problem of computing and representing results to relational queries, in particular, in a factorized manner. For various classes of queries, this work quantifies the succinctness gap between factorized and standard tabular representations for results of conjunctive queries, and surveys algorithms for computing factorized representations of query results. It further discusses the queries with aggregates and order-by clauses and their application to learning regression models over factorized databases, with very promising initial results.

The Vision Articles Column features the article, "Database Meets Deep Learning: Challenges and Opportunities". Motivated by the success of deep learning in applications such as computer vision and natural language processing, this article investigates the application of database techniques for optimizing deep learning systems. It further outlines the research problems in databases where deep learning techniques may help to improve performance. This is a timely article that aims to establish the connection between deep learning and database research, and to foster a synergy between the two disciplines to advance data-driven applications.

The Surveys Column features two articles. The first article, "A Time Machine for Information: Looking Back to Look Forward," is an invited article based on the tutorial by Dong, Kementsietsidis, and Tan at VLDB 2015. The article presents a vision of a time machine for information, in particular, Web information, which will help people "look back" so as to "look forward". With a focus on information extraction and temporal analysis, the article reviews the key ideas on three components (extraction, linking, and cleaning) that are central to the envisioned time machine, and further points out future research directions. The second article, by Wang, Song, and Chen, surveys recent research on accessing dataspaces. A dataspace system processes heterogeneous data sources. Without full control on its data, it gradually integrates data as necessary. Query processing is characterized by best-effort approximate answers where the correct semantic mappings have not been established. In this context, the article surveys major techniques for processing and optimizing search queries in dataspaces, and highlights future directions in accessing dataspaces.

The Open Forum Column features an article by Pavlo and Aslett, discussing "What is really New with NewSQL." NewSQL refers to a new class of database management systems (DBMSs) that claim the ability to scale modern on-line transaction processing (OLTP) workloads in a way that is not possible with legacy systems. Given the continuous development of relational DBMSs over the past four decades, this article examines whether the claim of NewSQL's superiority is true. To do this, the article reviews the history to explain how NewSQL systems came about, and provides a taxonomy of NewSQL systems with a detailed discussion of the different systems under this taxonomy. The main takeaway from the analysis is that NewSQL systems are not a radical departure from existing system architectures. What is innovative is that they incorporate various database technologies into single platforms with significant engineering effort. They are by-products of a new era where distributed computing resources are plentiful and affordable, but at the same time the de-

mands of applications are much greater.

The Distinguished Profiles column features H.V. Jagadish, Professor of Electrical Engineering and Computer Science at the University of Michigan and an ACM Fellow. In this interview, Jagadish talks about his experience of starting the Proceedings of VLDB (PVLDB), a hybrid conference and journal style publication, his involvement with CoRR (Computing Research Repository), and his perspective on data-driven research.

The Data Centers column features an article by Naumann and Krestel on the Information Systems Group at the Hasso Plattner Institute (HPI). The article describes the research focus of the group on data profiling, data cleansing, and text mining.

Finally, this issue closes with a message from the Editor-in-Chief of ACM TODS and Call for Papers for PODS 2017.

On behalf of the SIGMOD Record Editorial board, I hope that you all enjoy reading the June 2016 issue of the SIGMOD Record!

Your submissions to the Record are welcome via the submission site:
http://sigmod.hosting.acm.org/record

Prior to submission, please read the Editorial Policy on the SIGMOD Record's website:
http://sigmod.org/sigmodrecord/

Yanlei Diao

June 2016

Past SIGMOD Record Editors:

Ioana Manolescu (2009-2013)	Alexandros Labrinidis (2007–2009)	Mario Nascimento (2005–2007)
Ling Liu (2000–2004)	Michael Franklin (1996–2000)	Jennifer Widom (1995–1996)
Arie Segev (1989–1995)	Margaret H. Dunham (1986–1988)	Jon D. Clark (1984–1985)
Thomas J. Cook (1981–1983)	Douglas S. Kerr (1976-1978)	Randall Rustin (1974-1975)
Daniel O'Connell (1971–1973)	Harrison R. Morse (1969)	

Factorized Databases

http://www.cs.ox.ac.uk/projects/FDB/

Dan Olteanu Maximilian Schleich

Department of Computer Science, University of Oxford

ABSTRACT

This paper overviews factorized databases and their application to machine learning. The key observation underlying this work is that state-of-the-art relational query processing entails a high degree of redundancy in the computation and representation of query results. This redundancy can be avoided and is not necessary for subsequent analytics such as learning regression models.

1. INTRODUCTION

Succinct data representations have been developed across many fields including computer science, statistics, applied mathematics, and signal processing. Such representations are employed for instance for storing and transmitting otherwise large amounts of data, and for speeding up data analysis [22].

In this paper we overview recent developments on factorized databases, which are succinct lossless representations of relational data. They exploit laws of relational algebra, in particular the distributivity of the Cartesian product over union that underlies algebraic factorization, and data and computation sharing to reduce redundancy in the representation and computation of query results. The relationship between a flat, tabular representation of a relation as a set of tuples and an equivalent factorized representation is on a par with the relationship between logic functions in disjunctive normal form and their equivalent circuits.

Factorized databases naturally capture existing relational decompositions proposed in the literature: lossless decompositions defined by join dependencies, as investigated in the context of normal forms in database design [1], conditional independence in Bayesian networks [18], minimal constraint networks in constraint satisfaction [7], factorizations of provenance polynomials of query results [15] used for efficient computation in probabilistic databases [12, 20], and product decompositions of relations as studied in the context of incomplete information [13].

In the following we first exemplify the benefits of factorizing query results. We then discuss factorizations for various classes of queries, quantify the succinctness gap between factorized and standard tabular representations for results of conjunctive queries and survey worst-case optimal algorithms for computing factorized representations of query results [16, 17]. We then briefly mention the case of queries with aggregates and order-by clauses [3] and discuss in more detail their application to learning regression models over factorized databases [19, 14].

2. A FACTORIZATION EXAMPLE

Figure 1(a) depicts three relations and their natural join. `Branch` records the location, products and daily inventory of each branch store in the chain. There are many products per location and many inventories per product. `Competition` records the competitors (e.g., the distance to competitor stores) of a store branch at a given location, with several competitors per location. `Sales` records daily sales offered by the store chain for each product.

The join result exhibits a high degree of redundancy. The value l_1 occurs in 12 tuples, each value c_1 and c_2 occurs in six tuples and they are paired with the same tuples of values for the other attributes. Since l_1 is paired in `Competition` with c_1 and c_2 and in `Branch` with p_1 and p_2, the Cartesian product of $\{c_1, c_2\}$ and $\{p_1, p_2\}$ occurs in the join result. We can represent this product symbolically as $\{c_1, c_2\} \times \{p_1, p_2\}$ instead of materializing it. If we systematically apply this observation, we obtain an equivalent *factorized representation* of the entire join result that is much more compact than its flat representation. Each tuple in the flat join result is represented once in the factorization and can be constructed by following one branch of each union and all branches of each product. The flat join result in Figure 1(a) has 90 values (18 tuples of 5 values each), while the equivalent factorized join result in Figure 1(d) only has 20 values.

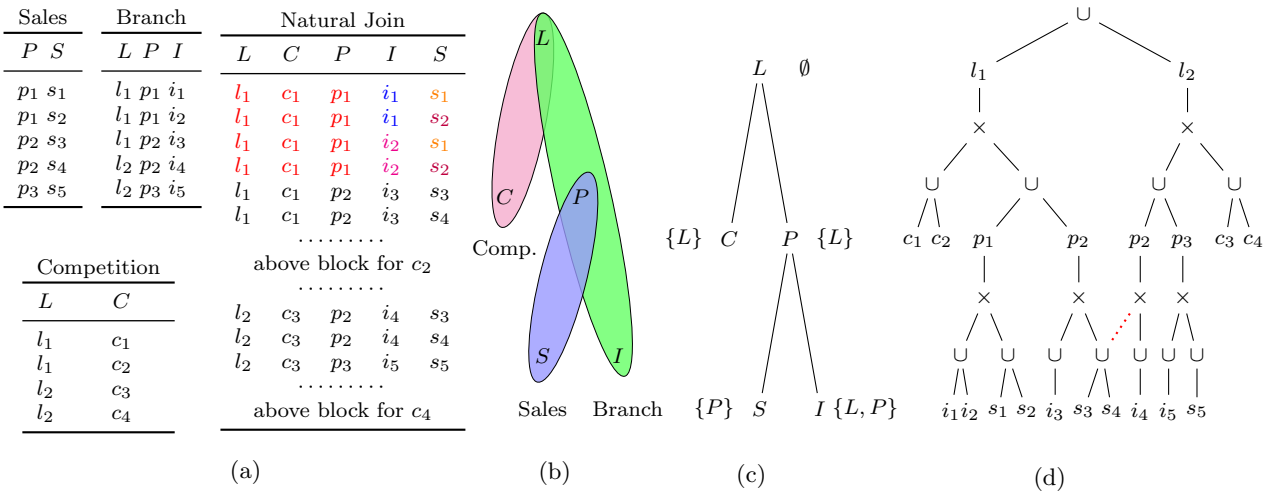


Figure 1: (a) Database with relations Branch(Location, Product, Inventory), Competition(Location, Competitor), Sales(Product, Sale), where the attribute names are abbreviated; (b) Hypergraph of the natural join of the relations; (c) Variable order Δ defining one possible nesting structure of the factorized join result given in (d). The union $s_3 \cup s_4$ is cached under the first occurrence of p_2 and referenced (via a dotted edge) from the second occurrence of p_2.

Figure 1(c) depicts the nesting structure of our factorized join result as a partial order Δ on the query variables: The factorization is a union of L-values occurring in both `Competitors` and `Branch`. For each L-value l, it is a product of the union of C-values paired with l in `Competitors` and of the union of P-values paired with l in `Branch` and with S-values in `Sales`. That is, given l, the C-values are *independent* of the P-values and can be stored separately. The factorization saves computation and space as it avoids the materialization of the product of the unions of C-values and of P-values for a given L-value. The same applies to the unions of S-values and of I-values under each P-value. Further saving is brought by *caching* expressions: The union of S-values $S_{34} = s_3 \cup s_4$ from `Sales` occurs with the P-value p_2 *regardless* of which L-values p_2 is paired with in `Branch`, so we can store the first occurrence of S_{34} and refer to it using a pointer $^{\uparrow}S_{34}$ from every subsequent occurrence of p_2. Like product factorization, caching is enabled by conditional independence: The variable S is independent of its ancestor L given its parent P. We encode it using a function *key* that maps each variable A to the set of its ancestors on which A and its descendants depend; this is given next to each variable in Δ.

Different variable orders are possible. We seek those variable orders that fully exploit the independence among variables and lead to succinct factorizations. Branching and caching are indicators of good variable orders. The total orders have no branching and caching, so they define factorizations with no asymptotic saving over flat representations.

For our join and any database, a factorized join result can be computed in linear time (modulo a log factor in the database size). In contrast, there are databases for which the flat join result requires cubic computation time, e.g., databases with one L and P-value and n distinct C, S, and I-values.

SQL aggregates can be computed in one pass over the factorized join result. For instance, to compute the aggregate `sum(1)` that computes the cardinality of the join result, we interpret each data value as 1 and turn unions into sums and products into multiplication. To compute `sum(P*C)` that sums over all multiplications of products and competitors (assuming they are numbers), we turn all values except for P and C into 1, unions into sums, and products into multiplications (the result of $\boxed{(1+1)}$ in the first line is cached and reused in the second line):

$$1 \cdot (c_1 + c_2) \cdot [p_1 \cdot (1+1) \cdot (1+1) + p_2 \cdot 1 \cdot \boxed{(1+1)}] +$$
$$1 \cdot [p_2 \cdot \boxed{(1+1)} \cdot 1 + p_3 \cdot 1 \cdot 1] \cdot (c_3 + c_4).$$

Learning regression models requires the computation of a family of aggregates `sum(`X_1`*...*`X_n`)` for any tuple of (not necessarily distinct) variables $(X_1, \ldots, X_n)$, such as the above aggregate `sum(P*C)`.

We may also compute aggregates with group-by clauses. Our factorized join result supports grouping by any set of variables that sit above all others in the variable order Δ, e.g., group by $\{L, C, P\}$.

3. QUERY FACTORIZATION

As exemplified in Section 2, factorized representations of relational data use Cartesian products to capture the independence in the data, unions to captures alternative values for an attribute, and references to capture caching.

DEFINITION 3.1. A *factorized representation* is a list $(D_i)_{i\in[m]}$, where each D_i is a relational algebra expression over a schema Σ and has one of the following forms:

- $\emptyset$, representing the empty relation over Σ,
- $\langle\rangle$, representing the relation consisting of the nullary tuple, if $\Sigma = \emptyset$,
- a, representing the relation $\{(a)\}$ with one tuple having one data value (a), if $\Sigma = \{A\}$ and the value $a \in \mathrm{Dom}(A)$,
- $\bigcup_{j\in[k]} E_j$, representing the union of the relations represented by E_j, where each E_j is an expression over Σ,
- $\times_{j\in[k]} E_j$, representing the Cartesian product of the relations represented by E_j, where each E_j is an expression over schema Σ_j such that Σ is the disjoint union of all Σ_j.
- a reference $^{\uparrow}E$ to an expression E over Σ.

The expression D_i may contain references to D_k for $k > i$ and is referenced at least once if $i > 1$. □

Definition 3.1 allows arbitrarily-nested factorized representations. In this paper, we focus on factorized representations of query results whose nesting structures are given by orders on query variables.

DEFINITION 3.2. Given a join query Q, a variable *depends* on another variable if they occur in the same relation symbol in Q.

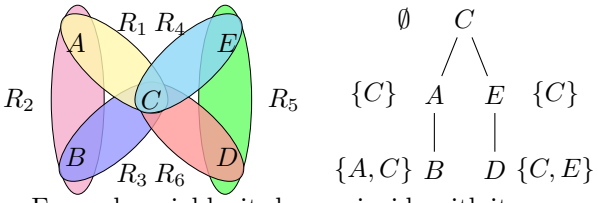
A *variable order* Δ for Q is a pair (T, key).

- T is a rooted forest with one node per variable in Q such that the variables of each relation symbol in Q lie along the same root-to-leaf path in T.
- The function key maps each variable A to the subset of its ancestor variables in T on which the variables in the subtree rooted at A depend, i.e., for every variable B that is a child of a variable A, $key(B) \subseteq key(A) \cup \{A\}$. □

If two variables A and B in Q depend on each other, then the choice of A-values may restrict the choice of B-values in a factorization of Q's result and we need to represent explicitly their possible combinations. If they are independent, then the set of A-values can be represented separately from the set of B-values and their combinations are only expressed symbolically. The succinctness of factorized representations lies in the exploitation of conditional independence between variables. In a variable order, this is reflected in branching, i.e., a variable has several children, and caching, i.e., a variable has ancestors that are not keys.

EXAMPLE 3.3. The key information in the variable order Δ in Figure 1(c) is given next to each variable. The variable L is an ancestor of S, yet it is not in $key(S)$ since S does not depend on it. We can thus cache the unions of S-values for a P-value and refer to them for each occurrence of that P-value. Δ also features branching: C and P are children of L, while S and I are children of P. This means that for a given L-value (P), we can store symbolically the product of the unions of C-values and of P-values (and respectively of S and I).

Consider now the cyclic bowtie join query over relations $R_1, \ldots, R_6$ and a variable order for it:



For each variable, its keys coincide with its ancestors, so there is no saving due to caching. There are however two branches under C, each of them defining a triangle query. For each C-value c we can thus compute and store the set of triangles $\{(c, A, B) \mid R_1(A, c), R_2(A, B), R_3(B, c)\}$ for the left branch separately from the set of triangles $\{(c, E, D) \mid R_4(c, E), R_5(E, D), R_4(c, D)\}$ for the right branch. □

Among the known classes of variable orders [17, 6], we consider here the most general class called d-trees. They are another syntax for hypertree decompositions of the join hypergraph [17].

3.1 Succinctness and Computation Time

The construction of variable orders is guided by the joins, their selectivities, and input cardinalities. They can lead to factorizations of greatly varying sizes, where the size of a representation (flat or factorized) is defined as the number of its values. Within the class of factorizations over variable orders, we can find the worst-case optimal ones and also compute them in worst-case optimal time:

THEOREM 3.4 ([2, 11, 17]). *Given a join query Q, for every database $\mathbf{D}$, the result $Q(\mathbf{D})$ admits*

- *a flat representation of size $O(|\mathbf{D}|^{\rho^*(Q)})$;*
- *a factorized representation of size $O(|\mathbf{D}|^{fhtw(Q)})$.*

There are classes of databases $\mathbf{D}$ for which the above size bounds are tight.

There are worst-case optimal join algorithms to compute the join result in these representations.

factorize (variable order Δ, varMap, ranges$[(start_i, end_i)_{i\in[r]}]$)

if $(\Delta = (\Delta_j)_{j\in[k]})$ **return** $\times_{j\in[k]}$ **factorize**$(\Delta_j, \text{varMap}, \text{ranges}[(start_i, end_i)_{i\in[r]}])$;

$A = var(\Delta)$; $E_\Delta = \emptyset$; context $= \pi_{key(A)}(\text{varMap})$;

if $(key(A) \neq anc(A))$ { ${}^{\uparrow}E_\Delta = \text{cache}_A[\text{context}]$; **if** ${}^{\uparrow}E_\Delta \neq 0$ **return** ${}^{\uparrow}E_\Delta$; }

foreach $a \in \bigcap_{i\in[r], A\in\mathsf{Schema}[R_i]} \pi_A(R_i[start_i, end_i])$ **do** {
 foreach $i \in [r]$ **do** find ranges $R_i[start'_i, end'_i] \subseteq R_i[start_i, end_i]$ s.t. $\pi_A(R_i[start'_i, end'_i]) = a$;
 switch(Δ) :
  leaf node A :
   $E_\Delta = E_\Delta \cup a$;
  inner node $A(\Delta_j)_{j\in[k]}$:
   foreach $j \in [k]$ **do** E_{Δ_j}= **factorize**$(\Delta_j, \text{varMap} \times a, \text{ranges}[(start'_i, end'_i)_{i\in[r]}])$;
   if $(\forall j \in [k] : E_{\Delta_j} \neq \emptyset)$ $E_\Delta = E_\Delta \cup \big(a \times (\times_{j\in[k]} E_{\Delta_j})\big)$;
}
if $(key(A) \neq anc(A))$ $\text{cache}_A[\text{context}] = {}^{\uparrow}E_\Delta$;

return E_Δ;

Figure 2: Grounding a variable order Δ over a database $(R_1, \ldots, R_r)$. The parameters of the initial call are Δ, an empty variable map, and the full range of tuples for each relation.

The measures $\rho^*(Q)$ and $fhtw(Q)$ are the fractional edge cover number and the fractional hypertree width respectively. We know that

$$1 \leq fhtw(Q) \leq \rho^*(Q) \leq |Q|$$

and the gap between them can be as large as $|Q|$, which is the number of relations in Q. The fractional hypertree width is fundamental to problem tractability with applications spanning constraint satisfaction, databases, matrix operations, logic, and probabilistic graphical models [9].

Example 3.5. The join query in Section 2 is acyclic and has $fhtw = 1$ and $\rho^* = 3$. The bowtie query has $fhtw = 3/2$, which already holds for each of its two triangles, and $\rho^* = 3$, which is the sum of the $\rho*$ values of the two triangles. □

3.2 Worst-case Optimal Join Algorithms

Worst-case optimal join algorithms for flat query results have been developed only recently [11]. At their outset is the observation that the classical relation-at-a-time query plans are suboptimal since their flat intermediate results may be larger than the flat query result [2]. To attain worst-case optimality, a new breed of join algorithms has been proposed that avoids intermediate results [11]. This monolithic recipe is however an artifact of the flat representation and not necessary for optimality: Using factorized intermediate results, optimality can be achieved by join-at-a-time query plans [6]. Such plans explore breadth-first the factorized space of assignments for query variables and can compute the join result in both factorized and flat form. An equivalent depth-first exploration leads to a monolithic worst-case optimal algorithm [17].

Figure 2 gives a worst-case optimal monolithic (depth-first) algorithm that computes the grounding E_Δ of a variable order Δ over an input database. If Δ is a variable order for a join query, then E_Δ is the factorized join result. As discussed in Section 3.3, Δ may also be a variable order for conjunctive queries with group-by and order-by clauses.

In case Δ is a forest, we construct a product of the factorizations over its trees. We next discuss the case where Δ is a tree with root variable A.

The relations are assumed sorted on their attributes following a depth-first pre-order traversal of Δ. Each call takes a range defined by start and end indices in each relation. Initially, these ranges span the entire relations. Once the root A is mapped to a value a in the intersection of possible A-values from the relations with attribute A, then these ranges are narrowed down to those tuples with value a for A. We may further narrow down these ranges using mappings for variables below A in Δ at higher recursion depths. Each A-value a in this intersection is the root of a factorization fragment over Δ. (Following Definition 3.1, a stands for relation $\{(a)\}$.)

We first check whether the factorization we are about to compute has been already computed. If this is the case, we simply return a reference to it from cache. If not, we compute it and place its reference in the cache. The key for the cache is the context of A, i.e., the current mapping of the variables in $key(A)$. The current variable mappings are kept in varMap. Caching is useful when $key(A)$ is strictly contained in $anc(A)$, since this means that the factorization fragments over the variable order rooted at A are repeated for every distinct combination of values for variables in $anc(A) \setminus key(A)$.

If Δ is a leaf node, then we construct a union of all mappings a of A. If it is an inner node, then for each mapping a we recurse to each child and construct a factorization that is a product of a and the factorizations at children.

This algorithm defaults to LeapFrog TrieJoin [23] if Δ is a path where $key(A) = anc(A)$ for each variable A in Δ, i.e., when there is no branching and no sharing. The resulting factorization is a trie.

The key operation dictating the time complexity of this algorithm is the intersection of the arrays of ordered values defined by the relation ranges. This takes time linear in the size of the smallest array (modulo log factor) [23].

3.3 Beyond Join Queries

The above framework is immediately extensible to queries with projections [16, 17] and order-by and group-by clauses [3] by appropriately restricting the variable orders of the factorized query results.

Projection. If a variable is projected away, then all variables depending on it now depend on each other. Following Definition 3.2, all dependent variables need to lie along the same path in the variable order. For instance, if we project away the variable P in our running example, then the variables S and I, which used to be independent given P, become dependent on each other. This restricts the possible variable orders of factorizations of the query result, possibly decrease the branching factor in the variable order, and likely increases the factorization size. Variable orders and their widths can be defined for conjunctive queries in immediate analogy to the case of join queries [16, 17].

Group-By and Order-By. For a relation representing a query result R, grouping by a set G of variables partitions the tuples of R into groups that agree on the G-value. Ordering R by a list O of variables sorts R lexicographically on the variables in the order given by O, where for each variable in O the sorting is in ascending or descending order.

Given a factorized representation R over a variable order, we can enumerate the tuples in the relation represented by R *in no particular order* with *constant delay*, i.e., the time between listing two consecutive tuples is independent of the number of tuples and thus constant under data complexity.

Group-by and order-by clauses require however to enumerate the tuples *in some desired order* as given by the explicit order O or by the group G so that all tuples with the same G-value are listed consecutively and can be aggregated. Constant-delay enumeration following an order O or a group G is not supported by arbitrary variable orders, but by those obeying specific constraints on the variables in G or O.

THEOREM 3.6 ([3]). *Given a factorized representation of a relation R over a variable order Δ, a set G of group-by variables, and a list O of order-by variables.*

The tuples within each G-group in R can be enumerated with constant delay if and only if each variable of G is either root in Δ or a child of another variable of G.

The tuples in R can be enumerated with constant delay in sorted lexicographic order by O if and only if each variable X of O is either root in Δ or a child of a variable appearing before X in O.

In other words, constant-delay enumeration of tuples within a group holds exactly when the group-by variables are above the other variables in the variable order Δ. For an order-by list O, the condition is stronger: there is a topological order of the variables in Δ that has O as prefix. Theorem 3.6 assumes that the values within each union are sorted. This is the case in the FDB system for factorized query processing [4] and the F system for factorized learning of regression models [19].

EXAMPLE 3.7. The variable order Δ from Figure 1(c) supports constant-delay tuple enumeration for the following sets of group-by variables: $\{L\}$, $\{L, C\}$, $\{L, P\}$, $\{L, C, P\}$, $\{L, P, I\}$, $\{L, P, I, S\}$, $\{L, C, P, S\}$, $\{L, C, P, I\}$, and $\{L, C, P, S, I\}$; and for the lists of order-by variables (with any variable in ascending or descending order) that can be constructed from the above sets such that they are prefixes of a topological order of Δ. □

There are two strategies for a factorized computation of a conjunctive query Q with order-by or group-by clauses. We derive a variable order Δ with minimum width that satisfies all constraints for the joins, projection, and order-by or group-by clauses. Then, given Δ and the input database, we compute

the factorized query result. Alternatively, we derive a variable order Δ' for the join of Q and compute the factorized join result. We then restructure Δ' and its factorization to support projection, grouping, and ordering[3]. The second strategy may be preferred if the join is very selective so the restructuring is not expensive.

Aggregates. We consider SQL aggregates based on expressions in the semiring $(\mathbb{N}[\Delta], +, \cdot, 0, 1)$ of polynomials with variables from Δ and coefficients from $\mathbb{N}$ [8], e.g., sums over expressions $2 \cdot X$ and $X \cdot Y$ for variables X and Y.

THEOREM 3.8 (GENERALIZATION OF [3, 19]). *Given a variable order Δ and a factorized representation E over Δ. Any SQL aggregate of the form* `sum(`X`)`*,* `min(`X`)`*, or* `max(`X`)`*, where X is an expression in the semiring $(\mathbb{N}[\Delta], +, \cdot, 0, 1)$, can be computed in one pass over E.*

If Δ supports constant-delay enumeration for a group-by clause G, then Theorem 3.8 applies to SQL aggregates with group-by G clause.

The algorithm in Figure 2 can be extended to compute aggregates on top of joins without the need to first materialize the factorized join result [19]: Instead of creating factorization fragments and possibly caching them, we compute (and possibly cache) the result of computing the aggregates on them and propagate these aggregates up through recursion. Since the aggregates we consider are distributive, we compute them at a variable in the variable order using the aggregates at its children.

Our earlier worst-case optimal factorization algorithm for conjunctive queries [17] coupled with one-pass aggregates and group-by clauses [3] can be recovered via the recent framework of Functional Aggregate Queries (FAQ) [9]. Both approaches are dynamic programming algorithms with the same runtime complexity. While FAQ is bottom-up, the factorization algorithm is top-down (memoized).

4. LEARNING REGRESSION MODELS

We show in this section how to learn polynomial regression models over factorized databases, generalizing earlier work on linear regression [19]. The core computation underlying the construction of such models concerns a set of aggregates with semiring expressions such as those from Theorem 3.8.

4.1 Polynomial Regression

We consider the setting where a polynomial regression model is learned over a training dataset defined by a query over a database:

$$\{(y^{(1)}, x_1^{(1)}, \ldots, x_n^{(1)}), \ldots, (y^{(m)}, x_1^{(m)}, \ldots, x_n^{(m)})\}.$$

The values $y^{(i)}$ are the *labels* and the values $x_j^{(i)}$ are the *features*. We assume that the intercept of the model is captured by one feature with value 1. All values are real numbers.

Polynomial regression models predict the value of the label based on a linear function of parameters and *feature interactions*, which are products of features. A polynomial regression model with all feature interactions up to degree d is given by:

$$h_\theta(x) = \sum_{t=1}^{d} \sum_{k_1=1}^{n} \cdots \sum_{k_t=k_{t-1}}^{n} \theta_{(k_1,\ldots,k_t)} x_{k_1} \cdot \ldots \cdot x_{k_t}.$$

The case of $d = 1$ corresponds to linear regression.

For each model h_θ, we define a set $\mathcal{I}$ such that each $K \in \mathcal{I}$ corresponds to the feature interaction of parameter θ_K in h_θ. For example, by substituting K by $(1, 3)$, the parameter $\theta_{(1,3)}$ corresponds to the interaction term $x_1 \cdot x_3$. For the model given above:

$$\mathcal{I} = \bigcup_{t \in [d]} \{(k_1, .., k_t) \mid \forall 1 \leq k_1 \leq \ldots \leq k_t \leq n\}.$$

Given the training dataset, the goal is to fit the parameters $\theta_{(k_1,\ldots,k_t)}$ of the model so as to minimize the error of an objective function. We consider the popular least squares regression objective function:

$$\mathcal{E}(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda R(\theta).$$

$R(\theta)$ is a regularization term used to overcome model overfitting, and λ determines its weight in $\mathcal{E}(\theta)$. Examples of regularization terms are: $\lambda \sum_{J \in \mathcal{I}} \theta_J^2$ (Ridge); $\lambda \sum_{J \in \mathcal{I}} |\theta_J|$ (Lasso); and $\lambda_1 \sum_{J \in \mathcal{I}} |\theta_J| + \lambda_2 \sum_{J \in \mathcal{I}} \theta_J^2$ (Elastic-Net). For uniformity in our equations, we treat the label y as a feature but with a predefined parameter -1.

We use *batch gradient descent* (BGD) [5] to learn the model. BGD repeatedly updates the model parameters in the direction of the gradient to decrease the error given by the objective function $\mathcal{E}(\theta)$ and to eventually converge to the optimal value:

$$\forall J \in \mathcal{I} : \theta_J := \theta_J - \alpha \frac{\delta}{\delta \theta_J} \mathcal{E}(\theta),$$

$$\frac{\delta}{\delta \theta_J} \mathcal{E}(\theta) = \sum_{i=1}^{m} h_\theta(x^{(i)}) \prod_{j \in J} x_j^{(i)} + \lambda \frac{\delta}{\delta \theta_J} R(\theta),$$

where the learning rate α determines the size of each convergence step. We use $j \in J$ to iterate over the elements in tuple J.

In standard BGD, each convergence step scans the training dataset and computes the sum aggregate, then updates the parameters, and repeats this process until convergence. This is inefficient because a large bulk of the computation is repeated

across the convergence steps. A rewriting of the sum aggregate can avoid the redundant work and make it very competitive.

4.2 Rewriting the Update Program

BGD has two logically independent tasks: The computation of the sum aggregate and convergence of the parameters. The data-dependent part is the sum aggregate S_J, where $J, K = (k_1, \ldots, k_t) \in \mathcal{I}$:

$$S_J = \sum_{i=1}^{m} \left(\sum_{t=1}^{d} \sum_{k_1=1}^{n} \cdots \sum_{k_t=k_{t-1}}^{n} \theta_K \prod_{k \in K} x_k^{(i)} \right) \prod_{j \in J} x_j^{(i)}$$

We can explicate the *cofactor* of θ_K in S_J:

$$S_J = \sum_{t=1}^{d} \sum_{k_1=1}^{n} \cdots \sum_{k_t=k_{t-1}}^{n} \theta_K \times \text{Cofactor}[K, J]$$

$$\text{where Cofactor}[K, J] = \sum_{i=1}^{m} \prod_{k \in K} x_k^{(i)} \prod_{j \in J} x_j^{(i)}.$$

For linear regression, $t = p = 1$ and the cofactors are sums over products of two features.

Remarkably, the data-dependent computation is captured fully by the cofactors, which are completely decoupled from the parameters. Therefore, this reformulation enables us to compute cofactors once and perform parameter convergence directly on the matrix of cofactors, whose size is independent of the data size m. This is crucial for performance as we do not require one pass over the entire training dataset for each convergence step.

The matrix of cofactors has desirable properties:

PROPOSITION 4.1. ([19]) *Given a query Q, database $\mathbf{D}$, where the query result $Q(\mathbf{D})$ has schema $\sigma = (A_i)_{i \in [n]}$. Let* Cofactor *be the cofactor matrix for learning a polynomial regression model h_θ using BGD over $Q(\mathbf{D})$.*

The cofactor matrix has the following properties:

1. Cofactor *is symmetric:*

$$\forall K, J \in \mathcal{I} : \text{Cofactor}[K, J] = \text{Cofactor}[J, K].$$

2. Cofactor *computation commutes with union: Given a disjoint partitioning $\mathbf{D} = \bigcup_{j \in [p]} (\mathbf{D}_j)$ and cofactors $(\text{Cofactor}_j)_{j \in [p]}$ over $(Q(\mathbf{D}_j))_{j \in [p]}$, then*

$$\forall K, J \in \mathcal{I} : \text{Cofactor}[K, J] = \sum_{j=1}^{p} \text{Cofactor}_j[K, J].$$

3. Cofactor *computation commutes with projection: Given a feature set $L \subseteq \sigma$ and cofactor matrix Cofactor_L for the training dataset $\pi_L(Q(\mathbf{D}))$, then*

$$\forall K, J \in \mathcal{I}(\text{ s.t. } \forall k \in K, j \in J : A_k, A_j \in L) : \text{Cofactor}_L[K, J] = \text{Cofactor}[K, J].$$

The symmetry property implies that we only need to compute one half of the cofactor matrix.

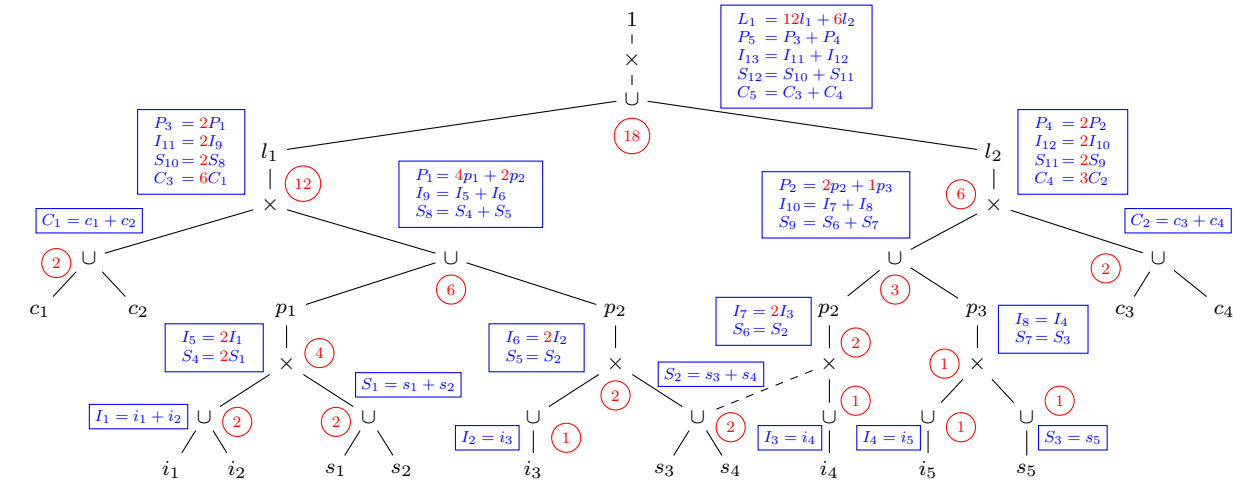
Commutativity with union means that the cofactor matrix for the union of several training datasets is the entry-wise sum of the cofactor matrices of these training datasets. This property is key to the efficiency of our approach, since we can locally compute partial cofactors over different partitions of the training dataset and then add them up. It is also desirable for *concurrent computation*, where partial cofactors can be computed on different cores.

The commutativity with projection implies that we can compute any regression model that consists of a subset of the parameters in the cofactor matrix. During convergence, we simply ignore from the matrix the columns and rows for the irrelevant parameters. This is beneficial if some features are necessary for constructing the dataset but irrelevant for learning, e.g., relation keys supporting the join such as location in our training dataset in Figure 1(a). It is also beneficial for *model selection*, a key challenge in machine learning centered around finding the subset of features that best predict a test dataset. Model selection is a laborious and time-intensive process, since it requires to learn independently parameters corresponding to subsets of the available features. With our reformulation, we first compute the cofactor matrix for all features and then perform convergence on top of the cofactor matrix for the entire lattice of parameters independently of the data. Besides choosing the features after cofactor computation, we may also choose the label and fix its parameter to -1.

The commutativity with projection is crucial for computing the most succinct factorization of the training dataset, since it does not restrict the choice of variable order and allows to retain the join variables in the variable order. Projecting away join variables from the variable order may break the conditional independence of other variables and increase the size of the factorization. Once the factorization is constructed, we may decide to only compute cofactors for a subset of the features.

4.3 Factorized Computation of Cofactors

There are several flavors for factorized cofactor computation [19, 14]: over the (non-)materialized factorized dataset or via an optimized SQL query. The materialized flavor is sketched in this section and the SQL flavor is presented in detail in Section 4.4. The underlying idea of all these flavors is that the algebraic factorization used to factorize the training dataset can be mirrored in the factorized computation of the cofactors.



	θ_L	θ_P	θ_I	θ_S	θ_C
Σ_T	$12l_1 + 6l_2$	$2(4p_1 + 2p_2) + 2(2p_2 + p_3)$	$4(i_1 + i_2 + i_3 + i_4) + 2i_5$	$4(s_1 + s_2 + s_3 + s_4) + 2s_5$	$6(c_1 + c_2) + 3(c_3 + c_4)$
Σ_L	$12l_1^2 + 6l_2^2$	$l_1P_3 + l_2P_4$	$l_1I_{11} + z_2I_{12}$	$l_1S_{10} + l_2S_{11}$	$l_1C_3 + l_2C_4$
Σ_P	Σ_Z/θ_P	$2(4p_1^2 + 2p_2^2) + 2(2p_2^2 + p_3^2)$	$2(p_1I_5 + p_2(I_6 + I_7) + p_3I_8)$	$2(p_1S_4 + p_2(S_5 + S_6) + p_3S_7)$	$P_1C_1 + P_2C_2$
Σ_I	Σ_L/θ_I	Σ_P/θ_I	$4(i_1^2 + i_2^2 + i_3^2 + i_4^2) + 2i_5^2$	$2(I_1S_1 + I_2S_2 + I_3S_2 + I_4S_3)$	$I_9C_1 + I_{10}C_2$
Σ_S	Σ_L/θ_S	Σ_P/θ_S	Σ_I/θ_S	$4(s_1^2 + s_2^2 + s_3^2 + s_4^2) + 2s_5^2$	$S_8C_1 + S_9C_2$
Σ_C	Σ_L/θ_C	Σ_P/θ_C	Σ_I/θ_C	Σ_S/θ_C	$6(c_1^2 + c_2^2) + 3(c_3^2 + c_4^2)$

Figure 3: (Top) The factorized join annotated with constant (counts) and linear (weighted sums) aggregates used for cofactor computation. (Bottom) Cofactor matrix based on the annotated factorized join (Column for intercept T not shown, the value for Σ_T/Θ_T is 18). For any model, the convergence of model parameters is run on top of this matrix.

EXAMPLE 4.2. Based on our running example in Figure 1(a), consider a linear regression model with all variables as features (the label can be decided later). The cofactor Cofactor$[(P),(I)]$ is the sum of products of features P and I. If this aggregate is computed over the factorized join result in Figure 1(d), then we can exploit the algebraic factorization rules

$$\sum_{i=1}^{n} x \to x \cdot n \quad \text{and} \quad \sum_{i=1}^{n} x \cdot a_i \to x \cdot \sum_{i=1}^{n} a_i$$

to obtain

$$\begin{aligned}\text{Cofactor}[(P),(I)] = {} & 2p_1 \cdot 2(i_1 + i_2) + 2p_2 \cdot 2i_3 + \\ & 2p_2 \cdot 2i_4 + 2p_3 \cdot i_5,\end{aligned}$$

where the coefficients are the number of C-values that are paired with P-values and I-values. Similarly, we can factorize the pairs of values of independent features as follows:

$$\sum_{i=1}^{r}\sum_{j=1}^{s}(x_i \cdot y_j) \to (\sum_{i=1}^{r} x_i) \cdot (\sum_{j=1}^{s} y_j).$$

For features P and C, the cofactor would then be:

$$\begin{aligned}\text{Cofactor}[(P),(C)] = {} & (4p_1 + 2p_2)(c_1 + c_2) + \\ & (2p_2 + p_3)(c_3 + c_4).\end{aligned}$$

For a factorization E representing a relation R, we compute the cofactors at the root of E using cofactors at children. They require the computation of constant (degree 0) aggregates corresponding to the number of tuples in R; linear (degree 1) aggregates for each feature A of E, which are sums of all A-values, weighted by the number of times they occur in R; and quadratic (degree 2) aggregates, which are products of values and/or linear aggregates, or of quadratic and constant aggregates. We call all these aggregates the *regression aggregates* for linear regression models [19].

Figure 3 displays the factorized join result, annotated with the constant (circles) and linear aggregates (rectangles), and an excerpt of the cofactor matrix, whose elements are quadratic aggregates. The $\langle 1 \rangle$ at the top of the factorization represents the intercept of the model. This can be obtained by extending each relation with one attribute T with value 1. The variable orders for the query would then have the variable T as root. □

We generalize the factorized computation of cofactors to polynomial regression models of arbitrary degree d. The difference to the linear case is that the cofactors are for more features due to feature interactions and thus the degrees of regression aggregates increase as well. For a given model of de-

gree d, the highest-degree aggregates have degree $2d$. We construct high-degree aggregates by combining lower-degree aggregates, following the factorization rules from Example 4.2.

We can compute the cofactors in one pass over a factorized query result.

PROPOSITION 4.3. ([3, 19]) *The cofactors of any polynomial regression model can be computed in one pass over any factorized representation.*

An immediate implication is that the redundancy in the flat join result is not necessary for learning:

THEOREM 4.4. *The parameters of any polynomial regression model of degree d can be learned over a query Q and database $\mathbf{D}$ in time $O(n^{2d} \cdot |\mathbf{D}|^{fhtw(Q)} + n^{2d} \cdot s)$, where n is the number of features and s is the number of convergence steps.*

Theorem 4.4 is a direct corollary of Propositions 3.4 and 4.3. Under data complexity, this becomes $O(|\mathbf{D}|^{fhtw(Q)})$ and coincides with the time needed to compute the factorized query result. For computing join queries, this is worst-case optimal within the class of factorized representations over variable orders. The factor n^{2d} is the total number of regression aggregates needed to learn h_θ. In contrast, the data complexity of any regression learner taking a flat join result as input would be at best $O(|\mathbf{D}|^{\rho^*(Q)})$. Furthermore, state-of-the-art learners typically do not decouple parameter convergence from data-dependent computation, so their total runtime to learn h_θ is $O(n^{2d} \cdot |\mathbf{D}|^{\rho^*(Q)} \cdot s)$.

4.4 Factorized Computation in SQL

A SQL encoding of factorized computation of cofactors has two desirable properties. It can be computed by any relational database system and is thus readily deployable in practice with a small implementation overhead. It leverages secondary-storage mechanisms and thus works for databases that do not fit in memory. For lack of space, we focus on learning over a join query Q_{in} (i.e., no projections).

Our approach has two steps. We first rewrite Q_{in} into an (α-)acyclic query Q_{out} over possibly cyclic subqueries. Since cyclic queries are not factorizable, we materialize them to new relations. To attain the overall complexity from Theorem 4.4, this materialization requires a worst-case optimal join algorithm like LeapFrog TrieJoin [23]. We then generate a SQL query that encodes the factorized computation of the regression aggregates over Q_{out}.

Rewriting queries with cycles. Figure 4 gives the rewriting procedure. It works on a variable order Δ of Q_{in}. The set QS is a disjoint partitioning

rewrite(variable order Δ)

$QS = \emptyset$; $A = var(\Delta)$;
if $(\Delta = A(\Delta_j)_{j\in[k]})$ $QS = \bigcup_{j\in[k]}$ **rewrite**(Δ_j);
$Q_A = \text{relations}(key(A) \cup \{A\})$;
if $(\nexists Q \in QS$ s.t. $Q_A \subseteq Q)$ **return** $QS \cup \{Q_A\}$
else return QS

Figure 4: Rewriting a join query over variable order Δ into an acyclic join query over possibly cyclic subqueries.

of the set of relations of Q_{in} into sets of relations or partitions. Each partition Q_A is a join query defined by the set $key(A) \cup \{A\}$ of variables. This partition is materialized to a relation with the same name Q_A. The materialization simplifies the variable order of Q_{in} to that of an acyclic query Q_{out} equivalent to Q_{in}. In case Q_{in} is already acyclic, then each partition has one relation and hence Q_{out} is syntactically equal to Q_{in}.

EXAMPLE 4.5. Let us consider the bowtie join query and its variable order from Example 3.3. We apply the rewriting algorithm. When we reach leaf B in the left branch, we create the join query Q_B over the relations $\{R_1, R_2, R_3\}$ and add it to QS. When we return from recursion to variable A, we create the query Q_A over the same relations, so we do not add it to QS. We proceed similarly in the right branch: We create the join query Q_D over relations $\{R_4, R_5, R_6\}$ and add it to QS. The queries at E and C are not added to QS. Whereas the original query and the two subqueries Q_B and Q_D are cyclic, the rewritten query Q_{out} is the join of Q_B and Q_D on C and is acyclic. The triangle queries Q_B and Q_D cannot be computed worst-case optimally with traditional relational query plans [2], but we can use specialized engines to compute them [23].

The join query in Figure 1(a) is already acyclic. Using its variable order from Figure 1(c), we obtain one identity query per relation. □

SQL query generation. The algorithm in Figure 5 generates one SQL query that computes all regression aggregates (thus including the cofactors) of a polynomial regression model. It takes as input an extended variable order Δ, which has one extra node per database relation placed under its lowest variable. The query is then constructed in a top-down traversal of Δ.

For a variable order Δ with root A, we materialize a relation A_{type} over the schema (A_n, A_d), where A_n is an identifier for A and A_d encodes the degrees of

factorize-sql (extended variable order Δ)

switch(Δ) :
leaf node R :
CREATE TABLE $R_{type}(R_n, R_d)$; **INSERT INTO** R_{type} **VALUES** $(R, 0)$;
let
$deg(G_\Delta) = R_d, \quad lineage(G_\Delta) = (R_n, R_d), \quad agg(G_\Delta) = 1$
in
$G_\Delta =$ **SELECT** $schema(R), lineage(G_\Delta), deg(G_\Delta), agg(G_\Delta)$ **FROM** R, R_{type};

inner node $A(\Delta_j)_{j\in[k]}$:
CREATE TABLE $A_{type}(A_n, A_d)$;
foreach $0 \le i \le 2d$ **do INSERT INTO** A_{type} **VALUES** (A, i);
foreach $j \in [k]$ **do** $\mathbf{G}_{\Delta_i} =$ **factorize-sql**(Δ_j);
let
$deg(G_\Delta) = \sum_{j\in[k]} deg(G_{\Delta_j}) + A_d, \quad lineage(G_\Delta) = (lineage(G_{\Delta_1}), \ldots, lineage(G_{\Delta_k}), A_n, A_d),$
$agg(G_\Delta) =$ **sum**(**power**$(A, A_d) * \prod_{j\in[k]} agg(G_{\Delta_j})$)
in
$G_\Delta =$ **SELECT** $key(A), lineage(G_\Delta), deg(G_\Delta), agg(G_\Delta)$
FROM G_{Δ_1}**NATURAL JOIN** $\ldots G_{\Delta_k}, A_{type}$
WHERE $deg(G_\Delta) \le 2d$ **GROUP BY** $key(A), lineage(G_\Delta), deg(G_\Delta)$;
return G_Δ;

Figure 5: Generation of one SQL query for computing all regression aggregates used to build a polynomial model over an acyclic query with extended variable order Δ.

the aggregates over A. A_{type} has $2d+1$ tuples, one for each degree from zero to $2d$.

We generate a query G_Δ, which computes the regression aggregates for the factorization over Δ. G_Δ is a natural join of the queries $(G_{\Delta_j})_{j\in[k]}$ constructed for the children of A and an inequality join with relation A_{type}. The query computes all aggregates ($agg(G_\Delta)$) of degree ($deg(G_\Delta)$) up to $2d$ along with the *lineage* of their computation. These aggregates are computed by combining aggregates computed at children and for A. The lineage is given by columns with indices n and d. It is a relational encoding of the set $\mathcal{I}$ of indexes of feature interactions as given in Section 4.1. Furthermore, the query retains the variables in $key(A)$ which are to be joined on in queries constructed for the ancestors of A.

For a leaf node representing an input relation R, the type relation R_{type} has only one tuple $(R, 0)$ and the generated query G_Δ is a product of R and R_{type}. We add a copy of column R_d to represent the degree $deg(G_\Delta)$ and we set $agg(G_\Delta)$ equal to 1. These additions allow us to treat all nodes uniformly.

For simplicity of exposition, we accommodate the intercept T as described in Example 4.2, where each relation is extended with one extra attribute T with value 1. T is used as a root node for any variable order, so variable orders cannot be forests.

Example 4.6. We show how to generate the SQL query for computing the regression aggregates for any polynomial regression model of degree d over the acyclic join in Figure 1(a). The queries constructed at different nodes in the variable order Δ in Figure 1(c), now extended with three relation nodes, are given in Figure 6.

For the path leading to the Sales node, we generate the queries Q_{Sales}, Q_S, and Q_P for the nodes Sales, S, and respectively P.

The query Q_{Sales} constructed at node Sales computes the product of $Sales$ and $Sales_{type}$ and adds the degree and aggregate columns.

Variable S only has Sales as child. Its query Q_S computes inequality join on Q_{Sales} and S_{type} to limit the combinations of aggregates to those of degree at most $2d$. It also propagates the lineage from Q_{Sales} and S_{types} and keeps the variable P because it is in its key: $key(S) = \{P\}$.

The query Q_P constructed at variable P computes the natural join of queries Q_S and Q_I and the inequality join with P_{type} on degree. This query also computes the aggregates (P_{agg}) with degree (P_{deg}) up to $2d$ and their lineage (columns with indexes n and d). The query keeps the variable L because it is in its key: $key(P) = \{L\}$. □

CREATE TABLE Q_{Sales} **AS**
SELECT P, S $Sales_n, Sales_d,$
$(Sales_d)$ **AS** $Sales_{deg}$, 1 **AS** $Sales_{agg}$
FROM $Sales, Sales_{type}$;

CREATE TABLE Q_S **AS**
SELECT $P,$ $Sales_n, Sales_d,$ $S_n, S_d,$
$(Sales_{deg} + S_d)$ **AS** S_{deg},
$\text{sum}(\text{power}(S, S_d) * Sales_{agg})$ **AS** S_{agg}
FROM Q_{Sales}, S_{type}
WHERE $(Sales_{deg} + S_d)$ $<= 2d$
GROUP BY $P,$ $Sales_n, Sales_d,$
S_n, S_d, S_{deg}

CREATE TABLE Q_P **AS**
SELECT $L,$ $I_n, I_d, Branch_n, Branch_d,$
$S_n, S_d, Sales_n, Sales_d,$ $P_n, P_d,$
$(I_{deg} + S_{deg} + P_d)$ **AS** P_{deg},
$\text{sum}(\text{power}(P, P_d) * I_{agg} * S_{agg})$ **AS** P_{agg}
FROM Q_I **NATURAL JOIN** Q_S, P_{type}
WHERE $(I_{deg} + S_{deg} + P_d)$ $<= 2d$
GROUP BY $L,$ $I_n, I_d, Branch_n, Branch_d,$
$S_n, S_d, Sales_n, Sales_d,$ P_n, P_d, P_{deg}

Figure 6: Queries generated by the algorithm in Figure 5 at nodes $Sales$, S, and P in the extended variable order of the rewritten query. In each of these queries, the aggregate, degree, and lineage columns are color-coded.

5. CONCLUSION AND FUTURE WORK

Factorized databases are a fresh look at the problem of computing and representing results to relational queries. So far, we addressed the worst-case optimal computation of factorized results for conjunctive queries under various factorized representation systems for relational data [17], the characterization of the succinctness gap between sizes of factorized and flat representations of query results and their provenance polynomials [16], and the factorized computation of aggregates [3], such as those needed for learning polynomial regression models over database joins [19].

These theoretical results form the foundation of the **FDB** system for query factorization [4, 3], and of the **F** system for learning regression models over factorized queries [19, 14].

We next discuss directions of future research.

Practical considerations. We have recently implemented an early prototype for learning regression models and preliminary benchmarks are very encouraging. For linear regression models, our prototype **F** can achieve up to three orders of magnitude performance speed-up over state-of-the-art systems R, Python StatsModels, and MADlib [19]. This performance gap can be further widened by accommodating systems aspects such as compilation of high-level code that only depends on the fixed set of features and not on the arbitrarily large data.

We are currently designing and implementing a second, more robust version of our in-memory query engine **FDB** for factorized databases, whose focus is on a cache-friendly representation and computation and the exploitation of many-cores architectures.

Beyond polynomial regression. The principles behind **F** are applicable to learning statistical models beyond least-squares polynomial regression models (with various regularizers) using gradient descent. It works for any model where the derivatives of the objective function are expressible in a semiring with multiplication and summation operations. It also works for classification, such as boosted trees and k-nearest neighbors, and other optimization algorithms, such as Newton optimization and coordinate descent. The semirings are necessary, since factorization relies on the commutativity and distributivity laws of semirings.

Distributed Factorized Computation. Massively parallel query processing incurs a high network communication cost [21]. This cost has been analyzed theoretically for the Massively-Parallel Communication model, with several recent results on communication optimality for join queries [10].

Since factorized databases were specifically designed for succinct and lossless representations of relational data, a natural idea would be to reduce the communication cost by shuffling factorized, instead of flat, data between computation rounds.

Acknowledgements

The authors acknowledge Nurzhan Bakibayev, Radu Ciucanu, Tomáš Kočiský, and Jakub Závodný for contributions to various aspects of factorized databases reported in this paper. This work has been supported by an Amazon AWS Research Grant, a Google Faculty Research Award, LogicBlox, and the ERC grant FADAMS agreement 682588.

6. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748, 2008.

[3] N. Bakibayev, T. Kociský, D. Olteanu, and J. Závodný. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.

[4] N. Bakibayev, D. Olteanu, and J. Závodný. FDB: A query engine for factorised relational databases. *PVLDB*, 5(11):1232–1243, 2012.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 2006.

[6] R. Ciucanu and D. Olteanu. Worst-case optimal join at a time. Technical report, Oxford, Nov 2015.

[7] G. Gottlob. On minimal constraint networks. *Artif. Intell.*, 191-192:42–60, 2012.

[8] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[9] M. A. Khamis, H. Q. Ngo, and A. Rudra. FAQ: Questions Asked Frequently, CoRR:1504.04044, April 2015.

[10] P. Koutris, P. Beame, and D. Suciu. Worst-case optimal algorithms for parallel query processing. In *ICDT*, pages 8:1–8:18, 2016.

[11] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *PODS*, pages 37–48, 2012.

[12] D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[13] D. Olteanu, C. Koch, and L. Antova. World-set decompositions: Expressiveness and efficient algorithms. *TCS*, 403(2-3):265–284, 2008.

[14] D. Olteanu and M. Schleich. F: Regression models over factorized views. *PVLDB*, 9(10), 2016.

[15] D. Olteanu and J. Závodný. On factorisation of provenance polynomials. In *TaPP*, 2011.

[16] D. Olteanu and J. Závodný. Factorised representations of query results: size bounds and readability. In *ICDT*, pages 285–298, 2012.

[17] D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *TODS*, 40(1):2, 2015.

[18] J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, 1989.

[19] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, 2016.

[20] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.

[21] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed SQL database that scales. *PVLDB*, 6(11):1068–1079, 2013.

[22] Simons Institute for the Theory of Computing, UC Berkeley. *Workshop on "Succinct Data Representations and Applications"*, September 2013.

[23] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

Database Meets Deep Learning: Challenges and Opportunities

Wei Wang[†], Meihui Zhang[‡], Gang Chen[§],
H. V. Jagadish[#], Beng Chin Ooi[†], Kian-Lee Tan[†]
[†]National University of Singapore [‡] Singapore University of Technology and Design
[§]Zhejiang University [#]University of Michigan
[†]{wangwei, ooibc, tankl}@comp.nus.edu.sg [‡]meihui_zhang@sutd.edu.sg
[§]cg@zju.edu.cn [#]jag@umich.edu

ABSTRACT

Deep learning has recently become very popular on account of its incredible success in many complex data-driven applications, including image classification and speech recognition. The database community has worked on data-driven applications for many years, and therefore should be playing a lead role in supporting this new wave. However, databases and deep learning are different in terms of both techniques and applications. In this paper, we discuss research problems at the intersection of the two fields. In particular, we discuss possible improvements for deep learning systems from a database perspective, and analyze database applications that may benefit from deep learning techniques.

1. INTRODUCTION

In recent years, we have witnessed the success of numerous data-driven machine-learning-based applications. This has prompted the database community to investigate the opportunities for integrating machine learning techniques in the design of database systems and applications [29]. A branch of machine learning, called deep learning [22, 18], has attracted worldwide interest in recent years due to its excellent performance in multiple areas including speech recognition, image classification and natural language processing (NLP). The foundation of deep learning was established about twenty years ago in the form of neural networks. Its recent resurgence is mainly fueled by three factors: immense computing power, which reduces the time to train and deploy new models, e.g. Graphic Processing Unit (GPU) enables the training systems to run much faster than those in the 1990s; massive (labeled) training datasets (e.g. ImageNet) enable a more comprehensive knowledge of the domain to be acquired; new deep learning models (e.g. AlexNet [20]) improve the ability to capture data regularities.

Database researchers have been working on system optimization and large scale data-driven applications since 1970s, which are closely related to the first two factors. It is natural to think about the relationships between databases and deep learning. First, are there any insights that the database community can offer to deep learning? It has been shown that larger training datasets and a deeper model structure improve the accuracy of deep learning models. However, the side effect is that the training becomes more costly. Approaches have been proposed to accelerate the training speed from both the system perspective [5, 19, 9, 28, 11] and the theory perspective [45, 12]. Since the database community has rich experience with system optimization, it would be opportune to discuss the applicability of database techniques for optimizing deep learning systems. For example, distributed computing and memory management are key database technologies. They are also central to deep learning.

Second, are there any deep learning techniques that can be adapted for database problems? Deep learning emerged from the machine learning and computer vision communities. Recently, it has been successfully applied to other domains, like NLP [13]. However, few studies have been conducted using deep learning techniques for database problems. This is partially because traditional database problems — like indexing, transaction and storage management — involve less uncertainty, whereas deep learning is good at predicting over uncertain events. Nevertheless, there are problems in databases like knowledge fusion [10] and crowdsourcing [27], which are probabilistic problems. It is possible to apply deep learning techniques in these areas. We will discuss specific problems like querying interface, knowledge fusion, etc. in this paper.

The rest of this paper is organized as follows: Section 2 provides background information about deep learning models and training algorithms; Section 3 discusses the application of database techniques for

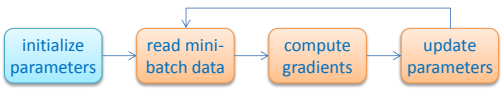


Figure 1: Stochastic Gradient Descent.

optimizing deep learning systems. Section 4 describes research problems in databases where deep learning techniques may help to improve performance. Some final thoughts are presented in Section 5.

2. BACKGROUND

Deep learning refers to a set of machine learning models which try to learn high-level abstractions (or representations) of raw data through multiple feature transformation layers. Large training datasets and deep complex structures enhance the ability of deep learning models for learning effective representations for tasks of interest. There are three popular categories of deep learning models according to the types of connections between layers [22], namely feedforward models (directed connection), energy models (undirected connection) and recurrent neural networks (recurrent connection). Feedforward models, including Convolution Neural Network (CNN), propagate input features through each layer to extract high-level features. CNN is the state-of-the-art model for many computer vision tasks. Energy models, including Deep Belief Network (DBN) are typically used to pre-train other models, e.g., feedforward models. Recurrent Neural Network (RNN) is widely used for modeling sequential data. Machine translation and language modeling are popular applications of RNN.

Before deploying a deep learning model, the model parameters involved in the transformation layers need to be trained. The training turns out to be a numeric optimization procedure to find parameter values that minimize the discrepancy (loss function) between the expected output and the real output. Stochastic Gradient Descent (SGD) is the most widely used training algorithm. As shown in Figure 1, SGD initializes the parameters with random values, and then iteratively refines them based on the computed gradients with respect to the loss function. There are three commonly used algorithms for gradient computation corresponding to the three model categories above: Back Propagation (BP), Contrastive Divergence (CD) and Back Propagation Through Time (BPTT). By regarding the layers of a neural net as nodes of a graph, these algorithms can be evaluated by traversing the graph in certain sequences. For instance, the BP algorithm

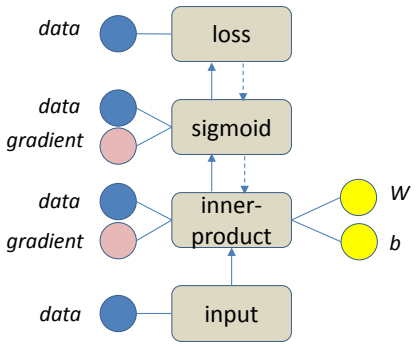


Figure 2: Data flow of Back-Propagation.

is illustrated in Figure 2, where a simple feedforward model is trained by traversing along the solid arrows to compute the data (feature) of each layer, and along the dashed arrows to compute the gradient of each layer and each parameter (W and b).

3. DATABASES TO DEEP LEARNING

In this section, we discuss the optimization techniques used in deep learning systems, and research opportunities from the perspective of databases.

3.1 Stand-alone Training

Currently, the most effective approach for improving the training speed of deep learning models is to use Nvidia GPU with the cuDNN library. Researchers are also working on other hardware, e.g. FPGA [21]. Besides exploiting advancements in hardware technology, operation scheduling and memory management are two important components to consider.

3.1.1 Operation Scheduling

Training algorithms of deep learning models typically involve expensive linear algebra operations as shown in Figure 3, where the matrix $W1$ and $W2$ could be larger than $4096*4096$. Operation scheduling is to first detect the data dependency of operations and then place the operations without dependencies onto executors, e.g., CUDA streams and CPU threads. Take the operations in Figure 3 as an example, $a1$ and $a2$ in Figure 3 could be computed in parallel because they have no dependencies. The first step could be done statically based on dataflow graph or dynamically [3] by analyzing the orders of read and write operations. Databases also have this kind of problems in optimizing transaction execution [44] and query plans. Those solutions should be considered for deep learning systems. For instance, databases use cost models to estimate query plans. For deep learning, we may also create a cost

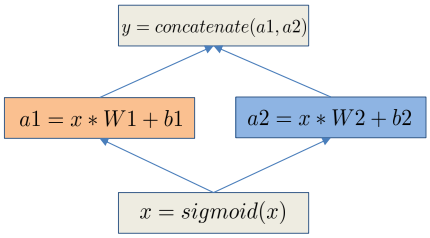


Figure 3: Sample operations from a deep learning model.

model to find an optimal operation placing strategy for the second step of operation scheduling given a fixed computing resources including executors and memory.

3.1.2 Memory Management

Deep learning models are becoming larger and larger, and already occupy a huge amount of memory space. For example, the VGG model [32] cannot be trained on normal GPU cards due to memory size constraints. Many approaches have been proposed towards reducing memory consumption. Shorter data representation, e.g. 16-bit float [7] is now supported by CUDA. Memory sharing is an effective approach for memory saving [3]. Take Figure 3 as an example, the input and output of the *sigmoid* function share the same variable and thus the same memory space. Such operations are called 'in-place' operations. Recently, two approaches were proposed to trade-off computation time for memory. Swapping memory between GPU and CPU resolves the problem of small GPU memory and large model size by swapping variables out to CPU and then swapping back manually[8]. Another approach drops some variables to free memory and recomputes them when necessary based on the static dataflow graph[4].

Memory management is a hot topic in the database community with a significant amount of research towards in-memory databases [35, 46], including locality, paging and cache optimization. To elaborate more, the paging strategies could be useful for deciding when and which variable to swap. In addition, failure recovery in databases is similar to the idea of dropping and recomputing approach, hence the logging techniques in databases could be considered. If all operations (and execution time) are logged, we can then do runtime analysis without the static dataflow graph. Other techniques, including garbage collection and memory pool, would also be useful for deep learning systems, especially for GPU memory management.

3.2 Distributed Training

Distributed training is a natural solution for accelerating the training speed of deep learning models. The parameter server architecture [9] is typically used, in which the workers compute parameter gradients and the servers update the parameter values after receiving gradients from workers. There are two basic parallelism schemes for distributed training, namely, data parallelism and model parallelism. In data parallelism, each worker is assigned a data partition and a model replica, while for model parallelism, each worker is assigned a partition of the model and the whole dataset. The database community has a long history of working on distributed environment, ranging from parallel databases [23] and peer-to-peer systems [37] to cloud computing [25]. We will discuss some research problems relevant to databases arising from distributed training in the following paragraphs.

3.2.1 Communication and Synchronization

Given that deep learning models have a large set of parameters, the communication overhead between workers and servers is likely to be the bottleneck of a training system, especially when the workers are running on GPUs which decrease the computation time. In addition, for large clusters, the synchronization between workers can be significant. Consequently, it is important to investigate efficient communication protocols for both single-node multiple GPU training and training over a large cluster. Possible research directions include : a) compressing the parameters and gradients for transmission [30]; b) organizing servers in an optimized topology to reduce the communication burden of each single node, e.g., tree structure [15] and AllReduce structure [42] (all-to-all connection); c) using more efficient networking hardware like RDMA [5].

3.2.2 Concurrency and Consistency

Concurrency and consistency are critical concepts in databases. For distributed training of deep learning models, they also matter. Currently, both declarative programming (e.g., Theano and TenforFlow) and imperative programming (e.g., Caffe and SINGA) have been adopted in existing systems for concurrency implementation. Most deep learning systems use threads and locks directly. Other concurrency implementation methods like actor model (good at failure recovery), co-routine and communicating sequential processes have not been explored.

Sequential consistency (from synchronous training) and eventual consistency (from asynchronous training) are typically used for distributed deep learn-

Table 1: Summary of optimization techniques used in existing systems as of July 2016.

	SINGA	Caffe	Mxnet	TensorFlow	Theano	Torch
1. operation scheduling	✓	x	✓	-	-	x
2. memory management	d+a+p	i	d+s	p	p	-
3. parallelism	d + m	d	d + m	d + m	-	d + m
4. consistency	s+a+h	s/a	s+a+h	s+a+h	-	s

1. x: not available: ✓: available 2. d: dynamic; a: swap; p: memory pool; i: in-place operation; s: static;
3. d: data parallelism; m: model parallelism; 4. s: synchronous; a: asynchronous; h:hybrid; -: unknown

ing. Both approaches have scalability issues [38]. Recently, there are studies for training convex models (deep learning models are non-linear and non-convex) using a value bounded consistency model [41]. Researchers are starting to investigate the influence of consistency models on distributed training [15, 16, 2]. There remains much research to be done on how to provide flexible consistency models for distributed training, and how each consistency model affects the scalability of the system, including communication overhead.

3.2.3 *Fault Tolerance*

Databases systems have good durability via logging (e.g., command log) and checkpointing. Current deep learning systems recover the training from crashes mainly based on checkpointing files [11]. However, frequent checkpointing would incur vast overhead. In contrast with database systems, which enforce strict consistency in transactions, the SGD algorithm used by deep learning training systems can tolerate a certain degree of inconsistency. Therefore, logging is not a must. How to exploit the SGD properties and system architectures to implement fault tolerance efficiently is an interesting problem. Considering that distributed training would replicate the model status, it is thus possible to recover from a replica instead of checkpointing files. Robust frameworks (or concurrency model) like actor model, could be adopted to implement this kind of failure recovery.

3.3 Existing Systems

A summary of existing systems in terms of the above mentioned optimization aspects is listed in Table 1. Many researchers have extended Caffe [19] with ad hoc optimizations, including memory swapping and communication optimization. However, the official version is not well optimized. Similarly, Torch [6] itself provides limited support for distributed training. Mxnet[3] has optimization for both memory and operations scheduling. Theano [1] is typically used for stand-alone training. TensorFlow [11] has the potential for the aforementioned static optimization based on the dataflow graph.

We are optimizing the Apache incubator SINGA system [28] starting from version 1.0. For stand-alone training, cost models are explored for runtime operation scheduling. Memory optimization including dropping, swapping and garbage collection with memory pool will be implemented. OpenCL is supported to run SINGA on a wide range of hardware including GPU, FPGA and ARM. For distributed training, SINGA (V0.3) has done much work on flexible parallelism and consistency, hence the focus would be on optimization of communication and fault-tolerance, which are missing in almost all systems.

4. DEEP LEARNING TO DATABASES

Deep learning applications, such as computer vision and NLP, may appear very different from database applications. However, the core idea of deep learning, known as feature (or representation) learning, is applicable to a wide range of applications. Intuitively, once we have effective representations for entities, e.g., images, words, table rows or columns, we can compute entity similarity, perform clustering, train prediction models, and retrieve data with different modalities [40, 39] etc. We shall highlight a few deep learning models that could be adapted for database applications below.

4.1 Query Interface

Natural language query interfaces have been attempted for decades [24], because of their great desirability, particularly for non-expert database users. However, it is challenging for database systems to interpret (or understand) the semantics of natural language queries. Recently, deep learning models have achieved state-of-the-art performance for NLP tasks [13]. Moreover, RNN has been shown to be able to learn structured output [34, 36]. As one solution, we can apply RNN models for parsing natural language queries to generate SQL queries, and refine it using existing database approaches. For instance, heuristic rules could be applied to correct grammar errors in the generated SQL queries. The

challenge is that a large amount of (labeled) training samples is required to train the model. One possible solution is to train a baseline model with a small dataset, and gradually refining it with users' feedback. For instance, users could help correct the generated SQL query, and these feedback essentially serve as labeled data for subsequent training.

4.2 Query Plans

Query plan optimization is a traditional database problem. Most current database systems use complex heuristic and cost models to generate the query plan. According to [17], each query plan of a parametric SQL query template has an optimality region. As long as the parameters of the SQL query are within this region, the optimal query plan does not change. In other words, query plans are insensitive to small variations of the input parameters. Therefore, we can train a query planner which learns from a set of pairs of SQL queries and optimal plans to generate (similar) plans for new (similar) queries. To elaborate more, we can learn a RNN model that accepts the SQL query elements and meta-data (like buffer size and primary key) as input, and generates a tree structure [36] representing the query plan. Reinforcement learning (like AlphaGo [31]) could also be incorporated to train the model on-line using the execution time and memory footprint as the reward. Note that approaches purely based on deep learning models may not be very effective. In particular, the training dataset may not be comprehensive to include all query patterns, e.g. some predicates could be missing in the training datasets. To solve these problems, a better approach would be to combine database solutions and deep learning.

4.3 Crowdsourcing and Knowledge Bases

Many crowdsourcing [43] and knowledge base [10] applications involve entity extraction, disambiguation and fusion problems, where the entity could be a row of a database, a node in a graph, etc. With the advancements of deep learning models in NLP [13], it is opportune to consider deep learning for these problems. In particular, we can learn representations for entities and then do entity relationship reasoning [33] and similarity calculation using the learned representations.

4.4 Spatial and Temporal Data

Spatial and temporal data are common data types in database systems [14], and are commonly used for trend analysis, progression modeling and predictive analytics. Spatial data is typically processed by mapping moving objects into rectangular blocks. If we regard each block as a pixel of one image, then deep learning models, e.g., CNN, could be exploited to extract the spatial locality between nearby blocks. For instance, if we have the real-time location data (e.g., GPS data) of moving objects, we could learn a CNN model to capture the density relationships of nearby areas for predicting the traffic congestion for a future time point. When temporal data is modeled as features over a time matrix, deep learning models, e.g. RNN, can be designed to model time dependency and predict the occurrence in a future time point. A particular example would be disease progression modeling [26] based on historical medical records, where doctors would want to estimate the onset of certain severity of a known disease.

5. CONCLUSIONS

In this paper, we have discussed databases and deep learning. Databases have many techniques for optimizing system performance, while deep learning is good at learning effective representation for data-driven applications. We note that these two "different" areas share some common techniques for improving the system performance, such as memory optimization and parallelism. We have discussed some possible improvements for deep learning systems using database techniques, and research problems applying deep learning techniques in database applications. Let us not miss the opportunity to contribute to the exciting challenges ahead!

6. ACKNOWLEDGEMENT

We would like to thank Divesh Srivastava for his valuable comments. This work is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08). Meihui Zhang is supported by SUTD Start-up Research Grant under Project No. SRG ISTD 2014 084.

7. REFERENCES

[1] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] J. Chen, R. Monga, S. Bengio, and R. Józefowicz. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016.

[3] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[4] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[5] A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng. Deep learning with COTS HPC systems. In *ICML*, pages 1337–1345, 2013.

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[7] M. Courbariaux, Y. Bengio, and J.-P. David. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

[8] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *EuroSys*, page 4. ACM, 2016.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, pages 1232–1240, 2012.

[10] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[11] M. A. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[12] J. Gao, H. Jagadish, and B. C. Ooi. Active sampler: Light-weight accelerator for complex data analytics at scale. *arXiv preprint arXiv:1512.03880*, 2015.

[13] Y. Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[14] C. Guo, C. S. Jensen, and B. Yang. Towards total traffic awareness. *ACM SIGMOD Record*, 43(3):18–23, 2014.

[15] S. Gupta, W. Zhang, and J. Milthorpe. Model accuracy and runtime tradeoff in distributed deep learning. *arXiv preprint arXiv:1509.04210*, 2015.

[16] S. Hadjis, C. Zhang, I. Mitliagkas, and C. Ré. Omnivore: An optimizer for multi-device deep learning on cpus and gpus. *CoRR*, abs/1606.04487, 2016.

[17] J. R. Haritsa. The picasso database query optimizer visualizer. *Proceedings of the VLDB Endowment*, 3(1-2):1517–1520, 2010.

[18] Y. B. Ian Goodfellow and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[21] G. Lacey, G. W. Taylor, and S. Areibi. Deep learning on fpgas: Past, present, and future. *CoRR*, abs/1602.04283, 2016.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[23] M. L. Lee, M. Kitsuregawa, B. C. Ooi, K.-L. Tan, and A. Mondal. Towards self-tuning data placement in parallel database systems. In *ACM SIGMOD Record*, volume 29, pages 225–236. ACM, 2000.

[24] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.

[25] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu. Distributed data management using mapreduce. *ACM Comput. Surv.*, 46(3):31:1–31:42, 2014.

[26] D. R. Mould. Models for disease progression: New approaches and uses. *Clinical Pharmacology & Therapeutics*, 92(1):125–131, 2012.

[27] B. C. Ooi, K. Tan, Q. T. Tran, J. W. L. Yip, G. Chen, Z. J. Ling, T. Nguyen, A. K. H. Tung, and M. Zhang. Contextual crowd intelligence. *SIGKDD Explorations*, 16(1):39–46, 2014.

[28] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. H. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng. SINGA: A distributed deep learning platform. In *ACM Multimedia*, 2015.

[29] C. Ré, D. Agrawal, M. Balazinska, M. I. Cafarella, M. I. Jordan, T. Kraska, and R. Ramakrishnan. Machine learning and databases: The sound of things to come or a cacophony of hype? In *SIGMOD*, pages 283–284, 2015.

[30] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *INTERSPEECH*, pages 1058–1062, 2014.

[31] D. Silver and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[33] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.

[34] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[35] K.-L. Tan, Q. Cai, B. C. Ooi, W.-F. Wong, C. Yao, and H. Zhang. In-memory databases: Challenges and opportunities from software and hardware perspectives. *ACM SIGMOD Record*, 44(2):35–40, 2015.

[36] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. *arXiv:1412.7449*, 2014.

[37] Q. H. Vu, M. Lupu, and B. C. Ooi. *Peer-to-peer computing*. Springer, 2010.

[38] W. Wang, G. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K.-L. Tan, and S. Wang. SINGA: Putting deep learning in the hands of multimedia users. In *ACM Multimedia*, 2015.

[39] W. Wang, B. C. Ooi, X. Yang, D. Zhang, and Y. Zhuang. Effective multi-modal retrieval based on stacked auto-encoders. *PVLDB*, 7(8):649–660, 2014.

[40] W. Wang, X. Yang, B. C. Ooi, D. Zhang, and Y. Zhuang. Effective deep learning-based multi-modal retrieval. *The VLDB Journal*, pages 1–23, 2015.

[41] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Managed communication and consistency for fast data-parallel iterative analytics. In *SoCC*, pages 381–394, 2015.

[42] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep image: Scaling up image recognition. *CoRR*, abs/1501.02876, 2015.

[43] T. Wu, L. Chen, P. Hui, C. J. Zhang, and W. Li. Hear the whole story: Towards the diversity of opinion in crowdsourcing markets. *PVLDB*, 8(5):485–496, 2015.

[44] C. Yao, D. Agrawal, G. Chen, Q. Lin, B. C. Ooi, W. F. Wong, and M. Zhang. Exploiting single-threaded model in multi-core in-memory systems. *IEEE Trans. Knowl. Data Eng.*, 2016.

[45] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv:1212.5701*, 2012.

[46] H. Zhang, G. Chen, B. C. Ooi, K. Tan, and M. Zhang. In-memory big data management and processing: A survey. *IEEE Trans. Knowl. Data Eng.*, 27(7):1920–1948, 2015.

A Time Machine for Information: Looking Back to Look Forward

Xin Luna Dong
Google Inc.
lunadong@google.com

Anastasios Kementsietsidis
Google Inc.
akement@google.com

Wang-Chiew Tan*
UC Santa Cruz
tan@cs.ucsc.edu

ABSTRACT

Historical data (also called *long data*) holds the key to understanding when facts are true. It is through long data that one can understand the trends that have developed in the past, form the audit trails needed for justification, and make predictions about the future. For searching, there is also increasing interest to develop search capabilities over long data.

In this article, we first motivate the need to develop a *time machine* for information that will help people "look back" so as to "look forward". We will overview key ideas on three components (extraction, linking, and cleaning) that we believe are central to the development of any time machine for information. Finally, we conclude with our thoughts on what we believe are some interesting open research problems. This article is based on the material presented in a tutorial at VLDB 2015.

1. INTRODUCTION

> *"The longer you can look back, the farther you can look forward."*
> — Winston Churchill

There is general consensus that while *big data* are important, *long data* (i.e., historical data or temporal data) are even more important [29, 48], as it provides the potential to understand when facts are true and forms the basis for audit trails needed by organizations and individuals. It is for this reason that companies use temporal databases to provide support for rollbacks and auditing. For Web pages, information from the Wayback machine of the Internet Archive [30], which periodically keeps a snapshot of most webpages on the Web, has been used as legal evidence.

With the abundant availability of information one can mine from the Web today, there is increasing interest to develop capabilities to search long data on the Web, to develop a complete understanding of the history of an entity (i.e., a person, a company, a music genre, a country, etc.), and to depict trends over time, based on Web data. An example is the search for answers to the query "*Google's CEO before Sundar Pichai*". Current search results on Google returns articles about Sundar Pichai and it is only after one reads through some articles about the history of Google that one can infer that the CEO of Google before Sundar Pichai is Larry Page.

Another compelling example to motivate the need to understand *when* a fact is true comes from reports that are filed with the U.S. Securities and Exchange Commission (SEC) [20] at different times. Companies are required, by federal regulations, to file reports periodically to SEC to disclose the stock holdings of its executives. There are now millions of electronic filings in EDGAR and the number of such filings is increasing over time. Given the millions of SEC reports, how can one find out the stock holdings of an executive during a certain period of time? Were *Ann* and *Bob* affiliated with the same *company X* during a certain time period? Or perhaps more interestingly, did *Ann* purchase a significant number of shares of *Company Y* before it was announced that *Company X* bought *Company Y*? Techniques for integrating and aggregating data over time have also been explored in a number of projects [2, 5, 25, 33, 38, 40, 41, 45].

Finally, historical data provides an understanding of what is important or trending over time. For example, the understanding that most people believed earth was flat versus spherical and the trending topics of discussions allow tremendous opportunities for further knowledge [19, 46, 47].

The task of aggregating long data into a meaningful whole remains a largely difficult and manual task despite more than a couple of decades of research in the areas of temporal databases and data integration. The difficulty to create a comprehensive understanding of entities over time largely stems from the lack of (explicit) temporal data, and tools for interpreting such data even if they were available. Ideally, we would like to develop a time machine for information, where one can

*Tan is partially supported by NSF Grant IIS-1450560 and IIS 1524382.

easily and incrementally ingest temporal data to develop an ever increasing understanding of entities over time, search and query facts for a particular time period, understand trending patterns over time, and perform analytics that would allow one to, for example, understand the prevalent "knowledge" in the previous decade. In this article, we describe the techniques critical in building such a *time machine for information*, and discuss how far (or close) we are in achieving this goal.

The development of such a time machine would necessarily involve many of the challenges that occur in data integration [13, 17] and knowledge curation (see, for example, [4, 15, 27]), which are notoriously difficult tasks. The data integration process often includes the fundamental steps of extracting information about different types of (heterogeneous) entities, transforming and cleaning the information, and curating facts regarding different aspects or properties of those entities consistently together. An additional challenge today is to perform these tasks at scale; that is, we need to collect information from a large number of data sources, where each source may contain lots of data, and the schemas of the sources may be diverse in their structure and quality or may even be unavailable. The ability to inter-operate amongst heterogeneous data sources with varying quality is thus a key ingredient to the successful development of this time machine.

Another key ingredient to the successful development of such time machine is to make every step of the data integration process *time-aware*. In other words, we need the capability to understand the valid time period for each fact. To achieve this goal, one would inevitably require text extraction rules or techniques to extract structured temporal data from unstructured and semi-structured data sources. Furthermore, techniques need to be developed to map and transform temporal data into a desired format before temporal entity resolution can be applied. And finally, information about the extracted entities is temporally integrated and conflicting information is resolved to arrive at an integrated archive. This process may repeat as new datasets are discovered or when new versions of the same datasets are available to further enrich the information time machine.

Outline In this article, we first look back at past work related to (bi-)temporal databases. We discuss why new techniques beyond (bi-)temporal databases need to be developed to integrate and manage long data in general (Section 2). We will then present some existing work on three components (extraction, linking, and cleaning) that we believe are central to the development of any time machine for information (Sections 3-5) before we conclude with our thoughts by looking forward on some of the interesting open research problems (Section 6).

While one goal of this article is to disseminate the above described material, a parallel goal is to motivate the reader to pursue research in the direction of managing and integrating temporal data. Ultimately, we hope that there will be more research along these directions to bring us closer to realizing the goal of building a time machine for information that will record and preserve history accurately, and to help people "look back" and, so as to, "look forward".

We note that this article is based on material presented in a tutorial at VLDB 2015 [18].

2. TEMPORAL DATABASES

The need for enterprises to track and query long data, roll back to previous states of the database to provide audit trials has led to the provision of temporal data management features in relational databases. Research in temporal databases has a long history (see, for example, [9, 10, 42, 43]) but the ability to create and manage temporal tables only found its way to the SQL standard in 2011, as part of the SQL:2011 standard. Since then, major database vendors such as IBM DB2 10 [12], Oracle's Total Recall, and Teradata [1] have also begun to support the temporal features.

There are two primary notions of time in temporal databases, namely, *valid time* (the time period during which a tuple is true) and *transaction time* (where the beginning of this time period represents when the tuple and its valid time period was first recorded in the database, and the duration of the transaction represents the time period during which the recorded tuple and its valid time was true). These are also known as *application time* and, respectively, *system time* in SQL:2011. Naturally, transaction time (or system time) can only increase since the next tuple that is recorded can only occur at a later time than the previously recorded tuple. This is in contrast with valid time, which may refer to the past or future regardless of when it was entered into the database.

For example, Anna was at restaurant Fleur De Lys at 12pm on March 28. Suppose this information was extracted and entered into the database on 10am March 29, the transaction time begins on 10am March 29, even though its valid time begins earlier on 12pm March 28. Later, we may also discover that Anna was at San Francisco's Museum of Modern Art (MOMA) on 11am on March 28. Suppose this information was only determined and entered into the database on 12pm March 30, the valid time begins on March 28, 11am, while the transaction time begins on March 30, 12noon. Each subsequent fact that is entered into the database has a later transaction time than the previous fact even though the valid time may occur at different (out-of-order) times. The first two records in Figure 1 illustrate the example we just discussed. The third record shows that Anna

Name	Location	When	Known since
Anna	Fleur De Lys	Mar 28, 12pm	Mar 29, 10am
Anna	SF MOMA	Mar 28, 11am	Mar 30, 12pm
Anna	SF MOMA	Mar 28, 1:15pm	Mar 30, 2pm

Figure 1: Records of locations of Anna.

was at MOMA on 1:15pm March 28 and this fact was entered into the database on 2pm March 30.

SQL statements can be issued to record the above events as-is with the corresponding application and system times into a bitemporal database. However, without additional application logic, these records are insufficient for the purposes of consistently integrating this information to arrive at the understanding that Anna was at SF MOMA from 11am to 12noon, at Fleur De Lys from 12noon to 1:15pm, and then back at SF MOMA from 1:15pm onwards. Even worse, in general, the time at which the records are entered into the database may not even correspond to the time when the information was known, since different pieces of information may be derived from different data sources at different times. The above inference of where and when is Anna at a location is done based on the preference that information with a later known time is preferred to information from an earlier known time, and that Anna can only be at one location at any point in time. With other types of preferences (e.g., information from one source is preferred over the other), other conclusions may be derived.

The above discussions point out the need for techniques to consistently integrate long data that goes beyond what bi-temporal databases and existing data integration support. In particular, support for user-defined preferences to decide how conflicting long information should be resolved is needed. In fact, techniques such as [2, 5] have taken a step in this direction to consistently aggregate possibly conflicting long information from different sources. In the next three sections, we will present some existing work on three major components (extraction, linking, and fusion) that are central to the development of any end-to-end data integration and management system for long data.

3. EXTRACTION

3.1 Techniques for conventional data

To build any type of time machine, or big data repository, we require data. While large repositories of proprietary data are often hidden behind corporate firewalls, or are available with restrictive licensing, the web is a rich source of information and is publicly available for use by anyone. In addition, web data provides a wide coverage on almost any topic of interest, and is updated constantly. The challenge is thus to *extract* useful data from this fairly unstructured world. In what follows, we discuss some of the challenges in extracting data directly from texts, short texts, and fairly structured web tables.

Information extraction, which refers to the task of understanding text and extracting structured data from texts, has been an active area of research for quite some time [31]. Its objective is to take a sentence such as *"Anna was at restaurant Fleur De Lys on March 28, 12 noon"* and extract from it structured entries that might look like a set of triples of the form

(Anna, location, Fleur De Lys),

(Anna, on, March 28).

Notice that not all information might be succesfully extracted in the process of extraction. For example, we might not extract the fact that Anna was at the restaurant at 12 noon, or that Fleur De Lys is actually a restaurant (and not a city). So, extractions are often incomplete by nature. The extraction task is typically further exacerbated by the need to consider (a) the richness of the language and that the same fact can be expressed in a number of different ways; and (b) processing cannot be limited to a single sentence at a time since understanding the semantics of a sentence often requires understanding its context (e.g., other sentences in the same paragraph). As an example, assume that the sentence above is followed by the statement *"She ordered pasta"*. To extract the simple fact that *(Anna, ate, pasta)* we need to dereference *"She"* and understand that it refers to Anna. Successful approaches in information extraction often rely on machine learning [49] or on a combination of machine learning and natural language understanding techniques [21].

More recently, the popularity of services such as Twitter gave momentum to a new class of short text sources that is rich in information. A distinguishing characteristic of such short text sources is that although there is a huge volume of data, the information content is very small in each short text (140 characters in the case of Twitter). On the surface, the analysis of a 140-character tweet seems like a much easier task when compared to the analysis of a document with many sentences. However, the simplicity of short texts is deceiving and in fact, the lack of context often makes the analysis of short texts significantly harder.

Consider for example two simple short texts *"I like pink songs"* and *"I like pink shoes"*. Even though a human might immediately understand that the former short text is likely to refer to the songs of the artist Pink and the latter short text refers to the fact that the author of the text likes shoes that are pink in color, the lack of context makes the automatic analysis of these short texts very challenging. In addition to the lack of context, part of the challenge also has to do with the fact that short texts often lack any syntax (due to the limitations in length). As a result, traditional natural lan-

guage based techniques often fail to properly extract information, and new techniques that are customized for the analysis of short text are necessary [28].

In a more ideal extraction scenario, one may be able to find the data of interest in a fairly structured setting. For example, the data of interest may exist as a web table [6, 36]. Web tables are typically small relational tables crawled from HTML tables on the Web. Consider Anna from our example in the previous section. If Anna is a celebrity whose sightings are tracked at some fan web page, one can conceivably find a table like the one shown in Figure 1 that offers a row for each fan encounter with Anna. While a table such as this one is a great starting point for data extraction, there are still significant challenges to be addressed. For one thing, it is non-trivial to identify the data types of each column since such tables often come either without or with ambiguous metadata (e.g., column headings). Sometimes, such tables do not even come with column headings. More importantly, while structured data are designed with explicit connections between the different parts of the schema (e.g., foreign keys), figuring out how different web tables relate to each other often requires techniques that go beyond the extraction of column types and understanding the text surrounding the table.

3.2 Techniques for temporal data

There is a large body of work on *temporal information extraction* [37] that is pertaining to the construction of a time-machine, since beyond extracting facts about the world we would also like to know the time that these facts were valid. Conceptually, there are three main challenges in temporal information extraction, namely (a) the identification of temporal references in the input, (b) the mapping of a temporal reference to a time point (or interval), and (c) the assignment of time point (or interval) to a fact. In what follows, we provide a high-level review of the first two time-related challenges.

Part of the challenge in identifying temporal references is that time is expressed in a variety of ways in text. So, while one can find a clear temporal reference in the sentence *"Anna was at restaurant Fleur De Lys on March 28, 12 noon"*, the following two sentences include temporal references that are less obvious: *"Last week, Anna was at restaurant Fleur De Lys."*, and *"Anna was at restaurant Fleur De Lys."*. Notice that both the first and second sentences have an *explicit* time reference (in the form of "March 28, 12 noon", and "Last week"), while the last sentence has an implicit time reference since the past tense of the verb implies that the event happened some time in the past (see [24] of a more complete list of explicit and implicit examples).

To be useful in practice, an identified time reference must be assigned to a time point (or interval). In the examples above, it seems fairly straighforward to assign the "March 28, 12 noon" to a point in time, assuming we know the year that this date is referring to. Similarly, for the assignment of the reference "Last week" we need some indication as to what is the week the text (paragraph) was written, or in the worst case ground the time reference to a time point based on a signal like the document creation time (which resembles the transaction time in temporal databases). Similarly, for the last sentence we need to identify the time the text was written (or is referring to) so that we assign a time interval to the event that is upper-bounded by the identified time. Not surprisingly, mapping time references to a time dimension is probably the most challenging part of temporal information extraction. Techniques in this space might rely on rules [44], or a combination of machine-learning and syntactic analysis [37].

4. RECORD LINKAGE

After structured data is obtained, another major step is to identify records that refer to the same real-world entities. The *record linkage* problem refers to the following problem: *given a set of records, each describing an entity by its attribute values, partition the records so that each partition contains records that refer to a distinct real-world entity.*

4.1 Techniques for conventional data

Solutions to the record linkage problem typically include the following three steps [17].

1. *Blocking* refers to the process where blocking functions are applied to partition input records into multiple smaller blocks such that records in different blocks are very unlikely to refer to the same real-world entity. Blocking helps reduce the number of pairwise comparison that would otherwise be needed in subsequent steps.
2. *Pairwise matching* refers to the process that compares every pair of records in a block to decide if they indeed refer to the same entity.
3. *Clustering* refers to the process that examines local pairwise matching decisions to arrive at a globally consistent decision of partitioning the records from the same block such that each partition refers to a distinct entity.

Among the three steps, the blocking step is performed to achieve *scalability* (i.e., to handle massive data), while the pairwise matching and clustering steps are used to ensure the *semantics* of record linkage. In other words, by blocking, one avoids unnecessary comparisons between two records in the pairwise matching and clustering steps, which determine whether or not two records refer to the same real-world entity.

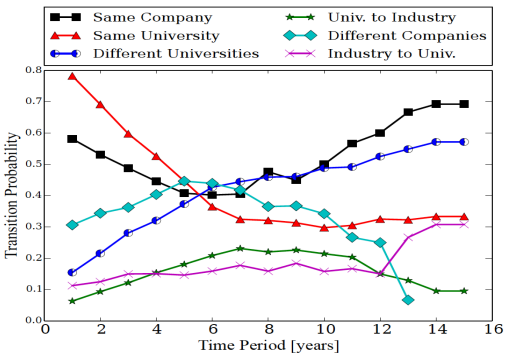


Figure 2: Transition rate on affiliation learned from DBLP [32].

4.2 Techniques for temporal data

Traditional linkage techniques typically reward high similarity between attribute values, and, likewise, penalize low similarity between attribute values. Such measures are not always appropriate for temporal record linkage. This is because, as time elapses, attribute values of an entity tend to evolve; for example, a person may change her name after marriage, change affiliations as a result of changing jobs, and may also change her phone number, address, and so on. Hence, blindly penalizing low value similarity can lead to a lot of false negatives in general; that is, records that refer to the same entity at different times are not matched because of value evolution. At the same time, as time elapses, different entities are increasingly likely to share the same attribute values, such as the shared name *Adam Smith* between a British philosopher in the 18th century and a current politician in the US. Hence, blindly rewarding high value similarity can lead to a lot of false positives in general; that is, records that refer to different entities at different times are wrongly matched together because they happen to share the same attribute values.

The discussions above point out that temporal records present new challenges to the *temporal record linkage problem* and new techniques are needed to perform temporal record linkage.

In the next two sections, we describe two key solutions for the temporal record linkage problem; one exploits pairwise matching techniques, and the other exploits clustering techniques. These techniques make crucial use of the following observations about temporal records in their solutions.

- First, entities typically evolve smoothly, and typically only a few attribute values of an entity change at any given time.
- Second, the values of an entity do not change erratically and, in particularly, they rarely change back and forth.
- Third, if the data set is fairly complete, records that refer to the same real-world entity typically (though not necessarily) observe continuity, or similarity in time gaps between adjacent records.

4.2.1 Time decay

A key insight by Li *et al.* is the notion of *time decay* [34] Time decay is often used in data analytics to reduce the impact of older records on the analysis and can be used effectively to capture the impact of time elapse on attribute-value evolution. Two types of decay, *disagreement decay* and *agreement decay*, were proposed in [34].

- Consider an attribute A and a time gap ΔT. The *disagreement decay* of A over ΔT is the probability that an entity changes its A-value within time ΔT. The dsiagreement decay is denoted by $d^{\neq}(A, \Delta T)$.
- Consider an attribute A and a time gap ΔT. The *agreement decay* of A over ΔT is the probability that the A-value is the same for two distinct entities within time ΔT. The notation we use for agreement decay is $d^{=}(A, \Delta T)$.

It is easy to see that both disagreement and agreement decays are values in $[0, 1]$, and typically monotonically non-decreasing as a function of their second argument ΔT. Intuitively, the disagreement decay is used to reduce the penalty for value disagreement, while the agreement decay is used to reduce the reward for value agreement, over a long time period. More formally, this is done by defining the pairwise similarity between two records R_1 and R_2 as

$$Sim(R_1, R_2) = \frac{\Sigma_{A \in \bar{A}} dw_A(s(R_1.A, R_2.A), \Delta T) * s(R_1.A, R_2.A)}{\Sigma_{A \in \bar{A}} dw_A(s(R_1.A, R_2.A), \Delta T)}$$

where $dw_A(s(), \Delta T)$ denotes the decayed weight of attribute A with value similarity $s()$ and time gap $\Delta T = |R_1.T - R_2.T|$. When the value similarity $s()$ is low, $dw_A(s(), \Delta T)$ is set to $w_A * (1 - d^{\neq}(A, \Delta T))$; when the value similarity $s()$ is high, $dw_A(s(), \Delta T)$ is set to $w_A * (1 - d^{=}(A, \Delta T))$, where w_A is the non-decayed weight of attribute A.

Li *et al.* [34] also describe ways to learn the disagreement and agreement decays empirically from a labeled data set. As an example, Figure 3 shows disagreement decay and agreement decay on attribute address learned from a data set that contains 1871 European Patent from 359 inventors in years of 1978-2003. We observe that while inventors over different periods of time seldom shared the same address over time, many of them have changed their address over time.

There are two extensions for time decays. First, Chiang *et al.* [7] define *recurrence rate*, which considers the

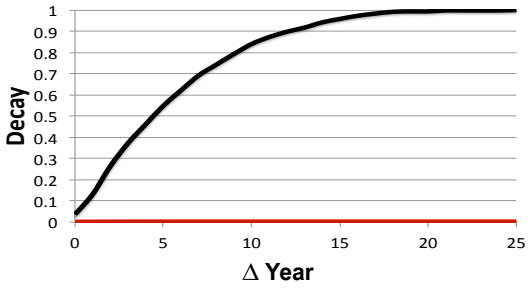


Figure 3: Agreement (flat line) and disagreement decay (curved line) for inventor address learned from a European patent data set [34].

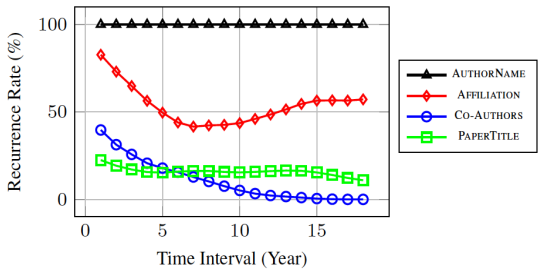


Figure 4: Recurrence rate on four attributes learned from DBLP [7].

probability of the same value re-occurring for an entity attribute. The intuition is that after a value changes, it may change back at a future time point; for example, a person may return to the same affiliation, may keep co-authoring with the same authors after a period of time, and so on. Figure 4 shows the recurrence rate on four attributes learnt from DBLP.

Second, Li *et al.* [32] consider a finer-grained disagreement decay, called the *transition probability*. Intuitively, the transition probability captures the likelihood of a particular (type of) value transitioning to different other (types of) values. These probabilities vary widely across different pairs of values. For example, it is far more likely for someone with the title of an *"Associate Professor"* to transit to the value *"Full Professor"* than *"Accountant"*. Figure 2 shows the transition rate between different types of values for affiliation learned from DBLP.

4.2.2 *Two-stage temporal clustering*

In addition to time decay, which is used in pairwise comparison, Li *et al.* [34] also propose a two-stage temporal clustering strategy, where the first stage requires high value consistency and obtains high-precision results, and then the second stage adjusts results from the first stage to improve recall. The key intuition is that, unlike traditional clustering techniques that are time-agnostic, considering the time order of records can often provide important clues for correct record linkage. The two stages proceed as follows.

1. The first stage considers the records in temporal order. It compares a record with each previously created cluster and computes a probability of merging. After processing all the records, it makes the clustering decision to maximize the overall probability of record-cluster matching.

2. The second stage augments the clustering results by also comparing a record with clusters that are created later. The comparisons take into account record continuity and clusters are adjusted based on these comparisons.

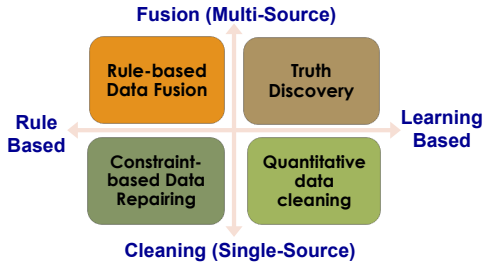


Figure 5: Data cleaning techniques classified along two dimensions.

There are also two extensions for the two-stage clustering method. The solution proposed by Chiang *et al.* [8] makes a clean cut on the two stages: the first stage, called *static stage*, requires high consistency on attribute values and does not consider time decay; the second stage, called *dynamic stage*, considers time decay and further merges clusters generated in the static stage.

In the solution proposed by Li *et al.* [32], the first stage may place a record into multiple different clusters, if it believes that the provider of the record is likely to provide stale data. The second stage then merges clusters from the first stage once it decides that the probability of transition is high, and ignores the clusters consisting of only stale records.

5. CLEANING AND FUSION

5.1 Techniques for conventional data

Data cleaning refers to the following problem: *given a snapshot of data, decide which piece of data is incorrect w.r.t. the ground truth and find the fix.*

As shown in Figure 5, data cleaning techniques can be classified along two dimensions. The first dimension concerns whether the data cleaning process considers only a single source or data over multiple sources. The term *data cleaning* generally refers to the task of

cleaning a single data source whereas the term *data fusion* is generally used to refer to the task of merging and cleaning data from multiple sources [16]. The second dimension concerns whether the underlying technology used for data cleaning is *rule-based* or *learning-based*. The combinations along these two dimensions give rise to four categories of methods.

- **Rule-based data cleaning–*Constraint-based data repairing*:** This type of data cleaning relies on the use of constraints, such as functional dependencies and, more recently, CFDs (Conditional Functional Dependencies), to specify the relationship between data values [22]. When such relationships are violated, the data are typically cleaned (or repaired) by finding the smallest set of changes to the data values such that the constraints are no longer violated.
- **Learning-based data cleaning–*Quantitative data cleaning*:** Such methods (e.g., [11, 26]) apply statistical techniques to detect outliers of the data. It suggests cleaning strategies such that the cleaned data have close distribution to an ideal data set; however, a quantitative repair is not guaranteed to be logically consistent.
- **Rule-based data fusion–*Declarative data fusion*:** Such methods (e.g., [3]) specify rules such as computing the average value from a list of values, finding the most popular value or the latest update to resolve conflicting values from multiple sources.
- **Learning-based data fusion–*Truth discovery*:** Such methods (e.g., [35]) find the truths that are consistent with the real world by applying machine learning models that consider trustworthiness of sources and copying relationships between the sources.

5.2 Techniques for temporal data

Temporal data add a new dimension, the time dimension, to the problem. The problem thus becomes the following: *given a set of temporal data, decide which piece of data is incorrect for the claimed time point or period of time and find the fix.*

Temporal data raise two new challenges for data cleaning and fusion. First, as mentioned before, the true values can evolve over time; so it is critical to distinguish between false data and out-of-date data, and to decide the time period that a value is true. Second, there are many causes for low-quality data: in addition to providing false and imprecise data, a previously good data source may stop updating (a part of) data or updating data infrequently and thus provide stale data.

To overcome these challenges, observe that (1) the update history from data sources can give hints for changes of the real world; for example, a source may remove information about a restaurant because the restaurant is closed; and (2) certain attribute values observe partial order; for example, for marry_status, value *single* should occur before *married* in a timeline, which in turn should occur before *divorced* in time.

We next describe how each category of data-cleaning techniques are extended for temporal data. We omit the category of learning-based data cleaning, as we are not aware of any quantitative cleaning strategies for temporal data.

5.2.1 Rule-based data cleaning: data currency

Fan *et al.* studied *data currency* [23] for cleaning temporal data. It considers data *without* timestamps as input, and aims at finding the *up-to-date* values.

The key idea of the solution is to extend conventional database constraints with *currency constraints*, which express currency relationships from the semantics of the data. An example currency constraint is stated below:

$$\forall s, t\text{: } s[\mathsf{status}] = \text{“married”} \wedge t[\mathsf{status}] = \text{“single”} \rightarrow t \prec_{status} s$$

This constraint states that given two tuples s and t about the same entity (*i.e.*, having the same EID), if s provides value *married* for marriage status while t provides value *single*, then s must have a more recent value than t on status.

In addition to currency constraints, their work also considers a simple class of CFDs, called *constant CFDs*. A constant CFD is a constraint that asserts an equality to a constant value. A *specification* is defined as a triple $S_e = (I_t, \Sigma, \Gamma)$, where I_t specifies existing knowledge on partial order of tuples regarding their timestamps, Σ denotes currency constraints, and Γ denotes constant CFDs.

Fan *et al.* [23] investigated four problems around data currency.

- *Satisfiability:* Decide if a specification is *valid*; in other words, whether the partial orders, the currency constraints, and the constant CFDs have conflicts. It is shown that this problem is NP-complete in general.
- *Implication:* Decide if any other currency orders can be implied by a given specification. It is shown that the problem is coNP-complete in general.
- *True value deduction:* Decide whether true values of an entity can be derived from a specification. Like implication, this problem is coNP-complete in general.
- *Coverage analysis:* Decide the minimum coverage of a given specification (*i.e.*, additional partial orders or currency constraints to be provided) to make true

values derivable. This problem is Σ_2^p-complete in general.

It is also proved that the above complexity results remain even if only currency constraints or only constant CFDs are present.

Although all these problems are intractable in general, Fan *et al.* [23] developed practical algorithms that integrate inferences of data consistency and currency in a single framework, deriving true values and finding a minimum set of attributes that require users' input to find their true values.

5.2.2 Rule-based data fusion: preference-aware union

Alexe *et al.* [2] propose a *preference-aware union* operator for fusing temporal data from multiple data sources. The intended use of the preference-aware union operator is typically at the final step of an entity integration workflow, where the outcome of this step is the set of integrated entity profiles derived by aggregating information from multiple sources.

Intuitively, the operator takes as input temporal data (from multiple sources) and resolves temporal conflicts that may occur in the data according to a given set of constraints and user-specified *preference rules*. For example, a constraint may assert that a person can have at most one affiliation at any time point, and a conflict arises if the input data corroborates there is more than one affiliation at some time point. A preference rule may state that one should prefer the values from a later timestamp or it may specify that one should prefer values from one source over another. With the given constraints and preference rules, the operator will then resolve a conflict, to the extent possible, to conclude what should be the affiliation at the time points when conflicts occur according to the specified preferences.

In the event that not all conflicts can be resolved through preferences, one can then enumerate each possible consistent interpretation of the result returned by the operator at a given time point through a polynomial-delay algorithm. A key property of the solution is that the operator produces the same integrated outcome, modulo representation of time, regardless of the order in which data sources are integrated.

5.2.3 Learning-based data fusion: temporal data fusion

Temporal data fusion takes data *with* timestamps as input, and tries to decide not only the currently correct values, but also *the correct values in the history and their valid time period*.

Specifically, consider a set $\mathcal{D}$ of *data items*, each describing the attribute of an entity, such as affiliation of a person. A data item is associated with a value at each particular time t and can be associated with different values at different times. The *life span* of a data item D is defined as a sequence of *transitions*, each associated with a value change regarding D at a particular time point. On the other hand, consider a set $\mathcal{S}$ of data sources, each providing values for data items in $\mathcal{D}$ and can change the data over time. Data provided by the sources are observed at different times; by comparing an observation with its previous observation, a series of *updates* can be inferred. Given $\mathcal{D}$ and $\mathcal{S}$, the *temporal data fusion* problem computes the life span of each data item in $\mathcal{D}$.

The solutions to this problem typically contains two major parts. The first component computes quality metrics of the sources for temporal data. The second component conducts inferences to decide the lifespan of each data item.

Source quality: For conventional static data, source quality is typically measured by its accuracy [35]. The metrics are far more complex for the dynamic case [14]. Ideally, a high-quality source should provide a new value for a data item *if and only if*, and *right after*, the value becomes true. These three conditions are captured by three measures: (a) the *coverage* of a source measures the percentage of all transitions of different data items that it captures (by updating to the correct value); (b) the *exactness* is the complement of the percentage of transitions that the source mis-captures (by providing a wrong value); and (c) the *freshness* is a function of time ΔT, measuring among the captured transitions, the percentage that are captured within time ΔT. These three measures are orthogonal and collectively referred to as the *CEF-measure*.

Inference: There are two inference approaches to decide the lifespan of a data item. The first inference approach uses Bayesian analysis [14]. Consider a data item $D \in \mathcal{D}$. To discover its life span, both the time and the value of each transition need to be decided. The Bayesian analysis is based on the CEF-measures of D's providers.

1. First, decide the value of D at a beginning time point.
2. Then, find for D's next transition the most likely time point and the most likely value, and repeat this process until it is decided that there is no more transition.

The second approach applies the Markov model [39]. As in the aforementioned Bayesian analysis, the Markov model considers the delay between real-world changes and source observation, and the delay between source observation and source update. However, there are two differences. First, the Markov model additionally encodes domain knowledge such as the partial order between values for a particular attribute. Second, it as-

sumes that the data contains only minor errors such as mis-spellings, and ignores possible erroneous data.

Finally, note that there can be copying relationships between the sources and the copying relationships may even change over time. Dong *et al.* [14] describe how to apply a Hidden Markov Model model to find such evolving relationships.

6. LOOKING FORWARD

There are many problems related to the extraction and integration of temporal data that are yet to be solved.

First, the Web is a rich source of information. It is possible to extract information from the Web, connect the dots, and recover the history of an entity. For example, by extracting information about the talks, tweets, or even resumes and home pages, one can build a profile of the affiliation (and even location) of a person over time. How can one automatically find the relevant web pages and tweets? How can one automatically ingest these sources of information and combine the temporal information for different attributes?

Second, as it is already challenging to build history for a single entity, it is even more challenging to build history for collective entities, such as the history of Rome, the history of World War II, and the history of rock music. The key here is to identify relevant single entities such as people and events, rank them according to their importance regarding the collective entity, and build the history taking into account the time dimension.

Third, as the facts for an entity are evolving over time, people's perspective on the entity can also evolve. For example, people used to believe that the earth is flat, and it is only in the recent two centuries that people started to believe that human evolved from apes. Distinguishing the fact changes and the perspective changes proposes new challenges in managing temporal data.

Finally, while there is already substantial body of work on temporal text extraction and temporal entity resolution, earlier foundational work on mapping and data exchange, conflict resolution, and query answering in inconsistent and incomplete databases have been largely time-agnostic. It will be desirable to develop a principled framework for managing inconsistent temporal data and managing incompleteness (that may change with time) in temporal data.

7. REFERENCES

[1] M. Al-Kateb, A. Ghazal, A. Crolotte, R. Bhashyam, J. Chimanchode, and S. P. Pakala. Temporal query processing in teradata. In *EDBT*, pages 573–578, 2013.

[2] B. Alexe, M. Roth, and W. Tan. Preference-aware integration of temporal data. *PVLDB*, 8(4):365–376, 2014.

[3] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2009.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[5] D. Burdick, M. A. Hernández, H. Ho, G. Koutrika, R. Krishnamurthy, L. Popa, I. Stanoi, S. Vaithyanathan, and S. R. Das. Extracting, linking and integrating data from public sources: A financial case study. *IEEE Data Eng. Bulletin.*, 34(3):60–67, 2011.

[6] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *PVLDB*, 1(1):538–549, Aug. 2008.

[7] Y.-H. Chiang, A. Doan, and J. F. Naughton. Modeling entity evolution for temporal record matching. In *Sigmod*, 2014.

[8] Y.-H. Chiang, A. Doan, and J. F. Naughton. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB*, pages 469–480, 2014.

[9] J. Chomicki. Temporal query languages: A survey. In *ICTL*, pages 506–534, 1994.

[10] J. Chomicki and D. Toman. Temporal databases. In *Foundations of Artificial Intelligence*, pages 429–467. Elsevier, 2005.

[11] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, 2003.

[12] A matter of time: Temporal data management in db2 10, 2012. `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/`.

[13] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[14] X. L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1):562–573, 2009.

[15] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, 2014.

[16] X. L. Dong and F. Naumann. Data fusion - resolving data conflicts for integration. *PVLDB*, 2(2):1654–1655, 2009.

[17] X. L. Dong and D. Srivastava. *Big Data Integration*. Morgan & Claypool, 2015.

[18] X. L. Dong and W. Tan. A time machine for information: Looking back to look forward. *PVLDB*, 8(12):2044–2055, 2015.

[19] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. *ACM Transactions on the Web (TWEB)*, 1(2):7, 2007.

[20] The EDGAR Public Dissemination Service. `http://www.sec.gov/edgar.shtml`.

[21] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and M. Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, 2011.

[22] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[23] W. Fan, F. Geerts, N. Tang, and W. Yu. Conflict resolution with data currency and consistency. *ACM Journal of Data and Information Quality*, 5, 2014.

[24] E. Filatova and E. Hovy. Assigning time-stamps to event-clauses. In *Workshop on Temporal and spatial inf. proc. -Volume 13*, page 13, 2001.

[25] D. Graus, M.-H. Peetz, D. Odijk, O. de Rooij, and M. de Rijke. yourhistory–semantic linking for a personalized timeline of historic events. *Workshop: LinkedUp Challenge at Open Knowledge Conference (OKCon) 2013*, 2013.

[26] J. Hellerstein. Quantitative data cleaning for large databases. Technical report, UC Berkeley, 2008.

[27] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. YAGO2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW*, pages 229–232, 2011.

[28] W. Hua, Z. Wang, H. Wang, K. Zheng, and X. Zhou. Short text understanding through lexical-semantic analysis. In *ICDE*, pages 495–506, 2015.

[29] Big Data, Meet Long Data. `http://www.informationweek.com/big-data/big-data-analytics/big-data-meet-long-data/d/d-id/1109325?`, Apr 1, 2013.

[30] Internet Archive Wayback Machine. http://waybackmachine.org, Aug. 26, 2011.

[31] J.-T. Kim and D. I. Moldovan. Acquisition of semantic patterns for information extraction from corpora. In *Artificial Intelligence for Appl.*, pages 171–176, 1993.

[32] F. Li, M. L. Lee, W. Hsu, and W.-C. Tan. Linking temporal records for profiling entities. In *SIGMOD*, 2015.

[33] J. Li and C. Cardie. Timeline generation: tracking individuals on twitter. In *WWW*, pages 643–652, 2014.

[34] P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.

[35] X. Li, X. L. Dong, K. Lyons, W. Meng, , and D. Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 2013.

[36] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1-2):1338–1347, 2010.

[37] X. Ling and D. S. Weld. Temporal information extraction. In *AAAI*, volume 10, pages 1385–1390, 2010.

[38] A. Mazeika, T. Tylenda, and G. Weikum. Entity timelines: Visual analytics and named entity evolution. In *CIKM*, pages 2585–2588, 2011.

[39] A. Pal, V. Rastogi, A. Machanavajjhala, and P. Bohannon. Information integration over time in unreliable and uncertain environment. In *WWW*, pages 789–798, 2012.

[40] R. Qian. Timeline: Understanding Important Events in Peoples Lives. http://blogs.bing.com/search/2014/02/21/timeline-understanding-important-events-in-peoples-lives/, February 2014. Last retrieved on Oct 27, 2014.

[41] M. Roth and W.-C. Tan. Data integration and data exchange: It's really about time. In *CIDR*, 2013.

[42] R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

[43] R. T. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, 1986.

[44] J. Strötgen and M. Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Intl. Workshop on Semantic Evaluation*, pages 321–324, 2010.

[45] T. A. Tuan, S. Elbassuoni, N. Preda, and G. Weikum. Cate: context-aware timeline for entity illustration. In *WWW*, pages 269–272, 2011.

[46] Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum. Timely yago: harvesting, querying, and visualizing temporal knowledge from wikipedia. In *EDBT*, pages 697–700, 2010.

[47] G. Weikum, N. Ntarmos, M. Spaniol, P. Triantafillou, A. A. Benczúr, S. Kirkpatrick, P. Rigaux, and M. Williamson. Longitudinal analytics on web archive data: It's about time! In *CIDR*, pages 199–202, 2011.

[48] Stop hyping big data and start paying attention to 'long data'. `http://www.wired.com/2013/01/forget-big-data-think-long-data/`, Jan 29, 2013.

[49] F. Wu, R. Hoffmann, and D. S. Weld. Information extraction from wikipedia: Moving down the long tail. In *SIGKDD*, pages 731–739, 2008.

A Survey on Accessing Dataspaces

Yihan Wang
Tsinghua University, Beijing, China
yihanwang@tsinghua.edu.cn

Shaoxu Song*
Tsinghua University, Beijing, China
sxsong@tsinghua.edu.cn

Lei Chen
The Hong Kong University of Science and Technology
leichen@cse.ust.hk

ABSTRACT

Dataspaces provide a co-existence approach for heterogeneous data. Relationships among these heterogeneous data are often incrementally identified, such as object associations or attribute synonyms. With the different degree of relationships recognized, various query answers may be obtained. In this paper, we review the major techniques for processing and optimizing queries in dataspaces, according to their different abilities of handling relationships, including 1) simple search query without considering relationships, 2) association query over object associations, 3) heterogeneity query with attribute correspondences, and 4) similarity query for similar objects. Techniques such as indexing, query rewriting, expansion, and semantic query optimization are discussed for these query types. Finally, we highlight possible directions in accessing dataspaces.

1. INTRODUCTION

A dataspace system [8, 10] processes data, with various formats, accessible through many systems with different interfaces, such as relational [11], sequential [30], XML [32], RDF [31], etc. Unlike data integration over DBMS, a dataspace system does not have full control on its data, and gradually integrates data as necessary.

Dataspace is a data co-existence approach, which provides base functionality over various data sources, regardless of how integrated they are. A dataspace system may return best-effort approximate answers, from multiple sources, where a set of correct semantic mappings have not been applied. In addition, dataspace query answering also takes into consideration a sequence of earlier queries leading up to it, not only for optimizing potential future queries (see [26]), but also for creating better semantic integration between sources in a dataspace [10].

Dataspace systems could apply existing approximate or uncertain mappings and keyword search techniques, however, as also indicated in [10], these works need to be generalized considerably to cases where we do not have semantic mappings of sources and where the data models of the sources differ. In particular, since dataspaces are loosely coupled, rather that providing exact answers, the goal of dataspaces is best-effort query answering.

Some of the challenges in accessing dataspaces have been (partially) addressed, such as studying a sequence of earlier queries (for query optimization and potentially better semantic integration), or ranking answers from multiple sources with various levels of semantic mappings. Other challenges remain open, e.g., handling inconsistencies in dataspaces. (See a detailed discussion in Section 7).

In this paper, we focus on search queries for accessing dataspaces, i.e., returning items already existing in the dataspace. Since well-established semantic mappings are often unavailable in dataspaces, more complex SPJ queries with transformations and views are not considered in this survey. Being aware of different levels of connections, we categorize recently proposed techniques for searching dataspaces into four potential categories.

Simple Search Query

Since the relationships between data objects may barely be obtained, it is necessary to provide an elementary way to access all the data in a dataspace. Simple search query, e.g., keyword queries (with or without attribute names) or attribute predicate queries, is a good choice to meet such requirements [16]. It returns a set of dataspace resources (objects) that directly match the query predicates without considering associations. Inverted index is thus naturally extended to dataspaces for efficient keyword query processing (see details in Section 3).

Association Query

A number of associations among objects may be naturally embedded in dataspace initialization, such as the tree relationships between file objects in a

*Corresponding Author

personal information management dataspace. The object associations could also be incrementally specified, e.g., all the persons graduated from the same university, known as *association trails*. These associations between objects are often modeled in a graph structure. Association query, performed on the association graph, returns not only the objects matching the query specified contents, but also those related objects connected via associations. Extensions on index for associations are discussed as well in Section 4.

Heterogeneity Query

Besides associations among objects, relationships between heterogeneous attributes could also be considered when querying a dataspace. For example, the matching between attributes "manu" and "prod" indicates that both of them specify similar information about manufacturers or producers. Such attribute matching is often recognized by schema matching techniques [18]. As mentioned, in dataspaces, a pay-as-you-go style [12] is usually applied to gradually identify attribute matching (according to users' feedback when necessary). The heterogeneity query extends query results by considering the matching relationships between heterogeneous attributes. For instance, a heterogeneity query on attribute manu searches not only the mentioned manu but also the identified prod attribute. Query rewrite is often employed for such query expansion (see details in Section 5).

Similarity Query

Rather than simply specifying keywords, a more advanced query may pose an object and return dataspace objects that are similar to the query object, known as similarity query. To accelerate similarity query processing, semantic query optimization [3, 13] can be employed. It relies on data dependencies introduced in dataspaces for query rewriting (in Section 6).

The remainder of this paper is organized as follow. In Section 2, we introduce models for representing dataspaces. The aforesaid four types of queries are discussed from Sections 3 to 6, respectively. Finally, we summarize this paper in Section 7 and discuss possible future directions.

2. MODELING

Owing to heterogeneity, dataspaces need to employ an elementary model to represent the most common part of data from various sources, and provide a very basic accessing way to begin with.

2.1 Data Model

The data model indicates how the contents of objects in dataspaces are organized. Usually, the objects consist of values on several attributes.

2.1.1 Triple Store

Since the objects in a dataspace are collected from various heterogeneous sources with distinct attributes, it is obviously inappropriate to manage the data as tables with fixed columns like the relational model. Instead, the data are often modeled as a collection of triples [12, 5], in the form of

$$\langle object, attribute, value \rangle.$$

For example, Table 1 presents 6 triples, recording the (School)Color attribute values of 6 objects.

Table 1: Triples in a dataspace

	$\langle object, attribute, value \rangle$
t_0	⟨Wisconsin, SchoolColor, Cardinal⟩
t_1	⟨Cal, SchoolColor, Blue⟩
t_2	⟨Washington, SchoolColor, Purple⟩
t_3	⟨Berkeley, Color, Navy⟩
t_4	⟨UW-Madison, Color, Red⟩
t_5	⟨Stanford, Color, Cardinal⟩

It is worth noting that the triples can be converted to graph representation. In order to present examples more intuitively, we use graph representation by default in the remainder of this paper.

2.1.2 Resource Views

The triple store scatters attribute values of an object, which may not be efficient in object oriented retrieval. The iMeMex Data Model (iDM), specialized for personal data management [4, 20], introduces an object oriented data model, by grouping the attribute-value pairs of an object together, known as *resource views*.

Table 2: Components of a resource view RV_i

Component	Description
$RV_i.name$	Name of a resource view RV_i
$RV_i.tuple$	Set of attribute value pairs $(att_0 : value_0), (att_1 : value_1), \ldots$
$RV_i.content$	Finite byte sequence of content

Table 2 lists the components that can be attached to a resource view RV_i. Besides the set of attribute-value pairs stored in $RV_i.tuple$, other components,

such as a finite byte sequence of content (a file in personal information management), could be further attached to a resource view.

2.2 Object Association

As mentioned in the introduction, we cannot identify ahead of time all the associations of objects in dataspaces. Ad hoc recognized associations among objects are managed in an extraordinary manner.

2.2.1 Association Graph

In iDM [20], the associations are modeled as a graph, $G := (\mathcal{RV}, E)$, where $\mathcal{RV} := \{RV_1, \ldots, RV_n\}$ denotes a set of n resource views (nodes). E is a sequence of directed edges between resource views.

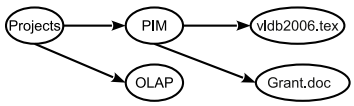


Figure 1: An example graph of associations among resource views (objects)

For instance, Figure 1 presents an example association graph of 5 resource views. The edges denote the associations between the corresponding resource views, e.g., (Projects→PIM) indicates that PIM is a project under the directory of all Projects in a personal file management dataspace.

2.2.2 Association Triple

The association between objects in a dataspace can be represented as a collection of triples as well, in the form of $\langle object, association, object \rangle$ [5]. Unlike the edges without any label in the aforesaid association graph, a type is indicated in the association triple for the association between two objects.

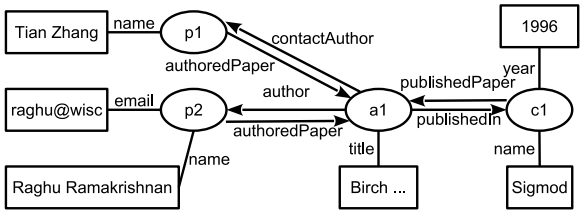


Figure 2: An example triple base

Figure 2 illustrate an example dataspace of 4 objects (p1, p2, a1 and c1, denoted by ellipses). Rectangles attached to each object denote its attribute values, e.g., "Tian Zhang" attached to p1 through an edge with label "name", representing an attribute value pair (name:Tian Zhang) of object p1. Each association triple corresponds to an edge in Figure 2 with a label indicating its association type. For example, ⟨a1, contactAuthor, p1⟩ means that a1 is connected to p1 as a contactAuthor.

2.2.3 Association Trail

Rather than representing specific single associations, an *association trail* [19], denotes a group of associations by a join predicate θ. Instead of originally embedded in data, such associations can be gradually declared.

Let $Q_L(G)$ and $Q_R(G)$ be two collections of objects specified by queries Q_L and Q_R, respectively, over an association graph G. An association trail $\Phi : Q_L \overset{\theta(l,r)}{\Longrightarrow} Q_R$ represents all the associations from objects in $Q_L(G)$ to objects in $Q_R(G)$ according to $\theta(l, r)$. It conceptually introduces in the association graph a directed edge from left to right and labeled Φ, for each pair of nodes given by $Q_L \bowtie_\theta Q_R$, namely intensional edge. Association trails cover relational and non-relational theta-joins as special cases. A bidirectional association trail $\Phi : Q_L \overset{\theta(l,r)}{\Longleftrightarrow} Q_R$ represents associations in both directions.

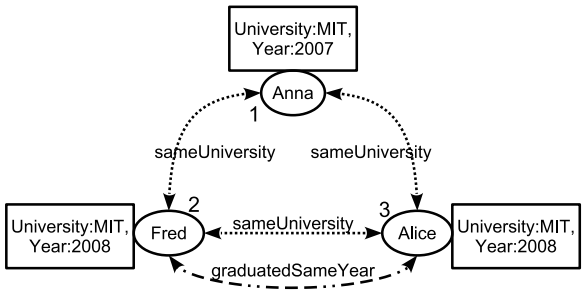


Figure 3: Example association trails

Consider the example in Figure 3. Let ellipses denote objects, while the rectangle attached to each object lists its attribute-value pairs. A bidirectional association trail is given by

$$sameUniversity : class = person \overset{\theta(l,r)}{\Longleftrightarrow} class = person,$$
$$\theta(l, r) : (l.university = r.university).$$

Arrows with dotted lines, labeled sameUniversity, represents 3 intensional edges introduced by the association trail. For instance, the intensional edge between Anna and Fred indicates that they share the same *university* attribute value.

2.2.4 Other Associations

Besides the aforesaid general associations between objects, a special category of "referenceOf" relationships in personal data are considered, namely *Context-Based Reference (CR)* [14]. Such associations are generated by user behaviors. Similar to association trails, the CR associations could be introduced gradually and represented in association graphs as well.

2.3 Semantic Correspondences

While associations represent the relationships between objects, the attribute correspondences indicate the identity relationships of attributes.

2.3.1 Attribute Synonyms

Synonym correspondence between two attributes (e.g., manu vs. prod) can be identified by schema matching techniques in data integration (see [18] for a survey). In dataspaces, the synonym correspondence between attributes are often incrementally recognized in a pay-as-you-go style [12]. Automated mechanisms such as schema matching and reference reconciliation provide initial correspondences, termed candidate matches, and then user feedback is used to incrementally confirm these matches (when necessary).

Two attributes A, B having synonym correspondence are denoted by $A \leftrightarrow B$, e.g., manu $\leftrightarrow$ prod. There may exist multiple attributes having synonym correspondences to an attribute A. A synonym table is introduced in [5] for attributes with correspondence. If attribute A is referred to as $A_1, \ldots, A_n$ in different data sources, having $A \leftrightarrow A_1, \ldots, A \leftrightarrow A_n$, the canonical name of A is chosen as one of $A_1, \ldots, A_n$.

2.3.2 Trails

Instead of the symmetric synonyms relationships between attributes, the concept of *trail* is also proposed to specify asymmetric correspondences [20].

A unidirectional trail is denoted as $\Psi : Q_L \rightarrow Q_R$. It means that the query on the left Q_L induces the query on the right Q_R, i.e., whenever we query for Q_L we should also query for Q_R.

For example, a trail

$$\Psi_1 : //*.tuple.created \rightarrow //*.tuple.date$$

indicates that whenever querying objects (resource views) on attribute *created*, it also needs to query objects on attribute *date*.

A bidirectional trail is denoted as $\Psi : Q_L \leftrightarrow Q_R$. It further indicates that the query on the right Q_R induces the query on the left Q_L.

Indeed, trails can specify correspondences between more general queries. For example, they could be used to transform a simple keyword query into a query to the mediated data source. Consider a trail "$dataspace \rightarrow //Projects/PIM/*$". With such a trail, the query will not only return resources containing keyword *dataspace*, but also "vldb2006.tex" and "Grant.doc" in Figure 1 (although the keyword *dataspace* does not appear in these documents).

Traditional data integration approaches, such as GAV, LAV, and GLAV, need high upfront effort to semantically integrate all source schemas and provide a mediated schema. Trails provide a declarative mechanism to enable semantic enrichment of a dataspace in a pay-as-you-go fashion.

2.3.3 Probabilistic Mapping

Instead of the synonym correspondence based on certain schema mapping, a probabilistic mapping describe a probability distribution of a set of possible schema mapping [21].

A probabilistic mapping is defined as $(S, T, \mathbf{m})$, where S and T are relations belonging to source schema and target schema respectively, and $\mathbf{m}$ is a set whose elements m_i consist of a one-to-one mapping between S and T and a probability, indicating the probability of each mapping.

For example, consider three possible mappings in Table 3. Each possible mapping consists of a set of attribute correspondences, and is associated with a corresponding probability. Note that the sum of all probability is 1.

It is worth noting that even though the studies [6, 7] focus now on attribute correspondences, probabilistic schema mappings could be much more complex semantic correspondences, e.g., more general GLAV mappings.

Table 3: Probabilistic schema mapping

	Possible Mapping	Prob
m_1	{(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-addr)}	0.5
m_2	{(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-addr)}	0.4
m_3	{(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

3. SIMPLE SEARCH QUERY

The primary and easy way for most people accessing dataspaces is the keyword predicate query [5]. Each predicate is of the form (attribute : keyword), denoted by $(A : K)$, where A is an attribute name and K is a keyword in the value of attribute A. For example, a keyword predicate query could be:

$$\{(title : Birch), (author : Raghu)\}.$$

To answer the keyword predicate query, we consider both the query and objects as sets of *items* of

attribute keyword pairs. An object O (or a query Q) in dataspaces is thus a set of items $\{(A_1 : K_1), (A_2 : K_2), \ldots, (A_{|O|} : K_{|O|})\}$. For example, an object with attribute value $(manu : Apple\ Inc.)$ can be represented by a set of items $\{(manu : Apple), (manu : Inc.)\}$, if each word is considered as a keyword.

The keyword predicate query returns the objects in the dataspace that match most items. Query answers are ranked by the following matching score in descending order $score(Q, O) = |Q \cap O|$.

A similar Filter operator is also considered in [11]. It returns resources (objects) satisfying the given conditions, which can be specified as a set of (attribute : keyword) pairs as well.

3.1 Index

Inverted lists are utilized for indexing dataspaces [5]. The *inverted index*, also known as *inverted files* or *inverted lists*, consists of a vocabulary of items and a set of inverted lists. Each item e corresponds to an inverted list of object IDs, where each ID reports the occurrence of item e in that object.

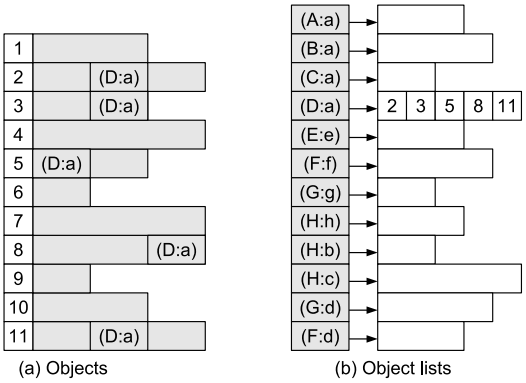


Figure 4: Indexing by attribute inverted lists (ATIL) for items of attribute-keyword pairs

The items of objects in dataspaces are indeed attribute-keyword pairs. The *attribute inverted lists* (ATIL) are lists of objects where the corresponding item (attribute-keyword pair) appears [5]. Figure 4 shows an example dataspace $\mathcal{S}$ which consists of 11 objects with a vocabulary $\mathcal{I}$ of 12 items. For each item (an attribute-keyword pair), we have a pointer referring to a specific list of object IDs, where the item appears. For instance, consider the inverted list of item (D : a) in Figure 4(b). It indicates that the keyword a on attribute D appears in the objects 2, 3, 5, 8, 11, as presented in in Figure 4(a). In real implementation, each object ID in the list is associated with a value, which denotes the weight of the item (D : a) in that object.

3.2 Compression

The heavy cost of inverted index based query processing arises from two aspects, 1) I/O cost of reading inverted lists from disks, and 2) aggregation of these inverted lists for ranking results. To reduce the cost of retrieving and merging lists, compression of inverted lists is studied [22]. The basic idea is to store the merged lists for items that appear together frequently in queries or dataspace objects.

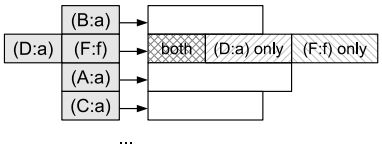


Figure 5: Compressing inverted lists on items (an item corresponds to only one list)

For instance, in Figure 5, the inverted lists of items, attribute-keyword pairs (D:a), (F:f), are combined together as a big single list. We have several sections in each compressed list, such as the sublist of tuples with both (D:a) and (F:f), the sublist with only (D:a), and the sublist with only (F:f).

3.3 Materialization

Materialized views in relational databases are often utilized to find equivalent view-based re-writings of relational queries [9], such as conjunctive queries or aggregate queries in databases. The concept of materialization is also extended to dataspaces for exploring query optimization opportunities [26].

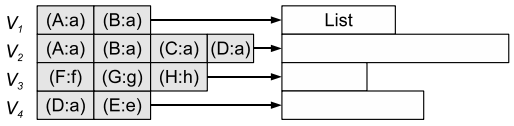


Figure 6: Materializing views of items (an item may be materialized multiple times)

In Figure 6, we show an example of materialized lists of item views. For instance, the first view, denoted by $V_1 = \{(\mathsf{A} : \mathsf{a}), (\mathsf{B} : \mathsf{a})\}$, materializes the merge results of lists corresponding to item $(\mathsf{A} : \mathsf{a})$ and $(\mathsf{B} : \mathsf{a})$ in the example of Figure 4. Without materialization, we need two random disk accesses for query predicates $(\mathsf{A} : \mathsf{a})$ and $(\mathsf{B} : \mathsf{a})$, respectively. In contrast, by materialized views, only one random disk access is needed for the same query. Moreover, besides reducing the I/O cost, we also avoid the aggregation of the aforesaid two separate inverted tuple lists in the query.

4. ASSOCIATION QUERY

Next, we discuss the queries specifying association requirements of objects. Again, both data models, triple store and resource view, support keyword predicate queries.

4.1 Association Search Query

4.1.1 Neighborhood Keyword Query

Neighborhood keyword query extends traditional keyword query by taking associations into account, that is, it also explores associations between data items. A neighborhood keyword query specifies a set of keywords, and the result consists of (1) relevant instance, which contains at least one of the query keywords; (2) associated instance, which is associated with a relevant instance.

For example, consider a query whose keyword is "*Birch*" in Figure 2. Instance a_1 is a relevant instance as it contains "*Birch*" in the title attribute, and p_1, p_2 and c_1 are associated instances as they have association with a_1.

4.1.2 Association Predicate Query

Besides the predicates on attribute values, predicates on associations can also be specified [5], of the form $(R : K)$, where R in the predicate is an association name. Objects satisfy the predicate if they have associations of type R with objects that contain the keyword K in attribute values.

For example, a query "Raghu's Birch paper in Sigmod" can be described with three predicates:

$$\{(title : Birch), (author : Raghu), (publishedIn : Sigmod)\}.$$

The query is satisfied by object a1 in Figure 2, which has attribute value $(title : Birch)$, and in association with c1 (containing keyword *Sigmod*) in type *publishedIn* and p2 (containing keyword *Raghu*) with type *author*.

4.1.3 Indexing Associations

Similar to indexing attribute keywords, *attribute-association inverted lists* (AAIL) are introduced for indexing association information (together with the aforesaid attribute-keyword information). Suppose that an object O has an association R with objects $O_1, \ldots, O_n$ in the dataspace, and each of $O_1, \ldots, O_n$ has the keyword K in one of its attribute values. An inverted list will be generated for the association:value item $(R : K)$, which contains n objects $O_1, \ldots, O_n$ in the list. For instance, in Figure 7, object $a1$ appears in the list of $(publishedIn : 1996)$, since $a1$ is in association (with type *publishedIn*) to another object ($c1$) containing keyword 1996 in its attribute ($year$), as illustrated in Figure 2.

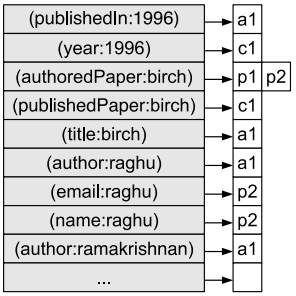


Figure 7: Attribute-association inverted lists (AAIL) for both associations and attribute-keyword items (for the example dataspace in Figure 2)

An association predicate query $\{(R : K1), \ldots, (R : K_n)\}$ can be answered over the AAIL. For example, when searching for "Raghu's papers", the query contains an association predicate $(author : Raghu)$. Based on the AAIL in Figure 2, it returns object $a1$.

4.2 Association Trail Query

Other than query over arbitrary individual associations, [19] considers queries with specific groups of associations, specified by association trails.

4.2.1 Neighborhood Query

Given a query Q (of attribute-keyword items) and an association trail Φ_i, the association trail query results are given by $(Q \cap Q_L^i) \rtimes_{\theta_i} Q_R^i$, where Q_L^i and Q_R^i are the queries on the left and right sides of trail Φ_i, respectively, and θ_i is the θ-predicate of Φ_i.

For example, consider the dataspace in Figure 3 and an association trail

$$sameUniversity : class = person \overset{\theta(l,r)}{\Longleftrightarrow} class = person,$$
$$\theta(l, r) : (l.university = r.university).$$

A query $Q = \{(name : Anna)\}$ with the association trail returns not only object 1 in Figure 3 (matching the query content), but also objects 2 and 3 who share the same university (identified by $\theta(l, r)$).

When considering a set of association trails $\Phi^* := \{\Phi_1, \ldots, \Phi_n\}$, the query results could be the union of the corresponding results given by Φ_i.

4.2.2 Indexing Association Trails

The most intuitive strategy for processing association trail queries is to explicitly materialize all intensional edges in the graph, through a join index [28]. For instance, Figure 8 lists all the objects 2 and 3, which have association to object 1 specified by association trail 1 (sameUniversity). Similarly, objects 1 and 3 have association to object 2 as well.

OidLeft	OidRight	trailList
1	2	{1}
	3	{1}
2	1	{1}
	3	{1, 2}
3	1	{1}
	2	{1, 2}

trailID	trail label
1	sameUniversity
2	graduatedSameYear

Figure 8: Association trail index

At query time, it looks up materialization to obtain the neighborhoods for each object returned by the original query Q. A query $Q = \{(name : Anna)\}$ thus returns object 1 (containing the specified keyword in Q) as well as objects 2 and 3 with the specified association 1 (sameUniversity) to object 1.

lookupEdges

OidLeft	OidRight	trailList
2	1	{1}
3	1	{1}
	2	{2}

trailID	trail label
1	sameUniversity
2	graduatedSameYear

normalEdges

OidLeft	OidRight	trailList
1	2	{1}
	3	{1}
2	3	{2}

Figure 9: Grouping-compressed index (GCI) for association trail

Association (join) relationships can be grouped together as cliques. The grouping-compressed index (GCI) [19] explicitly represents the edges from a given object (node) O_1 in the clique to all the other objects $\{O_2, \ldots, O_C\}$, and for each remaining node in the clique, represents special lookup edges $\langle O_j, O_1, lookup\rangle$ that state O_j connects to the same objects as O_1. Thus, it represents the information in the clique with C normal edges for O_1 plus $C-1$ edges for the lookup edges of all remaining nodes. In short, a reduction in storage space (and consequently join indexing time) from C^2 edges to $2C-1$ edges.

For example, consider the clique of three objects $\{1, 2, 3\}$ in Figure 3. It corresponds to three groups of edges $\{(1, 2), (1, 3)\}$, $\{(2, 1), (2, 3)\}$, $\{(3, 1), (3, 2)\}$, in Figure 8. In the grouping-compressed index as shown in Figure 9, only one group of (normal) edges $\{(1, 2), (1, 3)\}$ is preserved, while other two groups of edges are represented by lookupEdges $(2, 1)$ and $(3, 1)$. The lookupEdges denote that objects 2 and 3 share the same connections as object 1 w.r.t. association trail 1 (sameUniversity). The total number of edges (inverted lists) reduce.

4.3 Path Expression Query

Path expressions (as well as keywords and predicates) have been used in XML search engines referring to the NEXI language [27]. The queries have been adapted to dataspaces by iDM [4, 20] and [32]. A most typical path expression query, $//A//B$, returns resource views named B, from which there exists a path to another resource view named A.

The path expression can also be combined with attribute predicates, such as $//A//B[b = 42]$. It further requires $RV.tuple.b = 42$ for the results specified by $//A//B$. For example,

$$//Projects//Grant[created = \text{“2006”}]$$

returns resource views which are documents entitled *Grant* and *created* in 2006 in all the *Projects*. Consequently, the object *Grant.doc* in Figure 1 will be returned, which has a path from *Project* to *Grant*.

A Traverse operator, similar to the path expression query, is also introduced in [11], which is used to find resources referenced by a property. It accepts a sequence of resources with attribute predicates and a set of conditions. The returned resources satisfy the conditions and have association to the given resources successively, i.e., there is a path from the first resource to the result.

5. HETEROGENEITY QUERY

Dataspace queries could be extended to not only objects with associations, but also attributes with correspondence owing to information heterogeneity.

5.1 Attribute Hierarchy Query

Owing to information heterogeneity, an attribute (such as *name*) may correspond to multiple descendants in hierarchies, e.g., $firstName$, $lastName$, $nickName$, etc. A query $(name : Tian)$ may refer to $firstName$, $lastName$, or $nickName$.

To support such hierarchies in attribute heterogeneity, a natural idea is to index dataspaces with duplication. *Attribute inverted lists with duplication* (Dup-ATIL) [5] is constructed as follows. If the keyword K appears in the value of attribute A_0, and A is an ancestor of A_0 in the hierarchy, then there is a list for $(A : K)$. It records the number of occurrences of K in values of the attribute A and A's sub-attributes. Consequently, a predicate query with the Dup-ATIL is answered in the same way as we use the ATIL.

Indexing Attributes with Hierarchy Path

The size of Dup-ATIL could be very large if the attribute hierarchy contains long paths from the root

attribute to the leaf attributes and most values in the dataspace belong to leaf attributes.

A more concise way is to generalize the inverted lists without introducing duplicates [5]. *Attribute inverted lists with hierarchies* (Hier-ATIL) is constructed by extending the attribute inverted list as follows. Let $A_0, \ldots, A_n$ be attributes such that for each $i \in [0, n-1]$, attribute A_i is the super-attribute of A_{i+1}, and A_0 does not have super-attribute. It introduces a hierarchy path $A_0// \ldots //A_n$ for attribute A_n. For each keyword K in the value of attribute A_n, there is an inverted list generated for $(A_0// \ldots //A_n : K)$. Each object in the list denotes that the keyword K appears in the attribute A_n of the object.

The attribute-keyword predicate query $(A : K)$ can be answered by considering all the inverted lists whose hierarchy paths contain the attribute A.

5.2 Attribute Synonym Query

Besides attribute hierarchies, a more general form of information heterogeneity is the attribute synonyms, $A \leftrightarrow B$, in more arbitrary attribute pairs.

Query Rewrite with Canonical Name

To accommodate synonyms, a straightforward idea is to introduce a synonym table for attribute and association names [5]. If attribute A is referred to as $A_1, \ldots, A_n$ in different data sources, it chooses the canonical name of A as one of $A_1, \ldots, A_n$.

In the index, when a keyword K appears in a value of the A_i attribute, there is an inverted list for $(A : K)$. The object in the list for $(A : K)$ denotes the occurrence of K in its attribute $A_1, \ldots,$ or A_n.

To answer a predicate query with attribute predicate $(A_i, K), i \in [1, n]$, we transform it into a keyword search for $(K : A)$. For example, suppose that *author* is considered as a canonical name for *author* and *authorship*. The predicate $(authorship : Tian)$ will be transformed into $(author : Tian)$.

Query Rewrite with Predicate Expansion

The attribute synonym relationship is often incrementally determined, in a pay-as-you-go style. To avoid updating the existing index (w.r.t. synonym attributes), we may also consider to expand the query with synonym attributes [26], rather than replacing them with canonical names.

Consider a query $Q = \{(A_1 : K_1), \ldots, (A_{|Q|} : K_{|Q|})\}$ specifying predicates on a set of attribute-keyword pairs. The expanded query $\hat{Q}$ of Q is

$$\hat{Q} = \{(B_i : K_i) \mid B_i \leftrightarrow A_i, (A_i : K_i) \in Q\} \cup Q.$$

For example, we consider a query

$$Q = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{post} : \mathsf{Infinite})\}.$$

The query evaluation searches not only in the manu and post attributes specified in the query, but also in the attributes prod and addr according to the attribute synonym correspondences manu $\leftrightarrow$ prod and addr $\leftrightarrow$ post, respectively, having

$$\hat{Q} = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{prod} : \mathsf{Apple}), (\mathsf{post} : \mathsf{Infinite}), (\mathsf{addr} : \mathsf{Infinite})\}.$$

5.3 Trail Query

Rather than simple attribute pairs or hierarchies, trails specify more complicated relationships among attributes of query answers [20]. Query processing in the presence of trails first detects whether a trail should be applied to a given query Q. The matching of a unidirectional trail is performed on the left side of the trail. If the left side of a trail was matched by a query Q, the right side of that trail will be used to compute the transformation of Q. Finally, the original query Q is merged with the transformation as a new query. The new query extends the semantics of the original query based on the information provided by the trail definition.

Query Rewrite with Trails

Let Ψ_i^L and Ψ_i^R denote the left and right sides of a trail Ψ_i, respectively. The rewrite processing consists of three phases: Matching, Transformation, and Merging.

(1) Matching. A trail Ψ_i matches a query Q whenever its left side query, Φ_i^L, is contained in Q as a subset.

(2) Transformation. The query Q is transformed by substituting Ψ_i^L with Ψ_i^R, denoted by $Q^T_{\Psi_i}$.

(3) Merging. The transformed query $Q^T_{\Psi_i}$ is merged with the matched subexpression.

Consequently, the query is expanded not only on Ψ_i^L but also the related Ψ_i^R.

For example, consider a query

$$Q := //Projects//Grant[created = \text{“2006”}],$$

and a trail

$$\Psi_1 := //*.tuple.created \rightarrow //*.tuple.date.$$

The trail states that when querying the *created* attribute of an object (resource view), it should also consider the query on the *date* attribute. Ψ_1 matches Q as its left side query Ψ_1^L is contained in Q. After transformation, we have

$$Q^T_{\Psi_1} := //*[date = \text{“2006”}].$$

The final merged query $Q^*_{\{\Psi_1\}}$ is

$$Q^*_{\{\Psi_1\}} := //Projects//Grant[created=\text{“2006” OR } date=\text{“2006”}].$$

5.4 Query with Probabilistic Mapping

Instead of certain attribute synonym correspondence, the matching of attributes (especially the one generated by auto-matching tools) is often uncertain. Query answering is thus performed on such probabilistic mapping [6, 7, 21].

Consider the possible mappings in Table 3. By-table semantics [6, 7] could be considered for query answering based on p-mapping, i.e., one mapping for all tuples. A query Q with attribute-keyword predicate $\{(mailing\text{-}addr : Sunnyvale)\}$ will be expanded as

$$\hat{Q} = \{\{(current\text{-}addr : Sunnyvale)\}, \{(permanent\text{-}addr : Sunnyvale)\}, \{(email\text{-}addr : Sunnyvale)\}\}.$$

The probability of an object O matching the query is computed by aggregation w.r.t. possible mappings. Suppose that there is an object O with the following attribute-value pairs

$$O = \{(pname : Bob), (email\text{-}addr, bob), (current\text{-}addr : Sunnyvale), (permanent\text{-}addr : Sunnyvale)\}.$$

As shown, two predicates on attributes *current-addr* and *permanent-addr* match, w.r.t. mappings m_1 and m_2, respectively. Referring to the probabilities of m_1 and m_2 in Table 3, the probability of this object O matching the query Q is 0.9.

Besides the simple predicate queries, more complex SPJ queries could be answered based on probabilistic mapping [6, 7, 21]. As aforesaid, semantic mappings are often not well-established in dataspaces, we do not consider the complex SPJ queries in this survey.

In addition, queries could also be answered based on trails with probability variants, named probabilistic trails [20]. Specifically, a probability value $0 \leq p \leq 1$ is assigned to a trail. The probability reflects the likelihood that the results obtained by the trail are correct. Similarly, there is another variant named scored trails, which bind a score to a trail, reflecting the relevance of the trail.

6. SIMILARITY QUERY

Other than given a set of attribute-keyword predicates, a similarity query poses a query object with attribute-value pairs, and returns dataspace objects that are similar to the query object.

Similarity Query Answering

Given a query of attribute value pairs, $Q = \{(A_1 : W_1), \ldots, (A_{|Q|} : W_{|Q|})\}$, the similarity query returns a set of objects, which are similar to Q on each attribute w.r.t. attribute similarity function [25].

A *similarity function* $\theta(A_i, A_j) : [A_i \approx_{ii} A_i, A_i \approx_{ij} A_j, A_j \approx_{jj} A_j]$ specifies a constraint on similarity correspondence of two values from attribute A_i or A_j, according to their corresponding similarity operators $\approx_{ii}, \approx_{ij}$ or $\approx_{jj}$. Here, A_i, A_j are often synonym attributes, and the similarity function often comes together with the attribute matching on how their attribute values should be compared. For instance, a similarity function specified on two attributes $(\mathsf{manu}, \mathsf{prod})$ for the example dataspace in Figure 10 can be

$$\theta(\mathsf{manu}, \mathsf{prod}) : [\mathsf{manu} \approx_{\leq 5} \mathsf{manu}, \mathsf{manu} \approx_{\leq 5} \mathsf{prod}, \mathsf{prod} \approx_{\leq 5} \mathsf{prod}].$$

Two objects O_1, O_2 are said to be similar w.r.t. $\theta(A_i, A_j)$, denoted by $(O_1, O_2) \asymp \theta(A_i, A_j)$, if at least one of three similarity operators in $\theta(A_i, A_j)$ evaluates to true. For example, (t_1, t_2) are similar w.r.t. $\theta(\mathsf{manu}, \mathsf{prod})$, since the edit distance of $(t_1[\mathsf{manu}], t_2[\mathsf{prod}])$ is $4 \leq 5$, satisfying the similarity operator $\mathsf{manu} \approx_{\leq 5} \mathsf{prod}$.

t_1:{(name : iPod), (color : red), (manu : Apple Inc.), (addr : InfiniteLoop, CA), (tel : 567), (website : itunes.com)};
t_2:{(name : iPod), (color : cardinal), (prod : Apple), (post : InfiniteLoop, Cupert), (tel : 123), (website : apple.com)};
t_3:{(name : iPad), (color : white), (manu : Apple Inc.), (post : InfiniteLoop), (phn : 567), (website : apple.com)}.

Figure 10: Example dataspace with three objects

Consider a similarity query Q over dataspace $\mathcal{S}$ with a set Θ of similarity functions. It returns all the objects O in $\mathcal{S}$ with similar attribute values to Q, i.e., for each A_i specified in Q, having $(Q, O) \asymp \theta(A_i, B_i)$ for some $\theta(A_i, B_i) \in \Theta$. For example, let

$$\theta(\mathsf{addr}, \mathsf{post}) : [\mathsf{addr} \approx_{\leq 9} \mathsf{addr}, \mathsf{addr} \approx_{\leq 9} \mathsf{post}, \mathsf{post} \approx_{\leq 9} \mathsf{post}]$$

be another similarity function. Consider a query $Q = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{post} : \mathsf{InfiniteLoop}, \mathsf{CA})\}$. It returns all the 3 objects in Figure 10 as similarity query answers.

Semantic Query Optimization

Integrity constraints (e.g., FDs) can be utilized to rewrite and optimize queries, which is known as the semantic query optimization [3, 13]. In [24, 25], data dependencies are extended for similarity query optimization in dataspaces.

Data dependencies in dataspaces are generally in the form of $\varphi : \theta(A_i, A_j) \rightarrow \theta(B_i, B_j)$ defined on similarity functions $\theta(A_i, A_j)$ and $\theta(B_i, B_j)$. It states that for any two objects O_1, O_2 that are similar w.r.t. $\theta(A_i, A_j)$, it always implies $(O_1, O_2) \asymp \theta(B_i, B_j)$ as well. For example,

$$\varphi_1 : \theta(\mathsf{manu}, \mathsf{prod}) \rightarrow \theta(\mathsf{addr}, \mathsf{post}).$$

states that if the manu or prod values of two objects are similar, then their corresponding addr or post values should also be similar.

For instance, consider again the query object with (post : InfiniteLoop, CA) and (manu : Apple). As mentioned, the similarity query searches not only in the manu, post attributes specified in the query, but also in the synonym attributes prod, addr according to the similarity functions θ(manu, prod) and θ(addr, post), respectively. Recall the semantics of the above dependency φ_1. If (manu, prod) of the query object and a data object are found to be similar, then the data object can be directly returned as answer without evaluation on post, addr since their corresponding (post, addr) values must be similar as well. The query efficiency is improved.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we review several cases of accessing dataspaces (listed in Table 4), depending on the relationship information obtained thus far. When a dataspace system is first launched, with relationships barely identified, simple keyword queries apply. With the pay-as-you-go identification of relationships among data in dataspaces, the query answers are enhanced, e.g., trail queries, or probabilistic query answering with the partial probabilistic mapping. The gradually identified attribute relationships further enable the similarity query.

A dataspace system offers a suite of interrelated services and guarantees, where techniques, such as keyword query and probabilistic query answering, could be applied and considerably generalized [8, 10]. While some of the challenges in accessing dataspaces have been (partially) addressed, such as studying a sequence of earlier queries (for query optimization and potentially better semantic integration), or ranking answers from multiple sources with various levels of semantic mappings, the others remain open, e.g., handling inconsistencies in dataspaces.

We list some advice of future directions.

(1) To address inconsistencies in dataspaces, the comparable dependencies [24, 25] could be applied. However, such a notation is defined at schema level (originally for query optimization). Following the same line of extending data dependencies with conditions in databases, known as conditional dependencies [2, 17], we may also study data dependencies declared with conditions (instances) in dataspaces. For example, we may consider a keyword dependency $\phi : ([A_i \leftrightarrow B_i] : K_i) \rightarrow ([A_j \leftrightarrow B_j] : K_j)$, by extending matching keys [23]. It states that if an object contains a keyword K_i in either attribute A_i or its synonym B_i, then it must also have a keyword K_j appearing in either attribute A_j or B_j.

(2) To answer queries over inconsistent dataspaces, consistent query answering [1] for dataspaces needs to be studied. Existing study [15] could handle uncertainty and inconsistency together, but do not consider the heterogeneous data in dataspaces. In essence, we need to manipulate simultaneously the uncertainty originated from both schema mapping and data inconsistency.

(3) Beyond relational, tree-structured (XML) or graph-structured (RDF), more data types are expected to be supported in dataspaces, e.g., sequential (event logs). While the integration of sequential event data [33, 34] and inconsistency detection [30, 29] have been investigated, searching and consistent query answering over such heterogeneous event sequences, especially ranking together with other data types, remain open.

Acknowledgement

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; China NSFC under Grants 61572272 and 61202008.

8. REFERENCES

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79, 1999.

[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 746–755, 2007.

[3] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query

Table 4: Summary of the surveyed works

Query Case	Method	Detail
Simple Search	index [5]	Extend inverted lists for indexing dataspaces. Cost of list merging could be heavy in query processing
	compression [22]	Compress inverted lists to reduce the cost of retrieving and merging list
	materialization [26]	Materialize inverted lists to reduce the I/O and merging cost. May introduce large additional space cost
Association	association predicate [5]	Consider queries specifying association requirements of objects by keywords or predicates.
	association trail [19]	Consider queries with specific groups of associations, specified by association trails.
	path expression [4]	Consider relationships expressed in paths
Heterogeneity	hierarchy [5]	Handle the situation that an attribute corresponds to multiple descendants in hierarchies
	synonym [5, 26]	Consider attribute synonyms
	trail [20]	Expand query by tail associations
	probabilistic [21]	Enhance semantic mappings by modeling uncertainty
Similarity	dependency [25]	Return similar objects

optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.

[4] J. Dittrich and M. A. V. Salles. idm: A unified and versatile data model for personal dataspace management. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 367–378, 2006.

[5] X. Dong and A. Y. Halevy. Indexing dataspaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 43–54, 2007.

[6] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 687–698, 2007.

[7] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[8] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[9] G. Gou, M. Kormilitsin, and R. Chirkova. Query evaluation using overlapping views: completeness and efficiency. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 37–48, 2006.

[10] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 1–9, 2006.

[11] B. Howe, D. Maier, N. Rayner, and J. Rucker. Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 270–277, 2008.

[12] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 847–860, 2008.

[13] A. Y. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages

163–173, 1995.

[14] Y. Li and X. Meng. Supporting context-based query in personal dataspace. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1437–1440, 2009.

[15] X. Lian, L. Chen, and S. Song. Consistent query answers in inconsistent probabilistic databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 303–314, 2010.

[16] J. Liu, X. Dong, and A. Y. Halevy. Answering structured queries on unstructured data. In *Ninth International Workshop on the Web and Databases, WebDB 2006, Chicago, Illinois, USA, June 30, 2006*, 2006.

[17] S. Ma, W. Fan, and L. Bravo. Extending inclusion dependencies with conditions. *Theor. Comput. Sci.*, 515:64–95, 2014.

[18] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[19] M. A. V. Salles, J. Dittrich, and L. Blunschi. Intensional associations in dataspaces. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 984–987, 2010.

[20] M. A. V. Salles, J. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. itrails: Pay-as-you-go information integration in dataspaces. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 663–674, 2007.

[21] A. D. Sarma, X. L. Dong, and A. Y. Halevy. Uncertainty in data integration and dataspace support platforms. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 75–108. Springer, 2011.

[22] S. Song and L. Chen. Indexing dataspaces with partitions. *World Wide Web*, 16(2):141–170, 2013.

[23] S. Song, L. Chen, and H. Cheng. On concise set of relative candidate keys. *PVLDB*, 7(12):1179–1190, 2014.

[24] S. Song, L. Chen, and P. S. Yu. On data dependencies in dataspaces. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 470–481, 2011.

[25] S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *VLDB J.*, 22(2):253–274, 2013.

[26] S. Song, L. Chen, and M. Yuan. Materialization and decomposition of dataspaces for efficient search. *IEEE Trans. Knowl. Data Eng.*, 23(12):1872–1887, 2011.

[27] A. Trotman and B. Sigurbjörnsson. Narrowed extended xpath I (NEXI). In *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl Castle, Germany, December 6-8, 2004, Revised Selected Papers*, pages 16–40, 2004.

[28] P. Valduriez. Join indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.

[29] J. Wang, S. Song, X. Lin, X. Zhu, and J. Pei. Cleaning structured event logs: A graph repair approach. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 30–41, 2015.

[30] J. Wang, S. Song, X. Zhu, and X. Lin. Efficient recovery of missing events. *PVLDB*, 6(10):841–852, 2013.

[31] W. Zheng, L. Zou, X. Lian, J. X. Yu, S. Song, and D. Zhao. How to build templates for RDF question/answering: An uncertain graph similarity join approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1809–1824, 2015.

[32] M. Zhong, M. Liu, and Y. He. 3sepias: A semi-structured search engine for personal information in dataspace system. *Inf. Sci.*, 218:31–50, 2013.

[33] X. Zhu, S. Song, X. Lian, J. Wang, and L. Zou. Matching heterogeneous event data. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1211–1222, 2014.

[34] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun. Matching heterogeneous events with patterns. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 376–387, 2014.

What's Really New with NewSQL?

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Matthew Aslett
451 Research
matthew.aslett@451research.com

ABSTRACT

A new class of database management systems (DBMSs) called **NewSQL** tout their ability to scale modern on-line transaction processing (OLTP) workloads in a way that is not possible with legacy systems. The term NewSQL was first used by one of the authors of this article in a 2011 business analysis report discussing the rise of new database systems as challengers to these established vendors (Oracle, IBM, Microsoft). The other author was working on what became one of the first examples of a NewSQL DBMS. Since then several companies and research projects have used this term (rightly and wrongly) to describe their systems.

Given that relational DBMSs have been around for over four decades, it is justifiable to ask whether the claim of NewSQL's superiority is actually true or whether it is simply marketing. If they are indeed able to get better performance, then the next question is whether there is anything scientifically new about them that enables them to achieve these gains or is it just that hardware has advanced so much that now the bottlenecks from earlier years are no longer a problem.

To do this, we first discuss the history of databases to understand how NewSQL systems came about. We then provide a detailed explanation of what the term NewSQL means and the different categories of systems that fall under this definition.

1. A BRIEF HISTORY OF DBMSS

The first DBMSs came on-line in the mid 1960s. One of the first was IBM's IMS that was built to keep track of the supplies and parts inventory for the Saturn V and Apollo space exploration projects. It helped introduce the idea that an application's code should be separate from the data that it operates on. This allows developers to write applications that only focus on the access and manipulation of data, and not the complications and overhead associated with how to actually perform these operations. IMS was later followed by the pioneering work in the early 1970s on the first relational DBMSs, IBM's System R and the University of California's INGRES. INGRES was soon adopted at other universities for their information systems and was subsequently commercialized in the late 1970s. Around the same time, Oracle released the first version of their DBMS that was similar to System R's design. Other companies were founded in the early 1980s that sought to repeat the success of the first commercial DBMSs, including Sybase and Informix. Although IBM never made System R available to the public, it later released a new relational DBMS (DB2) in 1983 that used parts of the System R code base.

The late 1980s and early 1990s brought about a new class of DBMSs that were designed to overcome the much touted impedance mismatch between the relational model and object-oriented programming languages [65]. These object-oriented DBMSs, however, never saw wide-spread market adoption because they lacked a standard interface like SQL. But many of the ideas from them were eventually incorporated in relational DBMSs when the major vendors added object and XML support a decade later, and then again in document-oriented NoSQL systems over two decades later.

The other notable event during the 1990s was the start of today's two major open-source DBMS projects. MySQL was started in Sweden in 1995 based on the earlier ISAM-based mSQL system. PostgreSQL began in 1994 when two Berkeley graduate students forked the original QUEL-based Postgres code from the 1980s to add support for SQL.

The 2000s brought the arrival of Internet applications that had more challenging resource requirements than applications from previous years. They needed to scale to support large number of concurrent users and had to be on-line all the time. But the database for these new applications was consistently found to be a bottleneck because the resource demands were much greater than what DBMSs and hardware could support at the time. Many tried the most obvious option of scaling their DBMS vertically by moving the database to a machine with better hardware. This, however, only improves performance so much and has diminishing returns. Furthermore, moving the database from one machine to another is a complex process and often requires significant downtime, which is unacceptable for these Web-based applications. To overcome this problem, some companies created custom *middleware* to shard single-node DBMSs over a cluster of less expensive machines. Such middleware presents a single logical database to the application that is stored across multiple physical nodes. When the application issues queries against this database, the middleware redirects and/or rewrites them to distribute their execution on one or more nodes in the cluster. The nodes execute these queries and send the results back to the middleware, which then coalesces them into a single response to the application. Two notable examples of this middleware approach were eBay's Oracle-based cluster [53] and Google's MySQL-based cluster [54]. This approach was later adopted by Facebook for their own MySQL cluster that is still used today.

Sharding middleware works well for simple operations like reading or updating a single record. It is more difficult, however, to execute queries that update more than one record in a transaction or join tables. As such, these early middleware

systems did not support these types of operations. eBay's middleware in 2002, for example, required their developers to implement all join operations in application-level code.

Eventually some of these companies moved away from using middleware and developed their own distributed DBMSs. The motivation for this was three-fold. Foremost was that traditional DBMSs at that time were focused on consistency and correctness at the expense of availability and performance. But this trade-off was deemed inappropriate for Web-based applications that need to be on-line all the time and have to support a large number of concurrent operations. Secondly, it was thought that there was too much overhead in using a full-featured DBMS like MySQL as a "dumb" data store. Likewise, it was also thought that the relational model was not the best way to represent an application's data and that using SQL was an overkill for simple look-up queries.

These problems turned out to be the origin of the impetus for the *NoSQL*[1] movement in the mid to late 2000s [22]. The key aspect of these NoSQL systems is that they forgo strong transactional guarantees and the relational model of traditional DBMSs in favor of eventual consistency and alternative data models (e.g., key/value, graphs, documents). This is because it was believed that these aspects of existing DBMSs inhibit their ability to scale out and achieve the high availability that is needed to support Web-based applications. The two most well-known systems that first followed this creed are Google's BigTable [23] and Amazon's Dynamo [26]. Neither of these two systems were available outside of their respective company at first (although they are now as cloud services), thus other organizations created their own open source clones of them. These include Facebook's Cassandra (based on BigTable and Dynamo) and PowerSet's Hbase (based on BigTable). Other start-ups created their own systems that were not necessarily copies of Google's or Amazon's systems but still followed the tenets of the NoSQL philosophy; the most well-known of these is MongoDB.

By the end of the 2000s, there was now a diverse set of scalable and more affordable distributed DBMSs available. The advantage of using a NoSQL system (or so people thought) was that developers could focus on the aspects of their application that were more beneficial to their business or organization, rather than having to worry about how to scale the DBMS. Many applications, however, are unable to use these NoSQL systems because they cannot give up strong transactional and consistency requirements. This is common for enterprise systems that handle high-profile data (e.g., financial and order processing systems). Some organizations, most notably Google [24], have found that NoSQL DBMSs cause their developers to spend too much time writing code to handle inconsistent data and that using transactions makes them more productive because they provide a useful abstraction that is easier for humans to reason about. Thus, the only options available for these organizations were to either purchase a more powerful single-node machine and to scale the DBMS vertically, or to develop their own custom sharding middleware that supports transactions. Both approaches are prohibitively expensive and are therefore not an option for many. It is in this environment that brought about NewSQL systems.

[1]The NoSQL community argues that the sobriquet should now be interpreted as "Not Only SQL", since some of these systems have since support some dialect of SQL.

2. THE RISE OF NEWSQL

Our definition of NewSQL is that they are a class of modern relational DBMSs that seek to provide the same scalable performance of NoSQL for OLTP read-write workloads while still maintaining ACID guarantees for transactions. In other words, these systems want to achieve the same scalability of NoSQL DBMSs from the 2000s, but still keep the relational model (with SQL) and transaction support of the legacy DBMSs from the 1970–80s. This enables applications to execute a large number of concurrent transactions to ingest new information and modify the state of the database using SQL (instead of a proprietary API). If an application uses a NewSQL DBMS, then developers do not have to write logic to deal with eventually consistent updates as they would in a NoSQL system. As we discuss below, this interpretation covers a number of both academic and commercial systems.

We note that there are data warehouse DBMSs that came out in the mid-2000s that some people think meet this criteria (e.g., Vertica, Greenplum, Aster Data). These DBMSs target on-line analytical processing (OLAP) workloads and should not be considered NewSQL systems. OLAP DBMSs are focused on executing complex read-only queries (i.e., aggregations, multi-way joins) that take a long time to process large data sets (e.g., seconds or even minutes). Each of these queries can be significantly different than the previous. The applications targeted by NewSQL DBMSs, on the other hand, are characterized as executing read-write transactions that (1) are short-lived (i.e., no user stalls), (2) touch a small subset of data using index lookups (i.e., no full table scans or large distributed joins), and (3) are repetitive (i.e., executing the same queries with different inputs). Others have argued for a more narrow definition where a NewSQL system's implementation has to use (1) a lock-free concurrency control scheme and (2) a shared-nothing distributed architecture [57]. All of the DBMSs that we classify as NewSQL in Section 3 indeed share these properties and thus we agree with this assessment.

3. CATEGORIZATION

Given the above definition, we now examine the landscape of today's NewSQL DBMSs. To simplify this analysis, we will group systems based on the salient aspects of their implementation. The three categories that we believe best represent NewSQL systems are (1) novel systems that are built from the ground-up using a new architecture, (2) middleware that re-implement the same sharding infrastructure that was developed in the 2000s by Google and others, and (3) database-as-a-service offerings from cloud computing providers that are also based on new architectures.

Both authors have previously included alternative storage engines for existing single-node DBMSs in our categorization of NewSQL systems. The most common examples of these are replacements for MySQL's default InnoDB storage engine (e.g., TokuDB, ScaleDB, Akiban, deepSQL). The advantage of using a new engine is that an organization can get better performance without having to change anything in their application and still leverage the DBMS's existing ecosystem (e.g., tools, APIs). The most interesting of these was ScaleDB because it provided transparent sharding underneath the system without using middleware by redistributing execution between storage engines; the company, however, has since pivoted to another problem domain. There has been other sim-

ilar extensions for systems other than MySQL. Microsoft's in-memory Hekaton OLTP engine for SQL Server integrates almost seamlessly with the traditional, disk-resident tables. Others use Postgres' foreign data wrappers and API hooks to achieve the same type of integration but target OLAP workloads (e.g., Vitesse, CitusDB).

We now assert that such storage engines and extensions for single-node DBMSs are not representative of NewSQL systems and omit them from our taxonomy. MySQL's InnoDB has improved significantly in terms of reliability and performance, so the benefits of switching to another engine for OLTP applications are not that pronounced. We acknowledge that the benefits from switching from the row-oriented InnoDB engine to a column-store engine for OLAP workloads are more significant (e.g., Infobright, InfiniDB). But in general, the MySQL storage engine replacement business for OLTP workloads is the graveyard of failed database projects.

3.1 New Architectures

This category contains the most interesting NewSQL systems for us because they are new DBMSs built from scratch. That is, rather than extending an existing system (e.g., Microsoft's Hekaton for SQL Server), they are designed from a new codebase without any of the architectural baggage of legacy systems. All of the DBMSs in this category are based on distributed architectures that operate on shared-nothing resources and contain components to support multi-node concurrency control, fault tolerance through replication, flow control, and distributed query processing. The advantage of using a new DBMS that is built for distributed execution is that all parts of the system can be optimized for multi-node environments. This includes things like the query optimizer and communication protocol between nodes. For example, most NewSQL DBMSs are able to send intra-query data directly between nodes rather than having to route them to a central location like with some middleware systems.

Every one of the DBMSs in this category (with the exception of Google Spanner) also manages their own primary storage, either in-memory or on disk. This means that the DBMS is responsible for distributing the database across its resources with a custom engine instead of relying on an off-the-shelf distributed filesystem (e.g., HDFS) or storage fabric (e.g., Apache Ignite). This is an important aspect of them because it allows the DBMS to "send the query to the data" rather than "bring the data to the query," which results in significantly less network traffic since transmitting the queries is typically less network traffic than having to transmit data (not just tuples, but also indexes and materialized views) to the computation.

Managing their own storage also enables a DBMS to employ more sophisticated replication schemes than what is possible with the block-based replication scheme used in HDFS. In general, it allows these DBMSs to achieve better performance than other systems that are layered on top of other existing technologies; examples of this include the "SQL on Hadoop" systems like Trafodion [4] and Splice Machine [16] that provide transactions on top of Hbase. As such, we believe that such systems should not be considered NewSQL.

But there are downsides to using a DBMS based on a new architecture. Foremost is that many organizations are wary of adopting technologies that are too new and un-vetted with a large installation base. This means that the number of people that are experienced in the system is much smaller compared to the more popular DBMS vendors. It also means that an organization will potentially lose access to existing administration and reporting tools. Some DBMSs, like Clustrix and MemSQL, avoid this problem by maintaining compatibility with the MySQL wire protocol.

Examples: Clustrix [6], CockroachDB [7], Google Spanner [24], H-Store [8], HyPer [39], MemSQL [11], NuoDB [14], SAP HANA [55], VoltDB [17].

3.2 Transparent Sharding Middleware

There are now products available that provide the same kind of sharding middleware that eBay, Google, Facebook, and other companies developed in the 2000s. These allow an organization to split a database into multiple shards that are stored across a cluster of single-node DBMS instances. Sharding is different than database federation technologies of the 1990s because each node (1) runs the same DBMS, (2) only has a portion of the overall database, and (3) is not meant to be accessed and updated independently by separate applications.

The centralized middleware component routes queries, coordinates transactions, as well as manages data placement, replication, and partitioning across the nodes. There is typically a shim layer installed on each DBMS node that communicates with the middleware. This component is responsible for executing queries on behalf of the middleware at its local DBMS instance and returning results. All together, these allow middleware products to present a single logical database to the application without needing to modify the underlying DBMS.

The key advantage of using a sharding middleware is that they are often a drop-in replacement for an application that is already using an existing single-node DBMS. Developers do not need to make any changes to their application to use the new sharded database. The most common target for middleware systems is MySQL. This means that in order to be MySQL compatible, the middleware must support the MySQL wire protocol. Oracle provides the MySQL Proxy [13] and Fabric [12] toolkits to do this, but others have written their owning protocol handler library to avoid GPL licensing issues.

Although middleware makes it easy for an organization to scale their database out across multiple nodes, such systems still have to use a traditional DBMS on each node (e.g., MySQL, Postgres, Oracle). These DBMSs are based on the disk-oriented architecture that was developed in the 1970s, and thus they cannot use a storage manager or concurrency control scheme that is optimized for memory-oriented storage like in some of the NewSQL systems that are built on new architectures. Previous research has shown that the legacy components of disk-oriented architectures is a significant encumbrance that prevents these traditional DBMSs from scaling up to take advantage of higher CPU core counts and larger memory capacities [38]. The middleware approach can also incur redundant query planning and optimization on sharded nodes for complex queries (i.e., once at the middleware and once on the individual DBMS nodes), but this does allow each node to apply their own local optimizations on each query.

Examples: AgilData Scalable Cluster [2] [1], MariaDB MaxScale [10], ScaleArc [15], ScaleBase[3].

[2] Prior to 2015, AgilData Cluster was known as dbShards.

[3] ScaleBase was acquired by ScaleArc in 2015 and is no longer sold.

3.3 Database-as-a-Service

Lastly, there are cloud computing providers that offer NewSQL database-as-a-service (DBaaS) products. With these services, organizations do not have to maintain the DBMS on either their own private hardware or on a cloud-hosted virtual machine (VM). Instead, the DBaaS provider is responsible for maintaining the physical configuration of the database, including system tuning (e.g., buffer pool size), replication, and backups. The customer is provided with a connection URL to the DBMS, along with a dashboard or API to control the system.

DBaaS customers pay according to their expected application's resource utilization. Since database queries vary widely in how they use computing resources, DBaaS providers typically do not meter query invocations in the same way that they meter operations in block-oriented storage services (e.g., Amazon's S3, Google's Cloud Storage). Instead, customers subscribe to a pricing tier that specifies the maximum resource utilization threshold (e.g., storage size, computation power, memory allocation) that the provider will guarantee.

As in most aspects of cloud computing, the largest companies are the major players in the DBaaS field due to the economies of scale. But almost all of the DBaaSs just provide a managed instance of a traditional, single-node DBMS (e.g., MySQL): notable examples include Google Cloud SQL, Microsoft Azure SQL, Rackspace Cloud Database, and Salesforce Heroku. We do not consider these to be NewSQL systems as they use the same underlying disk-oriented DBMSs based on the 1970s architectures. Some vendors, like Microsoft, retro-fitted their DBMS to provide better support for multi-tenant deployments [21].

We instead regard only those DBaaS products that are based on a new architecture as NewSQL. The most notable examples is Amazon's Aurora for their MySQL RDS. Its distinguishing feature over InnoDB is that it uses a log-structured storage manager to improve I/O parallelism.

There are also companies that do not maintain their own data centers but rather sell DBaaS software that run on top of these public cloud platforms. ClearDB provides their own custom DBaaS that can be deployed on all of the major cloud platforms. This has the advantage that it can distribute a database across different providers in the same geographical region to avoid downtimes due to service outages.

Aurora and ClearDB are the only two products available in this NewSQL category as of 2016. We note that several companies in this space have failed (e.g., GenieDB, Xeround), forcing their customers to scramble to find a new provider and migrate their data out of those DBaaS before they were shut down. We attribute their failure due to being ahead of market demand and from being out-priced from the major vendors.

Examples: Amazon Aurora [3], ClearDB [5].

4. THE STATE OF THE ART

We next discuss the features of NewSQL DBMSs to understand what (if anything) is novel in these systems. A summary of our analysis is shown in Table 1.

4.1 Main Memory Storage

All of the major DBMSs use a disk-oriented storage architecture based on the original DBMSs from the 1970s. In these systems, the primary storage location of the database is assumed to be on a block-addressable durable storage device, like an SSD or HDD. Since reading and writing to these devices is slow, DBMSs use memory to cache blocks read from disk and to buffer updates from transactions. This was necessary because historically memory was much more expensive and had a limited capacity compared to disks. We have now reached the point, however, where capacities and prices are such that it is affordable to store all but the largest OLTP databases entirely in memory. The benefit of this approach is that it enables certain optimizations because the DBMS no longer has to assume that a transaction could access data at any time that is not in memory and will have to stall. Thus, these systems can get better performance because many of the components that are necessary to handle these cases, like a buffer pool manager or heavy-weight concurrency control schemes, are not needed [38].

There are several NewSQL DBMSs that are based on a main memory storage architecture, including both academic (e.g., H-Store, HyPer) and commercial (e.g., MemSQL, SAP HANA, VoltDB) systems. These systems perform significantly better than disk-based DBMSs for OLTP workloads because of this main memory orientation.

The idea of storing a database entirely in main memory is not a new one [28, 33]. The seminal research at the University of Wisconsin-Madison in the early 1980s established the foundation for many aspects of main memory DBMSs [43], including indexes, query processing, and recovery algorithms. In that same decade, the first distributed main-memory DBMSs, PRISMA/DB, was also developed [40]. The first commercial main memory DBMSs appeared in 1990s; Altibase [2], Oracle's TimesTen [60], and AT&T's DataBlitz [20] were early proponents of this approach.

One thing that is new with main memory NewSQL systems is the ability to evict a subset of the database out to persistent storage to reduce its memory footprint. This allows the DBMS to support databases that are larger than the amount of memory available without having to switch back to a disk-oriented architecture. The general approach is to use an internal tracking mechanism inside of the system to identify which tuples are not being accessed anymore and then chose them for eviction. H-Store's *anti-caching* component moves cold tuples to a disk-resident store and then installs a "tombstone" record in the database with the location of the original data [25]. When a transaction tries to access a tuple through one of these tombstones, it is aborted and then a separate thread asynchronously retrieves that record and moves it back into memory. Another variant for supporting larger-than-memory databases is an academic project from EPFL that uses OS virtual memory paging in VoltDB [56]. To avoid false negatives, all of these DBMSs retain the keys for evicted tuples in databases' indexes, which inhibits the potential memory savings for those applications with many secondary indexes. Although not a NewSQL DBMS, Microsoft's *Project Siberia* [29] for Hekaton maintains a Bloom filter per index to reduce the in-memory storage overhead of tracking evicted tuples.

Another DBMS that takes a different approach for larger-than-memory databases is MemSQL where an administrator can manually instruct the DBMS to store a table in a columnar format. MemSQL does not maintain any in-memory tracking meta-data for these disk-resident tuples. It organizes this data in log-structured storage to reduce the overhead of updates,

which are traditionally slow in OLAP data warehouses.

4.2 Partitioning / Sharding

The way that almost all of the distributed NewSQL DBMSs scale out is to split a database up into disjoint subsets, called either partitions or shards.

Distributed transaction processing on partitioned databases is not a new idea. Many of the fundamentals of these systems came from the seminal work by the great Phil Bernstein (and others) in the SDD-1 project in the late 1970s [51]. In the early 1980s, the teams behind the two pioneering, single-node DBMSs, System R and INGRES, both also created distributed versions of their respective systems. IBM's R* was a shared-nothing, disk-oriented distributed DBMS like SDD-1 [63]. The distributed version of INGRES is mostly remembered for its dynamic query optimization algorithm that recursively breaks a distributed query into smaller pieces [31]. Later, the GAMMA project [27] from the University of Wisconsin-Madison explored different partitioning strategies.

But these earlier distributed DBMSs never caught on for two reasons. The first of these was that computing hardware in the 20th century was so expensive that most organizations could not afford to deploy their database on a cluster of machines. The second issue was that the application demand for a high-performance distributed DBMS was simply not there. Back then the expected peak throughput of a DBMS was typically measured at tens to hundreds of transactions per second. We now live in an era where both of these assumptions are no longer true. Creating a large-scale, data-intensive application is easier now than it ever has been, in part due to the proliferation of open-source distributed system tools, cloud computing platforms, and affordable mobile devices.

The database's tables are horizontally divided into multiple fragments whose boundaries are based on the values of one (or more) of the table's columns (i.e., the partitioning attributes). The DBMS assigns each tuple to a fragment based on the values of these attributes using either range or hash partitioning. Related fragments from multiple tables are combined together to form a partition that is managed by a single node. That node is responsible for executing any query that needs to access data stored in its partition. Only the DBaaS systems (Amazon Aurora, ClearDB) do not support this type of partitioning.

Ideally, the DBMS should be able to also distribute the execution of a query to multiple partitions and then combine their results together into a single result. All of the NewSQL systems except for ScaleArc that support native partitionining provide this functionality.

The databases for many OLTP applications have a key property that makes them amenable to partitioning. Their database schemas can be transposed into a tree-like structure where descendants in the tree have a foreign key relationship to the root [58]. The tables are then partitioned on the attributes involved in these relationships such that all of the data for a single entity are co-located together in the same partition. For example, the root of the tree could be the customer table, and the database is partitioned such that each customer, along with their order records and account information, are stored together. The benefit of this is that it allows most (if not all) transactions to only need to access data at a single partition. This in turn reduces the communication overhead of the system because it does not have to use an atomic commitment protocol (e.g., two-phase commit) to make sure that transactions finish correctly at different nodes.

The NewSQL DBMSs that deviate from the homegenous cluster node architecture are NuoDB and MemSQL. For NuoDB, it designates one or more nodes as storage managers (SM) that each store a partition of the database. The SMs splits a database into blocks (called "atoms" in NuoDB parlance). All other nodes in the cluster are designated as transaction engines (TEs) that act as an in-memory cache of atoms. To process a query, a TE node retrieves all of the atoms that it needs for that query (either from the appropriate SMs or from other TEs). TEs acquire write-locks on tuples and then broadcasts any changes to atoms to the other TEs and the SM. To avoid atoms from moving back and forth between nodes, NuoDB exposes load-balancing schemes to ensure that data that is used together often reside at the same TE. This means that NuoDB ends up with the same partitioning scheme as the other distributed DBMSs but without having to pre-partition the database or identify the relationships between tables.

MemSQL also uses a similar heterogeneous architecture comprised of execution-only aggregator nodes and leaf nodes that store the actual data. The difference between these two systems is in how they reduce the amount of data that is pulled from the storage nodes to the execution nodes. With NuoDB, the TEs cache atoms to reduce the amount data that they read from the SMs. MemSQL's aggregator nodes do not cache any data, but the leaf nodes execute parts of queries to reduce the amount of data that is sent to the aggregator nodes; this is not possible in NuoDB because the SMs are only a data store.

These two systems are able to add additional execution resources to the DBMS's cluster (NuoDB's TE nodes, MemSQL's aggregator nodes) without needing to re-partition the database. A research prototype of SAP HANA also explored using this approach [36]. It remains to be seen, however, whether such a heterogeneous architecture is superior to a homegenous one (i.e., were each node both stores data and executes queries) in terms of either performance or operational complexity.

Another aspect of partitioning in NewSQL systems that is new is that some of them support live migration. This allows the DBMS to move data between physical resources to re-balance and alleviate hotspots, or to increase/decrease the DBMS's capacity without any interruption to service. This is similar to re-balancing in NoSQL systems, but it is more difficult because a NewSQL DBMS has to maintain ACID guarantees for transactions during the migration [30]. There two approaches that DBMSs use to achieve this. The first is to organize the database in many coarse-grained "virtual" (i.e., logical) partitions that are spread amongst the physical nodes [52]. Then when the DBMS needs to re-balance, it moves these virtual partitions between nodes. This is the approach used in Clustrix and AgilData, as well as in NoSQL systems like Cassandra and DynamoDB. The other approach is for the DBMS to perform more fine-grained re-balancing by redistributing individual tuples or groups of tuples through range partitioning. This is akin to the auto-sharding feature in the MongoDB NoSQL DBMS. It is used in systems like ScaleBase and H-Store [30].

4.3 Concurrency Control

Concurrency control scheme is the most salient and important implementation detail of a transaction processing DBMS as it affects almost all aspects of the system. Concurrency control permits end-users to access a database in a multi-program-

med fashion while preserving the illusion that each of them is executing their transaction alone on a dedicated system. It essentially provides the atomicity and isolation guarantees in the system, and as such it influences the entire system's behavior.

Beyond which scheme a system uses, another important aspect of the design of a distributed DBMS is whether the system uses a centralized or decentralized transaction coordination protocol. In a system with a centralized coordinator, all transactions' operations have to go through the coordinator, which then makes decisions about whether transactions are allowed to proceed or not. This is the same approach used by the TP monitors of the 1970–1980s (e.g., IBM CICS, Oracle Tuxedo). In a decentralized system, each node maintains the state of transactions that access the data that it manages. The nodes then have to coordinate with each other to determine whether concurrent transactions conflict. A decentralized coordinator is better for scalability but requires that the clocks in the DBMS nodes are highly synchronized in order to generate a global ordering of transactions [24].

The first distributed DBMSs from the 1970–80s used two-phase locking (2PL) schemes. SDD-1 was the first DBMS specifically designed for distributed transaction processing across a cluster of shared-nothing nodes managed by a centralized coordinator. IBM's R* was similar to SDD-1, but the main difference was that the coordination of transactions in R* was completely decentralized; it used distributed 2PL protocol where transactions locked data items that they access directly at nodes. The distributed version of INGRES also used decentralized 2PL with centralized deadlock detection.

Almost all of the NewSQL systems based on new architectures eschew 2PL because the complexity of dealing with deadlocks. Instead, the current trend is to use variants of timestamp ordering (TO) concurrency control where the DBMS assumes that transactions will not execute interleaved operations that will violate serializable ordering. The most widely used protocol in NewSQL systems is decentralized multi-version concurrency control (MVCC) where the DBMS creates a new version of a tuple in the database when it is updated by a transaction. Maintaining multiple versions potentially allows transactions to still complete even if another transaction updates the same tuples. It also allows for long-running, read-only transactions to not block on writers. This protocol is used in almost all of the NewSQL systems based on new architectures, like MemSQL, HyPer, HANA, and CockroachDB. Although there are engineering optimizations and tweaks that these systems use in their MVCC implementations to improve performance, the basic concepts of the scheme are not new. The first known work describing MVCC is a MIT PhD dissertation from 1979 [49], while the first commercial DBMSs to use it were Digital's VAX Rdb and InterBase in the early 1980s. We note that the architecture of InterBase was designed by Jim Starkey, who is also the original designer of NuoDB and the failed Falcon MySQL storage engine project.

Other systems use a combination of 2PL and MVCC together. With this approach, transactions still have to acquire locks under the 2PL scheme to modify the database. When a transaction modifies a record, the DBMS creates a new version of that record just as it would with MVCC. This scheme allows read-only queries to avoid having to acquire locks and therefore not block on writing transactions. The most famous implementation of this approach is MySQL's InnoDB, but it is also used in both Google's Spanner, NuoDB, and Clustrix. NuoDB improves on the original MVCC by employing a gossip protocol to broadcast versioning information between nodes.

All of the middleware and DBaaS services inherit the concurrency control scheme of their underlying DBMS architecture; since most of them use MySQL, this makes them 2PL with MVCC systems.

We regard the concurrency control implementation in Spanner (along with its descendants F1 [54] and SpannerSQL) as one of the most novel of the NewSQL systems. The actual scheme itself is based on the 2PL and MVCC combination developed in previous decades. But what makes Spanner different is that it uses hardware devices (e.g., GPS, atomic clocks) for high-precision clock synchronization. The DBMS uses these clocks to assign timestamps to transactions to enforce consistent views of its multi-version database over wide-area networks. CockroachDB also purports to provide the same kind of consistency for transactions across data centers as Spanner but without the use of atomic clocks. They instead rely on a hybrid clock protocol that combines loosely synchronized hardware clocks and logical counters [41].

Spanner is also noteworthy because it heralds Google's return to using transactions for its most critical services. The authors of Spanner even remark that it is better to have their application programmers deal with performance problems due to overuse of transactions, rather than writing code to deal with the lack of transactions as one does with a NoSQL DBMS [24].

Lastly, the only commercial NewSQL DBMS that is not using some MVCC variant is VoltDB. This system still uses TO concurrency control, but instead of interleaving transactions like in MVCC, it schedules transactions to execute one-at-a-time at each partition. It also uses a hybrid architecture where single-partition transactions are scheduled in a decentralized manner but multi-partition transactions are scheduled with a centralized coordinator. VoltDB orders transactions based on logical timestamps and then schedules them for execution at a partition when it is their turn. When a transaction executes at a partition, it has exclusive access to all of the data at that partition and thus the system does not have to set fine-grained locks and latches on its data structures. This allows transactions that only have to access a single partition to execute efficiently because there is no contention from other transactions. The downside of partition-based concurrency control is that it does not work well if transactions span multiple partitions because the network communication delays cause nodes to sit idle while they wait for messages. This partition-based concurrency is not a new idea. An early variant of it was first proposed in a 1992 paper by Hector Garcia-Molina [34] and implemented in the kdb system in late 1990s [62] and in H-Store (which is the academic predecessor of VoltDB).

In general, we find that there is nothing significantly new about the core concurrency control schemes in NewSQL systems other than laudable engineering to make these algorithms work well in the context of modern hardware and distributed operating environments.

4.4 Secondary Indexes

A secondary index contains a subset of attributes from a table that are different than its primary key(s). This allows the DBMS to support fast queries beyond primary key or partitioning key look-ups. They are trivial to support in a non-partitioned DBMS because the entire database is located on a

single node. The challenge with secondary indexes in a distributed DBMS is that they cannot always be partitioned in the same manner as with the rest of the database. For example, suppose that the tables of a database are partitioned based on the customer's table primary key. But then there are some queries that want to do a reverse look-up from the customer's email address to the account. Since the tables are partitioned on the primary key, the DBMS will have to broadcast these queries to every node, which is obviously inefficient.

The two design decisions for supporting secondary indexes in a distributed DBMS are (1) where the system will store them and (2) how it will maintain them in the context of transactions. In a system with a centralized coordinator, like with sharding middleware, secondary indexes can reside on both the coordinator node and the shard nodes. The advantage of this approach is that there is only a single version of the index in the entire system, and thus it is easier to maintain.

All of the NewSQL systems based on new architectures are decentralized and use partitioned secondary indexes. This means that each node stores a portion of the index, rather than each node having a complete copy of it. The trade-off between partitioned and replicated indexes is that with the former queries may need to span multiple nodes to find what they are looking for but if a transaction updates an index it will only have to modify one node. In a replicated index, the roles are reversed: a look-up query can be satisfied by just one node in the cluster, but any time a transaction modifies the attributes referenced in secondary index's underlying table (i.e., the key or the value), the DBMS has to execute a distributed transaction that updates all copies of the index.

An example of a decentralized secondary index that mixes both of these concepts is in Clustrix. The DBMS first maintains a replicated, coarse-grained (i.e., range-based) index at each node that maps values to partitions. This mapping allows the DBMS to route queries to the appropriate node using an attribute that is not the table's partitioning attribute. These queries will then access a second partitioned index at that node that maps exact values to tuples. Such a two-tier approach reduces the amount of coordination that is needed to keep the replicated index in sync across the cluster since it only maps ranges instead of individual values.

The most common way that developers create secondary indexes when using a NewSQL DBMS that does not support them is to deploy an index using an in-memory, distributed cache, such as Memcached [32]. But using an external system requires the application to maintain the cache since the DBMSs will not automatically invalidate the external cache.

4.5 Replication

The best way that an organization can ensure high availability and data durability for their OLTP application is to replicate their database. All modern DBMSs, including NewSQL systems, support some kind of replication mechanism. DBaaS have a distinct advantage in this area because they hide all of the gritty details of setting of replication from their customers. They make it easy to deploy a replicated DBMS without the administrator having to worry about transmitting logs and making sure that nodes are in sync.

There are two design decisions when it comes to database replication. The first is how the DBMS enforces data consistency across nodes. In a *strongly consistent* DBMS, a transaction's writes must be acknowledged and installed at all replicas before that transaction is considered committed (i.e., durable). The advantage of this approach is that replicas can serve read-only queries and still be consistent. That is, if the application receives an acknowledgement that a transaction has committed, then any modifications made by that transaction are visible to any subsequent transaction in the future regardless of what DBMS node they access. It also means that when a replica fails, there are no lost updates because all the other nodes are synchronized. But maintaining this synchronization requires the DBMS to use an atomic commitment protocol (e..g, two-phase commit) to ensure that all replicas agree with the outcome of a transaction, which has additional overhead and can lead to stalls if a node fails or if there is a network partition/delay. This is why NoSQL systems opt for a *weakly consistent* model (also called eventual consistency) where not all replicas have to acknowledge a modification before the DBMS notifies the application that the write succeeded.

All of the NewSQL systems that we are aware of support strongly consistent replication. But there is nothing novel about how these systems ensure this consistency. The fundamentals of state machine replication for DBMSs were studied back in the 1970s [37, 42]. NonStop SQL was one of the first distributed DBMSs built in the 1980s using strongly consistency replication to provide fault tolerance in this same manner [59].

In addition to the policy of when a DBMS propagates updates to replicas, there are also two different execution models for how the DBMS performs this propagation. The first, known as *active-active* replication, is where each replica node processes the same request simultaneously. For example, when a transaction executes a query, the DBMS executes that query in parallel at all of the replicas. This is different from *active-passive* replication where a request is first processed at a single node and then the DBMS transfers the resultant state to the other replicas. Most NewSQL DBMSs implement this second approach because they use a non-deterministic concurrency control scheme. This means that they cannot send queries to replicas as they arrive on the master because they may get executed in a different order on the replicas and the state of the databases will diverge at each replica. This is because their execution order depends on several factors, including network delays, cache stalls, and clock skew.

Deterministic DBMSs (e.g., H-Store, VoltDB, ClearDB) on the other hand do not perform these additional coordination steps. This is because the DBMS guarantees that transactions' operations execute in the same order on each replica and thus the state of the database is guaranteed to be the same [44]. Both VoltDB and ClearDB also ensure that the application does not execute queries that utilize sources of information that are external to the DBMS that may be different on each replica (e.g., setting a timestamp field to the local system clock).

One aspect of the NewSQL systems that is different than previous work outside of academia is the consideration of replication over the wide-area network (WAN). This is a byproduct of modern operating environments where it is now trivial to deploy systems across multiple data centers that are separated by large geographical differences. Any NewSQL DBMS can be configured to provide synchronous updates of data over the WAN, but this would cause significant slowdown for normal operations. Thus, they instead provide asynchronous replication methods. To the best of our knowledge, Spanner and CockroachDB are the only NewSQL systems to provide a repli-

cation scheme that is optimized for strongly consistent replicas over the WAN. They again achieve this through a combination of atomic and GPS hardware clocks (in case of Spanner [24]), or hybrid clocks (in the case of CockroachDB [41]).

4.6 Crash Recovery

Another important feature of a NewSQL DBMS for providing fault tolerance is its crash recovery mechanism. But unlike traditional DBMSs where the main concern of fault tolerance is to ensure that no updates are lost [47], newer DBMSs must also minimize downtime. Modern web applications are expected to be on-line all the time and site outages are costly.

The traditional approach to recovery in a single-node system without replicas is that when the DBMS comes back online after a crash, it loads in the last checkpoint that it took from disk and then replays its write-ahead log (WAL) to return the state of the database to where it was at the moment of the crash. The canonical method of this approach, known as ARIES [47], was invented by IBM researchers in the 1990s. All major DBMSs implement some variant of ARIES.

In a distributed DBMS with replicas, however, the traditional single-node approach is not directly applicable. This is because when the master node crashes, the system will promote one of the slave nodes to be the new master. When the previous master comes back on-line, it cannot just load in its last checkpoint and rerun its WAL because the DBMS has continued to process transactions and therefore the state of the database has moved forward. The recovering node needs to get the updates from the new master (and potentially other replicas) that it missed while it was down. There are two potential ways to do this. The first is for the recovering node to load in its last checkpoint and WAL from its local storage and then pull log entries that it missed from the other nodes. As long as the node can process the log faster than new updates are appended to it, the node will eventually converge to the same state as the other replica nodes. This is possible if the DBMS uses physical or physiological logging, since the time to apply the log updates directly to tuples is much less than the time it takes to execute the original SQL statement. To reduce the time it takes to recover, the other option is for the recovering node to discard its checkpoint and have system take a new one that the node will recover from. One additional benefit of this approach is that this same mechanism can also be used in the DBMS to add a new replica node.

The middleware and DBaaS systems rely on the built-in mechanisms of their underlying single-node DBMSs, but add additional infrastructure for leader election and other management capabilities. The NewSQL systems that are based on new architectures use a combination of off-the-shelf components (e.g., ZooKeeper, Raft) and their own custom implementations of existing algorithms (e.g., Paxos). All of these are standard procedures and technologies that have been available in commercial distributed systems since the 1990s.

5. FUTURE TRENDS

We foresee the next trend for database applications in the near future is the ability to execute analytical queries and machine learning algorithms on freshly obtained data. Such workloads, colloquially known as "real-time analytics" or hybrid transaction-analytical processing (HTAP), seek to extrapolate insights and knowledge by analyzing a combination of historical data sets with new data [35]. This differs from traditional business intelligence operations from the previous decade that could only perform this analysis on historical data. Having a shorter turnaround time is important in modern applications because data has immense value as soon as it is created, but that value diminishes over time.

There are three approaches to supporting HTAP pipelines in a database application. The most common is to deploy separate DBMSs: one for transactions and another for analytical queries. With this architecture, the front-end OLTP DBMS stores all of the new information generated from transactions. Then in the background, the system uses an extract-transform-load utility to migrate data from this OLTP DBMS to a second back-end data warehouse DBMS. The application executes all complex OLAP queries in the back-end DBMS to avoid slowing down the OLTP system. Any new information generated from the OLAP system is pushed forward to front-end DBMS.

Another prevailing system design, known as the lambda architecture [45], is to use a separate batch processing system (e.g., Hadoop, Spark) to compute a comprehensive view on historical data, while simultaneously using a stream processing system (e.g., Storm [61], Spark Streaming [64]) to provide views of incoming data. In this split architecture, the batch processing system periodically rescans the data set and performs a bulk upload of the result to the stream processing system, which then makes modifications based on new updates.

There are several problems inherent with the bifurcated environment of these two approaches. Foremost is that the time it takes to propagate changes between the separate systems is often measured in minutes or even hours. This data transfer inhibits an application's ability to act on data immediately when it is entered in the database. Second, the administrative overhead of deploying and maintaining two different DBMSs is non-trivial as personnel is estimated to be almost 50% of the total ownership cost of a large-scale database system [50]. It also requires the application developer to write a query for multiple systems if they want to combine data from different databases. Some systems that try to achieve a single platform by hiding this split system architecture; an example of this is Splice Machine [16], but this approach has other technical issues due to copying data from the OLTP system (Hbase) before it can be used in the OLAP system (Spark).

The third (and in our opinion better) approach is to use a single HTAP DBMS that supports the high throughput and low latency demands of OLTP workloads, while also allowing for complex, longer running OLAP queries to operate on both hot (transactional) and cold (historical) data. What makes these newer HTAP systems different from legacy general-purpose DBMSs is that they incorporate the advancements from the last decade in the specialized OLTP (e.g., in-memory storage, lock-free execution) and OLAP (e.g., columnar storage, vectorized execution) systems, but within a single DBMS.

SAP HANA and MemSQL were the first NewSQL DBMSs to market themselves as HTAP systems. HANA achieves this by using multiple execution engines internally: one engine for row-oriented data that is better for transactions and a different engine for column-oriented data that is better for analytical queries. MemSQL uses two different storage managers (one for rows, one for columns) but mixes them together in a single execution engine. HyPer switched from a row-oriented system with H-Store-style concurrency control that was focused on

		Year Released	Main Memory Storage	Partitioning	Concurrency Control	Replication	Summary
NEW ARCHITECTURES	**Clustrix** [6]	2006	No	Yes	MVCC+2PL	Strong+Passive	MySQL-compatible DBMS that supports shared-nothing, distributed execution.
	CockroachDB [7]	2014	No	Yes	MVCC	Strong+Passive	Built on top of distributed key/value store. Uses software hybrid clocks for WAN replication.
	Google Spanner [24]	2012	No	Yes	MVCC+2PL	Strong+Passive	WAN-replicated, shared-nothing DBMS that uses special hardware for timestamp generation.
	H-Store [8]	2007	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Optimized for stored procedures.
	HyPer [9]	2010	Yes	Yes	MVCC	Strong+Passive	HTAP DBMS that uses query compilation and memory efficient indexes.
	MemSQL [11]	2012	Yes	Yes	MVCC	Strong+Passive	Distributed, shared-nothing DBMS using compiled queries. Supports MySQL wire protocol.
	NuoDB [14]	2013	Yes	Yes	MVCC	Strong+Passive	Split architecture with multiple in-memory executor nodes and a single shared storage node.
	SAP HANA [55]	2010	Yes	Yes	MVCC	Strong+Passive	Hybrid storage (rows + cols). Amalgamation of previous TREX, P*TIME, and MaxDB systems.
	VoltDB [17]	2008	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Supports streaming operators.
MIDDLEWARE	**AgilData** [1]	2007	No	Yes	MVCC+2PL	Strong+Passive	Shared-nothing database sharding over single-node MySQL instances.
	MariaDB MaxScale [10]	2015	No	Yes	MVCC+2PL	Strong+Passive	Query router that supports custom SQL rewriting. Relies on MySQL Cluster for coordination.
	ScaleArc [15]	2009	No	Yes	Mixed	Strong+Passive	Rule-based query router for MySQL, SQL Server, and Oracle.
DBAAS	**Amazon Aurora** [3]	2014	No	No	MVCC	Strong+Passive	Custom log-structured MySQL engine for RDS.
	ClearDB [5]	2010	No	No	MVCC+2PL	Strong+Active	Centralized router that mirrors a single-node MySQL instance in multiple data centers.

Table 1: NewSQL Systems – Summary of the system features described in Section 4 for the different DBMSs. Note that the year released is either when the project was announced publicly or when the company was first formed.

OLTP to use an HTAP column-store architecture with MVCC to allow it support more complex OLAP queries [48]. Even VoltDB has pivoted their marketing strategy from pure OLTP performance to providing streaming semantics. Similarly, the S-Store project seeks to add support for stream processing operations on top of the H-Store architecture [46]. It is likely that the specialized OLAP systems from the mid-2000s (e.g., Greenplum) will start to add support for better OLTP.

We note, however, that the rise of HTAP DBMSs does mean the end of giant, monolithic OLAP warehouses. Such systems will still be necessary in the short-term as they stand to be the universal back-end database for all of an organization's front-end OLTP silos. But eventually the resurgence of database federation will allow organization's to execute analytical queries that span multiple OLTP databases (including even multiple vendors) without needing to move data around.

6. CONCLUSION

The main takeaway from our analysis is that NewSQL database systems are not a radical departure from existing system architectures but rather represent the next chapter in the continuous development of database technologies. Most of the techniques that these systems employ have existed in previous DBMSs from academia and industry. But many of them were only implemented one-at-a-time in a single system and never all together. What is therefore innovative about these NewSQL DBMSs is that they incorporate these ideas into single platforms. Achieving this is by no means a trivial engineering effort. They are by-products of a new era where distributed computing resources are plentiful and affordable, but at the same time the demands of applications is much greater.

It is also interesting to consider the potential impact and future direction of NewSQL DBMSs in the marketplace. Given that the legacy DBMS vendors are entrenched and well funded, NewSQL systems have an uphill battle to gain market share. In the last five years since we first coined the term NewSQL [18], several NewSQL companies have folded (e.g., GenieDB, Xeround, Translattice) or pivoted to focus on other problem domains (e.g., ScaleBase, ParElastic). Based on our analysis and interviews with several companies, we have found that NewSQL systems have had a relatively slow rate of adoption, especially compared to the developer-driven NoSQL uptake. This is because NewSQL DBMSs are designed to support the transactional workloads that are mostly found in enterprise applications. Decisions regarding database choices for these enterprise applications are likely to be more conservative than for new Web application workloads. This is also evident from the fact that we find that NewSQL DBMSs are used to complement or replace existing RDBMS deployments, whereas NoSQL are being deployed in new application workloads [19].

Unlike with the OLAP DBMS start-ups from the 2000s, where almost all of the vendors were acquired by major technology companies, up until now there has been only one acquisition made of a NewSQL company. In March 2016, Tableau announced that it purchased the start-up formed for the HyPer project. The two other possible exceptions to this are (1) Apple acquiring FoundationDB in March 2015, but we exclude them because this system was at its core a NoSQL key-value store with an inefficient SQL layer grafted on top of it, and (2) ScaleArc acquiring ScaleBase, but this was one competitor buying out another. None of these examples are the same kind of acquisition where a legacy vendor purchasing an upstart

system (e.g., Teradata buying Aster Data Systems in 2011). We instead see that the large vendors are choosing to innovate and improve their own systems rather than acquire NewSQL start-ups. Microsoft added the in-memory Hekaton engine to SQL Server in 2014 to improve OLTP workloads. Oracle and IBM have been slightly slower to innovate; they recently added column-oriented storage extensions to their systems to compete with the rising popularity of OLAP DBMSs like HP Vertica and Amazon Redshift. It is possible that they will add an in-memory option for OLTP workloads in the future.

More long term, we believe that there will be a convergence of features in the four classes of systems that we discussed here: (1) the older DBMSs from the 1980-1990s, (2) the OLAP data warehouses from the 2000s, (3) the NoSQL DBMSs from the 2000s, and (4) the NewSQL DBMSs from the 2010s. We expect that all of the key systems in these groups will support some form of the relational model and SQL (if they do not already), as well as both OLTP operations and OLAP queries together like HTAP DBMSs. When this occurs, such labels will be meaningless.

ACKNOWLEDGMENTS

The authors would like to thank the following people for their feedback: Andy Grove (AgilData), Prakhar Verma (Amazon), Cashton Coleman (ClearDB), Dave Anselmi (Clustrix), Spencer Kimball (CockroachDB), Peter Mattis (CockroachDB), Ankur Goyal (MemSQL), Seth Proctor (NuoDB), Anil Goel (SAP HANA), Ryan Betts (VoltDB). This work was supported (in part) by the National Science Foundation (Award CCF-1438955).

For questions or comments about this paper, please call the CMU Database Hotline at `+1-844-88-CMUDB`.

7. REFERENCES

[1] AgilData Scalable Cluster for MySQL. `http://www.agildata.com/`.

[2] Altibase. `http://altibase.com`.

[3] Amazon Aurora. `https://aws.amazon.com/rds/aurora`.

[4] Apache Trafodion. `http://trafodion.apache.org`.

[5] ClearDB. `https://www.cleardb.com`.

[6] Clustrix. `http://www.clustrix.com`.

[7] CockroachDB. `https://www.cockroachlabs.com/`.

[8] H-Store. `http://hstore.cs.brown.edu`.

[9] HyPer. `http://hyper-db.de`.

[10] MariaDB MaxScale. `https://mariadb.com/products/mariadb-maxscale`.

[11] MemSQL. `http://www.memsql.com`.

[12] MySQL Fabric. `https://www.mysql.com/products/enterprise/fabric.html`.

[13] MySQL Proxy. `http://dev.mysql.com/doc/mysql-proxy/en/`.

[14] NuoDB. `http://www.nuodb.com`.

[15] ScaleArc. `http://scalearc.com`.

[16] Splice Machine. `http://www.splicemachine.com`.

[17] VoltDB. `http://www.voltdb.com`.

[18] M. Aslett. How will the database incumbents respond to NoSQL and NewSQL? The 451 Group, April 2011.

[19] M. Aslett. MySQL vs. NoSQL and NewSQL: 2011-2015. The 451 Group, May 2012.

[20] J. Baulier, P. Bohannon, S. Gogate, S. Joshi, C. Gupta, A. Khivesera, H. F. Korth, P. McIlroy, J. Miller, P. P. S. Narayan, M. Nemeth, R. Rastogi, A. Silberschatz, and S. Sudarshan. DataBlitz: A high performance main-memory storage manager. VLDB, pages 701–, 1998.

[21] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manne, L. Novik, and T. Talius. Adapting microsoft SQL server for cloud computing. In *ICDE*, pages 1255–1263, 2011.

[22] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39:12–27, 2011.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[24] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's Globally-Distributed Database. In *OSDI*, 2012.

[25] J. DeBrabant, A. Pavlo, S. Tu, M. Stonebraker, and S. B. Zdonik. Anti-caching: A new approach to database management system architecture. *PVLDB*, 6(14):1942–1953, 2013.

[26] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

[27] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna. GAMMA - a high performance dataflow database machine. In *VLDB*, pages 228–237, 1986.

[28] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood. Implementation techniques for main memory database systems. *SIGMOD Rec.*, 14(2):1–8, 1984.

[29] A. Eldawy, J. Levandoski, and P.-Å. Larson. Trekking through siberia: Managing cold data in a memory-optimized database. *Proceedings of the VLDB Endowment*, 7(11):931–942, 2014.

[30] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. E. Abbadi. Squall: Fine-grained live reconfiguration for partitioned main memory databases. SIGMOD, pages 299–313, 2015.

[31] R. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. SIGMOD, pages 169–180, 1978.

[32] B. Fitzpatrick. Distributed Caching with Memcached. *Linux J.*, 2004(124):5–, Aug. 2004.

[33] H. Garcia-Molina, R. J. Lipton, and J. Valdes. A massive memory machine. *IEEE Trans. Comput.*, 33(5):391–399, May 1984.

[34] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Trans. on Knowl. and Data Eng.*, 4(6):509–516, Dec. 1992.

[35] Gartner. Hybrid Transaction/Analytical Processing Will Foster Opportunities for Dramatic Business Innovation. https://www.gartner.com/doc/2657815/, 2014.

[36] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Färber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner. Towards scalable real-time analytics: An architecture for scale-out of olxp workloads. *Proc. VLDB Endow.*, 8(12):1716–1727, Aug. 2015.

[37] J. Gray. *Concurrency Control and Recovery in Database Systems*, chapter Notes on data base operating systems, pages 393–481. Springer-Verlag, 1978.

[38] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. OLTP through the looking glass, and what we found there. In *SIGMOD*, pages 981–992, 2008.

[39] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. ICDE, pages 195–206, 2011.

[40] M. L. Kersten, P. M. Apers, M. A. Houtsma, E. J. Kuyk, and R. L. Weg. A distributed, main-memory database machine. In *Database Machines and Knowledge Base Machines*, volume 43 of *The Kluwer International Series in Engineering and Computer Science*, pages 353–369. 1988.

[41] S. Kimball. Living without atomic clocks. https://www.cockroachlabs.com/blog/living-without-atomic-clocks/, February 2016.

[42] L. Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.

[43] T. J. Lehman. *Design and performance evaluation of a main memory relational database system*. PhD thesis, University of Wisconsin–Madison, 1986.

[44] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory oltp recovery. In *ICDE*, pages 604–615, 2014.

[45] N. Marz and J. Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, 2013.

[46] J. Meehan, N. Tatbul, S. Zdonik, C. Aslantas, U. Çetintemel, J. Du, T. Kraska, S. Madden, D. Maier, A. Pavlo, M. Stonebraker, K. Tufte, and H. Wang. S-store: Streaming meets transaction processing. *PVLDB*, 8(13):2134–2145, 2015.

[47] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans. Database Syst.*, 17(1):94–162, 1992.

[48] T. Neumann, T. Mühlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. SIGMOD, pages 677–689, 2015.

[49] D. P. Reed. *Naming and synchronization in a decentralized computer system*. PhD thesis, MIT, 1979.

[50] A. Rosenberg. Improving query performance in data warehouses. *Business Intelligence Journal*, 11, Jan. 2006.

[51] J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. Introduction to a system for distributed databases (SDD-1). *ACM Trans. Database Syst.*, 5(1):1–17, Mar. 1980.

[52] M. Serafini, E. Mansour, A. Aboulnaga, K. Salem, T. Rafiq, and U. F. Minhas. Accordion: Elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.*, 7(12):1035–1046, Aug. 2014.

[53] R. Shoup and D. Pritchett. The ebay architecture. SD Forum, November 2006.

[54] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed sql database that scales. *Proc. VLDB Endow.*, 6(11):1068–1079, Aug. 2013.

[55] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd. Efficient transaction processing in sap hana database: The end of a column store myth. SIGMOD, pages 731–742, 2012.

[56] R. Stoica and A. Ailamaki. Enabling efficient os paging for main-memory OLTP databases. In *DaMon*, 2013.

[57] M. Stonebraker. New sql: An alternative to nosql and old sql for new oltp apps. BLOG@CACM, June 2011.

[58] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB*, pages 1150–1160, 2007.

[59] Tandem Database Group. NonStop SQL, a distributed, high-performance, high-availability implementation of sql. Technical report, Tandem, Apr. 1987.

[60] T. Team. In-memory data management for consumer transactions the timesten approach. SIGMOD '99, pages 528–529, 1999.

[61] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *SIGMOD*, pages 147–156, 2014.

[62] A. Whitney, D. Shasha, and S. Apter. High Volume Transaction Processing Without Concurrency Control, Two Phase Commit, SQL or C++. In *HPTS*, 1997.

[63] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, and R. Yost. Distributed systems, vol. ii: distributed data base systems. chapter R*: an overview of the architecture, pages 435–461. 1986.

[64] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *SOSP*, 2013.

[65] S. B. Zdonik and D. Maier, editors. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann, 1990.

H V Jagadish Speaks Out on PVLDB, CoRR and Data-driven Research

Marianne Winslett and Vanessa Braganholo



H V Jagadish
http://web.eecs.umich.edu/~jag/

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Phoenix, cite of the 2012 SIGMOD and PODS conference. I have here with me H. V. Jagadish, who is the Bernard A. Galler Professor of Electrical Engineering and Computer Science at the University of Michigan. Jag has served as the editor-in-chief of the Proceedings of the VLDB, the database area editor for CoRR, and a board member for the Computing Research Association. Jag's PhD is from Stanford University and he's an ACM Fellow. So, welcome Jag!

Thank you, Marianne.

Jag, you've been very involved with the Proceedings of the VLDB. How did that get started?

The Proceedings of the VLDB just happened to come together. This is really how it happened. I was part of the VLDB Endowment Board of Trustees and there had been a lot of discussion amongst various people who had been on the board before me about publication models and what one should do. There had been a broadening effort that Phil Bernstein and others had been pushing. There were some who felt that conference publications did not get the same level of respect as journal publications, particularly in some countries. There were others who were concerned about how we did our reviews and what our reviewing processes were. Somehow all the vectors lined up at the right time and I just happened to be able to make use of all the forces that were there at that time. When it suddenly happened, it was actually very quick. There were a number of pieces that needed to come together for PVLDB to happen and all of that happened within one VLDB Endowment board meeting, which is typically not the way that VLDB operates. That's because there had been many years of preparatory work and so people sort of knew all the issues. There had not been partial solutions before, they were just issues, what we should do, and inconclusive discussions. When there was something that seemed like it had a chance of working, I think the trustees were very enthusiastic about trying out the experiment and seeing how it worked.

[…] a great deal of the work that people are doing in a data-driven manner in many disciplines is often prey to all kinds of biases and errors. It's very easy not to have enough statistical power.

Is this experiment helping with the communities that want to see a journal publication? So is PVLDB a journal?

PVLDB is a journal when its advantageous to be one. I think that it is truly a hybrid. It is not a standard conference publication. It is not a standard journal publication. For those who keep books, the fact that we have an explicit ISSN, which is what makes it a journal as opposed to just an ISBN, which is a one-shot thing that each conference proceedings gets, makes it technically a journal for classification purposes. Other than that, I think that the nature of the publication, and the spirit of it and the way in which one reviews it and evaluates it, is very much in what we think of as conference style as opposed to journal style.

And is it in that ISI index that some countries rely on?

The ISI index is one place for instance where having an ISSN is very important.

How is the impact factor looking for it so far?

It's too early. It turns out that Thomson Reuters has a process for putting things into their index and among the things they want to see is a minimum bar of three years of publication history on schedule**.** So apparently they deal with a lot of publications that do not end up having enough volume and so the issues get delayed. Now, we are all used to backlogs in our journals and people are trying to minimize the backlogs and editors are wheedling the extra pages from the publishers, and things like this. There are places where journals just do not know how to fill their pages. They are promised a quarterly journal but they have a hard time actually bringing one out. So for whatever reasons, that is a rule and we have not had three years of regular publications[1]. So we are not yet indexed in the ISI, for instance.

So you've been very involved with this CoRR (Computing Research Repository). What's that all about?

The Computing Research Repository is something that some people put all of their work in and others may have never even heard of. It seems to have an uneven uptake in our community. The idea behind this is that there is a central place where people put any work that they think others may want to read. There isn't a review process other than for appropriateness. So we do talk about the category and we want to make sure that if you claim that your paper is about databases then it is about databases. So it isn't just that we keep a political creed out, but one of my jobs as the database section editor for CoRR is to make sure that a paper that is about "software engineering" doesn't have a

[1] This interview was conducted in 2012.

"database" label by mistake. Sometimes there are questions about where exactly do we draw a boundary, does a paper deserves two labels, and things of this nature. Anyway, the thing with this is simply to have one place where people can find work on a particular topic. This is something that physicists have had great success in using extensively. I think most areas in physics do this and some areas in mathematics and other fields. Computer science was not even the first to the game here, but the same infrastructure is being used for computing. Some people and some sub communities seem to have adopted it with gusto and others have just totally ignored it.

I think that the value, if there is good adoption, is that it saves you the effort of doing things like a web search to find papers. We have organized collections for things that have been published in good places (things like our SIGMOD DiSC -- Digital Symposium Collection) and things of that nature, which for our sub community works very well. So the need for the database community for something like CoRR may be less. We note though that CoRR isn't just for things that are published in good places. It is for everything.

I think of it as a pre-print place or a pre-submission place.

Well, so one of the things we've also done is when papers appear in PVLDB, they also get deposited in CoRR. So it is not something that gets there as a pre-print, it is put there at acceptance. And many conferences, workshops and journals do this with CoRR on a regular basis. When papers are accepted, there is a batch upload.

The people I know who are enthusiastically depositing and looking in CoRR are using it as a place to put in their papers that usually they haven't even submitted yet so they're sort of at the tech-report stage. That is different from what you were talking about a minute ago and that usage in my mind conflicts with the double-blind reviewing philosophy. Do you have any comments on that?

Yes, I agree with you that there are people who put things into CoRR at a very early stage, at a pre-print stage or to establish "first in time" for some idea. But I view this as not much different from people putting out papers on the web. There are a lot of people who put up tech reports on the web and I think that double-blind reviewing is impacted even if there weren't CoRR, just because there is a web and there is web search. That is a challenge for double-blind reviewing. I don't think that CoRR makes it that much worse. I personally believe that changing how we manage our archival to support the blindness of reviewing is the tail wagging the dog. I think that whatever one might want to do with respect to whether we use double-blind reviews or not, that is a concern with respect to how we evaluate papers and that should be a second order concern with respect to how we disseminate knowledge: how we store and share knowledge should be the primary concern. Even if it were the case that it mattered that there was a negative impact on double-blind reviewing, I would still say CoRR is dealing with a more important issue than double-blindness does.

So given that we have this IEEE, ACM Digital Library access, do we also need CoRR?

I think that the digital libraries are very good and actually in terms of stuff that I really use in my research there is very little that isn't in either IEEE or ACM's Digital Library. Again, there are some issues: they are both proprietary (they are owned by professional societies), they're available for a fee (so they're not free to use), and they have standards that they have in terms of what material is included. And so things that don't make the cut with respect to the venues that get incorporated wouldn't be in there, whereas CoRR just has everything in it. The other point is that, to the extent that people put pre-prints or tech reports you get faster dissemination. For conferences, for example, things get into the digital library usually several months after the conference, it isn't even at the time of the conference.

One side effect of all the work we've been doing in the database community over the last 50 years (according to Rick Snodgrass, we're 50 years old now) is that scientists have huge amounts of data available to them that they didn't have in the past. How has this changed the way that they do science?

The way I think about this is that the standard way of doing science is what is called hypothesis-driven. You first pose a research question that you're going to ask, you have a hypothesis and then you do one or more of the things that you just mentioned -- let's say an experiment. The result of that experiment will either verify the hypothesis or refute it. And that's the classical scientific method of doing research. The thing that has now become possible is not to have a hypothesis but to have a goal that says "I'll find out something of interest in this space." So if one were to take a cartoon picture of data mining, as we would have talked about it even 15-20 years ago, we would simply say "Well, we have a lot of data and we look for patterns in the data." Let's say we stop there. That is what one can think of as data-driven scientific

research. One can say "I am looking for genes that have some role in some disease. I don't have a clue about anything, except I know how to do sequencing and so I'm going to take a bunch of people who have this disease and a bunch of people who don't. I don't have a hypothesis other than to say that there must be some genes that are different. Then I'm just going to run their DNA and look at where the differences are." This is not hypothesis-driven research. You can state it in terms of a hypothesis, but it is not a very interesting hypothesis.

In the example that I just gave, there actually is a data generation face to the research. One could do this with secondary data. One could say "I'm going to make use of other people's data that's published and do a secondary study with that". The point is that we're learning new things without knowing beforehand what we're going to learn. I think that this is very powerful because it decreases the burden on us to specify a hypothesis in advance. On the other hand, if one doesn't have a good explanation, at least in a post-hoc manner, one ends up with things that are intellectually dissatisfying and possibly even statistical flukes. I think that there is need for people who undertake this kind of scientific investigation to think harder from first principles about the statistics and what the likelihood is that they are seeing results that are not the result of over-fitting or the result of just multiple hypothesis testing or some other issue of this nature. I think that the standards of statistical evidence need to be much higher as a threshold for acceptance when one is doing it without a hypothesis.

One problem I see in the non-hypothesis-driven approach is that I'm not sure how well it's accepted by other people in science. So here's a direct quote from someone who is in the medical industry: "Oh those epidemiologists, they just want to go on fishing expeditions".

I think such statements are actually warranted in many situations because a great deal of the work that people are doing in a data-driven manner in many disciplines is often prey to all kinds of biases and errors. It's very easy not to have enough statistical power. It's very easy to have results that are incorrect because of multiple hypothesis testing or because of over-fitting or because of some other bias in terms of the way things were done. There are well-documented cases, for instance, of people showing things like moving objects in the distance through thought. You know, things of this nature which one shouldn't have a scientific basis to expect. Every now and then there is some such paper that gets published. If one conducts enough experiments there would be some case where just in terms of random association, things will turn out the way that you would like them to be. So, when one is considering some small data sample, and saying "Well in a data-driven manner I see this result, therefore it is", I think one has to take that with a very big pinch of salt.

I don't think that the database community is actually doing very much for scientists.

That having been said, I think that there is a question of the comfort zone for somebody who has been trained in a certain way of doing work. As a person who begins with the data, which is what I'm certainly trained to do, I often have discussion with people who are used to thinking about the hypothesis first and just feel uncomfortable at a gut level because they are being forced to think about things in ways that they're not used to. That will instinctively make them react negatively and then it's a question of them thinking it through, with their knowledge, training and wisdom and coming to a conclusion about whether some new piece of work done in a new style makes sense or not.

How well is the database community doing at supporting the needs of scientists?

I don't think that the database community is actually doing very much for scientists. I think that many scientists have a lot of data. I think they struggle with the data and they do all kinds of things with the data that may seem ridiculous to people who attend SIGMOD for instance, but they do it because that's what they know how to do. I think being able to provide tools to support their work, particularly as the amount of data that scientists are dealing with increases, is something we as the community should embrace and I know that at least some segments of our community are thinking hard about things like this. I think we have a long way to go.

Do you have a list of top challenges that we should be working on for the sake of scientists?

Actually, my view is that what we do for scientists is probably not that much different from what we would do for an end user in the consumer arena. I work with scientists as you've said, and I think about things that we might need to do in terms of data management to

help scientists do what they need to do. But when I write a paper in the database world, describing some result, it's usually not hard for me to take the same thing and cast it in terms of a hotel reservation or managing an address book, or something of this nature -- just very simple, personal tasks that end users would do for themselves. So I really think that the challenges are what one would expect if we just sat down and said "Who is using this stuff? What do they need to do?". I think that we get too wrapped up in dealing with what needs to be done inside the box with tightly defined boundaries and I think taking that one extra step of seeing what is it that someone is trying to accomplish with whatever is running on this box would make a world of a difference.

So if I am understanding things correctly, you're saying that the core guts of what they need is already there but it's not friendly enough, accessible enough, missing some layer on top perhaps for them to actually make use of it. Or maybe they don't know it exists…

Yes, to all of the above.

What is the right way to design a usable data management system?

My soapbox position on this has been that usability isn't skin deep. Which is to say, that you can't build a database system first and then throw a pretty interface on top of it and say that you now have a usable database system. Instead, I think you need to start at the beginning from what task the user is trying to accomplish and what knowledge the user brings to the task when they're trying to accomplish this and then see what the workflow should be to maximize their ability to accomplish that task directly and quickly.

To some extent the interface matters, but I think that even beyond the interface, as one thinks it through in terms of breaking a task into subtasks, and what is actually being done, one ends up with an interaction model. This is in effect the query model, the thing we should then have efficient support for. We had better design our database to be able to support that kind of interaction model, not to make people think about it that way. Usually we start with "I got this box and what can this box do?". And so we naturally end up with things that are not particularly usable.

So if I understand correctly then you're saying is we might need to redesign the core, the guts of the system once we figure out what these people really need.

I think that we will need to redesign significant aspects of it. We may not need to re-do all of it. If we get down to things like the actual data store which is, for most purposes, probably not something that would become visible, even there, I can give you a counterexample. So a paper that one of my students had last year[2] was on the system called CRIUS for organically grown database systems where the idea is that the user doesn't have a schema in mind before they start throwing data into the database and so as they come up with new instances, they realize that the schema needed to be richer than what they previously had. So, you start with a single column in a single table, and you grow it from there. Well, even though a lot of our contribution there had to do with how the user does this and what support the user gets and what dependencies mean and how do you keep the user from making errors, etc., the fact that the schema is evolving (and you expect the schema to evolve) on a continuous basis has implications on the kind of storage that you do. So for instance even if you didn't otherwise have a reason to do a vertical storage, the fact that you need to support something like schema evolution might tip the balance. So there are things like this that could affect decisions even at the gut level.

Is database research turning into informatics research?

So a thing I've been trying to do with very little success, when people ask me what area I work in, is to say "I work in information management". Quite often, I get a blank stare. They say "Oh, you mean databases", and that means something to them. I think the reason that I want to say information management and not databases is because, to me, a database is a very specific engine that does something we all understand, whereas information management is the broader universe. Databases have a significant role to play in information management, but I want to lay claim to the broader turf and somehow that has been difficult.

XML query optimization: should we give up and walk away, like we did for relational query optimization and call it done?

I think that at some point things matured to a point that the academic community has done pretty much what could be done. It doesn't mean that everybody should

[2] H. V. Jagadish, Arnab Nandi, Li Qian: Organic Databases. DNIS 2011: 49-63. There is also a more recent paper on the subject: H. V. Jagadish, Li Qian, Arnab Nandi: Organic databases. IJCSE 11(3): 270-283 (2015).

walk away but I think that the bulk of the interest moves on and every now and then there will be some willing person that comes in and changes the paradigm and makes us all think something new about something and such things may happen.

Why should we care about the Computing Research Association?

The Computing Research Association actually is something that does a great deal of good. It is an organization that not too many of us may have that much familiarity with. It is a professional society, except that the members of the society are computing research departments as opposed to individuals as say in the case of something like ACM. What the CRA does is think about what is good to support the computing research enterprise with a little bit of an administrative view much more so than say, something like the ACM. So in terms what they specifically do, there are things that are bread and butter everyday things. They do what is known as the Taulbee Survey of salaries and placements of graduates and things of this nature and this is something that helps us keep track of where things are in the field. Helps us keep track of the health of the field. Helps our department heads fight for larger raises…

> ***[…] usability isn't skin deep. […] you can't build a database system first and then throw a pretty interface on top of it and say that you now have a usable database system.***

(laughs)

Yes it does! That is one reason to pay attention to the CRA. But I think beyond this, the CRA is a good place because of the way it is set up to take action on items that are of broad interest to the computing community. For example, the CRA was responsible for a postdoctoral fellowship program that was put into place exactly when the downturn hit about three years ago and jobs dried up. This program is now being phased out as the economy recovers and as hiring is coming back up to normal levels. I think that if the organization weren't there, this wouldn't have happened. I forgot to mention that the CRA is very much North American, so it's not a worldwide thing unlike say, the ACM. So one of the things the CRA does spend significant effort on is in educating government officials on the benefits of funding computer science research. Again, in that, there have been many activities that the CRA has undertaken and the fact that there is generally bipartisan agreement in congress with regards to funding for computing research, is, to some extent, because the CRA has been very effective in making the case of the value that it brings to the economy and the society as a whole in return for a small amount of investment.

Do you have any words of advice for fledgling or midcareer database researchers?

Glad you're doing it.

Good choice! Among all your past research, do you have a favorite piece of work?

Yeah, there are a couple of things I could pull out. One is a paper that I wrote with Abraham Silberschatz and Inderpal Mumick on what we call the chronicle data model and this was published in PODS[3] and nobody paid attention to it, but the whole point of it was that there is often too much data coming at too fast a volume for you to be able to store it before you process it. So we developed a data model for dealing with it in an online manner with data streaming. About 5-7 years later the database community discovered data streaming and the name was streams and not chronicles, but I am proud of having been there first and first by several years. The other piece of work that I'm really proud of is the TAX paper[4], which is the algebra that underlay the TIMBER XML database. This is a paper that was rejected at all the major venues and we eventually published in DBPL and I just think that, of all of my work, is the piece that I find the most elegant and it underlay the entire TIMBER system that came afterwards.

Can you say a little more about what the central result of the paper was?

Yeah, the problem has to do with how do you do set-oriented processing for something like XML where you are going to deal with different fragments and fragments may have different shapes. So you don't

[3] H. V. Jagadish, Inderpal Singh Mumick, Abraham Silberschatz: View Maintenance Issues for the Chronicle Data Model. PODS 1995: 113-124

[4] H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, Keith Thompson: TAX: A Tree Algebra for XML. DBPL 2001: 149-164

have a set of uniform structures that you can deal with. Our basic solution was that every operator would, as its first step, have something that renders things uniform and afterwards you would apply the operator and then we can develop a set-oriented algebra. So that was the idea.

If you magically had enough extra time to do one additional thing at work that you are not doing now, what would it that be?

I would blog.

Oh! Well, you blogged recently[5].

That was one of the first times, and that was more of a community thing. I would blog more[6].

Good. We'll watch for a blog appearing soon on your webpage. If you could change one thing about yourself as a computer science researcher, what would it be?

I wish I were better trained.

What an indictment of Stanford!

You know I got my degree in Electrical Engineering.

Yeah… but there were computer scientists in Electrical Engineering too.

Yeah, this is not an indictment of the university at all. In any case times change, things change and what we need to know changes. Its just that I seem to come up against the limits of what I know how to do all the time. I wish I knew how to do *X* and if I just knew how to do *X*, I would be in so much a better place to address some problem. Then I say "Well, I much teach myself *X*", and of course I never get the time to teach myself *X* and so, that's how it goes.

What are some example X's that you wished you knew more about?

I wish I were a better theoretician.

Oh, more theory! Okay. Anything else comes to mind?

I wish I were a better systems builder.

Woah! We covered both sides right there, okay. Well thanks very much for talking with me today.

Thanks Marianne!

[5] H V Jagadish. Big Data: it's not just the analytics. ACM SIGMOD Blog. http://wp.sigmod.org/?p=430

[6] Jagadish's blog is available at http://www.bigdatadialog.com/

The Information Systems Group at HPI

Felix Naumann and Ralf Krestel
Hasso Plattner Institute
Potsdam, Germany
firstname.lastname@hpi.de

ABSTRACT

The Hasso Plattner Institute (HPI) is a private computer science institute funded by the eponymous SAP co-founder. It is affiliated with the University of Potsdam in Germany and is dedicated to research and teaching, awarding B.Sc., M.Sc., and Ph.D. degrees.

The Information Systems group was founded in 2006, currently has around ten Ph.D. students and about 15 masters students actively involved in our research activities. Our initial and still ongoing research focus has been the area of data cleansing and *duplicate detection*. More recently we have become active in the area of *text mining* to extract structured information from text, and even more recently in *data profiling*, i.e., the task of discovering various metadata and dependencies from a data instance.

1. MOTIVATION

Data abounds – it appears in many forms ranging from traditional relational or XML databases over semi-structured data, often published as linked open data, to textual data from documents on the Web. This wealth of data is ever growing, and many organizations and researchers have recognized the benefit of integrating it into larger sets of homogeneous, consistent, and clean data. Integrated data consolidates disconnected sources in organizations; it combines experimental results to gain new scientific insights; it provides consumers with a more complete view of product offers, etc.

Yet integration of such data is difficult due to its often extreme heterogeneity: Syntactic heterogeneity in data formats, access protocols, and query languages is typically the most simple to overcome, usually by building appropriate source-specific wrapper components. Next, structural heterogeneity must be overcome by aligning the different schemata of the datasets: Schema matching techniques automatically detect similarity and correspondence among schema elements, while schema mapping techniques interpret these to actually transform the data. Finally, to overcome semantic heterogeneity the different meanings of data and the similar but different representations of real-world entities must be recognized. Here, similarity search and data cleansing techniques are employed.

While the first two challenges have been research topics of our's in the past, the last and arguably most difficult challenge is a main focus of our current research endeavors. This focus manifests itself in three main research directions, which are motivated in the following sections: First, and most recently, in the area of *data profiling*, i.e., the development of methods to discover interesting properties about unknown datasets. Second, in the area of *data cleansing*, i.e., the development of methods to automatically correct errors and inconsistencies in databases and in particular to search and consolidate duplicates. Third, the area of *text mining*, i.e., the extraction of information from textual data, such as Wikipedia articles, tweets, or other text.

Where possible we aim at making our data and our algorithms available. A good starting point to find them is `http://hpi.de/naumann/projects/repeatability.html`.

2. DATA PROFILING

"Data profiling is the set of activities and processes to determine the metadata about a given dataset." [1] The need to profile a new or unfamiliar data arises in many situations, in general to prepare for some subsequent task. Data profiling comprises a broad range of methods to efficiently analyze a given dataset. In a typical scenario, mirroring the capabilities of commercial data profiling tools, tables of a relational database are scanned to derive metadata including data types and typical value patterns, completeness and uniqueness of columns, keys and foreign keys, and occasionally functional dependencies and association rules. In addition, research (ours and others') has proposed many methods for further tasks, such as the discov-

ery of inclusion dependencies or conditional functional dependencies. There are a number of concrete use cases for data profiling results, including:

- Query optimization: counts and histograms for selectivity estimation, dependencies for query simplification
- Data cleansing: pattern and dependency detection to identify violations
- Data integration: inter-database inclusion dependencies to enrich datasets and find join-paths
- Data analytics: data preparation and initial insights
- Database reverse engineering: foreign key discovery to understand a schema and identify its core components

Our survey [1] highlights the community's significant research progress in this area in the recent past. Data profiling is becoming a more and more popular topic as researchers and practitioners are recognizing that just gathering data into data lakes is not sufficient: "If we just have a bunch of data sets in a repository, it is unlikely anyone will ever be able to find, let alone reuse, any of this data. With adequate metadata, there is some hope [...]" [4]

2.1 Profiling relational data

Apart from computationally more simple tasks, such as counting the number of distinct values in a column, data profiling is typically concerned with discovering dependencies in a given, possibly large dataset. We, and other groups, have developed various methods to efficiently discover all minimal functional dependencies, inclusion dependencies, unique column combinations, and order dependencies. More dependencies are to come, such as join dependencies, matching dependencies, denial constraints, etc. Instead of listing and explaining each technique in any detail, we highlight some general difficulties we have encountered that make data profiling both challenging and interesting:

Schema size: Because dependencies can occur among any column or column combination, not only the number of records, but also the number of columns is a decisive factor of complexity.

Size of dependencies: One way to handle the exponential search space is to limit the size of the dependencies, i.e., the number of involved attributes. For instance, one could argue that key-candidates with more than ten attributes are not useful. On the other hand, a complete set of metadata can be useful, for instance to normalize a relation based on its functional dependencies.

Number of dependencies: While much dependency-focussed research, such as normalization theory or reasoning with dependencies, assumes a handful of dependencies as input, we typically observe thousands, millions and in some cases even billions of dependencies in typical real-world datasets. Just storing them becomes a problem, not to mention reasoning about them or interpreting them manually.

Treatment of nulls: The semantics of missing values is an interesting problem for almost any data management and analysis task, likewise for data profiling [13].

Intricate pruning: Huhtala et al. already showed quite complex insights to efficiently prune the search space for FD discovery [10]. When profiling for various types of dependencies, cross-dependency pruning becomes possible.

Relaxed dependencies: Apart from strict dependencies, it is also of interest to discover partial dependencies, which are true for only a part of the dataset, and conditional dependencies, which are true for a well-defined such part.

Dynamic data: While most of our focus has been on algorithms for a given, static dataset, we are also interested in efficiently updating data profiling results after changes in the data.

Experiments: Testing correctness of algorithms for given, real-world datasets is straightforward, but generating artificial testdata with certain properties, such as a certain number and distribution of functional dependencies, is very challenging.

Interpreting results: Any discovered metadata can only be validated for the dataset at hand. Some might be true in general, some might be spurious. We discuss this arguably most important and most difficult challenge of making sense of profiling results in Section 2.4.

In conclusion, research has many avenues to follow!

2.2 The Metanome project

Metanome is our open Java-based framework and tool for managing relational datasets and data profiling algorithms [18]. Our motivation for this undertaking is to bundle the many algorithms developed in our group, to provide an easy interface and testing environment for developers of new algorithms, and finally to enable fair comparisons among competing algorithms. Our initial focus was on functional dependency discovery, and Metanome features implementations of already eight published FD-discovery algorithms including those evaluated in [19] plus seven further algorithms for other discovery tasks (`www.metanome.de`).

2.3 Profiling RDF data

Among the datasets that are particularly worthy to profile, due to their variety and their general interest, are linked datasets. We are applying traditional and novel data mining technology to linked data in its RDF representation as subject-predicate-object triples. For instance, the discovery of frequent itemsets of predicates or objects in the context of subjects allows enriching datasets with missing triples. Another configuration – mining for frequent subjects in the context of predicates – achieves a clustering of entities. We have also applied data mining techniques for the discovery of conditional inclusion dependencies [16]. The volume of available linked data (a popular dataset is from the Billion Triples Challenge, which currently comprises over 3 billion facts) necessitates space-efficient algorithms.

Again, much of our work enters our browser-based discovery tool, ProLOD++ [2], which features techniques to discovery key-candidates, explore class and property distributed, discover frequent graph patterns, and more (see Figure 1).

2.4 From metadata to semantics

Finding all (and thus very many) dependencies in a given dataset is only the first part of a meaningful discovery process. The vast majority of metadata is spurious: It might be valid only in the current instance, or it might be valid for any reasonable instance but meaningless nonetheless. Separating the wheat from the chaff is extremely difficult, as it is a jump from (meta-)data to semantics; only a human can promote a unique column combination to a key, an inclusion dependency to a foreign key, or a functional dependency to an enforced constraint.

But computer science can help: We are currently investing much of our time to transform large amounts of metadata to schematic information. A first step is a metadata management system to store and query many different types of metadata. Next, we are developing selection and ranking methods to present to users only the most promising metadata. And finally, the visualization of metadata is an important tool to aid experts in understanding their data. Figure 2, for instance, shows connected components created by discovering inclusion dependencies among millions of web tables.

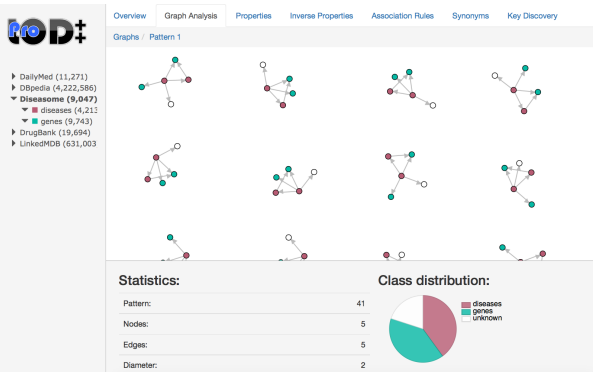


Figure 1: Exploring frequent patterns in a Linked Dataset with ProLOD (`www.prolod.org`)

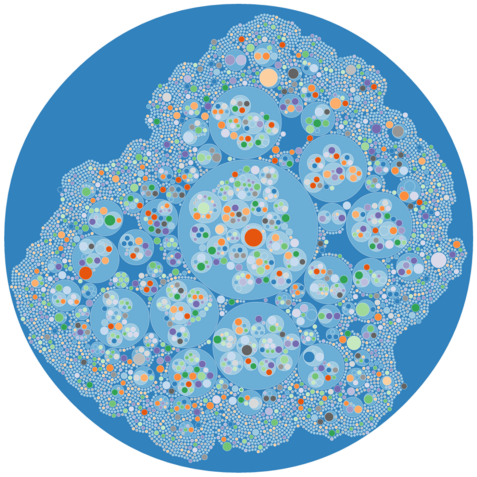


Figure 2: Clusters of web tables, connected through (reasonable) inclusion dependencies

3. DATA CLEANSING

With the ever-increasing volume of data, data quality problems arise. One of the most intriguing problems is that of multiple, yet different representations of the same real-world object in the data: duplicates. Such duplicates have many detrimental effects, for instance bank customers can obtain duplicate identities, inventory levels are monitored in-

correctly, catalogs are mailed multiple times to the same household, etc. A related problem is that of similarity search in structured data: given a query record, find the most similar candidate records in a database and identify whether one of them is a match.

The areas of similarity search and duplicate detection are experiencing a renaissance both in research and industry. Apart from scientific contributions we cooperate with companies to transfer our technology. Both our similarity search and our duplicate detection techniques have been adopted by industry partners.

3.1 Duplicate detection

Detecting duplicates is difficult: First, duplicate representations are usually not identical but slightly differ in their values. Second, in principle, all pairs of records should be compared, which is infeasible for large volumes of data [9]. Our research addresses both aspects by designing effective similarity measures and by developing efficient algorithms to reduce the search space.

One focus of our work is to develop improved variations of the elegant and simple sorted neighborhood method [8], for instance adapting it to nested XML data, making it progressive, parallelizing it for GPU-processing, or creating an adaptive version that is provably more efficient than the original [5].

In our experience, research(ers) in duplicate detection suffers particularly when trying to transfer technology and methods to industrial settings: Availability of data is a first issue, that arises even if a cooperation is firmly established and all participating parties in principle agree to the effort. Next, domain- and partner-specific similarity measures are needed that satisfy the specific use-case. Companies can have widely differing views of what constitutes a duplicate: Measuring recall is impossible due to large dataset sizes, and precision is surprisingly malleable, depending on whom one asks for validation. And finally, the real world holds many nitty, gritty details that can be conveniently ignored in a research setting[1]. With [20] we were able to overcome these difficulties and have had a lasting impact on the data quality of our partner.

3.2 Similarity search

A problem related to duplicate detection, but offering quite different requirements is that of efficient similarity search. Instead of comparing all or many pairs of records in an offline fashion ($n \times n$), online similarity search asks for all records matching a given query record ($1 \times n$). A typical use case is a call center agent pulling up customer information based on a customer's name and city. The main challenge is to develop a suitable similarity index, a much more difficult undertaking that an exact-match index.

One of our solutions matches the problem to a query plan optimization task, choosing similarity index accesses based on their selectivity and their cost, each of which is again modified by the dynamically chosen threshold: A low thresholds yields more candidates, but also more access to disk to retrieve the candidates [17]. A further insight is the importance of frequency-aware similarity measures, which apply different weights depending on the frequency of the query terms (Schwarzenegger vs. Miller).

We are currently extending this work to solve the problem of an ever-growing set of data that shall be held duplicate free: each query can simultaneously be an insert-operation.

[1]For instance, providing a machine with 16GB main memory but insisting on a 32-bit operating system.

4. TEXT MINING

Unstructured data in the form of textual documents can be found everywhere, from medical records to game chats, and from politicians' speeches to tweets. These documents cover a variety of genres, from serious to fun, from entire novels to single words. This diversity makes dealing with textual data particularly challenging and there is no one-size-fits-all text mining method yet. We are currently working on the topics of named entity linking, topic modeling, and bias detection on various document collections from the web. We cover the research areas natural language processing, information extraction, and recommender systems.

4.1 Named entity linking

A first step in analyzing texts is to find entities. Named entity linking is a rather new task composed of named entity recognition and linking the textual mentions in a document to corresponding entries in a knowledge base, thus disambiguating the mentions. The disambiguation can be performed using additional information in the knowledge base and the context of the mentions in the documents. We developed a named entity linking approach that operates on a textual range of relevant terms. We then aggregate decisions from an ensemble of simple classifiers, each of which operates on a randomly sampled subset from the above range [21]. The obtained results are very good with respect to precision and recall.

Some tasks, such as topic-based clustering, require near perfect precision and therefore we enhanced our named entity linking approach using random walks [6]. This allows for efficient computation of the linking and improves the precision at a minimal expense of recall.

4.2 Relationship extraction

Once entities are successfully extracted and disambiguated, finding relations between those entities is a next logical step. In the context of an industry project with a large German bank, we aim at building company networks to support their risk management department. These company networks are extracted automatically from newspaper articles, posing new challenges to the named entity recognition task, which is particularly difficult for German company names, due to complex, often ambiguous naming. Further, the relationship types we are interested in differ from standard, binary relations, such as "married with" or "located in". Our company networks require the detection of relations that are not necessarily binary, e.g. "competitor with" or "supplier to". To this end, we developed a holistic, seed-based algorithm to find these types of relations by providing a handful of example instantiations. The algorithm is based on Snowball [3] and can deal with any type of user-provided relations to extract relationship types with high precision.

4.3 Recommender systems

As with the information extraction tasks, we have a strong focus on the application of our research. Therefore, personalization, prediction, and recommendation play a major role in our group's work. From predicting accepted answers in MOOC forums [11], to recommending hashtags in Twitter [7], we analyzed diverse text collections accessible through the web. We also experimented with recommending serendipitous news articles [12] to present to the user not only relevant and novel articles, but also some surprising ones.

In an attempt to bridge the gap between traditional news and social media, we developed a tweet recommender system [15]. The goal was to provide the reader of a news article about some event with an overview of the reactions in Twitter. While Twitter is often only used to share and distribute information, it is also used to express opinions, reject ideas, or support certain viewpoints. To detect these (subtle) opinions, traditional sentiment analysis techniques have to be adapted to recognize emojis, abbreviations, slang, etc. The mismatch between the language used in news articles and tweets makes recommending one based on the other challenging.

4.4 Bias detection

Finally, we have to deal with another mismatch between used languages when trying to detect political bias of mainstream media. Initial experiments on comparing parliamentary speeches with news articles of various German news outlets [14] have shown that perceived bias can be automatically quantified. Given the very different genres (speeches vs. articles), detecting biased statements based on their comparison is rather cumbersome. Only rarely vocabulary use is a good indicator (e.g., "nuclear energy" vs. "atomic energy" in Germany). Nevertheless, identifying this bias in mainstream media and making it visible to the reader is an important piece of information.

Beside this *statement bias*, newspapers can also influence their readers by only reporting about certain topics (*gate-keeping bias*) or covering certain positions more thoroughly than others (*coverage bias*). Automatically detecting all three kinds of bias is our current goal, making it necessary to extract not only entities (politicians, parties, domain experts) and their relations, but also to do fine-grained opinion mining and sentiment analysis.

5. ACKNOWLEDGMENTS.

Our research has been funded by various partners including the DFG and companies interested in understanding and improving their data. We are very grateful for being able to work with our great Ph.D. students, who currently are Toni Gruetze, Hazar Harmouch, Maximilian Jenders, Anja Jentzsch, John Koumarelas, Sebastian Kruse, Konstantina Lazaridou, Michael Loster, Thorsten Papenbrock, Ahmad Samiei, and Zhe Zuo.

Two further groups at HPI, with which we collaborate, are based in the database community: The Enterprise Platforms and Integration Concepts (EPIC) group headed by Hasso Plattner and Matthias Uflacker, and the Knowledge Discovery and Data Mining (KDD) group headed by Emmanuel Müller.

6. REFERENCES

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB Journal*, 24(4):557–581, 2015.

[2] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. Profiling and mining RDF data with ProLOD++. In *Proceedings of the*

International Conference on Data Engineering (ICDE), 2014. Demo.

[3] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the ACM Conference on Digital Libraries*, pages 85–94, 2000.

[4] D. Agrawal et al. Challenges and opportunities with Big Data. Technical report, Computing Community Consortium, `http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf`, 2012.

[5] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg. Adaptive windows for duplicate detection. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1073–1083, Washington, D.C., 2012.

[6] T. Gruetze, G. Kasneci, Z. Zuo, and F. Naumann. CohEEL: Coherent and efficient named entity linking through random walks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2016.

[7] T. Gruetze, G. Yao, and R. Krestel. Learning temporal tagging behaviour. In *Proceedings of the Temporal Web Analytics Workshop (TempWeb) at the International World Wide Web Conference (WWW)*, 2015.

[8] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[9] M. Herschel, F. Naumann, S. Szott, and M. Taubert. Scalable iterative graph duplicate detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(11), 2012.

[10] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2):100–111, 1999.

[11] M. Jenders, R. Krestel, and F. Naumann. Which answer is best? Predicting accepted answers in MOOC forums. In *WWW Companion*, 2016.

[12] M. Jenders, T. Lindhauer, G. Kasneci, R. Krestel, and F. Naumann. A serendipity model for news recommendation. In *Advances in Artificial Intelligence - Annual German Conference on AI (KI)*, volume 9324 of *Lecture Notes in Computer Science*, pages 111–123, 2015.

[13] H. Köhler, S. Link, and X. Zhou. Possible and certain SQL keys. *Proceedings of the VLDB Endowment*, 8(11):1118–1129, 2015.

[14] R. Krestel, A. Wall, and W. Nejdl. Treehugger or Petrolhead? Identifying Bias by Comparing Online News Articles with Political Speeches. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 547–548, 2012.

[15] R. Krestel, T. Werkmeister, T. P. Wiradarma, and G. Kasneci. Tweet-recommender: Finding relevant tweets for news articles. In *Proceedings of the International World Wide Web Conference (WWW)*, 5 2015.

[16] S. Kruse, A. Jentzsch, T. Papenbrock, Z. Kaoudi, J.-A. Quiane-Ruiz, and F. Naumann. RDFind: Scalable conditional inclusion dependency discovery in RDF datasets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2016.

[17] D. Lange and F. Naumann. Efficient similarity search: Arbitrary similarity measures, arbitrary composition. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, Glasgow, UK, 2011.

[18] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome (demo). *Proceedings of the VLDB Endowment*, 8(12):1860–1871, 2015.

[19] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schnberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[20] M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster. Industry-scale duplicate detection. *Proceedings of the VLDB Endowment*, 1(2):1253–1264, 2008.

[21] Z. Zuo, G. Kasneci, T. Gruetze, and F. Naumann. BEL: Bagging for entity linking. In *International Conference on Computational Linguistics (COLING)*, 2014.

The Dark Citations of TODS Papers and What to Do About It—Or: Cite the Journal Paper

Christian S. Jensen
csj@cs.aau.dk

When assessing the excellence of a scientific paper, e.g., in a review, important aspects include the novelty and significance of its contribution, its scientific depth, and its mastery of the pertinent apparatus of computer science. The *excellence* of a researcher can be measured by their ability to publish in the scientific outlets with the highest reputation.

In contrast, the academic impact of the content of a paper can be measured by the number of citations to the paper. In some areas, it is easier to get citations than in other areas. However, when comparing two papers from the same area, one paper with many citations and one paper with few, the former can generally be considered as the more interesting, relevant, important, and/or impactful one. The academic *impact* of a researcher can then be measured by the number of citations to their papers.

However, although impact as measured by citations is then different from excellence, citations are still used for the rating of journals. Notably, journals are rated according to their citation-based impact factors, and a number of publishers advertise these statistics of their journals. Further, in some countries, the impact factors of a journal play an important role when different institutions assess the excellence of the journal. If a journal is not rated highly by funding agencies, researchers who rely on funding from those agencies are effectively encouraged to publish in other journals. Likewise, if a journal is not rated highly by hiring or promotion committees, candidates are effectively encouraged to publish in other journals. Because of reasons such as these, I find that it is not advisable to simply ignore citations.

A journal's two-year impact factor for a particular year n is calculated as the sum of the number of citations given during year n to each paper published in the journal during years $n-1$ and $n-2$, divided by the count of papers published during years $n-1$ and $n-2$. Thus, an impact factor of 2.5 for year 2015 means that papers published in that journal during 2013 and 2014 received an average of 2.5 citations during 2015. This definition does not state explicitly which citations are counted. When considering the two-year impact factor computed by Thomson Reuters, it is not entirely transparent which citations are counted. Thomson Reuters maintains a master journal list. Presumably, citations from papers in journals on this list are counted, but the extent to which other journals and also conferences are counted is not transparent. It is important for computer science that citations from conference papers are counted.

Having argued that citations are important, I will argue next that many citations to results published in TODS are not counted and that TODS papers should really have many more citations. This would substantially increase the citation statistics of TODS, including its two-year impact factor, and it would thus better reflect the externally perceived excellence of the journal and its papers.

A concrete example illustrates the issue. In June 2011, I and three coauthors published a paper in TODS entitled Design and analysis of a ranking approach to private location-based services. This paper is an extension

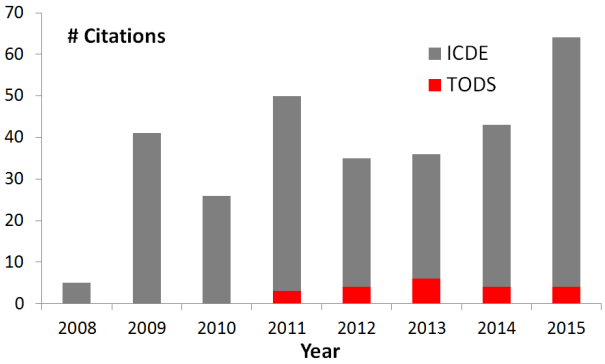


Figure 1: Citations to a 2008 ICDE paper and its TODS 2011 extended version (Source: Google Scholar as of May 14, 2016)

of a conference paper entitled *SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services* that we published in ICDE in 2008. We chose to extend this paper into a journal paper because we felt that its ap-

proach was quite novel. Also, the paper received encouraging reviews and was considered for the best paper award at ICDE. The journal paper offers more comprehensive coverage; for example, we involved a statistics professor in order to be able to analyze better the paper's ranking approach. Thus, the journal paper contains everything that the conference paper contains, and significantly more.

Figure 1 shows the citations to the two papers. The journal paper received 3, 4, 6, 4, and 4 citations in the years 2011 to 2015, respectively. If these citations are all counted, the paper contributes 4 citations to the TODS 2012 impact factor and 6 citations to the 2013 impact factor. The conference paper received 5, 41, 26, 47, 31, 30, 39, and 60 citations in the years 2008 to 2015. If these citations are all counted, the paper contributes 41 citations to the ICDE 2009 impact factor and 26 citations to the 2010 impact factor.

In this example, a total of 21 citations are counted for the results published in the journal paper from 2011 to 2015, but considering also the citations to the conference paper, the citations to the results are 228 from 2011 to 2015. The 207 concurrent citations to the conference paper are the dark citations that are not counted. The difference between the counted citations and the uncounted dark citations is an order of magnitude! Imagine the difference it would make if these citations were counted.

It is common practice in the database area and other areas of computer science to first publish papers in conferences and only then publish extended versions in journals. Indeed, database and other journals accept extended conference papers, and they publish many papers that are extensions of conference papers. TODS requires that extended versions include at least 30% of new content material (see `http://tods.acm.org/ThirtyPercentRulePolicy.cfm`), and I estimate that around three quarters of the papers published each year are extensions of conference papers.

So far, I have argued that we cannot simply ignore citations and that results published in TODS receive many more citations than are actually counted. Why does the problem occur and how can we fix the problem?

The example shows that other papers continue to cite the conference paper even when it has been superseded by an extended journal paper. This practice may occur because the conference paper is cited initially, as only it exists. (This was true for 2008 through 2010.) Then the authors of subsequent papers just keep citing the conference paper. They may not have noticed that an extended journal version had becomes available, as they already have something to cite. That said, in my view, this practice is generally not one that makes the most sense from an academic perspective.

On possible action that addresses the problem is for TODS to publish a higher fraction of papers that do not extend a conference paper. Such papers have no dark citations. TODS has already started to encourage more submissions of such original papers, by making them eligible for presentation at SIGMOD (see the editorial *The Best of Two Worlds—Present Your TODS Paper at SIGMOD* in the June 2015 issue of TODS). Other journals have established fast-track publication schemes for original papers. TODS could do something similar. However, this action can only partially fix the problem.

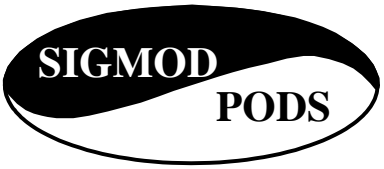
Another possible action is to develop a citation metric and system that takes the dark citations into account when assessing the citation performance for the results published in journals. While I think that such a metric and system make sense, the result is yet another metric that may not be adopted where it counts. Specifically, it is going to be a long, tedious, and up-hill battle to get publishers to use yet another metric, and it may be even harder to get institutions to adopt the new metric.

I propose a very practical action that authors can start taking right now and that I think is good for science. Specifically, I propose to address the problem of dark citations by always citing the extended journal version of a paper whenever it is available. The journal version is the definitive and most recent account of the research. The journal version has gone through an additional and more formal review process. The journal version extends and, likely, consolidates the conference version's results. And the journal version is likely to offer a better and more up-to-date coverage of related work. These are all good arguments for citing the journal version.

There can be reasons for also citing the conference version. One is that it may be important to establish the order of invention. It may have taken several years for an extended version to appear in a journal because it takes time to develop the new results, because the review process and revisions take time, and because there may be a delay from acceptance to actual publication in an issue. A possible reason for citing only the conference version occurs if one wants to make reference to content in the conference version that is not present in the journal version. However, in my experience as an editor and an author, this situation occurs rarely.

In summary, it is important for the database community to have journals that are not only excellent, but are also highly cited. Results published in TODS have many more citations than are counted. *You can help by citing the extended journal paper when one exists.*

Acknowledgments My colleague Rick Snodgrass, a former TODS Editor-in-Chief, provided valuable comments that helped improve the presentation.



CALL FOR PAPERS

36th ACM SIGMOD–SIGACT–SIGART Symposium on

PRINCIPLES OF DATABASE SYSTEMS (PODS 2017)

May 14 – May 19, 2017, Raleigh, North Carolina, USA

http://sigmod2017.org

Program Chair:

Floris Geerts
University of Antwerp
floris.geerts@uantwerpen.be

Program Committee:

Leopoldo Bertossi *(Carleton Univ.)*
Meghyn Bienvenu *(CNRS, Univ. of Montpellier)*
Angela Bonifati *(Univ. de Lyon)*
Andrea Calì *(Univ. of London)*
Rada Chirkova *(NC State Univ.)*
Giuseppe De Giacomo *(Sapienza Univ. di Roma)*
Ting Deng *(Beihang Univ.)*
Diego Figueira *(CNRS)*
Georg Gottlob *(Oxford Univ.)*
Paraschos Koutris *(Univ. of Wisconsin-Madison)*
Andrew McGregor *(Univ. of Massachusetts)*
Gerome Miklau *(Univ. of Massachusetts)*
Jeff Phillips *(Univ. of Utah)*
Andreas Pieris *(Vienna Univ. of Technology)*
Juan L. Reutter *(Pontificia Univ. Católica)*
Thomas Schwentick *(Univ. Dortmund)*
Francesco Silvestri *(IT Univ. of Copenhagen)*
Yufei Tao *(Chinese Univ. of Hong Kong)*
Stijn Vansummeren *(Univ. Libre de Bruxelles)*
Jef Wijsen *(Univ. de Mons)*
Qin Zhang *(Indiana Univ.)*

PODS General Chair:

Tova Milo *(Tel Aviv Univ.)*

Proceedings Chair:

Emanuel Sallinger *(Oxford Univ.)*

Publicity Chair:

Paolo Guagliardo *(Univ. of Edinburgh)*

Important Dates

First Submission Cycle:

Abstract submission	Jun 12, 2016
Paper submission	Jun 19, 2016
Accept/Reject/Revise	Aug 28, 2016
Revision deadline	Sep 25, 2016
Accept/Reject (Revisions)	Oct 30, 2016

Second Submission Cycle:

Abstract submission	Dec 11, 2016
Paper submission	Dec 18, 2016
Accept/Reject notification	Feb 26, 2017
Camera ready	Mar 19, 2017

All deadlines end at 11:59pm PST.

The PODS symposium series, held in conjunction with the SIGMOD conference series, provides a premier annual forum for the communication of new advances in the theoretical foundations of data management, traditional or non-traditional.

For the 36th edition, PODS continues to aim to broaden its scope, and calls for research papers providing original, substantial contributions along one or more of the following aspects:

- *deep theoretical exploration of topical areas central to data management;*
- *new formal frameworks that aim at providing the basis for deeper theoretical investigation of important emerging issues in data management; and*
- *validation of theoretical approaches from the lens of practical applicability in data management.*

Topics that fit the interests of the symposium include:

- *design, semantics, query languages*
- *databases and knowledge representation*
- *data models, data structures, algorithms for data management*
- *concurrency & recovery, distributed/parallel databases, cloud computing*
- *model theory, logics, algebras, computational complexity*
- *graph databases and (semantic) Web data*
- *data mining, information extraction, search*
- *data streams*
- *database aspects of machine learning*
- *data-centric (business) process management, workflows, web services*
- *incompleteness, inconsistency, uncertainty in data management*
- *data and knowledge integration and exchange, data provenance, views and data warehouses, metadata management*
- *domain-specific databases (multi-media, scientific, spatial, temporal, text)*
- *data privacy and security*

Submission Guidelines: Submitted papers must be formatted using the designated style file (sig-alternate-10.cls) which uses 10pt font size and line spacing of 11pt using the "\documentclass{sig-alternate-10}" command in your LaTeX document. Submitted papers should be at most twelve pages, excluding bibliography. Additional details may be included in an appendix, which, however, will be read at the discretion of the PC. Submissions that do not conform to these guidelines risk rejection without consideration of their merits.

The submission process is online, using https://easychair.org/conferences/?conf=pods2017. Note that, unlike the SIGMOD conference, PODS does not use double-blind reviewing, and therefore PODS submissions should have the names and affiliations of authors listed on the paper.

The results of submitted paper must be unpublished and not submitted elsewhere, including the formal proceedings of other symposia or workshops. Authors of an accepted paper will be expected to sign copyright release forms, and one author is expected to present it at the conference.

Awards: Awards may be given, as judged by the program committee, for best paper and best student paper. More details can be found on the conference website.