Reasoning on data partitioning for single-round multi-join evaluation in massively parallel systems

Peer-reviewed author version

AMELOOT, Tom; GECK, Gaetano; KETSMAN, Bas; NEVEN, Frank & Schwentick, Thomas (2017) Reasoning on data partitioning for single-round multi-join evaluation in massively parallel systems. In: Communications of the ACM, 60(3), p. 93-100.

DOI: 10.1145/3041063
Handle: http://hdl.handle.net/1942/23351

Reasoning on data partitioning for single-round multi-join evaluation in massively parallel systems

Tom J. Ameloot[*]
Hasselt University &
transnational University of
Limburg
tom.ameloot@uhasselt.be

Gaetano Geck
TU Dortmund University
gaetano.geck@udo.edu

Bas Ketsman[†]
Hasselt University &
transnational University of
Limburg
bas.ketsman@uhasselt.be

Frank Neven
Hasselt University &
transnational University of
Limburg
frank.neven@uhasselt.be

Thomas Schwentick
TU Dortmund University
thomas.schwentick@udo.edu

ABSTRACT

Evaluating queries over massive amounts of data is a major challenge in the big data era. Modern massively parallel systems, like e.g. Spark, organize query answering as a sequence of rounds each consisting of a distinct communication phase followed by a computation phase. The communication phase redistributes data over the available servers, while in the subsequent computation phase each server performs the actual computation on its local data. There is a growing interest in single-round algorithms for evaluating multiway joins where data is first reshuffled over the servers and then evaluated in a parallel but communication-free way. As the amount of communication induced by a reshuffling of the data is a dominating cost in such systems, we introduce a framework for reasoning about data partitioning to detect when we can avoid the data reshuffling step. Specifically, we formalize the decision problems parallel-correctness and transfer of parallel-correctness, provide semantical characterizations, and obtain tight complexity bounds.

1. INTRODUCTION

The background scenario for this work is that of large-scale data analytics where massive parallelism is utilized to answer complex join queries over multiple database tables. For instance, as described by Chu et al. [6], data analytics engines face new kinds of workloads, where multiple large tables are joined, or where the query graph has cycles. Furthermore, recent in-memory systems (e.g., [10, 12, 18, 22]) can fit data in main memory by utilizing a multitude of servers. Koutris and Suciu [11] introduced the Massively Parallel Communication model (MPC model) to facilitate an understanding of the complexity of query processing on shared-nothing parallel architectures. For such systems, performance is no longer dominated by the number of I/O requests to external memory as in traditional systems but by the communication cost for reshuffling data during query execution. When queries need to be evaluated in several rounds, such reshuffling can repartition the whole database and can thus be very expensive.

While in traditional distributed query evaluation, multi-join queries are computed in several stages over a join tree possibly transferring data over the network at each step, we focus on query evaluation algorithms within the MPC model that only require *one* round of communication. Such algorithms consist of two phases: a *distribution phase* (where data is repartitioned or reshuffled over the servers) followed by an *evaluation phase*, where each server contributes to the query answer in isolation, by evaluating the query at hand over the local data without any further communication. We refer to such algorithms as *generic one-round algorithms.* Afrati and Ullman [1] describe an algorithm that computes a multi-join query in a *single* communication round. The algorithm uses a technique that can be traced back to Ganguly, Silberschatz, and Tsur [8]. Beame, Koutris and Suciu [3, 4] refined the algorithm, named it *HyperCube*, and showed that it is a communication-optimal algorithm for single-round distributed evaluation of conjunctive queries.

The original version of this paper is entitled "Parallel-Correctness and Transferability for Conjunctive Queries" and was first published in the *Proceedings of the 2015 ACM Symposium on Principles of Database Systems.* A modified version entitled "Data partitioning for single-round multi-join evaluation in massively parallel systems" appeared in the March 2016 issue of *ACM Sigmod Record.*

The generic one-round HyperCube algorithm requires a reshuffling of the base data for *every* separate query. As the amount of communication induced by a reshuffling of the data can be huge, it is important to detect when the reshuffle step can be avoided. We present a framework for reasoning about data partitioning for generic one-round algorithms for the evaluation of queries under *arbitrary* distribution policies, not just those resulting from the HyperCube algorithm. To target the widest possible range of repartitioning strategies, the initial distribution phase is therefore modeled by a distribution policy that can be *any* mapping from facts to subsets of servers.

The optimization framework is motivated by two concrete

[*] Postdoctoral Fellow of the Research Foundation - Flanders (FWO).

[†] PhD Fellow of the Research Foundation - Flanders (FWO).

scenarios. In the first scenario, we assume that the data is already partitioned over the servers and we want to know whether a given query can be evaluated correctly over the given data distribution *without reshuffling the data.* In the second scenario, the data distribution might be unknown or hidden, but it is known that it allowed the correct evaluation of the *previous* query. Here, we ask whether this knowledge guarantees that the given (next) query can be evaluated correctly without reshuffling. To this end, we formalize the following decision problems:

Parallel-Correctness: Given a distribution policy and a query, can we be sure that the corresponding generic one-round algorithm will always compute the query result correctly—no matter the actual data?

Parallel-Correctness Transfer: Given two queries $\mathcal{Q}$ and $\mathcal{Q}'$, can we infer from the fact that $\mathcal{Q}$ is computed correctly under the current distribution policy, that $\mathcal{Q}'$ is computed correctly as well?

We say that parallel-correctness *transfers* from $\mathcal{Q}$ to $\mathcal{Q}'$, denoted $\mathcal{Q} \xrightarrow{\text{pc}} \mathcal{Q}'$, when $\mathcal{Q}'$ is parallel-correct under *every* distribution policy for which $\mathcal{Q}$ is parallel-correct. Parallel-correctness transfer is particularly relevant in a setting of automatic data partitioning where an optimizer tries to automatically partition the data across multiple nodes to achieve overall optimal performance for a specific workload of queries (see, e.g., [14, 17]). Indeed, when parallel-correctness transfers from a query $\mathcal{Q}$ to a set of queries $\mathcal{S}$, then any distribution policy under which $\mathcal{Q}$ is parallel-correct can be picked to evaluate all queries in $\mathcal{S}$ without reshuffling the data.

We focus in this paper on conjunctive queries and first study the parallel-correctness problem. We give a characterization of parallel-correctness: a distribution policy is parallel-correct for a query, if and only if for every *minimal* valuation of the query there is a node in the network to which the distribution assigns all facts required by that valuation. This criterion immediately yields a Π_2^P upper bound for parallel-correctness, for various representations of distribution policies. It turns out that this is essentially optimal, since the problem is actually Π_2^P-complete. These results also hold in the presence of union and inequalities. When negation is added, deciding parallel-correctness might involve counterexample databases of exponential size. More specifically, in the presence of negation deciding parallel-correctness is coNEXPTIME-complete. The latter result is related to the new result that query containment for conjunctive queries with negation is coNEXPTIME-complete, as well.

For parallel-correctness transfer we also first provide a semantical characterization in terms of a (value-based) containment condition for minimal valuations of $\mathcal{Q}'$ and $\mathcal{Q}$ (Proposition 6.4). Deciding transferability of parallel-correctness for conjunctive queries is Π_3^P-complete, again even in the presence of unions and inequalities. We emphasize that the implied exponential time algorithm for parallel-correctness transfer does not rule out practical applicability since the running time is exponential in the size of the queries not in the size of a database.

Outline. In Section 2, we introduce the necessary preliminaries regarding databases and conjunctive queries. In Section 3, we discuss the MPC model. In Section 4, we exemplify the HyperCube algorithm. In Section 5 and Section 6, we explore parallel-correctness and parallel-correctness transfer. We present concluding remarks together with direction for further research in Section 7.

2. CONJUNCTIVE QUERIES

In this article, a *(database) instance* I is a finite set of facts of the form $R(a_1,\dots,a_n)$, where R is an n-ary relation symbol from a given database schema and each a_i is an element from some given infinite domain **dom**.

A *conjunctive query (CQ)* $\mathcal{Q}$ is an expression of the form

$$H(\mathbf{x}) \leftarrow R_1(\mathbf{y_1}),\dots,R_m(\mathbf{y_m}),$$

where every R_i is a relation name, every tuple $\mathbf{y_i}$ matches the arity of R_i, and every variable in $\mathbf{x}$ occurs in some $\mathbf{y}_i$. We refer to the *head atom* $H(\mathbf{x})$ by $head_{\mathcal{Q}}$ and to the set $\{R_1(\mathbf{y_1}),\dots,R_m(\mathbf{y_m})\}$ by $body_{\mathcal{Q}}$. We denote by $vars(\mathcal{Q})$ the set of all variables occurring in $\mathcal{Q}$.

A *valuation* for a CQ $\mathcal{Q}$ maps its variables to values, that is, it is a function $V : vars(\mathcal{Q}) \to \mathbf{dom}$. We refer to $V(body_{\mathcal{Q}})$ as the facts *required* by V. A valuation V *satisfies* $\mathcal{Q}$ on instance I if all facts required by V are in I. In that case, V *derives* the fact $V(head_{\mathcal{Q}})$. The *result* of $\mathcal{Q}$ on instance I, denoted $\mathcal{Q}(I)$, is defined as the set of facts that can be derived by satisfying valuations for $\mathcal{Q}$ on I. We denote the class of all CQs by **CQ**.

Example 2.1. Let I_e be the example database instance

$$\big\{\mathsf{Like}(a,b),\mathsf{Like}(b,a),\mathsf{Like}(b,c),\mathsf{Dislike}(a,a),\mathsf{Dislike}(c,a)\big\},$$

and $\mathcal{Q}_e$ be the example CQ

$$H(x_1,x_3) \leftarrow \mathsf{Like}(x_1,x_2),\mathsf{Like}(x_2,x_3),\mathsf{Dislike}(x_3,x_1).$$

Then $V_1 = \{x_1 \mapsto a, x_2 \mapsto b, x_3 \mapsto a\}$ and $V_2 = \{x_1 \mapsto a, x_2 \mapsto b, x_3 \mapsto c\}$ are the only satisfying valuations. Consequently, $\mathcal{Q}_e(I_e) = \big\{H(a,a), H(a,c)\big\}$. □

3. MPC MODEL

The Massively Parallel Communication (MPC) model was introduced by Koutris and Suciu [11] to study the parallel complexity of conjunctive queries. It is motivated by query processing on big data that is typically performed on a shared-nothing parallel architecture where data is stored on a large number of servers interconnected by a fast network. In the MPC model, computation is performed by p servers connected by a complete network of private channels. Examples of such systems include Pig [16], Hive [19], Dremel [12], and Spark [22]. The computation proceeds in rounds where each round consists of two distinct phases:

- *Communication Phase:* The servers exchange data by communicating with all other servers.
- *Computation Phase:* Each server performs only local computation (on its local data).

The number of rounds then corresponds to the number of synchronization barriers that an algorithm requires. The input data is initially partitioned among the p servers and every server receives $1/p$-th of the data. There are no assumptions on the particular partitioning scheme. At the end of the execution, the output must be present in the union of the p servers. As the model focuses primarily on quantifying the amount of communication there is no a priori bound on the computational power of a server. A relevant measure is the *load* at each server, which is the amount of data received by a server during a particular round. Examples of optimization goals are minimizing total load (e.g., [1]) and minimizing maximum load (e.g., [11]).

To get a feeling for the model, we next present simple examples of single- and multi-round algorithms in the MPC model for evaluating specific conjunctive queries.

Example 3.1. (1) Consider the query Q_1

$$H(x, y, z) \leftarrow R(x, y), S(y, z)$$

joining two binary relations R and S over a common attribute. Let h be a hash function mapping every domain value to one of the p servers. The following single-round algorithm computes Q_1. In the communication phase, executed by every server on its local data, every tuple $R(a, b)$ is sent to server $h(b)$ while every tuple $S(c, d)$ is sent to server $h(c)$. In the computation phase, every server evaluates Q_1 on the received data. The output of the algorithm is the union of the results computed at the computation phase. This strategy is called a repartition join in [5].

(2) Let Q_2 be the triangle query:

$$H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x).$$

One way to evaluate Q_2 is by two binary joins leading to a two-round algorithm. We assume two hash functions h and h'. In the first round, all tuples $R(a, b)$ and $S(c, d)$ are sent to servers $h(b)$ and $h(c)$, respectively. The computation phase computes the join of R and S at each server in a relation K. In the second round, each resulting triple $K(e, f, g)$ is sent to $h'(e, g)$, while each tuple $T(i, j)$ is sent to $h'(j, i)$. Finally, K and T are joined at each server. □

We note that every MapReduce [7] program can be seen as an algorithm within the MPC model since the map phase and reducer phase readily translate to the communication and computation phase of MPC.

4. HYPERCUBE ALGORITHM

To illustrate the HyperCube algorithm, we show in the following example that the triangle query of Example 3.1(2) can be evaluated by a single-round MPC algorithm.

Example 4.1. Consider again the triangle query Q_2 of Example 3.1(2):

$$H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x).$$

Let α_x, α_y, and α_z be positive natural numbers such that $\alpha_x \alpha_y \alpha_z = p$. Every server can then uniquely be identified by a triple in $[1, \alpha_x] \times [1, \alpha_y] \times [1, \alpha_z]$. For $c \in \{x, y, z\}$, let h_c be a hash function mapping each domain value to a number in $[1, \alpha_c]$. The algorithm then operates as follows. In the communication phase, every fact

- $R(a, b)$ is sent to every server with coordinate $(h_x(a), h_y(b), \alpha)$ for every $\alpha \in [1, \alpha_z]$; so, $R(a, b)$ is sent to the subcube determined by the hash values $h_x(a)$ and $h_y(b)$ in the x- and y-dimension, respectively, as illustrated in Figure 1(a);
- $S(b, c)$ is sent to every server with coordinate $(\alpha, h_y(b), h_z(c))$ for every $\alpha \in [1, \alpha_x]$; and,
- $T(c, a)$ is sent to every server with coordinate $(h_x(a), \alpha, h_z(c))$ for every $\alpha \in [1, \alpha_y]$.

We note that every R-tuple is replicated α_z times and similarly for S- and T-tuples.

The computation phase consists of evaluating Q_2 on the local data at each server. The algorithm is correct since for every valuation V for Q_2 some server contains the facts

$$\{V(R(x, y)), V(S(y, z)), V(T(z, x))\}$$

if and only if the (hypothetical) centralized database contains those facts. In this sense, the algorithm distributes the space of all valuations of Q_2 over the computing servers in an instance independent way through hashing of domain values. In the special case that $\alpha_x = \alpha_y = \alpha_z = p^{1/3}$, each tuple is replicated $p^{1/3}$ times. Assuming each relation consists of m tuples and there is no skew, each server will receive $m/p^{2/3}$ tuples for each of the relations R, S, and T. So, the maximum load per server is $O(m/p^{2/3})$. □

The technique in Example 4.1 can be generalized to arbitrary conjunctive queries and was first introduced in the context of MapReduce by Afrati and Ullman [1] as the *Shares algorithm*. The values α_x, α_y, and α_z are called shares (hence, the name) and the work of Afrati and Ullman focuses on computing optimal values for the shares minimizing the total load (as a measure for the communication cost).

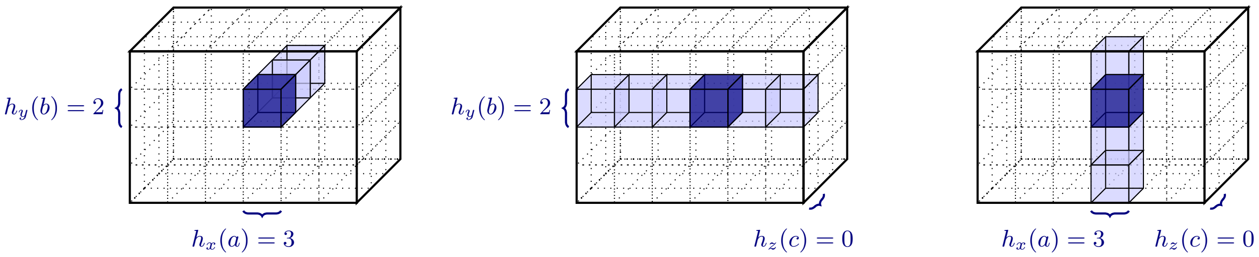
Beame, Koutris, and Suciu [3, 4] show that the method underlying Example 4.1 is essentially communication optimal for full conjunctive queries $\mathcal{Q}$. Assuming that the sizes of all relations are equal to m and under the assumption that there is no skew, the maximum load per server is bounded by $O(m/p^{1/\tau^*})$ with high probability. Here, τ^* depends on the structure of $\mathcal{Q}$ and corresponds to the optimal fractional edge packing (which for Q_2 is $\tau^* = 3/2$). The algorithm is referred to as *HyperCube* in [3, 4]. Additionally, the bound is tight over all one-round MPC algorithms indicating that HyperCube is a fundamental algorithm.

Chu, Balazinska, and Suciu [6], provide an empirical study of HyperCube (in combination with a worst-case optimal algorithm for sequential evaluation [15, 21]) for complex join queries, and establish, among other things, that HyperCube performs well for join queries with large intermediate results. On the other hand, HyperCube can perform badly on queries with small output.

5. PARALLEL-CORRECTNESS

In the remainder of this paper, we present a framework for reasoning about data partitioning for generic one-round algorithms for the evaluation of queries under *arbitrary* distribution policies. We recall from the introduction that such algorithms consist of a distribution phase (where data is repartitioned or reshuffled over the servers) followed by an evaluation phase where each server evaluates the query at hand over the local data. In particular, generic one-round algorithms are one-round MPC algorithms where every server in the computation phase evaluates the same given query.

When such algorithms are used in a multi-query setting, there is room for optimization. We recall that the HyperCube algorithm requires a reshuffling of the base data for *every* separate query. As the amount of communication induced by a reshuffling of the data can be huge, it is relevant to detect when the reshuffle step can be avoided and



(a) Replication of $R(a, b)$ (b) Replication of $S(b, c)$ (c) Replication of $T(c, a)$

Figure 1: HyperCube distribution policies view the computing nodes in the network as arranged in a multi-dimensional grid. Each dimension corresponds to a variable of the query to be computed. Replication happens in a structurally restricted way: along a line, a plane or a hyperplane. This figure illustrates the replication of facts $R(a, b), S(b, c), T(c, a)$ as required by a valuation for the triangle query in Example 4.1 for values $p = 72$, $\alpha_x = 6$, $\alpha_y = 4$, and $\alpha_z = 3$. All facts meet at the node with coordinate $\big(h_x(a), h_y(b), h_z(c)\big) = (3, 2, 0)$. Therefore the fact $H(a, b, c)$ can be derived locally, as desired.

the current distribution of the data can be reused to evaluate another query. Here, parallel-correctness and parallel-correctness transfer become relevant static analysis tasks. We study parallel-correctness in this section and parallel-correctness transfer in the next section.

Before we can address the parallel-correctness problem in detail, we first need to fix our model and our notation.

A characteristic of the HyperCube algorithm is that it reshuffles data on the granularity of facts and assigns each fact in isolation (that is, independent of the presence or absence of any other facts) to a subset of the servers. This means that the HyperCube reshuffling is independent of the current distribution of the data and can therefore be applied locally at every server. We therefore define distribution policies as arbitrary mappings linking facts to servers.

Following the MPC model, a *network* $\mathcal{N}$ is a nonempty finite set of node names. A *distribution policy* $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$ for a network $\mathcal{N}$ consists of a universe U and a total function $\mathit{rfacts}_{\boldsymbol{P}}$ that maps each node of $\mathcal{N}$ to a subset of facts[1] from $\mathit{facts}\big(U\big)$. Here, $\mathit{facts}\big(U\big)$ denotes the set of all possible facts over U. A node κ is *responsible for a fact* $\boldsymbol{f}$ (under policy $\boldsymbol{P}$) if $\boldsymbol{f} \in \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$. For an instance I and a $\kappa \in \mathcal{N}$, let $\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa)$ denote $I \cap \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$, that is, the set of facts in I for which node κ is responsible. We refer to a given database instance I as the *global instance* and to $\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa)$ as the *local instance on node* κ.

The result $[\mathcal{Q}, \boldsymbol{P}](I)$ of the distributed evaluation in one round of a query $\mathcal{Q}$ on an instance I under a distribution policy $\boldsymbol{P}$ is defined as the union of the results of $\mathcal{Q}$ evaluated over every local instance. Formally,

$$[\mathcal{Q}, \boldsymbol{P}](I) \overset{\text{def}}{=} \bigcup_{\kappa \in \mathcal{N}} \mathcal{Q}\big(\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa)\big).$$

Example 5.1. Let I_e be the example database instance

$$\big\{\mathsf{Like}(a, b), \mathsf{Like}(b, a), \mathsf{Like}(b, c), \mathsf{Dislike}(a, a), \mathsf{Dislike}(c, a)\big\},$$

and $\mathcal{Q}_e$ be the example CQ

$$H(x_1, x_3) \leftarrow \mathsf{Like}(x_1, x_2), \mathsf{Like}(x_2, x_3), \mathsf{Dislike}(x_3, x_1)$$

from Example 2.1. Consider a network $\mathcal{N}_e$ consisting of two nodes $\{\kappa_1, \kappa_2\}$. Let $\boldsymbol{P}_1 = (\{a, b, c\}, \mathit{rfacts}_{\boldsymbol{P}}\})$ be the distribution policy that assigns all Like-facts to both nodes κ_1 and κ_2, and every fact $\mathsf{Dislike}(d_1, d_2)$ to node κ_1 when $d_1 = d_2$ and to node κ_2 otherwise. Then,

$$\mathit{loc\text{-}inst}_{\boldsymbol{P}_1,I_e}(\kappa_1) = \\ \big\{\mathsf{Like}(a, b), \mathsf{Like}(b, a), \mathsf{Like}(b, c), \mathsf{Dislike}(a, a)\big\},$$

and

$$\mathit{loc\text{-}inst}_{\boldsymbol{P}_1,I_e}(\kappa_2) = \\ \big\{\mathsf{Like}(a, b), \mathsf{Like}(b, a), \mathsf{Like}(b, c), \mathsf{Dislike}(c, a)\big\}.$$

Furthermore,

$$[\mathcal{Q}_e, \boldsymbol{P}_1](I_e) = \\ \mathcal{Q}_e\big(\mathit{loc\text{-}inst}_{\boldsymbol{P}_1,I_e}(\kappa_1)\big) \cup \mathcal{Q}_e\big(\mathit{loc\text{-}inst}_{\boldsymbol{P}_1,I_e}(\kappa_2)\big),$$

which is just $\{H(a, b)\} \cup \{H(a, c)\}$.

We get $[\mathcal{Q}_e, \boldsymbol{P}_2](I_e) = \emptyset$ for the distribution policy $\boldsymbol{P}_2$ that assigns all Like-facts to node κ_1 and all $\mathsf{Dislike}$-facts to node κ_2. □

Now we can define parallel-correctness:

Definition 5.2. A query $\mathcal{Q}$ is *parallel-correct on instance* I *under distribution policy* $\boldsymbol{P}$ if $\mathcal{Q}(I) = [\mathcal{Q}, \boldsymbol{P}](I)$.
$\mathcal{Q}$ is *parallel-correct under distribution policy* $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$, if it is parallel-correct on all instances $I \subseteq \mathit{facts}\big(U\big)$.

We note that parallel-correctness is the combination of

- *parallel-soundness*: $[\mathcal{Q}, \boldsymbol{P}](I) \subseteq \mathcal{Q}(I)$, and
- *parallel-completeness*: $\mathcal{Q}(I) \subseteq [\mathcal{Q}, \boldsymbol{P}](I)$.

For monotone queries, like conjunctive queries, parallel-soundness is guaranteed, and therefore parallel-correctness and parallel-completeness coincide.

While Definition 5.2 is in terms of general queries, in the rest of this section, we only consider (extensions of) conjunctive queries.

[1] We mention that for HyperCube distributions, the view is reversed: facts are assigned to nodes. However, both views are essentially equivalent and we will freely adopt the view that fits best for the purpose at hand.

5.1 Conjunctive queries

We first focus on a characterization of parallel-correctness. It is easy to see that a CQ $\mathcal{Q}$ is parallel-correct under distribution policy $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$ if, for each valuation for $\mathcal{Q}$, the required facts meet at some node. That is, if the following condition holds:

For every valuation V for $\mathcal{Q}$ over U, there is a node $\kappa \in \mathcal{N}$ such that $V(\mathit{body}_{\mathcal{Q}}) \subseteq \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$. (PC_0)

However, Condition (PC_0) is not necessary as the following example shows.

Example 5.3. Let $\mathcal{Q}_3$ be the CQ

$$H(x,z) \leftarrow R(x,y), R(y,z), R(x,x),$$

and V the valuation $\{x \mapsto a, y \mapsto b, z \mapsto a\}$. Let further $\mathcal{N} = \{\kappa_1, \kappa_2\}$ and let $\boldsymbol{P}$ distribute every fact except $R(a,b)$ onto node κ_1 and every fact except $R(b,a)$ onto node κ_2. Since $R(a,b)$ and $R(b,a)$ do not meet under $\boldsymbol{P}$, valuation V witnesses the failure of Condition (PC_0) for $\boldsymbol{P}$ and $\mathcal{Q}$.

However, $\mathcal{Q}_3$ is parallel-correct under $\boldsymbol{P}$. Indeed, every valuation that derives a fact $\boldsymbol{f}$ with the help of the facts $R(a,b)$ and $R(b,a)$, also requires the fact $R(a,a)$ (or $R(b,b)$). But then, $R(a,a)$ (or $R(b,b)$) alone is sufficient to derive $\boldsymbol{f}$ by mapping all variables to a (or b). Therefore, if $\boldsymbol{f} \in \mathcal{Q}(I)$, for some instance I, then $\boldsymbol{f} \in [\mathcal{Q}, \boldsymbol{P}](I)$ and thus $\mathcal{Q}_3$ is parallel-correct under $\boldsymbol{P}$. □

It turns out that it suffices to consider only valuations that are minimal in the following sense:

Definition 5.4. A valuation V for $\mathcal{Q}$ is *minimal* for a CQ $\mathcal{Q}$, if there is *no valuation* V' for $\mathcal{Q}$ that derives the same head fact with a strict subset of body facts, that is, such that $V'(\mathit{body}_{\mathcal{Q}}) \subsetneq V(\mathit{body}_{\mathcal{Q}})$ and $V(\mathit{head}_{\mathcal{Q}}) = V'(\mathit{head}_{\mathcal{Q}})$.

Example 5.5. For a simple example of a minimal valuation and a non-minimal valuation, consider again the CQ $\mathcal{Q}_3$,

$$H(x,z) \leftarrow R(x,y), R(y,z), R(x,x).$$

Both valuations $V_1 = \{x \mapsto a, y \mapsto b, z \mapsto a\}$ and $V_2 = \{x \mapsto a, y \mapsto a, z \mapsto a\}$ for $\mathcal{Q}_3$ agree on the head variables of $\mathcal{Q}_3$, but they require different sets of facts. In particular, for V_1 to be satisfying on I, instance I must contain the facts $R(a,b), R(b,a)$, and $R(a,a)$, while V_2 only requires $R(a,a)$. Thus V_1 is not minimal for $\mathcal{Q}_3$. Further, since V_2 requires only one fact it is minimal for $\mathcal{Q}_3$. □

The next lemma shows that it suffices to restrict valuations to minimal valuations in Condition (PC_0) to get a sufficient *and* necessary condition for parallel-correctness.

Proposition 5.6. *Let $\mathcal{Q}$ be a CQ. Then $\mathcal{Q}$ is parallel-correct under distribution policy $\boldsymbol{P}$ if and only if the following holds:*

For every minimal valuation V for $\mathcal{Q}$ over U, there is a node $\kappa \in \mathcal{N}$ such that $V(\mathit{body}_{\mathcal{Q}}) \subseteq \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$. (PC_1)

We emphasize that the word *minimal* is the only difference between Conditions (PC_0) and (PC_1). We now turn to algorithmic questions, that is, we study the following two algorithmic problems, parameterized by classes $\mathcal{P}$ of distribution policies.

PCI($\mathbf{CQ}, \mathcal{P}$)

Input: $\mathcal{Q} \in \mathbf{CQ}$, $\boldsymbol{P} \in \mathcal{P}$, instance I
Question: Is $\mathcal{Q}$ parallel-correct on I under $\boldsymbol{P}$?

PC($\mathbf{CQ}, \mathcal{P}$)

Input: $\mathcal{Q} \in \mathbf{CQ}$, $\boldsymbol{P} \in \mathcal{P}$
Question: Is $\mathcal{Q}$ parallel-correct under $\boldsymbol{P}$?

The quantifier structure in Condition (PC_1) hints at a Π_2^p upper bound for the complexity of parallel-correctness.[2] The exact complexity can not be judged without having a bound on the number of nodes κ and the complexity of the test $V(\mathit{body}_{\mathcal{Q}}) \subseteq \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$. The largest classes of distribution policies for which we established the Π_2^p upper bound, are gathered in the set $\mathfrak{P}_{\mathrm{npoly}}$: it contains classes $\mathcal{P}$ of distribution policies, for which each policy comes with an algorithm $\mathcal{A}$ and a bound n on the representation size of nodes in the network, respectively, such that whether a node κ is responsible for a fact $\boldsymbol{f}$ is decided by $\mathcal{A}$ *non-deterministically* in time $\mathcal{O}(n^k)$, for some k that depends only on $\mathcal{P}$.

It turns out that the problem of testing parallel-correctness is also Π_2^p-*hard*, even for the simple class $\mathcal{P}_{\mathrm{fin}}$ of distribution policies, for which all pairs $(\kappa, \boldsymbol{f})$ of a node and a fact are explicitly enumerated. Thus, in a sense, Condition (PC_1) can essentially not be simplified.

Theorem 5.7. *Problems* PC($\boldsymbol{CQ}, \mathcal{P}$) *and* PCI($\boldsymbol{CQ}, \mathcal{P}$) *are Π_2^p-complete, for every policy class $\mathcal{P} \in \{\mathcal{P}_{\mathit{fin}}\} \cup \mathfrak{P}_{\mathit{npoly}}$.*

The upper bounds follow from the characterization in Proposition 5.6 and the fact that pairs $(\kappa, \boldsymbol{f})$ can be tested in NP.

We note that Proposition 5.6 continues to hold true in the presence of union and inequalities (under a suitable definition of minimal valuation for unions of CQs) leading to the same complexity bounds as stated in Theorem 5.7 [9].

5.2 Conjunctive queries with negation

In this section, we consider conjunctive queries with negation. Specifically, queries can be of the form

$$H(\mathbf{x}) \leftarrow R_1(\mathbf{y_1}), \ldots, R_m(\mathbf{y_m}), \neg S_1(\mathbf{z_1}), \ldots, \neg S_n(\mathbf{z_n}),$$

where, to ensure safety, we require that every variable in $\mathbf{x}$ occurs in some $\mathbf{y}_i$ or $\mathbf{z}_j$, and that every variable occurring in a negated atom has to occur in a positive atom as well. A valuation V now derives a fact $V\big(H(\mathbf{x})\big)$ on an instance I if every positive atom $V\big(R_i(\mathbf{y_i})\big)$ occurs in I while none of the negative atoms $V\big(S_j(\mathbf{z_j})\big)$ do. We refer to the class of conjunctive queries with negation as $\mathbf{CQ}^{\neg}$.

We note that, since queries in $\mathbf{CQ}^{\neg}$ need not be monotone, parallel-soundness is no longer guaranteed and thus parallel-correctness need not coincide with parallel-completeness.

We illustrate through an example that in the case of conjunctive queries *with negation*, the parallel-correctness problem becomes much more intricate, since it might involve counterexample databases of exponential size. We emphasize that this exponential explosion can only occur if, as in our framework, the arity of the relations in the database schema are not a-priori bounded by some constant.

[2] Indeed, testing minimality of V does not introduce another alternation of quantifiers, since it only requires an additional existential quantification of a valuation V' that serves as a witness, in case V is not minimal.

Example 5.8. Let $\mathcal{Q}_4$ be the following query:

$$\begin{aligned} H() \quad \leftarrow \quad & \mathtt{Val}(w_0, w_0), \mathtt{Val}(w_1, w_1), \neg\mathtt{Val}(w_0, w_1), \\ & \mathtt{Val}(x_1, x_1), \ldots, \mathtt{Val}(x_n, x_n), \neg\mathtt{Rel}(x_1, \ldots, x_n). \end{aligned}$$

Let $\boldsymbol{P}$ be the policy with universe $U = \{0, 1\}$ and two-node network $\{\kappa_1, \kappa_2\}$, which distributes all facts except $\mathtt{Rel}(0, \ldots, 0)$ to node κ_1 and only fact $\mathtt{Rel}(0, \ldots, 0)$ to node κ_2.

Query $\mathcal{Q}_4$ is not parallel-sound under policy $\boldsymbol{P}$, due to the counterexample $I \overset{\text{def}}{=} \{\mathtt{Val}(0, 0), \mathtt{Val}(1, 1)\} \cup \{\mathtt{Rel}(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in \{0, 1\}^n\}$. Indeed, $\mathcal{Q}_4(I) = \emptyset$ but the all-zero valuation witnesses $\mathcal{Q}_4(\textit{loc-inst}_{\boldsymbol{P},I}(\kappa_1)) \neq \emptyset$.

However, I has $2^n + 2$ facts and is a counterexample of minimal size as can easily be shown as follows. First, it is impossible that $\mathcal{Q}_4(I^*) \neq \emptyset$ and $\mathcal{Q}_4(\textit{loc-inst}_{\boldsymbol{P},I^*}(\kappa_1)) = \emptyset$, for any I^*, since $\mathtt{Rel}(0, \ldots, 0)$ is the only fact that can be missing at node κ_1, and $\mathcal{Q}_4$ is antimonotonic with respect to $\mathtt{Rel}$. On the other hand, if $\mathcal{Q}_4(\textit{loc-inst}_{\boldsymbol{P},I^*}(\kappa_1)) \neq \emptyset$, then the literals $\mathtt{Val}(w_0, w_0)$, $\mathtt{Val}(w_1, w_1)$, and $\neg\mathtt{Val}(w_0, w_1)$ ensure that there are at least two different data values (and thus 0 and 1) in I^*. But then $\mathcal{Q}_4(I^*) = \emptyset$ can only hold if all 2^n n-tuples over $\{0, 1\}$ are in I^*. □

Although this example requires an exponential size counterexample, in this particular case, the existence of the counterexample is easy to conclude. However, the following result shows that, in general, there is essentially no better algorithm than guessing an exponential size counterexample.

Theorem 5.9. *[9] For every class $\mathcal{P} \in \mathfrak{P}_{npoly}$ of distribution policies, the following problems are coNEXPTIME-complete.*

- Parallel-Sound($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)
- Parallel-Complete($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)
- Parallel-Correct($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)

The result and, in particular, the lower bound even holds if $\mathfrak{P}_{\text{npoly}}$ is replaced by the class $\mathfrak{P}_{\text{poly}}$, where the decision algorithm for pairs $(\kappa, \boldsymbol{f})$ is deterministic.

The proof of the lower bounds comes along an unexpected route and exhibits a reduction from query containment for $\mathbf{CQ}^{\neg}$ to parallel-correctness for $\mathbf{CQ}^{\neg}$. Query containment asks whether for two queries $\mathcal{Q}$ and $\mathcal{Q}'$, it holds that $\mathcal{Q}(I) \subseteq \mathcal{Q}'(I)$, for all instances I. It is shown in [9] that query containment for $\mathbf{CQ}^{\neg}$ is coNEXPTIME-complete, implying coNEXPTIME-hardness for parallel-correctness as well. The result regarding containment of $\mathbf{CQ}^{\neg}$ confirms the observation in [13] that the Π_2^p-completeness result for query containment for $\mathbf{CQ}^{\neg}$ mentioned in [20] only holds for fixed database schemas (or a fixed arity bound, for that matter).

6. PARALLEL-CORRECTNESS TRANSFER

Parallel-correctness is relative to a distribution policy. The idea of parallel-correctness *transfer* is to drop this dependence and to infer that a distribution policy is parallel-correct for the next query from the fact that it is parallel-correct for the current query.

Definition 6.1. For two queries $\mathcal{Q}$ and $\mathcal{Q}'$, *parallel-correctness transfers from $\mathcal{Q}$ to $\mathcal{Q}'$* if $\mathcal{Q}'$ is parallel-correct under every distribution policy for which $\mathcal{Q}$ is parallel-correct. In this case, we write $\mathcal{Q} \xrightarrow{\text{pc}} \mathcal{Q}'$.

Example 6.2. We consider a database of document IDs with a reference relation R among them: fact $R(22, 44)$ states that document 22 references document 44. Query $\mathcal{Q} : H(d_1, d_2) \leftarrow R(d_1, d_2), R(d_1, d_3), R(d_3, d_2)$ asks for all documents d_1, d_2 such that d_1 references d_2 directly as well as in two steps, that is, hopping over a document d_3.

One might expect that the syntactic subquery

$$\mathcal{Q}' : H(d_1, d_2) \leftarrow R(d_1, d_3), R(d_3, d_2),$$

which asks for a two-step reference only, is parallel-correct under every distribution policy that allows correct evaluation of query $\mathcal{Q}$. However, this is not the case because for derived facts (i, i), where document i references *itself* directly and in two steps (taking d_3 as i as well), query $\mathcal{Q}'$ requires both facts $R(i, j)$ and $R(j, i)$ to be present at some node for some j, while $\mathcal{Q}$ requires only $R(i, i)$ to be present. See Figure 2 for an example instance and distribution.

Parallel-correctness does transfer from a similar query, $\mathcal{Q}'' : H(d_1, d_2, d_3) \leftarrow R(d_1, d_2), R(d_1, d_3), R(d_3, d_1)$, where d_3 is part of the head, to $\mathcal{Q}'$ because all valuations for $\mathcal{Q}''$ are minimal and every valuation for $\mathcal{Q}'$ requires a subset of the facts required by the same valuation for $\mathcal{Q}''$. □

Like for parallel-correctness, the characterization of parallel-correctness transfer is in terms of minimal valuations. It turns out that the following notion yields the desired semantical characterization.

Definition 6.3. For two CQs $\mathcal{Q}$ and $\mathcal{Q}'$, we say that $\mathcal{Q}$ *covers* $\mathcal{Q}'$ if the following holds:

> for every minimal valuation V' for $\mathcal{Q}'$, there is a minimal valuation V for $\mathcal{Q}$, such that $V'(\textit{body}_{\mathcal{Q}'}) \subseteq V(\textit{body}_{\mathcal{Q}})$.

Proposition 6.4. *For two CQs $\mathcal{Q}$ and $\mathcal{Q}'$, parallel-correctness transfers from $\mathcal{Q}$ to $\mathcal{Q}'$ if and only if $\mathcal{Q}$ covers $\mathcal{Q}'$.*

One might be tempted to assume that parallel-correctness transfer is somehow directly linked with query containment. However, as the following example shows, this is not the case.

Example 6.5. We consider the following four queries:

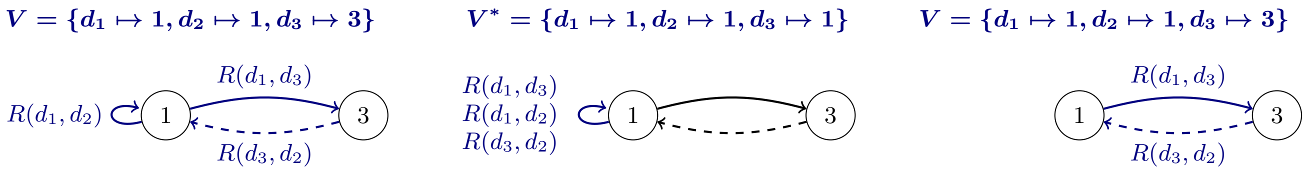
$$\begin{aligned} &\mathcal{Q}_1 : H() \leftarrow S(x), R(x, x), T(x). \\ &\mathcal{Q}_2 : H() \leftarrow R(x, x), T(x). \\ &\mathcal{Q}_3 : H() \leftarrow S(x), R(x, y), T(y). \\ &\mathcal{Q}_4 : H() \leftarrow R(x, y), T(y). \end{aligned}$$

Figure 3 (a) shows how these queries relate with respect to parallel-correctness transfer. As an example, $\mathcal{Q}_3 \xrightarrow{\text{pc}} \mathcal{Q}_1$. As Figure 3 (b) illustrates, these relationships are completely orthogonal to query containment. Indeed, there are examples where parallel-correctness transfer and query containment coincide ($\mathcal{Q}_3$ vs. $\mathcal{Q}_4$), where they hold in opposite directions ($\mathcal{Q}_4$ vs. $\mathcal{Q}_2$) and where one but not the other holds ($\mathcal{Q}_3$ vs. $\mathcal{Q}_2$ and $\mathcal{Q}_1$ vs. $\mathcal{Q}_4$, respectively). □

Proposition 6.4, allows us to pinpoint the complexity of parallel-correctness transferability. For a formal statement we define the following algorithmic problem:

pc-trans ($\mathbf{CQ}$)

Input:	Queries $\mathcal{Q}$ and $\mathcal{Q}'$ from $\mathbf{CQ}$
Question:	Does parallel-correctness transfer from $\mathcal{Q}$ to $\mathcal{Q}'$?


$V = \{d_1 \mapsto 1, d_2 \mapsto 1, d_3 \mapsto 3\}$
$R(d_1, d_3)$
$R(d_1, d_2)$
1
3
$R(d_3, d_2)$
$V^* = \{d_1 \mapsto 1, d_2 \mapsto 1, d_3 \mapsto 1\}$
$R(d_1, d_3)$
$R(d_1, d_2)$
$R(d_3, d_2)$
1
3
$V = \{d_1 \mapsto 1, d_2 \mapsto 1, d_3 \mapsto 3\}$
$R(d_1, d_3)$
1
3
$R(d_3, d_2)$

(a) Valuation V satisfies $\mathcal{Q}$ globally on I but not locally. It is not minimal for $\mathcal{Q}$.

(b) Valuation V^* satisfies $\mathcal{Q}$ globally on I and locally on κ_1. It is minimal for $\mathcal{Q}$ and derives the same fact as V.

(c) Valuation V satisfies $\mathcal{Q}'$ globally on I' but not locally. It is minimal for $\mathcal{Q}'$. It does not satisfy $\mathcal{Q}$.

Figure 2: Global instances $I = \{R(1,1), R(1,3), R(3,1)\}$ (left and middle) and $I' = \{R(1,3), R(3,1)\}$ (right) are represented by graphs. Solid edges (facts) are located at node κ_1, dashed edges at node κ_2; colored edges are required by the valuation under concern. Instance I has globally and locally satisfying valuations for query $\mathcal{Q}$, subinstance $I' \subseteq I$ has a globally satisfying valuation but no locally satisfying one—under the same distribution policy. There is thus a policy under which $\mathcal{Q}$ is parallel-correct but $\mathcal{Q}'$ is not, and therefore parallel-correctness does not transfer from $\mathcal{Q}$ to $\mathcal{Q}'$.

$$\begin{tikzcd}
 & R(x,y), T(y) \arrow[dl, "\text{pc}"'] & \\
R(x,x), T(x) & & S(x), R(x,y), T(y) \arrow[ul, "\text{pc}"'] \arrow[ll, "\text{pc}"'] \arrow[dl, "\text{pc}"] \\
 & S(x), R(x,x), T(x) \arrow[ul, "\text{pc}"] &
\end{tikzcd}$$

(a) parallel-correctness transfer

$$\begin{tikzcd}
 & R(x,y), T(y) & \\
R(x,x), T(x) \arrow[ur, dashed, "\subseteq"] & & S(x), R(x,y), T(y) \arrow[ul, dashed, "\supseteq"'] \\
 & S(x), R(x,x), T(x) \arrow[ul, dashed, "\supseteq"] \arrow[ur, dashed, "\subseteq"'] \arrow[uu, dashed, "\subseteq"] &
\end{tikzcd}$$

(b) containment

Figure 3: Relationship between the queries of Example 6.5 with respect to (a) parallel-correctness transfer (pc) and (b) query containment ($\subseteq$).

When the defining condition of "covers" is spelled out by rewriting "minimal valuations" one gets a characterization with a Π_3-structure. Again, it can be shown that this is essentially optimal.

Theorem 6.6. *Problem* PC-TRANS($\boldsymbol{CQ}$) *is* Π_3^p*-complete.*

The upper bounds follow directly from the characterization in Proposition 6.4, implying that these characterizations are essentially optimal. We note that the same complexity bounds continue to hold in the presence of inequalities and for unions of conjunctive queries [2].

The complexity of transferability is considerably better for a restricted class of conjunctive queries that we call strongly minimal.

Definition 6.7. A CQ query is *strongly minimal* if all its valuations are minimal.

Strong minimality generalizes two particularly simple classes of queries:

Lemma 6.8. *A conjunctive query $\mathcal{Q}$ is strongly minimal*

- *if it is a full query;*
- *if it contains no self-joins (every relation name occurs at most once).*

Theorem 6.9. PC-TRANS($\boldsymbol{CQ}$) *restricted to inputs* $(\mathcal{Q}, \mathcal{Q}')$*, where $\mathcal{Q}$ is strongly minimal, is* NP*-complete.*

7. DISCUSSION

Parallel-correctness serves as a framework for studying correctness and implications of data partitioning in the context of one-round query evaluation algorithms. A main insight of the work up to now is that testing for parallel-correctness as well as the related problem of parallel-correctness transfer boils down to reasoning about minimal valuations (of polynomial size) in the context of conjunctive queries (even in the presence of union and inequalities) but seems to require to reason about databases of exponential size when negation is allowed.

There are many questions left unexplored and we therefore highlight possible directions for further research.

From a foundational perspective, it would be interesting to explore the decidability boundary for parallel-correctness and transfer when considering more expressive query languages or even other data models. Obviously, the problems become undecidable when considering first-order logic, but one could consider monotone languages or, for instance, guarded fragment queries. At the same time, it would be interesting to find settings that render the problems tractable, for instance, by restricting the class of queries or by limiting to certain classes of distribution policies.

Parallel-correctness transfer is a rather strong notion as it requires that a query $\mathcal{Q}'$ is parallel-correct for *every* distribution policy for which another query $\mathcal{Q}$ is parallel-correct. From a practical perspective, however, it could be interesting to determine, given $\mathcal{Q}$ and $\mathcal{Q}'$, whether there is at least one distribution policy under which both queries are correct. Other questions concern the least costly way to migrate from one distribution to another. As an example, assume a distribution $\boldsymbol{P}$ on which $\mathcal{Q}$ is parallel-correct but $\mathcal{Q}'$ is not. Find a distribution $\boldsymbol{P}'$ under which $\mathcal{Q}'$ is parallel-correct and that minimizes the cost to migrate from $\boldsymbol{P}$ to $\boldsymbol{P}'$. Similar questions can be considered for a workload of queries.

Even though the naive one-round evaluation model considered in this paper suffices for HyperCube, it is rather restrictive. Other possibilities are to consider more complex aggregator functions than union and to allow for different queries than the original one to be executed at computing nodes. Furthermore, it could be interesting to generalize the framework beyond one-round algorithms, that is, towards evaluation algorithms that comprise of several rounds.

8. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1282–1298, 2011.

[2] T. J. Ameloot, G. Geck, B. Ketsman, F. Neven, and T. Schwentick. Parallel-correctness and transferability for conjunctive queries. Journal verion, 2015.

[3] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.

[4] P. Beame, P. Koutris, and D. Suciu. Skew in parallel query processing. In *PODS*, pages 212–223, 2014.

[5] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *SIGMOD*, pages 975–986, 2010.

[6] S. Chu, M. Balazinska, and D. Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *SIGMOD*, pages 63–78, 2015.

[7] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] S. Ganguly, A. Silberschatz, and S. Tsur. Parallel bottom-up processing of datalog queries. *J. Log. Program.*, 14(1&2):101–126, 1992.

[9] G. Geck, B. Ketsman, F. Neven, and T. Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. In *ICDT*, pages 9:1–9:17, 2016.

[10] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, and D. Suciu. Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 881–884, 2014.

[11] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011.

[12] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1-2):330–339, 2010.

[13] M. Mugnier, G. Simonet, and M. Thomazo. On the complexity of entailment in existential conjunctive first-order logic with atomic negation. *Inf. Comput.*, 215:8–31, 2012.

[14] R. Nehme and N. Bruno. Automated partitioning design in parallel database systems. In *SIGMOD*, pages 1137–1148, 2011.

[15] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012*, pages 37–48, 2012.

[16] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.

[17] J. Rao, C. Zhang, N. Megiddo, and G. Lohman. Automating physical database design in a parallel database. In *SIGMOD*, pages 558–569, 2002.

[18] 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.

[19] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *PVLDB*, pages 1626–1629, 2009.

[20] J. D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.

[21] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

[22] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: SQL and rich analytics at scale. In *SIGMOD*, pages 13–24, 2013.