Parallel-Correctness and Parallel-Boundedness for Datalog Programs

Frank Neven
Hasselt University and transnational University of Limburg, The Netherlands

Thomas Schwentick
Dortmund University, Germany

Christopher Spinrath
Dortmund University, Germany

Brecht Vandevoort[1]
Hasselt University and transnational University of Limburg, The Netherlands

Abstract

Recently, Ketsman et al. started the investigation of the parallel evaluation of recursive queries in the Massively Parallel Communication (MPC) model. Among other things, it was shown that parallel-correctness and parallel-boundedness for general Datalog programs is undecidable, by a reduction from the undecidable containment problem for Datalog. Furthermore, economic policies were introduced as a means to specify data distribution in a recursive setting. In this paper, we extend the latter framework to account for more general distributed evaluation strategies in terms of communication policies. We then show that the undecidability of parallel-correctness runs deeper: it already holds for fragments of Datalog, e.g., monadic and frontier-guarded Datalog, with a decidable containment problem, under relatively simple evaluation strategies. These simple evaluation strategies are defined w.r.t. data-moving distribution constraints. We then investigate restrictions of economic policies that yield decidability. In particular, we show that parallel-correctness is 2EXPTIME-complete for monadic and frontier-guarded Datalog under hash-based economic policies. Next, we consider restrictions of data-moving constraints and show that parallel-correctness and parallel-boundedness are 2EXPTIME-complete for frontier-guarded Datalog. Interestingly, distributed evaluation no longer preserves the usual containment relationships between fragments of Datalog. Indeed, not every monadic Datalog program is equivalent to a frontier-guarded one in the distributed setting. We illustrate the latter by considering two alternative settings where in one of these parallel-correctness is decidable for frontier-guarded Datalog but undecidable for monadic Datalog.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Datalog, distributed databases, distributed evaluation, decision problems, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.14

1 Introduction

Modern data management systems like Spark [6, 30] and Hadoop [5] embrace massive parallelism to speed up query processing over large volumes of data. The latter inspired an extensive line of research on the foundations of parallel query evaluation which mostly focused on join queries (e.g., [2, 3, 10, 23, 24]). These investigations can be cast within the massively parallel communication model [10] (MPC) where query evaluation proceeds in multiple rounds and where each round consists of a computation and a communication step. In the computation step, every server operates on its local data in isolation, whereas in the communication step data is exchanged between the servers. The Hypercube algorithm [3, 17]

[1] PhD Fellow of the Research Foundation – Flanders (FWO)

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti; Article No. 14; pp. 14:1–14:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which can compute any multiway join in one round was shown to play a fundamental role in several of the above mentioned papers. Ameloot et al. [4] introduced a framework to reason about such Hypercube-style single-round algorithms. The authors considered the problem of *parallel-correctness* asking whether one can always be sure that the corresponding single-round algorithm computes the query result correctly, no matter the actual data, starting from a particular distribution policy. Parallel-correctness and related problems were studied for conjunctive queries (with and without negation) and under set as well as bag semantics [4, 18, 22].

The relevance of Datalog as a query language for expressing recursive queries has only gained in importance in the last decade (e.g,[1, 7, 20, 28, 29]). Ketsman, Albarghouthi, and Koutris [22] started the investigation of the parallel evaluation of Datalog queries in the multi-round MPC model. To this end, *economic policies* were introduced as a means to specify data reshuffling in a recursive setting where intermediate derived facts can be communicated between servers. So, in addition to specifying the initial distribution of extensional database facts, economic policies also determine which servers can produce and consume intensional database facts during evaluation of a Datalog program. Among other things, the authors showed that parallel-correctness and parallel-boundedness (the problem that asks whether a Datalog program can be evaluated in a bounded number of evaluation rounds) for general Datalog programs is undecidable. For this, a reduction from the undecidable containment problem for Datalog was used.

In this paper, we revisit the parallel-correctness and parallel-boundedness problem for Datalog programs. We start by establishing that the undecidability of parallel-correctness runs deeper than the containment problem. We show that parallel-correctness is already undecidable under relatively simple distributed evaluation strategies for fragments of Datalog with a decidable containment problem: monadic and frontier-guarded Datalog. Here, monadic Datalog is the variant of Datalog where intensional predicates have arity at most one, whereas in frontier-guarded Datalog every rule should contain an extensional predicate mentioning all the variables occurring in the head of the rule (c.f., e.g., [8, 9, 13]).

With this undecidability result as a guideline, we focus in this paper on settings for which the above mentioned decision problems become decidable. We start by presenting a generic framework for the distributed evaluation of Datalog within the multi-round MPC model. This framework allows for more general evaluation strategies than the economic policies of [22]. In brief, for a given Datalog program P, its parallel evaluation is determined by a *policy pair* (δ, ρ) consisting of an initial distribution policy δ and a communication policy ρ. A database instance is first distributed according to δ assigning extensional database facts to the computation servers. Then the computation proceeds in rounds consisting of two steps: in the first step each server recursively evaluates the same program P over its local database, and in the second step all derived intensional database facts are communicated according to the communication policy ρ. In their most general form, communication policies are triples of the form $(\mathbf{f}, k, \ell)$ indicating that when a fact $\mathbf{f}$ is derived on server k it is sent to server ℓ.

Since in practice, data distribution is typically done using hash-partitioning (e.g.,[10, 25, 31]), we consider hash-based policies as initial distribution policies δ. For communication policies ρ we consider two approaches: economic hash-based policies and communication policies defined through data-moving constraints. The former is an instantiation of the framework of [22] with hash-partitioning while the latter is a declarative approach based on distribution constraints as introduced in [19], but used here in a much more restricted form to specify relocation of facts in between rounds of computation.

The contributions of this paper can be summarized as follows:

1. We introduce in Section 3 a general framework for the specification of distributed evaluation strategies for Datalog programs.
2. In Section 4 we first show that parallel-correctness for monadic Datalog and frontier-guarded Datalog is undecidable for communication policies defined by simple data-moving distribution constraints. We then study decidable cases for parallel-correctness and show that parallel-correctness is 2ExpTime-complete for monadic Datalog and frontier-guarded Datalog under hash-based communication policies, and that parallel-correctness is 2ExpTime-complete for frontier-guarded Datalog under communication policies defined by a restriction of data-moving distribution constraints.
3. The upper bounds in Section 4 rely on a fine-grained analysis of the complexity of the containment problem for frontier-guarded Datalog in terms of the number of variables, the number of rules and the size of the rules (Proposition 11). This analysis is based on a proof by Bourhis, Krötzsch, and Rudolph [13] and is carried out in Section 5.
4. We show in Section 6 that parallel-boundedness is 2ExpTime-complete for frontier-guarded Datalog and communication policies defined by the same kind of data-moving distribution constraints as for parallel-correctness.
5. In Section 7 we show that the results on parallel-correctness do not change if we restrict each server to derive only facts that it can communicate (similar to [21]), and that there is a family of communication policies for which parallel-correctness of frontier-guarded Datalog is decidable whereas parallel-correctness of monadic Datalog is undecidable.

The latter results in Section 7 show that the usual ability to transform programs from monadic Datalog into frontier-guarded Datalog can break drastically in a distributed setting, which we find quite surprising.

We are aware that 2ExpTime upper bounds are far from practical tractability. Still, this is the best we can hope for as containment of a Datalog fragment easily reduces to parallel-correctness for that fragment and containment for both monadic as well as frontier-guarded Datalog is complete for 2ExpTime [9, 11, 13].

Related work is addressed throughout the paper at the places where it is most relevant. Due to space limitations, many proofs are delegated to the full version of the paper which we plan to publish at arXiv by the time of the conference. We are grateful to Gaetano Geck for inspiring discussions on the subject of this paper.

2 Preliminaries

Datalog. We assume a fixed database *schema* $\mathcal{S}$ throughout the paper. An *instance* G is a finite set of facts over $\mathcal{S}$. We use standard notions, including $ar(R)$ for the arity of a relation name R, $R(\bar{x})$ for an atom (with variables in $\bar{x}$), $R(\bar{t})$ for a *fact* (with domain elements in $\bar{t}$) $\mathsf{val}(G)$ for the set of domain elements in G, and $I_{|S}$ for the set of facts in I over $S \subseteq \mathcal{S}$.

A *Datalog rule* τ is usually written[2] as $R(\bar{x}) \leftarrow S_1(\bar{y}_1), \dots, S_n(\bar{y}_n)$. By $head_\tau$ and $body_\tau$ we refer to its head and body, respectively. By $vars(\tau)$ and $vars(A)$, we denote the set of all variables in τ and in an atom A, respectively. A *Datalog program* P is a finite set of Datalog rules. By EDB(P), IDB(P) and $out(P)$, we denote the extensional, intensional and output relation names of P, respectively. P is *monadic* if the arity of every relation in IDB(P) is at most one. It is *frontier-guarded*, if every rule τ is frontier-guarded, that is, if there is an EDB-atom in $body_\tau$ that contains all the variables from $head_\tau$. We denote the class of monadic and frontier-guarded Datalog programs with mon-Datalog and fg-Datalog, respectively. The semantics of Datalog programs is defined in the usual way.

[2] We only consider Datalog programs without constant symbols.

▶ **Example 1.** Let $E_r(x, y)$ denote red edges and let $E_s(x, y)$ denote sea blue edges. The following monadic program P_{mon} computes all nodes reachable from a starting node x (indicated by $\text{Start}(x)$) by a path containing only red and a path containing only sea blue edges:

$$\begin{array}{lll} R(x) \leftarrow \text{Start}(x) & S(x) \leftarrow \text{Start}(x) & \text{Out}(x) \leftarrow R(x), S(x) \\ R(x) \leftarrow R(y), E_r(y, x) & S(x) \leftarrow S(y), E_s(y, x) & \end{array}$$

Let $I_{\text{mon}} = \{\text{Start}(1), E_r(1, 3), E_r(1, 4), E_s(1, 2), E_s(2, 3)\}$, then $P(I_{\text{mon}})_{|out(P)} = \{\text{Out}(1), \text{Out}(3)\}$.

Networks and Distributed Databases. We model a *network* of database servers as a finite set $\mathcal{N}$ of *servers*. A distributed instance $D = (G, \mathbf{I})$ consists of a global instance G over $\mathcal{S}$ and a family $\mathbf{I} = (I_k)_{k\in\mathcal{N}}$ of local instances over $\mathcal{S}$, one for each server of $\mathcal{N}$ such that $\bigcup_{k\in\mathcal{N}} I_k = G$. We also denote G by $\text{global}(D)$. We write $\mathbf{f}@k$ for a fact $\mathbf{f}$ in I_k. For two distributed instances $D = (G, \mathbf{I})$ and $D' = (G', \mathbf{I}')$ we write $D \subseteq D'$, if $G \subseteq G'$ and $I_k \subseteq I'_k$ for every $k \in \mathcal{N}$. We emphasise that we do not allow facts to be “skipped”. That is, every global fact should occur somewhere as a local fact.

3 Framework

We adapt the MPC model [10] and define a generic framework that allows us to reason about Datalog programs in distributed settings. The framework is based on *policy pairs* (δ, ρ), consisting of a (partial) specification of the initial data distribution δ and a communication policy ρ over the same network. Policy pairs allow to specify initial distributions and communication of facts independently and in different ways (thus extending the framework of [21]). We then define the hash-based initial distributions that we use and provide several concrete ways to define communication policies: hash-based and constraint-based. We also define the parallel-correctness problem.

As mentioned before, we reason about Datalog programs relative to policy pairs (δ, ρ), where δ is a distribution policy and ρ a communication policy. These terms are defined next. We require that the distribution of EDB-facts is only described by δ and that communication policies are restricted to IDB-facts.

In general, a *distribution policy* δ over a network $\mathcal{N}$ is a function mapping global instances G to distributed instances $D = (G, \mathbf{I})$ over $\mathcal{N}$. A distributed instance $D = (G, \mathbf{I})$ over $\mathcal{N}$ is *valid with respect to* a distribution policy δ over $\mathcal{N}$ if $\delta(G) \subseteq D$. Thus, we allow that D distributes facts to more servers than required by δ. This is because we do not understand δ as a specification of a communication round but rather as a constraint that is met at the beginning of the evaluation of a Datalog program. However, the definition ensures that the global instance of $\delta(G)$ is just G. We define the specific (hash-based) distribution policies used in this paper below in Section 3.2.

For a network $\mathcal{N}$ and a (global) database G, a *communicated fact* $(\mathbf{f}, k, \ell)$ consists of a fact $\mathbf{f} \in G$ and two servers k and ℓ of $\mathcal{N}$ and has the intended meaning that $\mathbf{f}$ is communicated from k to ℓ. A *communication policy* ρ for a Datalog program P over $\mathcal{N}$ is a monotone mapping that assigns to every distributed instance $D = (G, \mathbf{I})$ over $\mathcal{S} \cup \text{IDB}(P)$ and $\mathcal{N}$ a set of communicated facts. Here, monotonicity means that $\rho(D) \subseteq \rho(D')$ whenever $D \subseteq D'$. The communication policies studied in this paper are defined in Section 3.3.

3.1 Distributed Evaluation of Datalog Programs

A communication policy ρ induces a distributed multi-round evaluation strategy for P over $D = (G, \mathbf{I})$ as follows.[3] Each round consists of a computation and a communication phase. By $D^i = (G^i, \mathbf{I}^i)$ we denote the distributed instance after the i-th communication phase. The initial instance D^0 is just D. Then, for $i \geq 1$, the following phases are performed:

- *Computation:* Every server computes the local fixpoint of P over its local instance. That is, $D' = (G', \mathbf{J})$ where $G' = \bigcup_{k \in \mathcal{N}} J_k$ and, for each $k \in \mathcal{N}$, $J_k = P(I_k^{i-1})$.
- *Communication:* For each $(\mathbf{f}, k, \ell) \in \rho(D')$, $\mathbf{f}$ is copied from k to ℓ. That is, for each $\ell \in \mathcal{N}$, $I_\ell^i = J_\ell \cup \{\mathbf{f} \mid (\mathbf{f}, k, \ell) \in \rho(D'), \mathbf{f} \in J_k\}$. Then, $D^i = (G^i, \mathbf{I}^i)$ where $G^i = \bigcup_{k \in \mathcal{N}} I_k^i$.

The evaluation terminates when a global fixpoint is reached. We note that this fixpoint always exists. By $[P, \rho](D)$, we denote the union of all facts found at any server in this fixpoint. By $[P, \rho](D)_{|out(P)}$ we denote the output of the query computed by P on D according to ρ.

We note that our setting differs slightly from [21]. We provide more details in Section 7 and show that it does not have any influence on our results for parallel-correctness.

▶ **Example 2.** Consider P_{mon} and I_{mon} from Example 1. Let $D_{\text{mon}} = (I_{\text{mon}}, \mathbf{I})$ be defined over the network $\mathcal{N} = [1, 4]$ with four servers and let $I_1 = \{\text{Start}(1), E_r(1, 3), E_r(1, 4)\}$, $I_2 = \{\text{Start}(1), E_s(1, 2), E_s(2, 3)\}$ and $I_3 = I_4 = \emptyset$. That is, Start-facts are duplicated over server 1 and 2, while red edges are sent to server 1 and sea blue edges to server 2. Let ρ be the communication policy that maps a distributed database $D' = (G', \mathbf{I}')$ to the set of triples $\{(R(i), 1, f(i)) \mid R(i) \in I'_1\} \cup \{(S(i), 2, f(i)) \mid S(i) \in I'_2\}$ with f a function mapping each value i to $((i - 1) \bmod 4) + 1$. So, when server 1 derives a fact $R(c)$, it is sent to server $f(c)$. A fact $S(c)$ derived on server 2 is also sent to server $f(c)$. The distributed computation then proceeds as follows. Only newly added IDB facts are shown in each round. Underlined facts are received through communication.

$$\begin{array}{llll}
I_1^1 = \{R(1), R(3), R(4), \underline{S(1)}\}, & I_2^1 = \{S(1), S(2), S(3)\}, & I_3^1 = \{\underline{R(3)}, \underline{S(3)}\}, & I_4^1 = \{\underline{R(4)}\} \\
I_1^2 = \{\text{Out}(1)\}, & I_2^2 = \emptyset, & I_3^2 = \{\text{Out}(3)\}, & I_4^2 = \emptyset \\
I_1^3 = \emptyset, & I_2^3 = \emptyset, & I_3^3 = \emptyset, & I_4^3 = \emptyset
\end{array}$$

A global fixpoint is reached in the third iteration.

Now we can formally define parallel-correctness for Datalog programs and policy pairs.

▶ **Definition 3** (Parallel-Correctness). *Let P be a Datalog program and (δ, ρ) a policy pair, consisting of a distribution policy δ and a communication policy ρ for P over the same network. We call P* parallel-correct *w.r.t. (δ, ρ), if $[P, \rho](D)_{|out(P)} = P(global(D))_{|out(P)}$, for every distributed instance D that is valid with respect to δ. A program P is* parallel-correct w.r.t. a family $\mathcal{F}$ of policy pairs *if it is parallel-correct w.r.t. every policy pair in $\mathcal{F}$.*

For classes $\mathcal{P}$ of Datalog programs, and $\mathcal{C}$ of families of policy pairs, Para-Correct$(\mathcal{P}, \mathcal{C})$ denotes the decision problem that asks, for a program $P \in \mathcal{P}$ and a family $\mathcal{F} \in \mathcal{C}$, whether P is parallel-correct with respect to $\mathcal{F}$.

Since all our evaluation strategies are sound (i.e., $[P, \rho](D)_{|out(P)} \subseteq P(\text{global}(D))_{|out(P)}$), and communication strategies are monotone, parallel-correctness can be decided by testing parallel-completeness (i.e., $P(\text{global}(D))_{|out(P)} \subseteq [P, \rho](D)_{|out(P)}$).

[3] We recall that we do not view the distribution policy δ as a specification of a communication round.

3.2 Distribution Policies

As mentioned before, we only consider hash-based distribution policies for the specification of initial distrbutions.

Let $H = (h_1, \ldots, h_p)$ be a tuple of hash functions over some network $\mathcal{N}$, where, for each i, $h_i : U^{\alpha_i} \rightarrow 2^{\mathcal{N}} - \{\emptyset\}$, for some *arity* α_i and with U the set of domain elements. A *hash policy scheme* Z is a set of triples $(R, i, \bar{b})$ such that R is a relation symbol, i is a number and $\bar{b} \in [1, ar(R)]^n$ for some $n \geq 0$. We say that Z is *consistent* if for all triples $(R, i, \bar{b})$ and $(S, i, \bar{c})$ it holds $|\bar{b}| = |\bar{c}|$. We only consider consistent hash policy schemes. We say that Z is *compatible with* H, if $|\bar{b}| = \alpha_i$, for every triple $(R, i, \bar{b})$.

If Z is consistent and compatible with H, the two induce the distribution policy $\delta_{Z,H}$ that maps every fact $R(\bar{u})$ to the set of servers k, for which there is a triple $(R, i, \bar{b}) \in Z$ such that $k \in h_i(\bar{v})$, where $\bar{v} = \pi_{\bar{b}}(\bar{u})$ is the projection of $\bar{u}$ to $\bar{b}$. Formally, $\pi_{\bar{b}}(\bar{a})$ is $(a_{b_1}, \ldots, a_{b_{\alpha_i}})$. By $\mathcal{F}(Z)$ we denote the family of such distribution policies $\delta_{Z,H}$.

We note that $\delta_{Z,H}$ is *fact-based*, in the sense that it maps each fact over $\mathcal{S}$ to a set of servers from $\mathcal{N}$, independent of the other facts in G. Like [21] we only consider fact-based distribution policies for the specification of initial distributions.

▶ **Example 4.** Recall I_{mon} as defined in Example 1. Consider the hash policy scheme $Z = \{(\text{Start}, 1, ()), (\text{Start}, 2, ()), (E_r, 1, ()), (E_s, 2, ())\}$. Then Z is consistent and is compatible with any tuple $H = (h_1, h_2)$ of hash functions where both h_1 and h_2 are nullary. Let $\mathcal{N} = [1, 4]$. Furthermore, let h_1 map the empty tuple to 1 and let h_2 map the empty tuple to 2. Then $\delta_{Z,H}(I_{\text{mon}}) = D_{\text{mon}}$ where D_{mon} is as defined in Example 2. We refer to Example 5 and 6 below for more involved hash policy schemes.

3.3 Communication Policies

In this paper, we study two ways to specify communication policies. We first consider policies that are based on hash-based distribution policies and afterwards policies that are specified with the help of distribution constraints.

In [21] the communication policies that controlled the evaluation of Datalog programs, were called *economic policies*. An economic policy ρ is induced by a pair (π, γ) of fact-based distribution policies. Here, π is referred to as the *production policy* determining which servers are allowed to derive IDB-facts whereas γ is the *consumption policy* determining which servers can consume which IDB- and EDB-facts. Such a pair defines a communication policy in which each fact $\mathbf{f}$ induces the set $\{\mathbf{f}\} \times \pi(\mathbf{f}) \times \gamma(\mathbf{f})$ of communicated facts.

Hash-Based Communication. If Z_1, Z_2 are hash policy schemes such that $Z_1 \cup Z_2$ is consistent and compatible with a tuple H of hash functions, they induce a communication policy $\rho_{Z_1,Z_2,H}$ as $\rho_{\pi,\gamma}$ where $\pi = \delta_{Z_1,H}$ and $\gamma = \delta_{Z_2,H}$. So, hash-based communication policies are an instantiation of economic policies. If Z, Z_1, Z_2 are hash policy schemes such that $Z \cup Z_1 \cup Z_2$ is consistent[4], $\mathcal{F}(Z, Z_1, Z_2)$ denotes the set of policy pairs $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$, where H is any tuple of hash functions with which Z, Z_1, Z_2 are compatible. We denote by Hash-Hash the class of families that are defined in this way by triples Z, Z_1, Z_2 of hash policy schemes. We note that economic policies and hash-based communication policies are *fact-based* in the sense that they map $\mathcal{S}$-facts $\mathbf{f}$ to sets of communicated facts $(\mathbf{f}, k, \ell)$, independent of other facts.

[4] Further on, we will simply say that Z, Z_1, Z_2 are consistent.

▶ **Example 5.** Consider the hash policy scheme Z as described in Example 4. Let $Z_1 = \{(R, 1, ()), (S, 2, ())\}$ and $Z_2 = \{(R, 3, (1)), (S, 3, (1))\}$. Then, Z, Z_1, Z_2 are consistent. Consider now the tuple $H = (h_1, h_2, h_3)$ of hash functions, where h_1 and h_2 are as in Example 4, and h_3 is a unary hash function mapping each value i onto $((i - 1) \bmod 4) + 1$. Clearly, Z, Z_1 and Z_2 are compatible with H, and $\rho_{Z_1,Z_2,H} = \rho$, where ρ is as defined in Example 2. For $\delta_{Z,H}$ as defined in Example 4, P_{mon} is parallel-correct w.r.t. $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$. Furthermore, this holds for every tuple H of hash functions with which Z, Z_1, Z_2 are compatible. Or equivalently, P_{mon} is parallel-correct w.r.t. $\mathcal{F}(Z, Z_1, Z_2)$.

Constraint-Based Communication. We next introduce a constraint-based formalism to define communication policies. We make use of the formalism of distribution constraints as introduced in [19]. It is important to note that the distribution constraints in [19] are far more general and are targeted at specifying classes of distributions (including co-partitionings). They are in particular based on distributed tuple- and equality-generating dependencies. Here, we use the formalism of distribution constraints to define the communication of facts between servers and only consider so-called data-moving distribution constraints. In the terminology of [19] they are data-collecting and do not refer to the global database.

A *distributed atom* $A@\kappa$ consists of an atom A and a server variable κ. A *distribution constraint* is a rule of the form $\sigma = \mathcal{A} \to K$ for a set of distributed atoms $\mathcal{A}$ forming its *body* and a distributed atom K forming its *head*. We denote $\mathit{head}_\sigma = K$ and $\mathit{body}_\sigma = \mathcal{A}$, respectively. We further assume in this paper that body_σ contains a distributed atom of the form $B@\kappa$ when $\mathit{head}_\sigma = A@\kappa$, that is, using the same server variable κ as in the head. We denote by $\mathit{vars}(\sigma)$ and $\mathit{nvars}(\sigma)$ the set of data and server variables occurring in σ.

A distribution constraint is *data-moving* when the atom occurring in the distributed atom in the head also occurs in a distributed atom in the body.[5] That is, when its head equals $A@\kappa$ then its body contains a distributed atom of the form $A@\lambda$ (possibly $\kappa = \lambda$). This will then imply that the atom A will be sent from server λ to server κ.

A *valuation* for σ over a distributed instance D with network $\mathcal{N}$, is a mapping V which maps server variables to servers, and the other variables to domain values. For a distributed atom $A' = A@\kappa$, we write $V(A') \in D$ if $V(A) \in I_{V(\kappa)}$. Each set Σ of data-moving distribution constraints over $\mathcal{S}$ induces for every network $\mathcal{N}$ a communication policy $\rho_{\Sigma,\mathcal{N}}$ as follows: for each distributed instance D over $\mathcal{N}$, $(\mathbf{f}, k, \ell)$ is in $\rho_{\Sigma,\mathcal{N}}(D)$ if there is a valuation V and a constraint $\sigma \in \Sigma$ such that $V(\mathit{body}_\sigma) \subseteq D$, $V(\mathit{head}_\sigma) = \mathbf{f}@\ell$ and $\mathbf{f}@k \in V(\mathit{body}_\sigma)$.

Let Σ be a set of data-moving distribution constraints and let Z be a hash policy. By $\mathcal{F}(Z, \Sigma)$ we denote the set of all policy pairs $(\delta_{Z,H}, \rho_{\Sigma,\mathcal{N}})$, where H is a tuple of hash functions over $\mathcal{N}$ and Z is compatible with H. By Hash-Constraints we denote the class of families $\mathcal{F}(Z, \Sigma)$, where Z is a hash policy and Σ is a set of data-moving distribution constraints.

▶ **Example 6.** Consider the following set Σ of data-moving distribution constraints:

$$R(y)@\kappa, E_r(y, x)@\lambda \to R(y)@\lambda, \quad S(y)@\kappa, E_s(y, x)@\lambda \to S(y)@\lambda.$$

Intuitively, for an arbitrary network $\mathcal{N}$, the induced communication policy $\rho_{\Sigma,\mathcal{N}}$ communicates each fact $R(y)$ to every server containing a matching fact $E_r(y, x)$. Analogously, $S(y)$ is sent to every server containing at least one $E_s(y, x)$.

[5] While data-moving constraints are similar to VWL of [1], it is not the case that WL can express *all* distribution constraints: node-generating constraints can not be defined (c.f., [19]).

Next, consider the hash policy scheme $Z = \{(\text{Start}, 1, (1)), (E_r, 1, (2)), (E_s, 1, (2))\}$, and let P_{mon} be as in Example 1. Then P_{mon} is parallel-correct w.r.t. every $(\delta_{Z,H}, \rho_{\Sigma,\mathcal{N}})$ where $\mathcal{N}$ is a network and H is a tuple consisting of a single unary hash function h_1 over $\mathcal{N}$ with which Z is compatible. In other words, P_{mon} is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$.

Notice in particular that Σ does not have a constraint enforcing $R(c)$ and $S(c)$ to end up on the same server, as this already follows from Z and P_{mon}. Indeed, every derivation of $R(c)$ is witnessed by either $\text{Start}(c)$ or some $E_r(b, c)$, and every derivation of $S(c)$ is witnessed by $\text{Start}(c)$ or some $E_s(b, c)$. By construction of Z, these facts are always distributed onto the same set of servers $h_1(b)$. As a result, every $R(c)$ and corresponding $S(c)$ are always derived on the same set of servers $h_1(c)$.

4 Parallel-Correctness

In this section, we first show that for the fragments mon-Datalog and fg-Datalog with a decidable containment problem, parallel-correctness is undecidable for relatively simple distribution policies. Then we study settings where deciding parallel-correctness becomes decidable for mon-Datalog and fg-Datalog.

The following result sharpens the undecidability result for parallel-correctness for Datalog in [22], since it states undecidability for fragments of Datalog for which the containment problem is decidable.

▶ **Theorem 7.** Para-Correct*(fg-Datalog, Hash-Constraints) and* Para-Correct*(mon-Datalog, Hash-Constraints) are undecidable.*

The proof is by reduction from the halting problem of deterministic Minsky machines that is well-known to be undecidable (cf., e.g., [27]). The reduction yields a monadic *and* frontier-guarded Datalog program and therefore serves for both stated results.

We now turn to restrictions with decidable parallel-correctness. All our upper bounds rely on the fact that to decide parallel-correctness for all allowed distributed instances and communication policies it suffices to study distributed instances (and communication policies) that distribute facts in a maximally scattered way.

We first make this notion of scatteredness more precise and then present our results on parallel-correctness for Hash-Hash and Hash-MConstraints (a fragment of Hash-Constraints to be defined below) in the two subsequent subsections.

We say that a tuple $H = (h_1, \ldots, h_p)$ of hash functions *scatters* a global instance G if,

- $h_i(\bar{a}) \cap h_j(\bar{b}) = \emptyset$, for all $i \neq j$ and all tuples $\bar{a} \in \mathsf{val}(G)^{\alpha_i}$, $\bar{b} \in \mathsf{val}(G)^{\alpha_j}$, and
- $h_i(\bar{a}) \cap h_i(\bar{b}) = \emptyset$, for all tuples $\bar{a}, \bar{b} \in \mathsf{val}(G)^{\alpha_i}$ with $\bar{a} \neq \bar{b}$.

A distribution policy $\delta_{Z,H}$ *scatters* G if H scatters G. Thus, if $\delta_{Z,H}$ scatters G then two facts $R(\bar{a})$ and $R'(\bar{a}')$ meet at some server, if and only if there is some i and triples $(R, i, \bar{b})$ and $(R', i, \bar{b}')$ such that $\pi_{\bar{b}}(\bar{a}) = \pi_{\bar{b}'}(\bar{a}')$. Similarly, a pair $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$ *scatters* G if H scatters G. For a global instance G and a hash policy scheme Z, we define the canonical scattered instance $\delta_{Z,H}(G)$ where for each $i \leq p$ and $\bar{a} \in \mathsf{val}(G)^{\alpha_i}$, $h_i(\bar{a}) = \{(i, \bar{a})\}$.

4.1 Parallel-Correctness for Hash-Hash

The main result of this section is the following:

▶ **Theorem 8.** Para-Correct*(fg-Datalog, Hash-Hash) and* Para-Correct*(mon-Datalog, Hash-Hash) are* 2ExpTime*-complete.*

The lower bound is, in both cases, by a reduction from the containment problem for monadic Datalog, which is known to be 2EXPTIME-hard [11].

Towards the upper bound, the following lemma shows that it suffices to restrict attention to scattering policies to establish parallel-correctness.

▶ **Lemma 9.** *Let P be a Datalog program and Z, Z_1, Z_2 consistent hash policy schemes. Then P is parallel-correct w.r.t. $\mathcal{F}(Z, Z_1, Z_2)$ if and only if for all $(\delta, \rho) \in \mathcal{F}(Z, Z_1, Z_2)$ and all global instances G that are scattered by (δ, ρ), it holds that $[P, \rho](\delta(G))_{|out(P)} = P(G)_{|out(P)}$.*

Next we show that over scattered instances the distributed evaluation of mon-Datalog and fg-Datalog programs can be simulated by fg-Datalog programs over the global instance.

▶ **Lemma 10.** *For every frontier-guarded or monadic Datalog program P and family $\mathcal{F}(Z, Z_1, Z_2)$ of hash-based policy pairs, a frontier-guarded datalog program P' can be constructed such that for every pair (δ, ρ) from $\mathcal{F}$ and for every global instance G, for which (δ, ρ) scatters G, it holds $P'(G)_{|\text{IDB}(P)} = [P, \rho](\delta(G))_{|\text{IDB}(P)}$. Furthermore, the number of variables and the length of the rules in P' is polynomial in the size of P, Z, Z_1 and Z_2 and the number of rules of P' is at most exponential.*

Proof. We first construct a Datalog program P'' that has the claimed equivalence property and afterwards we "guard" it. The program P'' uses one relation symbol R_i of arity $\alpha_i + ar(R)$ for every relation symbol R from P and every $i \leq p$ occurring in the hash policy schemes. Furthermore it has four kinds of rules:

- For every triple $(R, i, \bar{b})$ in Z, P_1 has a rule $R_i(\bar{v}; \bar{x}) \leftarrow R(\bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, and $\bar{v} = \pi_{\bar{b}}(\bar{x})$ is the projection of $\bar{x}$ to $\bar{b}$.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P there is an indexed version $R_i(\bar{v}; \bar{x}) \leftarrow S_i^1(\bar{v}; \bar{y}_1), \ldots, S_i^n(\bar{v}; \bar{y}_n)$, where $\bar{v}$ is a tuple of pairwise distinct variables.
- For every triple $(R, i, \bar{b})$ in Z_1 and every triple $(R, j, \bar{c})$ in Z_2 there is a rule $R_i(\bar{v}; \bar{x}) \leftarrow R_j(\bar{w}; \bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, $\bar{v}$ is the projection of $\bar{x}$ to $\bar{b}$, and $\bar{w}$ is the projection of $\bar{x}$ to $\bar{c}$.
- Finally, for each IDB-relation of P and every i, there is a rule $R(\bar{x}) \leftarrow R_i(\bar{v}; \bar{x})$, where $\bar{v}$ and $\bar{x}$ are (disjoint) tuples of mutually different variables.

Let (δ, ρ) and G be as in the statement of the lemma. We argue that $P''(G)_{|\text{IDB}(P)} = [P, \rho](\delta(G))_{|\text{IDB}(P)}$. First of all, the rules of the first kind basically define $\delta(G)$, where a fact $R_i(\bar{c}, \bar{a})$ corresponds to $R(\bar{a})$ being at server $(i, \bar{c})$. The second set of rules mimics the local evaluation of Datalog rules at each such server. The third set of rules accounts for the communication of IDB-facts according to ρ. Finally, the last set of rules produces the output tuples. We show in the full paper how P'' can be guarded. ◀

The two lemmas show that testing for parallel-correctness reduces to testing of equivalence of fg-Datalog programs over global databases. Notice that we need both reductions in the previous lemma, as we show in Section 7 that monadic Datalog can not always be rewritten into an equivalent frontier-guarded Datalog program in the distributed setting.

As a final step, we need the following result, which already follows from the construction in [13, Theorem 7] but has a refined statement about the complexity with respect to the number of program rules. A proof sketch can be found in Section 5.

▶ **Proposition 11** ([13]). *The containment problem for Datalog programs (on the left-hand side) and frontier-guarded Datalog programs (on the right-hand side) is decidable in time*

- *doubly exponential in the number of variables in P and P',*
- *doubly exponential in the maximal size of a rule of P and P', and*
- *singly exponential in the number of rules of P and P'.*

Now we are ready to give a proof sketch for the upper bound in Theorem 8. The proof of the lower bound is given in the full version of this paper.

Proof of Theorem 8, upper bound. Let P be a mon-Datalog or fg-Datalog program and Z, Z_1, Z_2 consistent hash policy schemes. Let P' be the fg-Datalog program that can be constructed thanks to Lemma 10. By combing Lemma 10 with Lemma 9 it follows that P is parallel-correct for $\mathcal{F}(Z, Z_1, Z_2)$ if and only if P and P' are equivalent.

The containment problem (and therefore the equivalence problem) for frontier-guarded Datalog is known to be in 2ExpTime [13]. Since the size of P' is at most exponential in the size of P, Z, and Σ, and both programs are frontier-guarded, a triply exponential upper bound follows. However, only the number of rules of P' might be non-polynomial in P, Z, Z_1, Z_2, but the number of variables and the size of each rule is polynomial. Therefore, by Proposition 11 the desired 2ExpTime upper bound follows. ◀

4.2 Parallel-Correctness for Hash-MConstraints

Given the undecidability of Para-Correct(fg-Datalog, Hash-Constraints), we study in this subsection a restriction of distribution constraints that yields decidable parallel-correctness for fg-Datalog with the same complexity as Para-Correct(fg-Datalog, Hash-Hash).

The restriction has two ingredients. First, we require that to test whether a fact $\mathbf{f}$ should be communicated from a server to some other server, no other data values than those in $\mathbf{f}$ need to be communicated. In particular, for constraints, in which the body has only two node variables λ and κ, for the sending and the receiving server, we want to enable λ and κ to test "their" atoms independently, only communicating $\mathbf{f}$. To this end, we say that a data-moving distribution constraint σ has *guarded communication* if whenever its body contains atoms $A_1@\kappa_1$ and $A_2@\kappa_2$, with $\kappa_1 \neq \kappa_2$, then $vars(A_1) \cap vars(A_2) \subseteq vars(head_\sigma)$. The second ingredient of our restriction is a kind of linearity condition for the proof trees that result from computations. A set Σ of data-moving distribution constraints is *modest* if all its constraints have guarded communication and for every $\sigma \in \Sigma$, there is exactly[6] one atom in the body with a relation symbol that occurs in the head of some constraint of Σ. By Hash-MConstraints we denote the class of families $\mathcal{F}(Z, \Sigma)$, where Z is a hash policy and Σ is a modest set of data-moving distribution constraints. The main result of this subsection is the following theorem.

▶ **Theorem 12.** Para-Correct*(fg-Datalog, Hash-MConstraints) is* 2ExpTime*-complete.*

The lower bound is by a reduction from the containment problem for frontier-guarded Datalog, which is known to be 2ExpTime-hard (cf. [9, 13]).

The approach for the upper bound is very similar to that for Theorem 8. Let P be a Datalog program, $D = (G, \mathbf{I})$ be a distributed database over network $\mathcal{N}$, and Σ a set of data-moving distribution constraints. We use proof trees for facts that are produced in a distributed evaluation in the straightforward way by combining proof trees for Datalog facts with proof trees for communicated facts, whose communication is specified by some tgd. For a tree t, we denote its set of nodes by $nodes(t)$, its root by $root(t)$, and its set of leaves by $fringe(t)$. We denote the set of children of a node v by $children_t(v)$. Moreover, for a set $\mathbf{F}$ of facts and a server k, we denote by $\mathbf{F}@k$ the set $\{\mathbf{f}@k \mid \mathbf{f} \in \mathbf{F}\}$.

[6] Clearly, this is the atom $A@\lambda$ required by the definition of *data-moving*.

▶ **Definition 13.** *A* proof tree t *for a distributed fact* $\mathbf{f}@k$ *with respect to a Datalog program* P *and a set* Σ *of data-moving constraints is a tree, in which every node* v *is labelled by a distributed fact*[7] $\mathit{fact}_t(v)$*, and which has the following properties:*

- $\mathit{fact}_t(\mathit{root}(t)) = \mathbf{f}@k$*;*
- *for every inner node* v *labelled* $\mathbf{g}@\ell$*, there is either*
  - *a rule* $\tau \in P$ *and a valuation* V *for* τ *such that* $V(\mathit{head}_\tau) = \mathbf{g}$ *and* $V(\mathit{body}_\tau)@\ell = \{\mathit{fact}_t(w) \mid w \in \mathit{children}_t(v)\}$*, or* *(computed facts)*
  - *a constraint* $\sigma \in \Sigma$ *and a valuation* V *for* σ *such that* $V(\mathit{head}_\sigma) = \mathbf{g}@\ell$ *and* $V(\mathit{body}_\sigma) = \{\mathit{fact}_t(w) \mid w \in \mathit{children}_t(v)\}$*.* *(communicated facts)*

We call such a proof tree *computation-free*, if all its inner nodes are witnessed by constraints from Σ.

▶ **Definition 14.** *A class* $\mathcal{P}$ *of Datalog programs and a class* $\mathcal{C}$ *of distribution constraints have the* polynomial communication property *if there is a polynomial* q *such that the following holds, for each program* $P \in \mathcal{P}$ *and each finite set* $\Sigma \subseteq \mathcal{C}$*: if a distributed fact has a computation-free proof tree with respect to* P *and* Σ*, and with a set* $\mathbf{F}$ *of leaf facts, then there is such a proof tree of size at most* $q(|P|, |\Sigma|)$ *(with leaf facts from* $\mathbf{F}$*).*

▶ **Lemma 15.** *The class of Datalog programs and the class of modest distribution constraints have the polynomial communication property.*

The following lemma shows that it suffices to restrict attention to scattering policies to establish parallel-correctness.

▶ **Lemma 16.** *Let* P *be a Datalog program,* Z *be a hash policy scheme, and* Σ *be a set of data-moving distribution constraints. Then* P *is parallel-correct w.r.t.* $\mathcal{F}(Z, \Sigma)$ *if and only if for all* $(\delta, \rho) \in \mathcal{F}(Z, \Sigma)$ *and all global instances* G *that are scattered by* δ*, it holds that* $[P, \rho](\delta(G))_{|\mathit{out}(P)} = P(G)_{|\mathit{out}(P)}$*.*

The last step in the upper bound proof is the following lemma, similar to Lemma 10, showing that over scattered instances, a fg-Datalog program over the global instance can simulate distributed evaluation. As before, parallel-correctness can thus be reduced to testing equivalence of fg-Datalog programs.

▶ **Lemma 17.** *For every frontier-guarded Datalog program* P*, hash policy scheme* Z*, and set* Σ *of data-moving distribution constraints, which has the polynomial communication property, a frontier-guarded Datalog program* P' *can be constructed such that for every* $(\delta, \rho) \in \mathcal{F}(Z, \Sigma)$ *and every global instance* G *that is scattered by* δ*, it holds* $[P, \rho](\delta(G))_{|\mathit{out}(P)} = P'(G)_{|\mathit{out}(P)}$*. Furthermore, the number of variables and the length of the rules in* P' *is polynomial in the size of* P*,* Z*, and* Σ *and the number of rules of* P' *is at most exponential.*

Proof sketch. Similarly as in the proof of Lemma 10, we first construct an intermediate program P'' which is actually a sub-program of the one constructed there. However, P'' does not have any rules for the communication phases, therefore the second step takes care of communication *and* guarding.

As in the proof of Lemma 10, the program P'' uses one relation symbol R_i of arity $\alpha_i + \mathit{ar}(R)$, for every relation symbol R from P and every $i \leq p$ occurring in the hash policy scheme Z. Furthermore, it has frontier-guarded versions of three kinds of rules:

[7] We omit the subscript t, if it is clear from the context.

- For every triple $(R, i, \bar{b})$ in Z, P_1 has a rule $R_i(\bar{v}; \bar{x}) \leftarrow R(\bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, and $\bar{v}$ is the projection of $\bar{x}$ to $\bar{b}$.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P, there is an indexed version $R_i(\bar{v}, \bar{x}) \leftarrow S^1_i(\bar{v}, \bar{y}_1), \ldots, S^n_i(\bar{v}, \bar{y}_n)$, where $\bar{v}$ is a tuple of pairwise distinct variables.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P, there is a second version $R(\bar{x}) \leftarrow S^1_i(\bar{v}, \bar{y}_1), \ldots, S^n_i(\bar{v}, \bar{y}_n)$.

As before, the rules of the first kind basically define the canonical scattered instance of G with respect to Z, where a fact $R_i(\bar{c}, \bar{a})$ corresponds to $R(\bar{a})$ being at server $(i, \bar{c})$, and the second set of rules mimics the local evaluation of Datalog rules at each such server. The rules of the third kind are responsible for the production of output facts. They can be guarded similarly as in Lemma 10.

However, these rules do not account yet for the redistribution of facts during the communication phases. In a nutshell, we replace IDB-atoms by sets of leaf (IDB- and EDB-) atoms of computation-free subtrees of a proof tree. Thanks to the polynomial communication property these sets only have polynomial size. The rules of P' result from P'' by replacing, in all possible ways, *some* IDB-atoms in bodies by such (polynomial-size) sets of atoms accounting for communication. ◀

5 The Containment Problem for Frontier-Guarded Datalog

In this section we prove Proposition 11 which states an upper bound for the complexity of the containment problem for frontier-guarded Datalog. In principle, it was shown in [13, Theorem 7], but we need that the upper bound is at most singly exponential in the number of rules of the two programs.

Each Datalog program P induces a (possibly infinite) set $C(P)$ of conjunctive queries (CQs) in a natural way by finite "unravellings" of P. It is well-known that for two Datalog programs P, P' it holds $P \subseteq P'$ if and only if[8] each CQ from $C(P)$ is contained in some CQ from $C(P')$ [26], which in turn is equivalent to the existence of a homomorphism from every CQ in $C(P')$ to some CQ from $C(P)$ [14]. The idea in [13], going back to [16], is to use tree automata to test the existence of such homomorphisms. This approach requires to encode CQs by trees over a finite alphabet, even though the number of variables in CQs from $C(P)$ and $C(P')$ can be unlimited. The idea is to encode CQs by symbolic trees in which different, unconnected occurrences of the same variable can represent different variables in a CQ.

The main step of the proof constructs from two fg-Datalog programs P, P' an alternating two-way tree automaton $\mathcal{A}_{P \sqsubseteq P'}$ for the set $T_{P \sqsubseteq P'}$ of symbolic proof trees representing CQs resulting from P that are contained in CQs resulting from P'. In the following, we define symbolic proof trees and, afterwards, we give a high-level description of $\mathcal{A}_{P \sqsubseteq P'}$ in terms of a 2-player game on symbolic proof trees t (for P) and consider its size.

▶ **Definition 18.** *A* symbolic proof tree *for a Datalog program P is a tree in which every node v is labelled by a rule instantiation $head_v \leftarrow body_v$ of a rule of P induced by a variable substitution $s : vars \rightharpoonup vars$, and which has the following properties.*[9]

- *For every node v of t, $body_{v|\text{IDB}(P)} = \{head_w \mid w \in children_t(v)\}$ where $body_{v|\text{IDB}(P)}$ is the set of all the IDB-atoms in $body_v$.*
- *All variables that occur in $body_v$ but not in $head_v$, do* not *occur in the label of the parent of v.*

We note that, in particular, for every leaf v of t, $body_v$ only contains EDB-atoms.

[8] The result is only stated for finite unions in [26] but the proof is just the same for countable unions.
[9] This definition does not yet yield a finite alphabet, but this issue will be addressed below.

With each symbolic proof tree t we associate a CQ $q[t]$ which is the CQ obtained from t in the obvious way, but replacing variable x in t in each x-connected component by a different fresh variable. With a set T of symbolic proof trees we associate a (possibly infinite) set $q[T]$ of CQs via $q[T] = \{q[t] \mid t \in T\}$. Thanks to [15] we can assume that in every symbolic proof tree t of every Datalog program P with n variables only variables among $x_1, \dots, x_{2n}$ occur. For a Datalog program P we denote by T_P the set of symbolic proof trees t for P over set $\{x_1, \dots, x_{2n}\}$ of variables (which we fix for each program P). T_P is a tree language over a (finite) alphabet. More precisely, the alphabet size is at most polynomial in the number of rules of P and exponential in the number of variables occurring in P (i.e. the number of substitutions).

To test containment of P in P' it now suffices to test whether $T_P \subseteq T_{P \sqsubseteq P'}$, where the tree language $T_{P \sqsubseteq P'}$ is defined as

$$T_{P \sqsubseteq P'} = \{t \in T_P \mid \text{ there is a tree } t' \in T_{P'} \text{ s.t. } q[t] \subseteq q[t']\}.$$

We show the following result, which is a slight refinement of a result in [13] in which the complexity in terms of the number of rules in P is better suited for our purposes.

▶ **Proposition 19.** *For each Datalog program P and frontier-guarded Datalog program P' one can construct a two-way alternating tree automaton $\mathcal{A}_{P \sqsubseteq P'}$ for $T_{P \sqsubseteq P'}$ of size*

- *exponential in the number of variables in P and P',*
- *exponential in the maximal size of a rule of P and P', and*
- *polynomial in the number of rules of P and P'.*

To decide containment of P in P' one can construct a non-deterministic automaton for the complement of $T_{P \sqsubseteq P'}$ from $\mathcal{A}_{P \sqsubseteq P'}$ in size exponential in the size of $\mathcal{A}_{P \sqsubseteq P'}$. Then P is contained in P' if and only if the intersection (of the language) of the resulting automaton with an automaton for T_P is empty. Altogether this proves Proposition 11.

Our proof of Proposition 19 uses a direct construction of the automaton $\mathcal{A}_{P \sqsubseteq P'}$ constructed in [13] for fg-Datalog, avoiding some technical complications in [13] caused by the more general *Guarded Queries*. However, we describe $\mathcal{A}_{P \sqsubseteq P'}$ only in terms of a 2-player game on symbolic proof trees t for P. The players are Arthur and Morgana. Morgana's task is to show that t is in $T_{P \sqsubseteq P'}$ and if she succeeds, she wins. Arthur's task is to challenge her proof. We only need to consider the game for trees from T_P since other trees are not relevant for the further processing. For that purpose, Morgana builds up a tree $t' \in T_{P'}$ and a (partial) homomorphism. During the game, both players can traverse the tree t in a two-way fashion but t' is only "traversed" top-down along a single path.

After each round, there is a current node v and a pair (h, M) where h is a partial mapping $h : \mathsf{var}(\tau') \to \{x_1, \dots, x_{2|\mathsf{var}(P)|}\}$ and $M \subseteq \mathit{body}_{\tau'}$, for some rule instantiation τ' of P'. Intuitively, M consists of the atoms of $q[t']$, induced by τ', that still have to be mapped into $q[t]$ and h is a partial homomorphism that has to be extended by Morgana to do so.

In the first round, Morgana chooses a rule τ' of P' and a (partial) mapping h which maps the head of τ' to the head of root_t (i.e. the head of $q[t]$). Thus, $M = \mathit{body}_{\tau'}$. In all further rounds, there are three possible cases.

Case 1 $|M| > 1$: If there is a subset $M' \subsetneq M$ such that $\mathsf{var}(M') \cap \mathsf{var}(M - M') \subseteq \mathsf{dom}(h)$, i.e. if h is defined for all variables occurring in M' and its complement, Arthur chooses M' or $M - M'$ and the new state is (h', M') or $(h', M - M')$ where h' is the restriction to $\mathsf{var}(M')$ or $\mathsf{var}(M - M')$, respectively. Otherwise, Morgana has to extend the mapping

h to another variable x in M as follows. Morgana has to choose a (possibly new) current node v' of the input tree t such that all nodes on the path from v to v' contain all variables[10] of $h(M)$. Then she has to choose a variable appearing at v' for $h(x)$.

Case 2 $|M| = 1$ and $\mathsf{var}(M) \not\subseteq \mathsf{dom}(h)$: Morgana needs to extend the mapping h to all variables in the remaining atom $B \in M$. To this end, she has to choose a node v' such that all nodes on the path from v to v' contain all variables of $h(B)$. Then she has to extend h such that it maps all remaining variables of M to variables at v'.

Case 3 $|M| = 1$ and $\mathsf{var}(M) \subseteq \mathsf{dom}(h)$: Morgana chooses a node v' such that all nodes on the path from v to v' contain all variables of $h(B)$ where B is the remaining atom in M.

Case 3.1 B is extensional: Morgana wins if the image $h(B)$ is in the label of v', otherwise she loses.

Case 3.2 B is intensional: Morgana has to choose a rule instantiation $\tau' \in P'$ whose head equals B and a guard atom C of τ'. Then she extends, if necessary, h to all variables occurring in C (but not in B). If $h(C)$ is *not* at v', Morgana loses. Otherwise, the game continues at v' with the homomorphism h' that is the restriction of h to C and the atom set $\mathit{body}_{\tau'}$.

Due to lack of space we omit the formal definition of $\mathcal{A}_{P \sqsubseteq P'}$ and even the formal definition of alternating two-way tree automata. However, it should be clear that the game can be translated into such an automaton whose states basically consist of pairs (h, M) where $M \subseteq \mathit{body}_{\tau'}$ for a rule instantiation τ' of P' and a partial mapping $h : \mathsf{var}(\tau') \to \{x_1, \dots, x_{2|\mathsf{var}(P)|}\}$. In particular, the number of states is at most exponential in the number of variables of P and P' (i.e. the number of partial mappings h), exponential in the size of the rules in P', and polynomial in the number of rules of P' (choices for M).

6 Parallel-Boundedness

In this section, we study parallel boundedness of Datalog programs in our distributed setting.

▶ **Definition 20** (Parallel-Boundedness). *A Datalog program P is* parallel-bounded *w.r.t. a family $\mathcal{F}$ of policy pairs if there is an $r \in \mathbb{N}$ such that for all policy pairs $(\delta, \rho) \in \mathcal{F}$ and all distributed instances D valid w.r.t. δ, no new* output facts *are computed on any server after r rounds in the distributed evaluation of P on D w.r.t. ρ.*

We note that our definition of parallel-boundedness is more generous than the one in [21], since there the requirement is that no facts whatsoever are produced any more. Of course, complexity upper bounds for our notion translate to the notion of [21].

For classes $\mathcal{P}$ of Datalog programs, and $\mathcal{C}$ of families of policy pairs, Para-Bound($\mathcal{P}, \mathcal{C}$) denotes the decision problem that asks, for a program $P \in \mathcal{P}$ and a family $\mathcal{F} \in \mathcal{C}$, whether P is parallel-bounded *and* parallel-correct with respect to $\mathcal{F}$. Parallel-boundedness for programs that are not parallel-correct does not seem meaningful.

Our objective in this section is to decide parallel-boundedness for frontier-guarded Datalog programs and families of policy pairs in Hash-MConstraints.

▶ **Theorem 21.** Para-Bound(*fg-Datalog*, *Hash-MConstraints*) *is* 2ExpTime*-complete.*

Similarly to the lower bound proof for parallel-correctness (cf. Theorem 12), the lower bound is by a reduction from the containment problem for monadic Datalog, which is known to be 2ExpTime-hard [11].

[10] Recall that non-connected occurrences of a variable in t correspond to different variables in $q[t]$.

Towards the upper bound, let in the following P be a fg-Datalog program, Z a hash policy scheme and Σ a set of data-moving distribution constraints, such that P and Σ have the polynomial communication property. The starting point for the upper bound is the rewriting of P, Z, Σ into a fg-Datalog program P' from Lemma 17 which simulates the distributed evaluation of P on scattered instances. We show first that the consideration of scattered instances suffices also for parallel-boundedness.

▶ **Lemma 22.** *Let P be a Datalog program, Z a hash policy scheme, and Σ be a set of data-moving distribution constraints such that P is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$. Then P is* parallel-bounded *w.r.t. $\mathcal{F}(Z, \Sigma)$ if and only if there is an $r \in \mathbb{N}$ such that for all policy pairs $(\delta_S, \rho_S) \in \mathcal{F}$ and all distributed instances D_S scattered by δ_S, no new* output facts *are computed on any server after r rounds in the distributed evaluation of P on D_S w.r.t. ρ_S.*

The program P' has two kinds of rules: *communication-prone* rules that incorporate communication steps and *communication-free* rules. The body of each communication-prone rule consists of the leaves of some partial proof tree. With each body atom A, the number of communication steps in which A is sent in this proof tree can be computed when the rule is generated and symbolic proof trees corresponding to P' can be extended to carry these numbers as weights.

Then P is parallel-bounded w.r.t. $\mathcal{F}(Z, \Sigma)$ if and only if there is some constant r such that for each symbolic proof tree $t \in T_P$, there is a tree $t' \in T_{P'}$ such that $q[t] \subseteq q[t']$ and along each (root-to-leaf) path of t' the sum of the communication weights is at most r. Thus, testing parallel-boundedness asks for an extension of the approach of Section 5 which constructed an alternating automaton $\mathcal{A}_{P \sqsubseteq P'}$ for the set of trees $t \in T_P$ that are "captured" by trees $t' \in T_{P'}$. Now we are asking for a tree t' that respects the path bound. Since r is not known in advance, the "communication count" can not be incorporated into the states of $\mathcal{A}_{P \sqsubseteq P'}$. However, there is an extension of tree automata that was invented for exactly this kind of situation: cost automata. For our purposes, we need a cost automaton that has one counter which can be incremented but neither be decremented nor reset to zero. A cost automaton $\mathcal{A}$ is *limited* if there is some number s such that in each run of $\mathcal{A}$ the counter stays below s.

▶ **Proposition 23.** *For each frontier-guarded Datalog program P, hash policy scheme Z, and set Σ of modest distribution constraints such that P is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$, one can construct an alternating two-way tree cost automaton $\mathcal{A}$ that uses only a single counter, which is never reset, such that $\mathcal{A}$ is limited if and only if P is parallel-bounded w.r.t. $\mathcal{F}(Z, \Sigma)$. Furthermore, $\mathcal{A}$ has size (and can be constructed in time) exponential in P.*

The final step is then to decide whether $\mathcal{A}$ is limited which is decidable in exponential time thanks to the following result by [12]. In particular, Propositions 23 and 24 imply the upper bound of Theorem 21.

▶ **Proposition 24** ([12])**.** *The limitedness problem for alternating two-way tree cost automata with a single counter that is never reset, is decidable in exponential time.*

Proof idea for Proposition 23. Let P' be the program constructed in Lemma 17 for P, Z, and Σ. Similarly as in Section 5, we describe the cost automaton $\mathcal{A}$ in terms of a 2-player game on symbolic proof trees t for P'.

The rules of the game are almost exactly as in the game underlying $\mathcal{A}_{P' \sqsubseteq P'}$. We assign a value to each play, which is the sum of the weights of the IDB-atoms of communication-prone rules in t' that are visited in that play.[11] Morgana's task is to minimize the value of the

[11] We recall that a play can move arbitrarily in t but evolves along a path of t'.

play, while Arthur's task is to maximize it. Correspondingly, in the semantics of $\mathcal{A}$ the value for t is maximized over Arthur's moves and minimized over Morgana's moves. There is one difference to the rules for $\mathcal{A}_{P' \sqsubseteq P'}$: in situations where Morgana would lose immediately, Arthur is allowed to increment the counter arbitrarily. Altogether, the constructed cost automaton $\mathcal{A}$ is limited if and only if P is parallel-bounded.

The size of $\mathcal{A}$ is the same as for $\mathcal{A}_{P \sqsubseteq P'}$, i.e. polynomial in the number of rules of P' and at most exponential in the number of variables of P' and the maximal length of a rule in P'. Thus, it has size exponential in the original program P (cf. Lemma 17). ◀

7 Variants of the Framework

The precise formalisation of our framework leaves a lot of freedom and it is a fair question how "stable" our results are. To shed some light on this question, we consider in this section two slightly different settings for distributed evaluation strategies for Datalog. The answers that we get are not entirely unambiguous.

We call the first alternative the *locally restrained* setting. It was actually used in [21]. It requires that during each local fixpoint computation, every server only derives facts that it can communicate to other servers according to the communication policy.[12] It turns out that our results on parallel-correctness are not affected by this difference in the setting.

▶ **Proposition 25.**

(a) *In the locally restrained setting,* Para-Correct*(fg-Datalog, Hash-Hash) and* Para-Correct*(mon-Datalog, Hash-Hash) are in* 2ExpTime.

(b) *In the locally restrained setting,* Para-Correct*(fg-Datalog, Hash-MConstraints) is in* 2ExpTime.

The second variant attempts to restrict communication by disallowing servers to send facts that they did not derive themselves through a local computation. We refer to this variant as the *non-transitive communication* setting.

Interestingly, although this variant does not alter our result for frontier-guarded Datalog, it has a huge effect on parallel-correctness for monadic Datalog.

▶ **Proposition 26.**

(a) *In the non-transitive communication setting* Para-Correct*(fg-Datalog, Hash-Constraints) is in* 2ExpTime.

(b) *In the non-transitive communication setting* Para-Correct*(mon-Datalog, Hash-Constraints) is undecidable.*

In a nutshell, the difference between frontier-guarded Datalog and monadic Datalog in the non-transitive communication setting can be explained as follows. The reduction in the proof of Proposition 26b (stemming from the proof of Theorem 7) simulates a Minsky machine by repeated redistribution of facts. Although non-transitive communication only allows to send facts that were derived locally, for mon-Datalog this can be overcome by copy rules. Although copy rules are also allowed in fg-Datalog, they need to be guarded by EDB-atoms there, and this might change the semantics in a distributed setting.

In particular this setting shows that in a parallel setting it can happen that the usual "semantical inclusion" of monadic Datalog in frontier-guarded Datalog by rewriting programs

[12] Although it might seem useless to restrict local computation, this setting allows a more fine-grained control over the workload for each server if for example all servers start with the complete database.

of the former kind into programs of the latter kind fails to hold and monadic Datalog can indeed behave worse than frontier-guarded Datalog. The next example illustrates why this rewriting does not always work in a distributed setting.

▶ **Example 27.** Every monadic Datalog program can be rewritten into an equivalent (w.r.t. evaluation over a global database) frontier-guarded one by adding suitable guards [13]. Consider the program P_{mon} from Example 1. Then the rule $\text{Out}(x) \leftarrow R(x), S(x)$ is not frontier-guarded but can be replaced by the following set of frontier-guarded rules:

$$\begin{aligned}
&\text{Out}(x) \leftarrow R(x), S(x), \text{Start}(x) && \text{Out}(x) \leftarrow R(x), S(x), E_s(x, y)\\
&\text{Out}(x) \leftarrow R(x), S(x), E_r(x, y) && \text{Out}(x) \leftarrow R(x), S(x), E_s(y, x)\\
&\text{Out}(x) \leftarrow R(x), S(x), E_r(y, x) &&
\end{aligned}$$

We denote the such obtained program by P'_{mon}. Then for every global database G, $P_{\text{mon}}(G) = P'_{\text{mon}}(G)$, but $[P_{\text{mon}}, \rho](D_{\text{mon}}) \neq [P'_{\text{mon}}, \rho](D_{\text{mon}})$ for D_{mon} and ρ as defined in Example 2. Indeed, I_3 can never output $\text{Out}(3)$ as that server contains none of the required EDB-facts. The difference between the global and distributed evaluation is that in the former *all* guards are present in the global database while this does not necessarily hold true for all servers in the distributed evaluation.

8 Conclusion

While most effort has been devoted to study correctness properties in the simpler setting of the single-round MPC model (e.g., [4, 18, 22]), to date only Ketsman, Albarghouthi, and Koutris [21] considered correctness w.r.t. the multi-round MPC model (for Datalog). We continue in their footsteps by presenting a more general framework in terms of communication policies for the distributed evaluation of Datalog. We provide decidable instances for the parallel-correctness and parallel-boundedness problem for Datalog. Even though the obtained complexities are far from tractable, we do feel that they contribute to not only the study of reasoning for parallel Datalog but also for parallel queries in the multi-round MPC model in general. Indeed, while a most natural direction for future work is to investigate more tractable settings for Datalog, we are eager to explore communication policies to study the evaluation of non-recursive queries, like join queries, in the multi-round setting. In addition, we plan to investigate data-moving distribution constraints further, trying to characterize the class of distributed evaluations that extensions and restrictions of those can define. We also would like to explore how they can be efficiently evaluated, which is not obvious as such constraints can refer to multiple servers.

An important insight resulting from our investigations is the central role scattered instances play w.r.t. parallel-correctness and parallel-boundedness, and how they can be encoded into Datalog programs if they are defined through hash partitioning. It allowed to translate our reasoning problems for the distributed setting to static analysis problems for traditional Datalog, a topic that has received quite some attention in the literature.

References

1 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Émilien Antoine. A rule-based language for web data management. In *Principles of Database Systems*, pages 293–304, 2011.

2 Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. GYM: A multiround distributed join algorithm. In *International Conference on Database Theory, ICDT 2017*, pages 4:1–4:18, 2017.

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

4 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Transferability for Conjunctive Queries. *Journal of the ACM*, 64(5):36:1–36:38, 2017. doi:10.1145/3106412.

5 Apache Hadoop. https://hadoop.apache.org/.

6 Apache Spark. https://spark.apache.org/.

7 Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and Implementation of the LogicBlox System. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2015*, pages 1371–1382, 2015.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with Guarded Negation. *Proceedings of the Very Large Database Endowment*, 5(11):1328–1339, 2012.

9 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. In *International Colloquium on Automata, Languages, and Programming, ICALP 2011*, pages 356–367, 2011.

10 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *Journal of the ACM*, 64(6):40:1–40:58, 2017.

11 Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic Datalog Containment. In *International Colloquium on Automata, Languages, and Programming, ICALP 2012*, pages 79–91, 2012.

12 Michael Benedikt, Balder Ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *Logic in Computer Science, LICS 2015*, pages 293–304, 2015.

13 Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable Highly Expressive Query Languages. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 2826–2832, 2015.

14 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

15 Surajit Chaudhuri and Moshe Y Vardi. On the equivalence of recursive and nonrecursive datalog programs. *Journal of Computer and System Sciences*, 54(1):61–78, 1997.

16 Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable Optimization Problems for Database Logic Programs (Preliminary Report). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 477–490, 1988. doi:10.1145/62212.62259.

17 Sumit Ganguly, Avi Silberschatz, and Shalom Tsur. A Framework for the Parallel Processing of Datalog Queries. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 1990*, pages 143–152, 1990.

18 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation. In *International Conference on Database Theory, ICDT 2016*, pages 9:1–9:17, 2016.

19 Gaetano Geck, Frank Neven, and Thomas Schwentick. Distribution Constraints: a Declarative Framework for Reasoning about Data Distributions. Manuscript, 2018.

20 Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. Demonstration of the Myria Big Data Management Service. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pages 881–884, 2014.

21 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution Policies for Datalog. In *International Conference on Database Theory, ICDT 2018*, pages 17:1–17:22, 2018.

22 Bas Ketsman, Frank Neven, and Brecht Vandevoort. Parallel-Correctness and Transferability for Conjunctive Queries under Bag Semantics. In *International Conference on Database Theory, ICDT 2018*, pages 18:1–18:16, 2018.

23 Bas Ketsman and Dan Suciu. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *Principles of Database Systems, PODS 2017*, pages 417–428, 2017.

24 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-Case Optimal Algorithms for Parallel Query Processing. In *International Conference on Database Theory, ICDT 2016*, pages 8:1–8:18, 2016.

25 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

26 Yehoshua Sagiv and Mihalis Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM*, 27(4):633–655, 1980. doi:10.1145/322217.322221.

27 Mark V. Sapir. Minsky Machines and Algorithmic Problems. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2015.

28 Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big Data Analytics with Datalog Queries on Spark. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2016*, pages 1135–1149, 2016.

29 Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines. *Proceedings of the Very Large Database Endowment*, 8(12):1542–1553, 2015.

30 Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and Rich Analytics at Scale. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pages 13–24, 2013.

31 Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2015*, pages 17–30, 2015.