DatalogRA : Datalog with Recursive Aggregation in the Spark RDD Model

Marek Rogala [1] Jan Hidders [2] Jacek Sroka [1]

[1]Institute of Informatics, University of Warsaw

[2]Vrije Universiteit Brussel

24 June, 2016

Outline

Outline

Motivation

- Need for high-level declarative languages for *Graph Processing*
- *Datalog* seems an interesting starting point:
  - Well-understood semantics
  - Very parallellizable [Ganguly et al. 1990] [Zhang et al. 1995].
  - Large body of research on optimization [Tekle et al. 2010]
  - Limited recursion matches graph navigation
- Becomes more interesting when extended with basic arithmetic and *stratified* aggregation [Mumick et al. 1990] [Shkapsky et al. 2013]
  - *Counting triangles*
- And even better with *recursive* aggregation [Lam et al. 2013] (*Socialite*)
  - *Shortest Path*, *PageRank*

Contribution of Paper

- Implementation in Spark:
  - Leverages optimizations in Spark (but not yet *Spark SQL*)
  - Embedding in mature framework
  - DatalogRA program can be part of bigger Spark workflow
- Semantics:
  - Explicit and more general semantics then Socialite
  - Some investigation of well-definedness of result

Outline

Syntax of Plain Datalog

- A *database* is a finite set of *facts* of the form $\mathrm{R}(v_1, \ldots, v_n)$ where R is a *relation name* and $(v_1, \ldots, v_n)$ a vector of *domain values*.
  - E.g., $\{\mathrm{A}(1,2), \mathrm{A}(2,3), \mathrm{B}(3,1)\}$
  - We will assume all domains are finite.
- A *basic Datalog program* consist of a set of *rules* where a rule is an expression of the form:

$$\mathrm{R}(\bar{x}) \text{ :- } \mathrm{S}_1(\bar{y}_1), \ldots, \mathrm{S}_n(\bar{y}_n).$$

  where $n \geq 1$, $\mathrm{R}, \mathrm{S}_1, \ldots, \mathrm{S}_n$ are relation names and $\bar{x}, \bar{y}_1, \ldots \bar{y}_n$ are tuples of *variables* and *constants* (i.e., domain values).
  - **Head:** $\mathrm{R}(\bar{x})$
  - **Body:** $\mathrm{S}_1(x_1), \ldots, \mathrm{S}_n(x_n)$, which is a set of *subgoals*
- Operational semantics in terms of a minimal/first fixed point of a function that applies all rules to infer facts.

Semi-naive Evaluation

- **Basic idea:** compute inferred facts based on newly added atoms in previous interation
- For example:
  - a rule $\mathrm{R}(x, y)$:- $\mathrm{S}(x, y, z), \mathrm{R}(z, 2), \mathrm{R}(y, z)$
  - assume R' contains the tuples added in the previous step
  - the tuples added by this rule in the next step are the union of
    - $\{(x, y) \mid \mathrm{S}(x, y, z) \wedge \mathrm{R}'(z, 2), \mathrm{R}(y, z)\}$ and
    - $\{(x, y) \mid \mathrm{S}(x, y, z) \wedge \mathrm{R}(z, 2) \wedge \mathrm{R}'(y, z)\}$
  - after this we compute the next R' by subtracting existing tuples
- Prevents a lot of redundant computation, but same tuple may still be derived more than once

Outline

Basic Idea of DatalogRA

- Based on ideas in *Socialite* [Lam et al. 2013]
- Allows recursive aggregation, under certain conditions
  - i.e., optionally an aggregation function can be specified for the last column of a relation
- **Example:** *(compute length of shortest path from node 1)*

  $\textsc{Edge}(\text{int } src, \text{int } sink, \text{int } len)$
  $\textsc{Path}(\text{int } target, \text{int } dist \text{ aggregate } \textsc{Min})$

  $\textsc{Path}(t, d)$:- $t = 1, d = 0.$
  $\textsc{Path}(t, d)$:- $\textsc{Path}(s, d_1), \textsc{Edge}(s, t, d_2), d = d_1 + d_2.$

- Can be generalized to allow aggregation on multiple columns
- We also allow basic arithmetic predicates and stratified negation

Semantics of DatalogRA

Operational semantics

- The semantics of DatalogRA program P (without negation) is the *first* fixed point of immediate conseq. operator $\Gamma_P \circ \hat{T}_P$
  - $\hat{T}_P$ computes the *bag* of direct consequences of P
  - Γ_P is a function that aggregates as specified in P

Semantics of DatalogRA

The bag of direct consequences

$\hat{T}_P$ computes the *bag* of direct consequences of P:

- The *result bag of a rule r for database D*, $\hat{r}(D)$, is a bag over $r(D)$ such that
  - the multiplicity of each fact $\mathrm{R}(\bar{c})$ in this bag is the number of valuations of the variables in the tail that cause its inference
- The *bag of direct consequences of P for D*, is

$$\hat{T}_P(D) = D \uplus \biguplus_{r \in P} \hat{r}(D)$$

where $\uplus$ is the additive bag union.

Semantics of DatalogRA

The global aggregation function

Γ_P is a function that aggregates as specified in P:

- If relation R is aggregated in P with G:
  - for each vector $\bar{x}$ s.t. there is a fact of the form $R(\bar{x}, y)$ in the input:
    - replace these facts with $R(\bar{x}, G(\bar{Y}))$ where $\bar{Y}$ is the bag of domain values where the multiplicity of an element y is the multiplicity of $R(\bar{x}, y)$ in the input.
- If relation R is *not* aggregated in P:
  - remove duplicate facts for this relation
- **Note:** the result of Γ_P is in both cases without duplicates

Semantics of DatalogRA

Well-definedness

- So the semantics of $P(D)$ is the *first* fixed point of $\Gamma_P \circ \hat{T}_P$ on D
- **Questions:**
  - When is this defined?
  - Is result a minimal fixed point in some sense?
- *Sufficient condition:* for some partial ordering over databases $\Gamma_P \circ \hat{T}_P$ is monotonic
- Subset ordering is too strict when aggregation is used.

Semantics of DatalogRA

Aggregation-dependent partial order

- Assume G is based on a binary operator, say $\oplus_G$, that is commutative and associative:
  - G applied to non-empty bag $\{\!\{a_1, \ldots, a_n\}\!\}$ is $a_1 \oplus_G \ldots \oplus_G a_n$
- Implies sometimes a partial order: $a \sqsubseteq_G b$ iff $a = b$ or there is a c such that $a \oplus_G c = b$.
  - E.g., for $\textsc{Max}$ operator that ordering is $\leq$
  - for $\textsc{Min}$ it is $\geq$
  - for $\textsc{Sum}$ over nonnegative integers it is also $\leq$
  - for $\textsc{Sum}$ over all integers it is **not** a partial order
- We consider only those G where $\sqsubseteq_G$ is a partial order

Semantics of DatalogRA

Aggregation-based database ordering

- Assume $\sqsubseteq_G$ is a partial order for all G in a program P
- We let $\sqsubseteq_P$ define a partial order over *facts*:
  1. if relation R has aggregation operator G in P then $R(\bar{x}, y) \sqsubseteq_P R(\bar{x}', y')$ iff $\bar{x} = \bar{x}'$ and $y \sqsubseteq_G y'$ and
  2. if R has no aggregation operator in P then $R(\bar{x}) \sqsubseteq_P R(\bar{x}')$ iff $\bar{x} = \bar{x}'$.
- We let $\sqsubseteq_P$ also define a partial order over *databases*:
  1. $D_1 \sqsubseteq_P D_2$ holds iff for all $R(\bar{x}) \in D_1$ there is a fact $R(\bar{x}') \in D_2$ such that $R(\bar{x}) \sqsubseteq_P R(\bar{x}')$
- If P is monotonic w.r.t. to $\sqsubseteq_P$, i.e., $\Gamma_P \circ \hat{T}_P$ is monotonic under $\sqsubseteq_P$, then P always computes a minimal fixed point.

Semantics of DatalogRA

A sufficient condition for monotonicity

- Also assume all G are all idempotent, i.e., $a \oplus_G a = a$
  - e.g., for MIN and MAX
- Then multiplicity in the bags is ignored by Γ_P, so $\Gamma_P \circ \hat{T}_P = \Gamma_P \circ T_P$, where T_P is the classical Datalog inference function
- Since Γ_P is always monotonic under $\sqsubseteq_P$, it is sufficient to require that T_P is monotonic under $\sqsubseteq_P$.
- Complexity of deciding this property is still unclear
- Under such monotonicity we essentially can do semi-naive evaluation:
  1. "New facts" are those not subsumed (under $\sqsubseteq_P$) by an existing fact
  2. Infer additional results in T_P for these facts as usual
  3. Add these results and apply Γ_P

Outline

Integrating DatalogRA in Spark

- The main component is the *Database* class with a *datalog* method.
- Contains a set of named relation objects.

```
1  val edgesRdd = ... // Read from HDFS or computed using Spark
2
3  val database = Database(Relation.ternary("Edge", edgesRdd))
4  val resultDatabase = database.datalog("""
5      declare Path(int v, int dist aggregate Min).
6      Path(x, d) :- s == 1, Edge(s, x, d).
7      Path(x, d) :- Path(y, da), Edge(y, x, db), d = da + db.
8  """)
9  val resultPathsRdd = resultDatabase("Path")
10
11 ... // Save or use resultPathsRdd as any RDD.
```

Optimizations

- The rules are divided into a sequence of strata
  - evaluated one by one
  - non-recursive strata iterate only once
- Semi-naive evaluation if possible, each iteration determining a *delta database* of "new facts"
- Caching intermediate results: if relation referred to is from a lower stratum it is persisted as RDD after it is generated

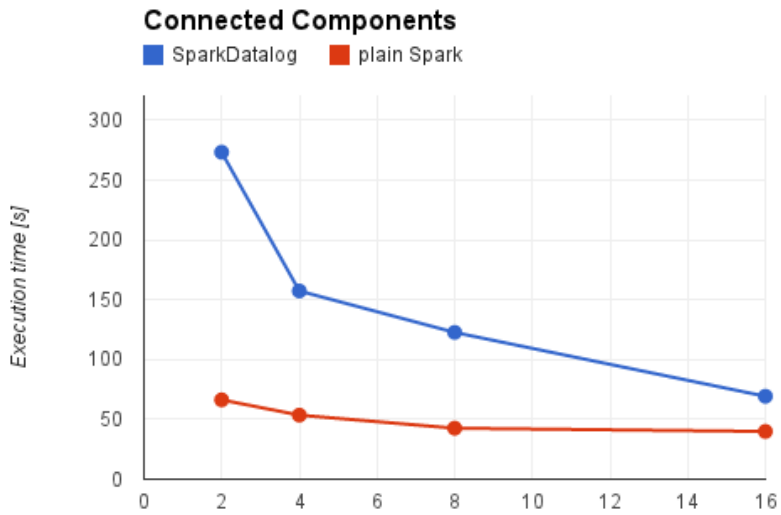
Outline

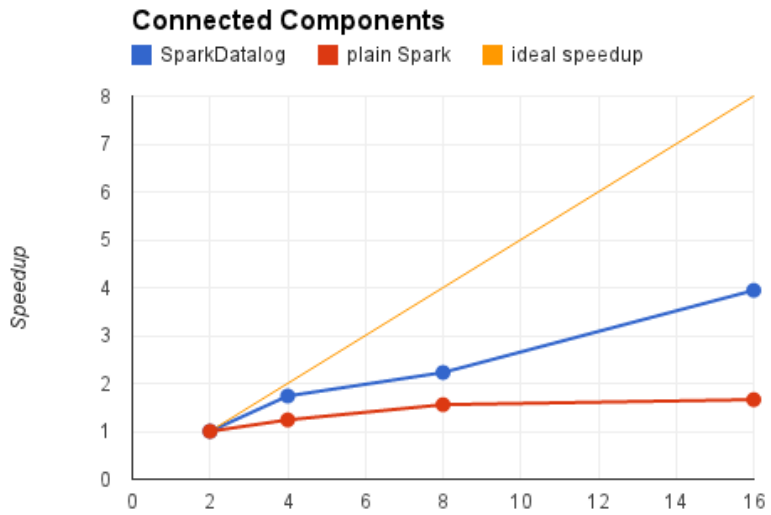
Experimental Setup

- Three classic graph problems:
  - Connected Components
  - Shortest Paths
  - Triangle Counting
- Compared to plain Spark using core methods and GraphX extension
- Executed using Amazon EC2 clusters consisting of 2, 4, 8 and 16 worker nodes and one master node.
  - Each node was a 2-core 64-bit machine with 7.5 GB of RAM memory.
- Dataset used: social graph of Twitter circles on SNAP, which has 2.4M edges.

Experimental Results

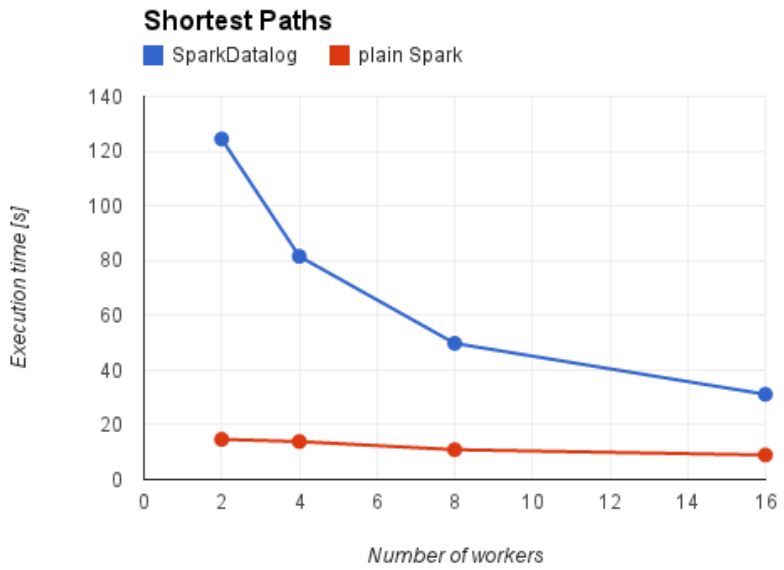
Efficiency: Connected Components

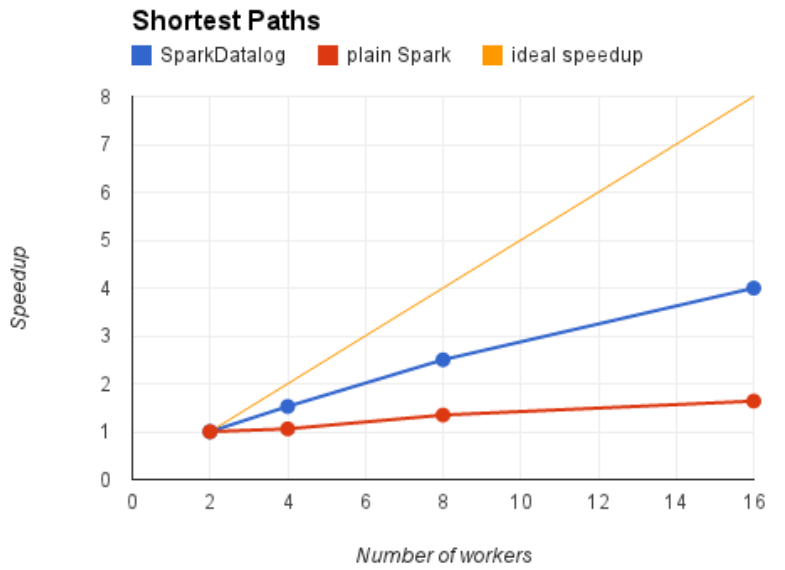




Experimental Results

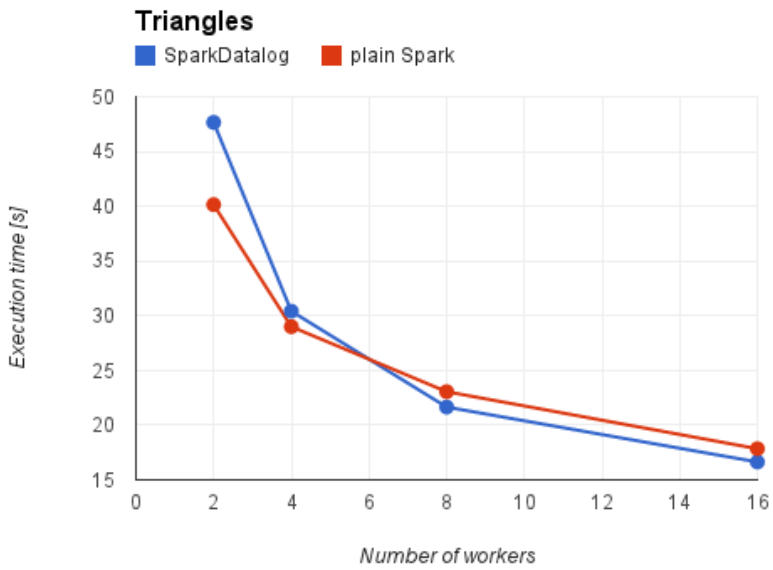
Efficiency: Shortest Paths

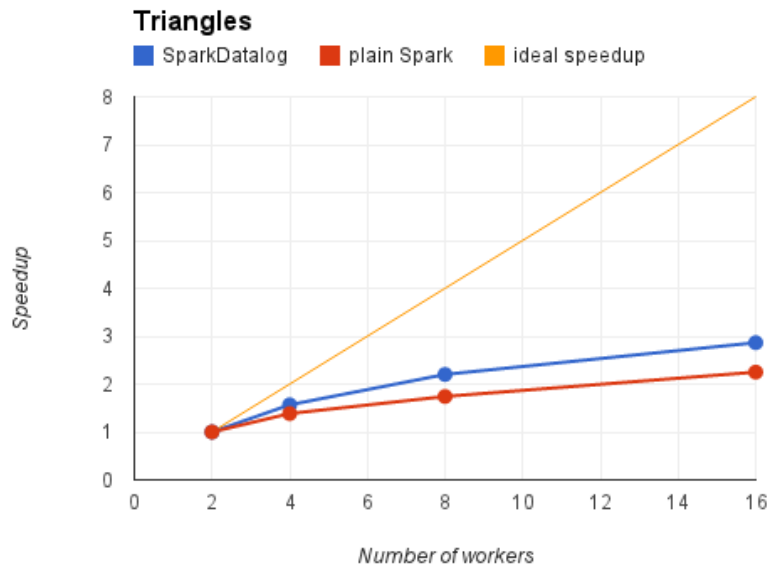




Experimental Results

Efficiency: Triangles





Experimental Results

Compactness

Number of lines in programs, excluding data loading and comments.

	plain Spark	**SparkDatalog**
Connected Components	11	6
Shortest Paths	12	4
Triangles	7	5

Outline

Conclusions and Future Work

- Studied implementing Datalog with recursive aggregation in Spark
- Ongoing work:
  - Leveraging Spark SQL
  - Support wider class of recursive aggregation
  - Magic sets
  - More optimized distributed execution of conjunctive queries
  - Optimizing more general classes of aggregation operations
  - Investigate decidability of aggregation monotonicity