I. Motivation

- Many big data platforms
- Still big data analysis is hard!
  - Low-level primitives
  - Too static (hard to make changes)
  - Complex configuration
  - Not efficient
- We want big data analysis as easy as Python (or Ruby)

II. Introducing SociaLite

SociaLite is a query language that is
- As easy as Python
- As efficient as optimized C++/Java
- Parallel execution
- High-level primitives for parallelization
- Python integration
High-level queries compiled to fast parallel/distributed code!

III. Syntax and Semantics of SociaLite

Local Tables

```
Foo(int i, String s).
```

This declares a table named Foo with two columns; each machine in the cluster has its own instance of the table.

Distributed Tables

```
Bar(int j(String s, double d)).
```

With the partitioning operator [ ], this declares a distributed table Bar; it is horizontally partitioned (or sharded) and stored across machines in the cluster.

Table Options

- Nesting
  ```
  Bar(int i(String s, double d)).
  ```
- Ranges
  ```
  Bar(int i)(String s, double d)).
  ```
- Indexing and sorting
  ```
  Bar(int i)(String s, double d) indexby i.
  Bar(int i)(String s, double d) sortby d.
  ```

Rules

```
Foo(a,b) :- Bar(a[b], Qux(b)).
```

Tables in the rule body are joined, and the result tuples are inserted to the table in the rule head.

Aggregation

```
Foo(a_OK, $min(a)).
```

The $min in the rule head is an aggregate function that is applied to tuples in Foo having the same first column value.

Python Integration

SociaLite queries in Python code are marked with a pair of backticks, as shown in following.

```
print "This is Python code."
print "Foo(int i, double d).
print Foo(a,b) : = a-10, b=42.0."
```

`for a,b in `Foo(a,b)`:
print a, b`

Tables can be accessed from Python code as following

```
Access Python in SociaLite

Python variables are referenced in SociaLite queries with a preceding dollar sign.

```python
var=42
"Foo(int i, double d).
print Foo(a,b) : = a-$var, b=42.0."
```

Python functions are referenced similarly, but requires type annotation.

```python
@returns(int)
def func(a,b): return a+b

"Foo(a,b) : = a+func(0,1), b=func(1,1)."
```

IV. Evaluation

Sequential

Parallel

Distributed

SociaLite vs Java

Connected Components

Triangle

Clustering Coefficients

Collaborative Filtering
**Logical Programming**

- Logic programming language as well as Database query language
- Clean syntax and semantics
- Express graph algorithms very well
- Recently used in many domains
  - (programming analysis, network/distributed systems, security, etc)
- Slow performance
- Lack of control in data structure and execution

---

**Datalog**

- The recursive rule computes shortest paths from source node (SRC) to the rest nodes, and stores the distances in Path table.
- The base case of the recursion (written in red) sets the distance to the source node to be 0, and the recursive part (written in green) computes the distances to the rest nodes recursively.

- `Path(int n, double dist) indexby i.`
  - `Edge(int a, (int t, double len)) indexby a.`
  - `Path(t, $min(d)) :- t=SRC, d=0.`
  - `Path(n, d), Edge(n, t, d, d1), d=d+d1.`

**Graph Algorithms**

- The Python function `getDiff` (in orange) computes the distance from a data point to a cluster center. The first rule in the loop (in red) finds a new center for each cluster by taking average of data points for each dimension. The second rule (in green) computes new mapping from data points to clusters with updated cluster centers.

- `Data(int n:0..$N, double[] p).`
  - `Weight(int n:iter, double[] w).`
  - `Cluster(int n:0..$K, int:iter, Avg[], ArgMin[]) groupby (2).`
  - `Cluster(int n:0..$K, int:iter, ArgMin min) groupby (2).`
  - `Cluster(int n:0..$K, int:iter, ArgMin min) groupby (2).`
  - `Gradient(int n:iter, double[] g).`
  - `Rank(n, 1, r)`
  - `Rank(n, 0, $sum(r)) :- Node(n), r=0.15*1.0/$N;` for i in range(20):
    - `Rank(n, $i+1, $sum(r)) :- Node(n), r=0.15*1.0/$N;` for i in range(0, 100):
      - `Gradient($i+1, $sum(w)) :- Node($i), r=0.01;` return 1.0/(1+math.exp(-x))
      - `Gradient($i+1, $sum(w)) :- Node($i), r=0.01;` return 1.0/(1+math.exp(-x))

**PageRank**

- The table stores friends-of-friends relation after evaluating the rule.

- `Friends(int i, (int f)) indexby i.`
  - `Friend(int i, (int f)) indexby i.`
  - `Friend(int i, (int f)) indexby i.`

**Data Mining Algorithms**

- The table stores friends-of-friends relation after evaluating the rule.

- `Data(int n:0..$N, double[] p).`
  - `Weight(int n:iter, double[] w).`
  - `Cluster(int n:0..$K, int:iter, Avg[], ArgMin[]) groupby (2).`

**Logistic Regression**

- Update rule (gradient descent) \( \theta_j := \theta_j + \alpha \frac{\partial J}{\partial \theta_j} \)

---

**Recursive Aggregate Functions**

- Aggregate functions that can be used in recursive rules
  - Must be a meet operator (idempotent, commutative, and associative)
  - e.g. Min, Max
- Optimizes execution of recursive rules
  - Guarantees iterative solution is same as greatest fixed-point solution
  - Semi-naive evaluation can be defined, and gives same result as naïve evaluation.

---

**Technical Background and Algorithms**

- SocialLite
- K-Means
- PageRank
- Recursive Aggregate Functions
- Data Mining Algorithms