Datalog extensions

Simone de Oliveira Santos

February 26, 2014

Distributed SociaLite: A Datalog-based language for large-scale graph analysis

Motivations

- With the rise of world-wide social networks, many graph-oriented databases are now available.
- The large graphs need large-scale distributed systems to be processed.
- MapRedure and Pregel are examples of techniques that masks the complexity of distributed processing.
- Pregel adopts a vertex-centric programming model, this feature makes it one of the most well-known languages designed in response for social network analysis.

Distributed SociaLite

- SociaLite makes it possible for an implementation that hides the low-level complexity in distributed execution from the programmers.
- It is built upon the sequential SociaLite as a new language extension and optimization for large-sale graph analysis on distributed machines.

Contributions

Data distributed through sharded tables The programmer does not have to worry about distributing the computation and managing the communication. Instead, he simply specifies how data are to be decomposed across the machines.

Parallel recursive aggregate function Delta stepping is effective for parallelizing the shortest path algorithm. Now it is made available to any recursive monotone aggregate functions.

Efficient approximation SociaLite automatically returns partial answers as it process the Datalog rules using semi-naive evaluation. Delta stepping and Bloom filters are used to deliver partial results.

Single-source paths problem

(a) Datalog:

$$\textsc{Path}(t, d) \ :- \ t = 1, d = 0.0.$$
$$\textsc{Path}(t, d) \ :- \ \textsc{Path}(s, d_1), \textsc{Edge}(s, t, d_2), d{=}d_1{+}d_2.$$
$$\textsc{MinPath}(t, d) \ :- \ \textsc{Path}(t, \$\textsc{Min}(d)).$$

(b) Sequential SociaLite:

$\textsc{Edge}$ (int s:0..9999, (int t, double $dist$)).
$\textsc{Path}$ (int t:0..9999, double $dist$).

$$\textsc{Path}(t, \$\textsc{Min}(d)) \ :- \ t = 1, d = 0.0;$$
$$:- \ \textsc{Path}(s, d_1), \textsc{Edge}(s, t, d_2), d{=}d_1{+}d_2.$$

(c) Parallel SociaLite:

$\textsc{Edge}$ [int s:0..9999] ((int t, double $dist$)).
$\textsc{Path}$ [int t:0..9999] (double $dist$).

$$\textsc{Path}[t](\$\textsc{Min}(d)) \ :- \ t = 1, d = 0.0;$$
$$:- \ \textsc{Path}[s](d_1), \textsc{Edge}[s](t, d_2), d{=}d_1{+}d_2.$$

Parallel SociaLite extensions

- Ask programmers to indicate how the date are to be partitioned across the distributes machines.
- It introduces a *location operator*([]) to be applied to the first column of a data declaration.
- The declaration PATH[int *t*](double *dist*) specifies that PATH is a relation with two columns and that the table is horizontally partitioned, or *sharded*.

Data distribution in SociaLite

- **Shards**, each containing a number of rows, are placed on different machines in a distributes system.
- The value of the first column in a relation is the *shard key*, and it dictates where the relation is located.
- The function $\textsc{ShardLoc}(r, x)$ returns the machine number based on the value of the *shard key* x in relation r.

Range-based and hash-based shards

- There are two kinds of sharding: **range-based** and **hash-based**
- If the first column of a sharded array has a range, then the range is divided up into consecutive subranges and evenly distributes across the machines.

Supose the shard key x in relation r has hange $l..u$, then

$$\textsc{ShardLoc}(r, x) = \left\lfloor \frac{x - l}{\lceil \frac{u-l+1}{n} \rceil} \right\rfloor$$

Where n is the number of machines in this system.

Range-based and hash-based shards

- If no range is given, it is used a standard hash function to map the shard key to a machine location.
- The range of the hashed values is also evenly distributed across all the machines.
- The location operator is also used in rule specifications to make it apparent to the programmer where the operands and results are placed.

Distribution of computation

$$\text{Bar}[\text{int } x : 0..9999](\text{int } z).$$
$$\text{Baz}[\text{int } z : 0..9999](\text{int } y).$$
$$\text{Foo}[\text{int } x : 0..9999](\text{int } y).$$
$$\text{Foo}[x](y) \text{ :- } \text{Bar}[x](z), \text{Baz}[z](y)$$

The rule specifies that data Bar[x](z) are to be transfered to a machine with ID ShardLoc(Baz,z), where join operation with Baz[z](y) is performed.
The result from the join operation is then transferred back to a machine with ID ShardLoc(Foo,x).

Batching the messages

The compiler rewrites the rules so that all the computation is performed on local data, and communication is necessary only for sending the result to the right machine.

BAR'[int I:0..9999](int x, int z).
BAR'[z](x, z) :- BAR[x](z). (1)
FOO[x](z) :- BAR'[z](x, z), BAZ[z](y)

The rule (1) distributes all the relations of BAR on each machine into a sharded relation BAR'. Each shard destined for a different machine, SHARDLOC(BAR',z) $\neq$ SHARDLOC(BAR,x), is sent to the appropriate machine in one message.

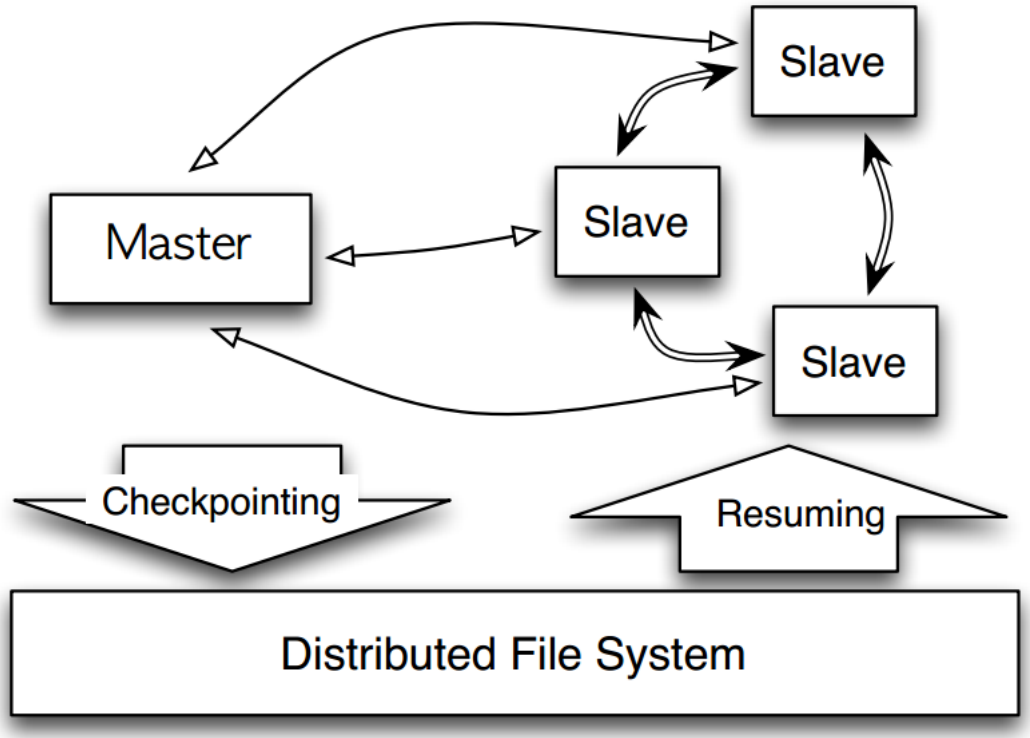
Distributed system architecture

The SociaLite parallel machine consists of a master machine which interprets the Datalog rules and issues work to a collection of slave machines.

Fault-tolerant distributed file system

If one or more workers fail, the intermediate states are restored from the latest check point and the evaluation is resumed from that point.

Distributed system architecture



Distributed system architecture

- The master compiles the SociaLite program into a dependence graph, where each node corresponds to a join operation and each edge represents a data dependence.
- After finding the strongly connected components, the master organize all the nodes within a stratum into epochs.
- Recursive strongly connected components are each placed into their own epoch.
- Non-recursive rules can be combined into epochs with the constraint that dependences across the epochs form an acyclic graph.

Distributed system architecture

- The master then visits the epoch graph in a topological order and instructs the slaves to work on an epoch at a time.
- Each slave node repeatedly executes the rules upon the arrival of communication from other nodes and updates the internal tables or sends messages to remote nodes as needed.
- Protocol used to detect when the slaves complete an epoch:
  - slaves report its idle status to the master if there are no rules to execute and no more data to send.
  - Upon receiving an idle status from all the slaves, the master confirms with each slave that it is still idle with the same last reported timestamp. This process is repeated until confirmation from each slave are received.

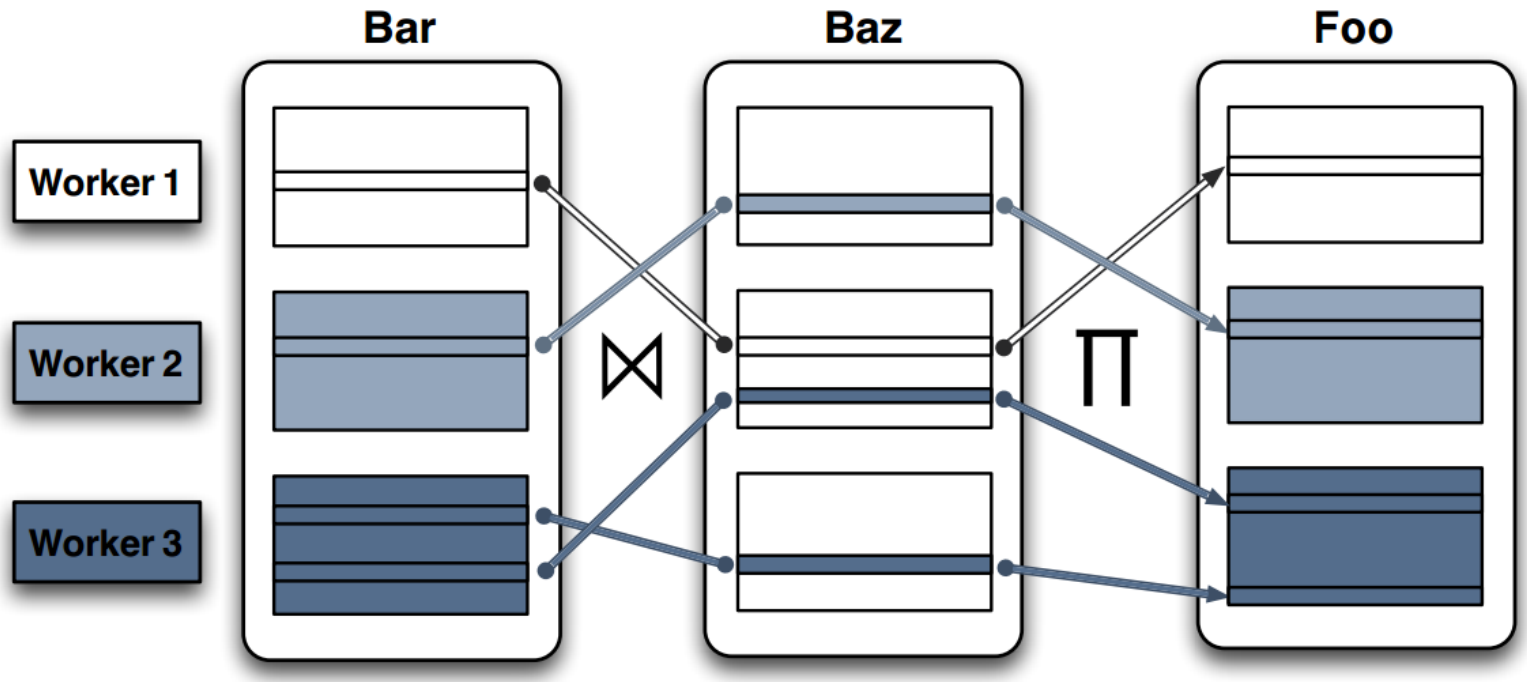
Multiple cores

- Each sharded table is further *subsharded*, with each subsharded $32n$ ways, where n is the number of cores supportes on the machine.
- SociaLite breaks up tasks into units that operate in a subshard at a time and are placed in a dynamic task queue.
- Each machine has a manager responsible for:
  - accepting epoch assignment
  - report and confirming idle status with the master
  - accepting inputs
  - placing the corresponding tasks on the work queue
  - sending data intended fir other machines
- Each worker fetches tasks from the work queue, performs the task and updates the resulting data tables.

Multiple cores

Two optimizations to minimize the synchronization overhead

- Non-recursive epochs are further broken down into *sub-epochs* whose rules are totally independent of each other. Synchronization is necessary only to enforce mutual exclusion between result update operations.



Multiple cores

Two optimizations to minimize the synchronization overhead

- No synchronization is necessary if the updated shard is guaranteed to be accessed by only one worker.

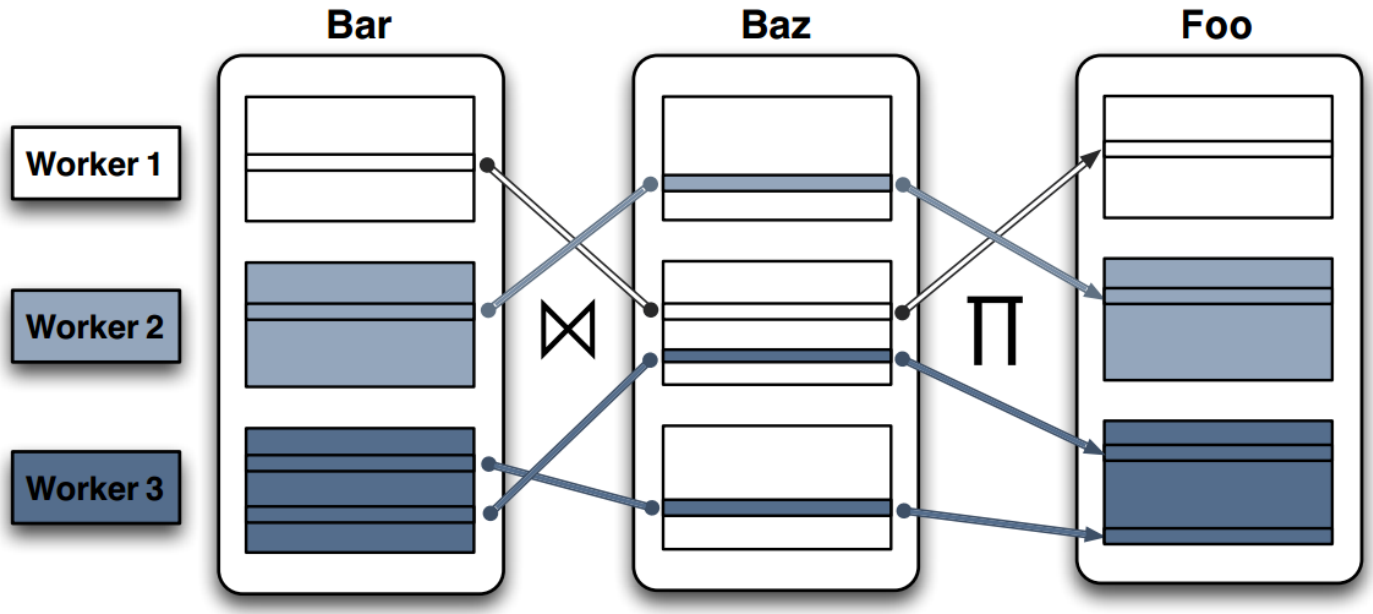


Figure : Rule $\textsc{Foo}[x](y)$:- $\textsc{Bar}[x](z)$, $\textsc{Baz}[z](y)$.

Parallelizing recursive aggregate functions

- The sequential SociaLite compiler uses a priority queue to keep track of all the newly tuples to be evaluated.
- By operating on the lowest values the solution converges quickly to the greatest fix point, yielding a behavior similar to that of Dijkstra's shortest-paths algorithm.
- **It is not suitable for distributed SociaLite to use priority queue, as it will serialize the evaluation of aggregate functions.**

Parallelizing recursive aggregate functions

- **Delta Stepping** has shown to be effective in computing shortest paths for large-scale graphs.
- The parallel implementation of Delta Stepping algorithm gives near linear speedup (shown in a experimental study).
- Distributes SociaLite have generalized this technique and incorporate it into the compiler.

Approximation Computation

- Results of a query in a large-scale graph may also be very large.
- For the sake of a faster response, it might be desirable to provide partial results to the user instead of waiting until all results are available.
- The semi-naive evaluation of SociaLite rules supports approximate computation trivially.
- Approximation is achieve by simply terminating each epochs of the execution before the fix point is reached.

Bloom-Filter based approximation

The Bloom filter

Used as a means to provide a quick approximation to the case where the final result may be relatively small, but the intermediate results are large.

- When intermediate sets gets too large to be represented exactly, Bloom filter is used to represent them compactly.

Friend of a friend program

$\textsc{Foaf}(n, f_2)$:- $\textsc{Friend}(n, f)$, $\textsc{Friend}(f, f_2)$.
$\textsc{FoafSum}(\text{n},\$\textsc{Sum}(\text{a}))$:- $\textsc{Foaf}(n, f_2)$, $\text{Attr}(f_2, a)$