Transitive Closure and Recursive Datalog Implemented on Clusters

Foto N. Afrati

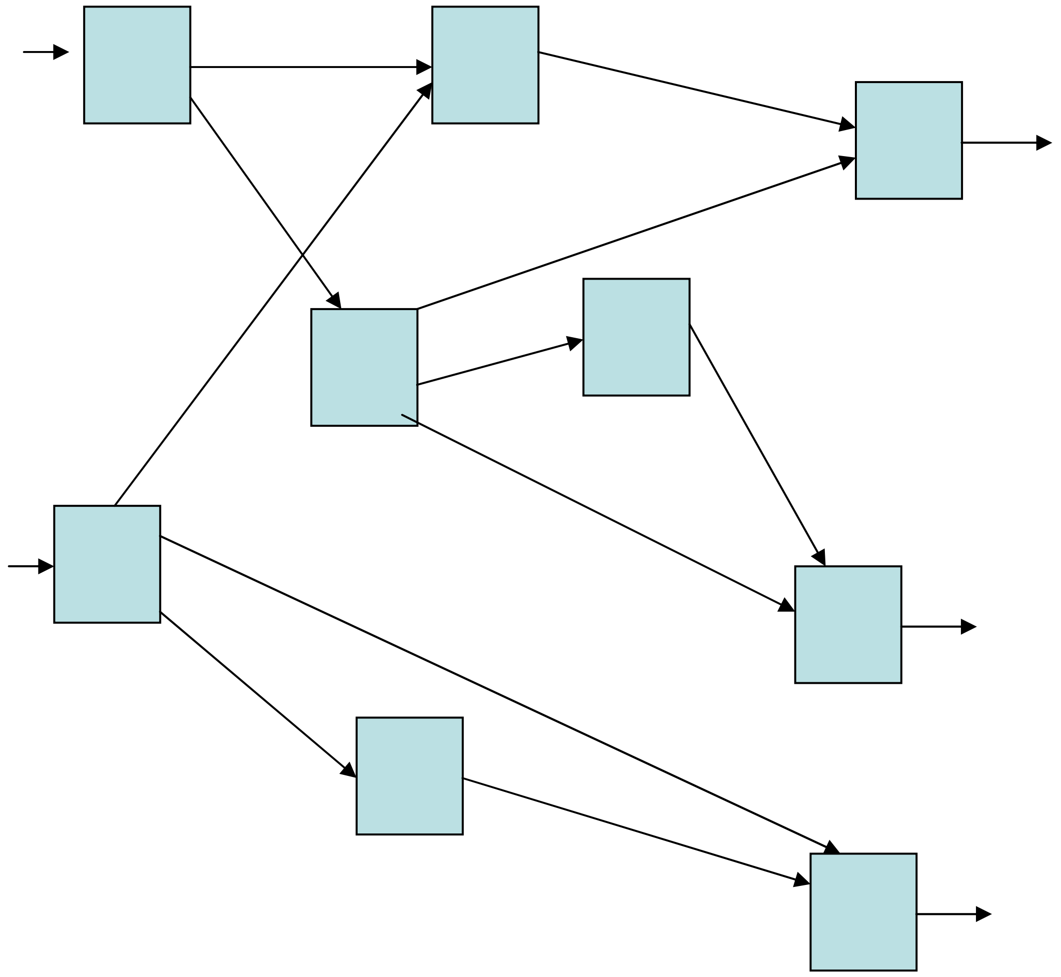
National Technical University of Athens, Greece

Joint work with Jeff Ullman

Dataflow systems vs. Recursive-task systems

- Computation in rounds
- Tasks feed input themselves several times during the computation
- “Blocking property” does not hold
- Recovery from node failure is not easy
- Need to transfer small files especially towards the end of the computation
- Small files incur large overhead

Dataflow Systems: an acyclic network of functions



Clustera – University of Wisconsin.

Hyracks – Univ. of California/Irvine.

Dryad/DryadLINQ – Microsoft.

Nephele/PACT – T. U. Berlin.

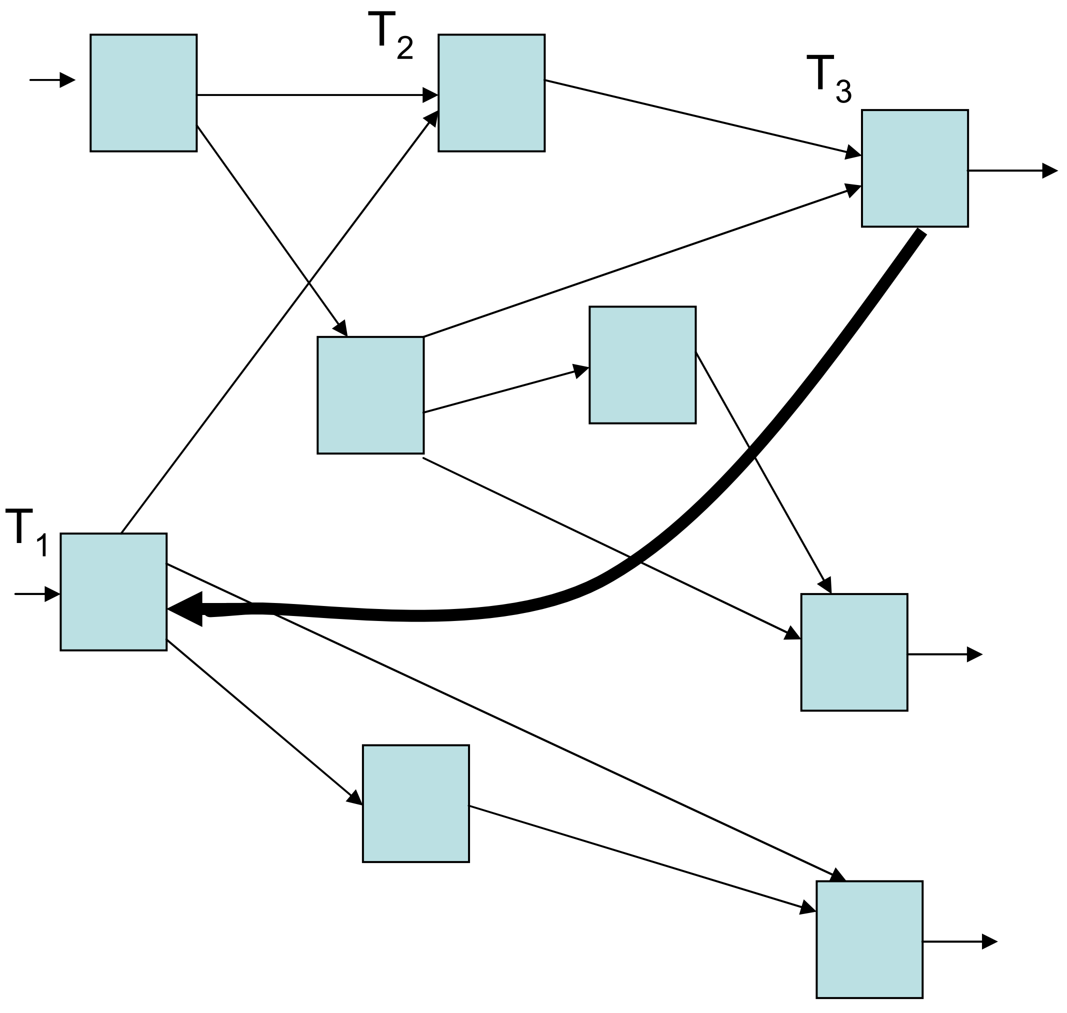
BOOM – Berkeley.

epiC – N. U. Singapore.

“Blocking property” holds

Not Map and Reduce tasks any more, could be anything.

Recursive Systems: a cyclic network of functions



Pregel
Giraph
Recursive Hyracks

"Blocking property" does **not** hold

Computation in rounds

Recovery from Node Failure: Blocking property

- Map-reduce deals with node failures in a simple way.
- Both Map tasks and Reduce tasks have the *blocking* property:

  A task does not deliver output to any other task until it has completely finished its work.
- Dataflow systems also have the blocking property.

Communication cost of the computations

- Individual communication for each task: Sum of all its inputs during the computation
- Total communication: Sum of all individual communication costs over all tasks.

Our focus: The Endgame

- Problem: in a cluster, transmitting small files carries much overhead.
- Minimum communication cost per transmission.
- Some recursions, like TC, take a large number of rounds, but the number of new discoveries in later rounds drops.

Transitive closure (TC)

Nonlinear O(log n) rounds

```
p(x,y) <- e(x,y)
p(x,y) <- p(x,z) & p(z,y)
```

Left-linear O(n) rounds

```
p(x,y) <- e(x,y)
p(x,y) <- p(x,z) & e(z,y)
```

Comparison

- Linear: 5=4+1; 4=3+1; 3=2+1; 2=1+1.
- Nonlinear: 5=4+1, 5=3+2, 5=2+3, 5=1+4; 4=3+1, 4=2+2, 4=1+3; 3=2+1, 3=1+2; 2=1+1.

Lower Bound on Query Execution Cost for Datalog

- Number of Derivations: the sum, over all the rules in the program, of the number of ways we can assign values to the variables in order to make the entire body (right side of the rule) true.
- Key point: An implementation of a Datalog program that executes the rules as written must take time on a single processor at least as great as the number of derivations.
- Seminaive evaluation: time proportional to the number of derivations

Number of Derivations for Transitive closure

- *Nonlinear TC:*
  the sum over all nodes c of the number of nodes that can reach c times the number of nodes reachable from c.
- *Left-linear TC:*
  the sum over all nodes c of the in-degree of c times the number of nodes reachable from c.

Nonlinear TC has more derivations than linear TC

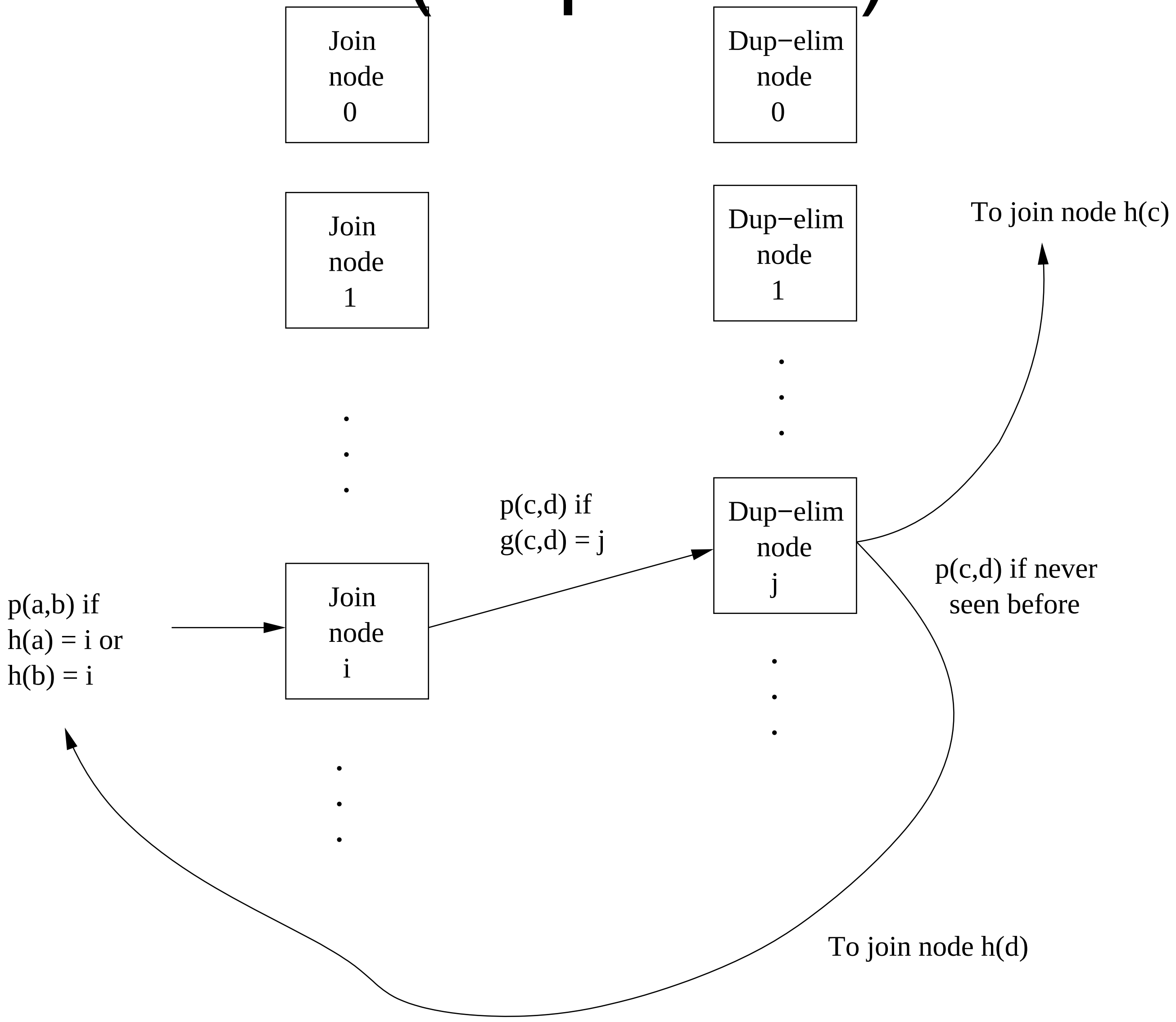
Implementing TC on a Cluster

- Use k tasks.
- Hash function h sends each node of the graph to one of the k tasks.
- Task i receives and stores Path(a,b) if either h(a) = i or h(b) = i, or both.
- Task i must join Path(a,c) with Path(c,b) if h(c) = i.

Another implementation of TC (Dup-Elim)

- *Join tasks*, which perform the join of tuples as in earlier slide.
- *Dup-elim tasks*, whose job is to catch duplicate *p*-tuples before they can propagate.

Nonlinear TC (Dup-Elim)



Improved algorithms

- Small number of rounds
- Small number of derivations

Unique decomposition property

- A path is discovered only once.
- Linear TC has this property.
- Nonlinear TC does not have this property.
- Question: Is there a Datalog program with the unique decomposition property, which can be executed in logarithmic number of rounds?
- Answer: This is the Smart Algorithm and many others.

Algorithm Smart

- In each round combine paths of length a power of 2 with paths of length no greater than that power of 2.
- Example: Round 1: paths of length 1.

Round 2: paths of length 1+1=2

Round 3: paths of length 2+1=3 and 2+2=4.

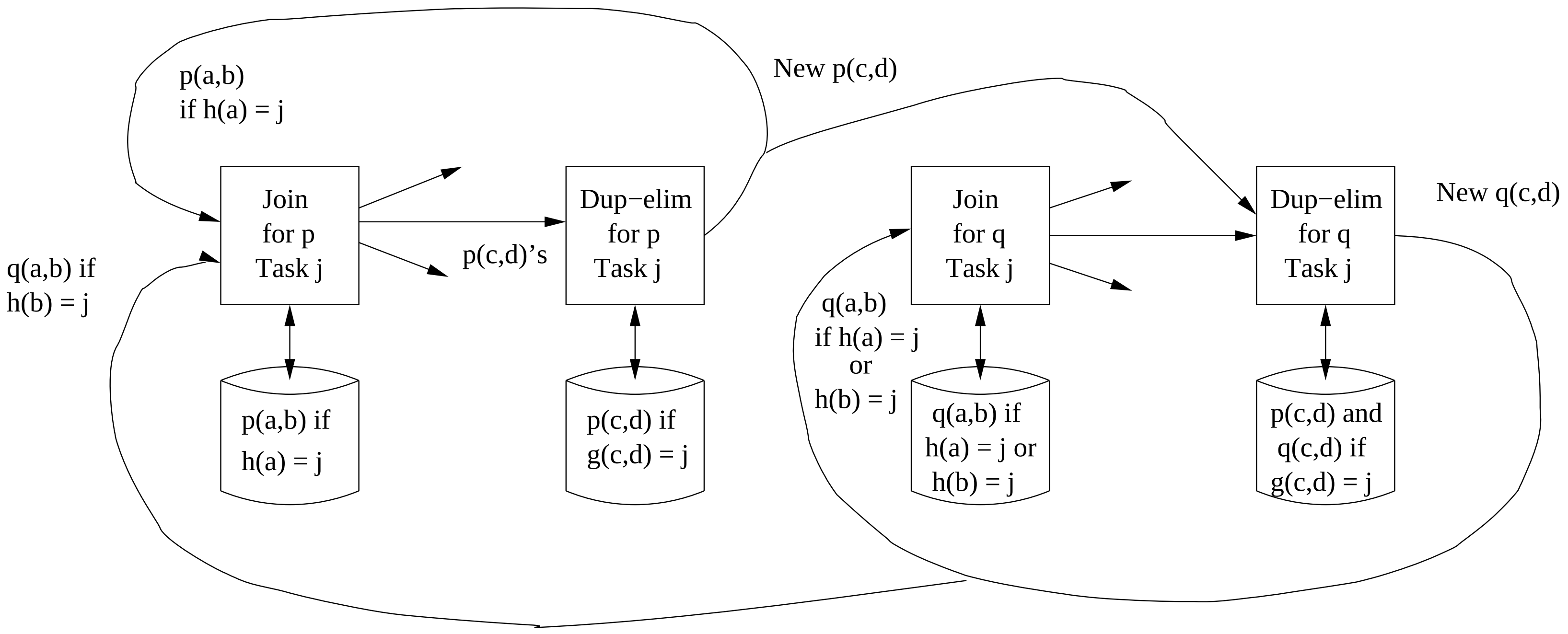
Round 4: paths of length $2^2 + 1=5$, $2^2 + 2=6$,

$2^2 + 3=7$ and $2^2 + 2^2 =8$.

Algorithm Smart

```
1) $q0(X,Y)$ :- $e(X,Y)$;
2) $p0(X,X)$ :- ;
3) $i := 0$;
4) repeat {
5) $i := i + 1$;
6)     $p_i(X,Y)$ :- $q_{i-1}(X,Z)$ & $p_{i-1}(Z,Y)$;
7)     $p_i(X,Y) := p_i(X,Y) \cup p_{i-1}(X,Y)$;
8)     $q_i(X,Y)$ :- $q_{i-1}(X,Z)$ & $q_{i-1}(Z,Y)$;
9)     $q_i(X,Y) := q_i(X,Y) - p_i(X,Y)$;
}
10) until ($q_i == \emptyset$)
```

Implementation of Algorithm Smart



Algorithm Balance

- Balance: We combine two paths if the ratio of the lengths of the two paths is as close to 1 as possible.
- Balance, Smart, Linear are incomparable wrto the number of derivations.

Comparison

- Linear: 5=4+1; 4=3+1; 3=2+1; 2=1+1.
- Nonlinear: 5=4+1, 5=3+2, 5=2+3, 5=1+4; 4=3+1, 4=2+2, 4=1+3; 3=2+1, 3=1+2; 2=1+1.
- Smart: $5=2^2 +1=4+1$; $4=2^1 +2=2+2$; 2=1+1.
- Balance: 5=3+2; 4=2+2; 3=2+1; 2=1+1.

Linear: linear number of rounds
Nonlinear: unique decomposition property lacking

A program that needs to increase the arity

- Reachability:

```
reach(X) :- source(X)
reach(X) :- reach(Y) & arc(Y,X)
```

Reachability

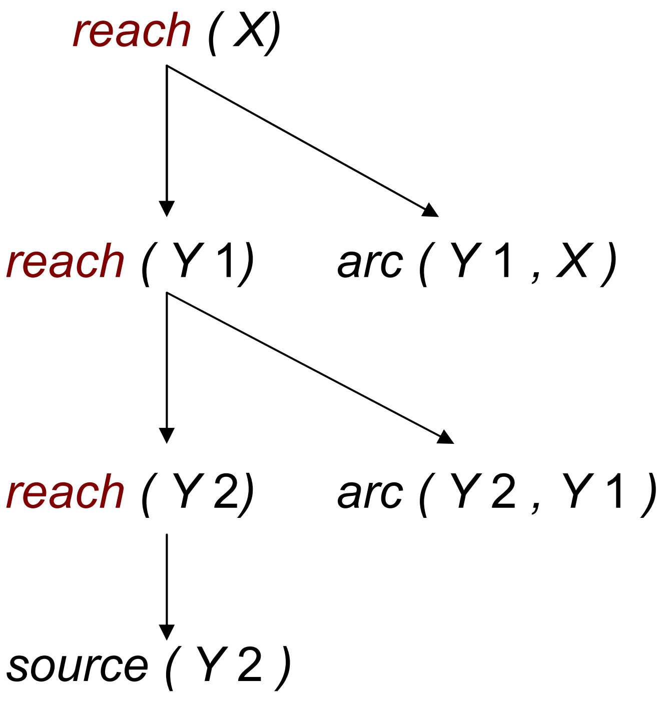
- As written the number of rounds equals longest path.
- You can have log n rounds but you have to implement TC.
- Can square the number of derived facts.
- E.g., reachability on the web is feasible, TC on the web is not feasible.

Proof trees

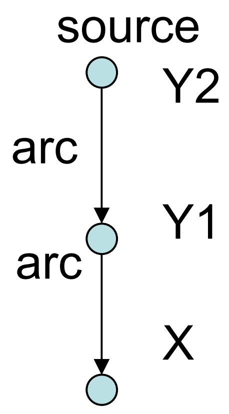
Proof tree

Input graph

reach(X) :- source(X)
reach(X) :- reach(Y) & arc(Y,X)



Derived predicate



Reachability

Theorem: *Suppose* Π *is a Datalog program for reachability with derived predicates of arity 1.*

Then there is a constant c > 0, *depending on* Π, *such that when* Π *is applied to relations arc and source, the height of any proof tree for the fact reach*(*a*) *is at least c times the length of the shortest path from the source node to a.*

Program transformation: iteration/arity tradeoff

Theorem: For every linear program Π *, there is an equivalent program* Π*' with the following properties:*

- *The maximum arity of IDB predicates in π' is twice the maximum arity of IDB predicates in* Π*.*
- *The program can be evaluated in O*(log *n*) *rounds on a database of size n.*

Polynomial fringe property (beyond linear recursion)

- These programs inlcude linear programs and are known to have algorithms in NC. It is a simple application of known techniques to prove that:

- When a program has the PFP then there is an equivalent program which can be executed in logarithmic number of rounds.

Open Problems (1)

- Find classes of linear recursions for which one can guarantee logarithmic rounds with no increase in the arity of recursive predicates.
- For any such class discovered in (1), are there unique-decomposition variants of the nonlinear recursion? Can we argue that these variants are comparable in communication cost to the original linear recursions?

Open Problems (2)

- Find general classes of linear recursions for which we can prove no equivalent recursion that completes in logarithmic rounds can use only predicates of the same arity as the linear recursion.
- It is reasonable to assume that when the arity of recursive predicates is increased, the number of facts deduced during the recursion grows significantly, and such is the case in the examples we have examined. Is this intuition correct in all cases?
- Note: a probabilistic single-source reachability algorithm which achieves a low number of derivations although the arity is equal to two.

Thank you