Declarative Networking

Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay,
Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan,
Timothy Roscoe, and Ion Stoica
CACM 52(11), Nov 2009

Presented By
Sandeep Kale

Agenda

- Datalog and Recursive Query Processing
- Motivation to Declarative Networking
- NDLog : Extensions to Datalog
- Execution Plan generation
- Incremental Maintenance
- Use cases
- Conclusion and References

Declarative Networking

- A declarative framework for networks:
  - Network protocols are declaratively specified using a database query language
  - Distributed query engine executes specifications to implement network protocols
- Success of database research:
  - 70's – today: Database research has revolutionized data management
  - Today: Similar opportunity to revolutionize the Internet architecture

Declarative Networking: Motivation

- Internet faces many challenges today:
  - Unwanted, harmful traffic
  - Complexity/fragility in Internet routing
  - Proliferation of new applications and services
- Efforts at improving the Internet:
  - Evolutionary: App-level "*Overlay*" networks
  - Revolutionary: Clean-slate designs
    - NSF GENI initiative, FIND program

Motivation

- Routing?
  - Gathering local network state : Base Data
  - Deriving forwarding tables : Derived Data
  - Enforcing constraints : Constraints

- Recursive Query processing
  - Using derived data again in query processing

Datalog

Introduction to datalog and recursive query processing

Datalog

- Declarative Paradigm
  - Give only what, and not how
  - Used extensively in SQL, functional programming
  - Easy and more natural to state

- Datalog
  - Non procedural query language
  - Based on Prolog

  - V1(A,B) :- account (A, “Mumbai”, B), B> 700
  - This gives account numbers and balances for accounts in Mumbai with balance greater than 700

Datalog

- A simple Datalog rule

p :- q1, q2, …, qn

  - “q1 and q2 and … and qn implies p.”

p : head
q1..qn : body(list of literals), literals are either predicates or functions,
Commas are Logical Conjunction (AND)

Datalog

- Program consists of rules
- Many rules can be used to define a relation
- Order of defining rules does not matter
- Predicates(relations) and matched (joined) to produce head

  - *interest_rate(A,10) :- account(A,N,B), B >= 10000*
  - *interest_rate(A,5) :-account(A,N,B) , B < 10000*
  - *interest(A,I) :- account(A, ”Mumbai”, B), interest_rate(A,R),I = B*I/100*

- This is an example of non-recursive datalog program

Datalog

- Recursive datalog program
- Consider a program to give all paths between nodes of a graph

  - *path(src,Dest,Path,Cost) :- link(src,Dest,Cost), path=f_init(src,Dest).*
  - *path(src,Dest,Path,Cost) :- link(src,nxt,Cost1), path(nxt,Dest,path2,Cost2), Cost=Cost1+Cost2, Path=f_concatpath(src,path2)*

- Recursive programs use fixpoint evaluation
  - Tuples produced in each iteration are added to the relation
  - Duplicate tuples are removed
  - Repeated until no new tuples are produces, i.e. *“fixpoint”* is reached

Semi Naïve Evaluation

- New tuples that are generated for the first time in the current iteration are then used in the next iteration.
- This is repeated until a fixpoint is achieved (i.e., no new tuples are produced).

$$\Delta \mathrm{p}_j^{new} :- p_1^{old}, \ldots, p_{k-1}^{old}, \Delta \mathrm{p}_k^{old}, p_{k+1}, \ldots, p_n, b_1, b_2, \ldots, b_m.$$

p1,...,pn are recursive predicates and b1,..are base predicates

Language

NDLog, Datalog extended for networking

Language

- Simple example to find path between two nodes
- Paths in fully connected network:

sp1 path(src,Dest,path,Cost) :-
link(src,Dest,Cost), path=f_init(src,Dest).

- To get paths in a general network, we need to add one more rule
- **sp2** path(src,Dest,path,Cost) :-
  link(src,nxt,Cost1),
  path(nxt,Dest,path2,Cost2),
  Cost=Cost1+Cost2,
  path=f_concatpath(src,path2),
  f_inPath(src,path2)=false

Language

- Aggregation
  - To get the cost of the shortest path we can use aggregation
  - min, max, avg all can be obtained in the same way

- **sp3** spCost(src,Dest,min<Cost>) :- path(src,Dest,path,Cost).

Language

- Location Specifiers (@)
  - Extension to datalog for networking
  - Specifies the location of data
  - Captures data partitioning
  - Enables distributed computations necessary for network protocols
  - Hides low level dataflow details

sp1 path(@src,Dest,path,Cost) :-
link(@src,Dest,Cost),
path=f_init(src,Dest).
sp2 path(@src,Dest,path,Cost) :-
link(@src,nxt,Cost1),
path(@nxt,Dest,path2,Cost2),
Cost=Cost1+Cost2,
path=f_concatpath(src,path2),
f_inPath(src,path2)=false

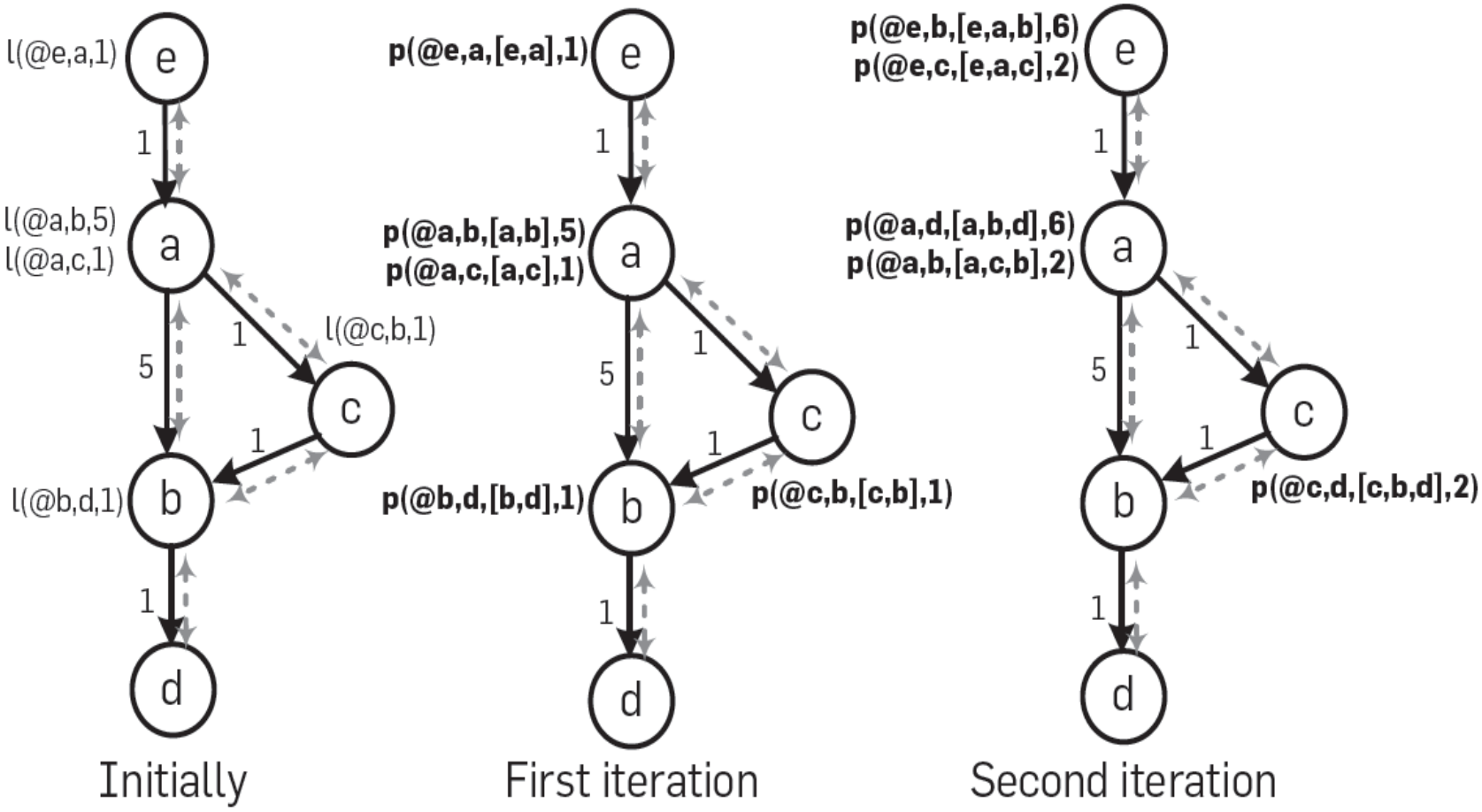
Language

- Link Restricted Rules
  - Models physical network where full connectivity is not available
  - One link predicate in each rule
  - There is exactly one link predicate in the body and rule is either..
    - Local rule or
    - All other predicates (including the head predicate)have their location specifier set to either the first (source) or second (destination) field of the link predicate.
  - p(@Dest,...):- link(@Src,Dest...),p1(@Src,...), …,pn(@Src,...).

Language

- **sp1** path(@src,Dest,path,Cost) :- link(@src,Dest,Cost), path=f_init(src,Dest).
- **sp2** path(@src,Dest,path,Cost) :- link(@src,nxt,Cost1), path(@nxt,Dest,path2,Cost2), Cost=Cost1+Cost2, path=f_concatpath(src,path2), f_inPath(src,path2)=false
- **sp3** spCost(@src,Dest,min<Cost>) :- path(@src,Dest,path,Cost).
- **sp4** shortestpath(@src,Dest,path,Cost) :- spCost(@src,Dest,Cost),path(@src,Dest,path,Cost).
- **Query** shortestpath(@src,Dest,path,Cost).

Language


l(@e,a,1)
e
1
l(@a,b,5)
l(@a,c,1)
a
5
1
l(@c,b,1)
c
1
l(@b,d,1)
b
1
d
Initially
p(@e,a,[e,a],1)
e
1
p(@a,b,[a,b],5)
p(@a,c,[a,c],1)
a
5
1
c
1
p(@b,d,[b,d],1)
b
p(@c,b,[c,b],1)
1
d
First iteration
p(@e,b,[e,a,b],6)
p(@e,c,[e,a,c],2)
e
1
p(@a,d,[a,b,d],6)
p(@a,b,[a,c,b],2)
a
5
1
c
1
b
p(@c,d,[c,b,d],2)
1
d
Second iteration

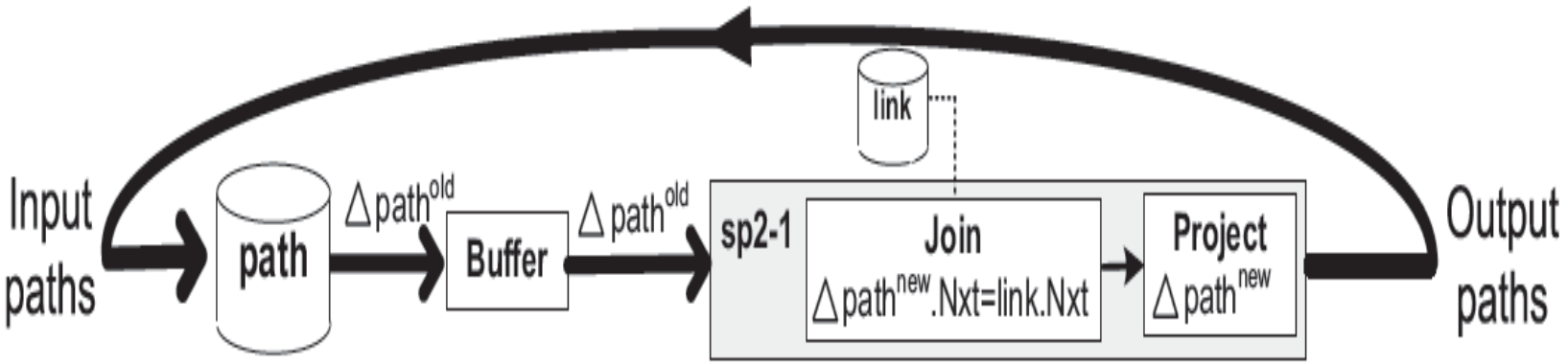
Execution Plan generation

Execution Plan Generation

- Network Protocols : State machines
- NDlog : Developed as Distributed dataflow execution engines (runtime similar to Parallel Database Systems)
- Recursive flows and the asynchronous communication difficult to handle
- Two types : Centralized / Distributed

Centralized Evaluation

sp2-1 $\Delta path^{new}$ (@Src, Dest,Path,Cost) :-
link(@Src,Nxt,Cost1),
$\Delta path^{old}$(@Nxt,Dest,Path2,Cost2),
Cost=Cost1+Cost2,
Path=*f_concatPath*(Src,Path2).



Centralized Evaluation

while $\exists B_k.size > 0$

$\forall B_k \; where \; B_k.size > 0, \Delta p_k^{old} \leftarrow B_k.flush()$

$execute \; all \; rule \; strands$

foreach $recursive \; predicate \; p_j$

$p_j^{old} \leftarrow p_j^{old} \cup \Delta p_j^{old}$

$B_j \leftarrow \Delta p_j^{new} - p_j^{old}$

$p_j \leftarrow p_j^{old} \cup B_j$

$\Delta p_j^{new} \leftarrow \phi$

Distributed Plan Generation

For Distributed implementation:

Nonlocal rules whose body predicates have different location Can not be executed as a single node

Tuples to be joined are situated at diff nodes

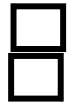
Rule localization rewrite ensures all tuples to be joined at same node

Localization Rewrite

- Rules may have body predicates at different locations:

  R2: path(@S,D,P) ← link(@S,Z), path(@Z,D,P2), P=S•P2.

Matching variable Z = “Join”



Rewritten rules:

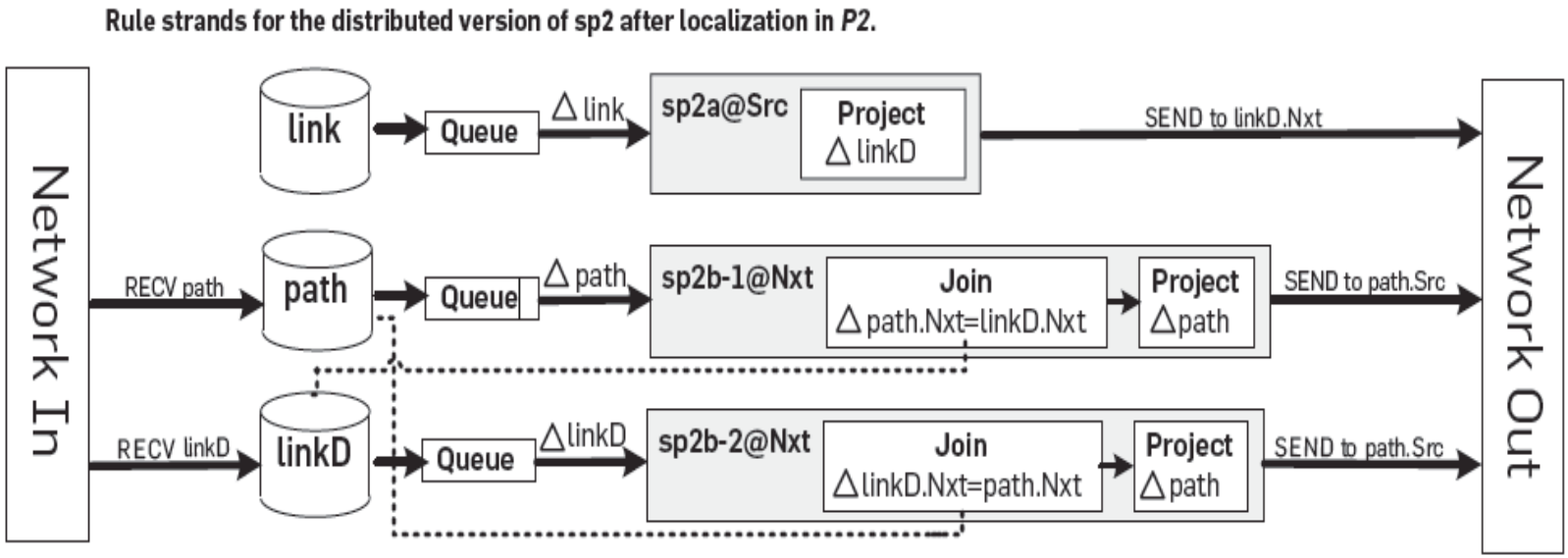
sp2a: linkD(S,@D) ← link(@S,D)

sp2b: path(@S,D,P) ← linkD(S,@Z), path(@Z,D,P2), P=S•P2.

Matching variable Z = “Join”

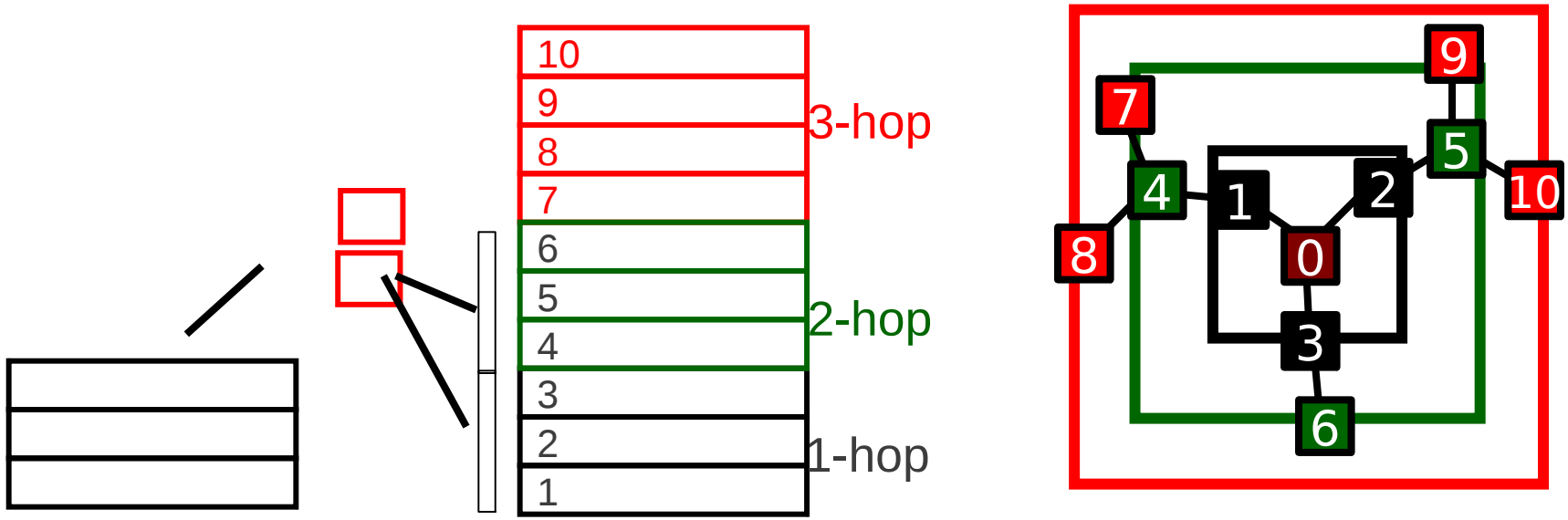
Distributed Plan Generation

- Rule sp2a@Src sends links to dest. add with as linkD tuples.
- Rule sp2b-2@Nxt takes linkD and performs join with path tuple.



Recursive Query Evaluation

- Semi-naïve evaluation:
  - Iterations (rounds) of synchronous computation
  - Results from iteration ith used in (i+1)th

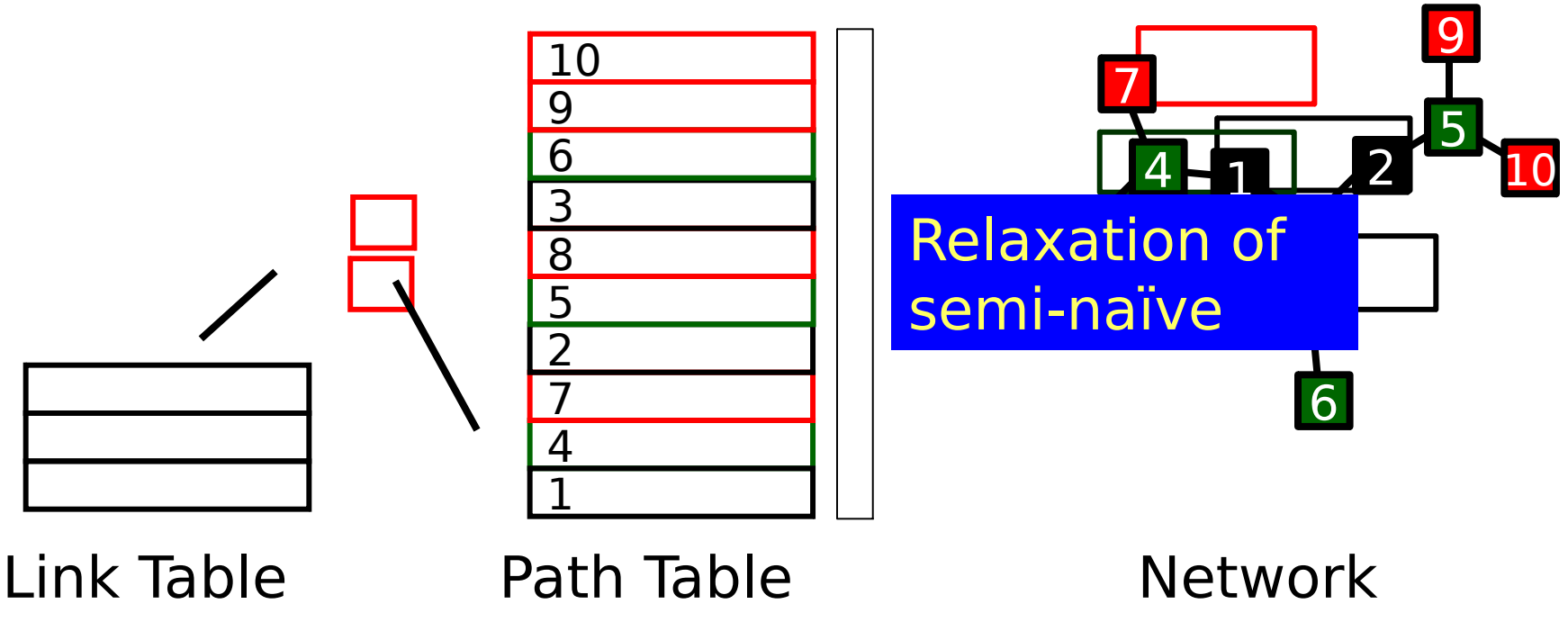


Slide Source: Reference 5

Problem: Unpredictable delays and failures

Pipelined Semi-naïve (PSN)

- Fully-asynchronous evaluation:
  - Computed tuples in *any* iteration pipelined to next iteration
  - Natural for distributed dataflows



Pipelined Semi-naïve (PSN)

- Node receiving new tuple can immediately process the tuple without waiting for local iteration to complete.

Algorithm Pipelined Semi-naïve (PSN) Evaluation

while $\exists\, Q_k.size > 0$

$t_k^{old,i} \leftarrow Q_k.dequeueTuple()$

foreach *rule strand execution*

$\Delta p_j^{new,i+1}$:—

$p_1,\ldots,p_{k-1}, t_k^{old,i}, p_{k+1},\ldots, p_n, b_1, b_2,\ldots, b_m$

foreach $t_j^{new,i+1} \in \Delta p_j^{new,i+1}$

if $t_j^{new,i+1} \notin p_j$

then $p_j \leftarrow p_j \cup t_j^{new,i+1}$

$Q_j.enqueueTuple(t_j^{new,i+1})$

Incremental Maintenance

Semantics in Dynamic Network

- Underlying Network Changes.
- Use Bursty update model for re-computations

- Three types of changes:
  - Insertion : Tuple can be inserted at any stage , handled by pipelined Evaluation.
  - Deletion : Deletion of base tuple leads to deletion of tuples derived from base tuple.
  - Update : Treated as deletion followed by insertion. Updating base tuple leads to more updates propagated further.

Semantics in Dynamic Network

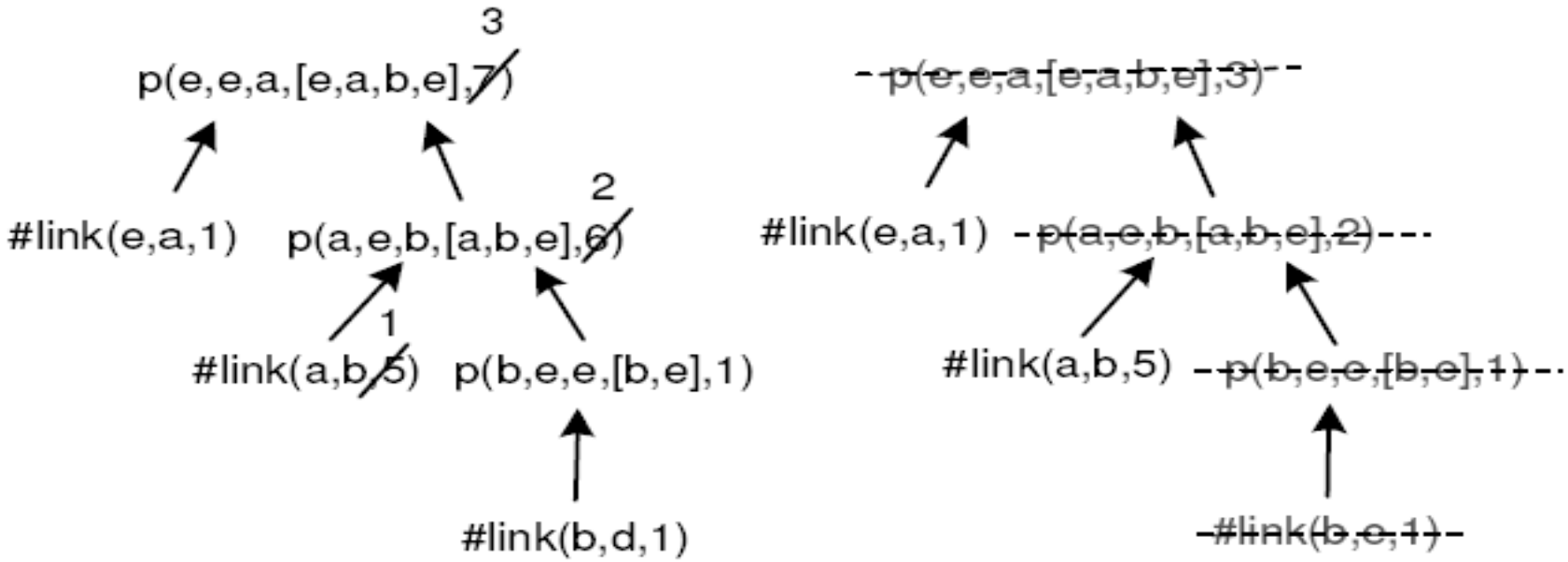


Figure : *Derivation tree for derived path tuple from a to e. The left diagram shows updating the tree due to a change in base tuple* $\#link(\mathbf{a}, b, 5)$*, and the right diagram shows the deletion of* $\#link(\mathbf{b}, e, 1)$.

Semantics in Dynamic network

- Use of Count Algorithm for multiple derivations
- Standard view maintenance technique for aggregates
- Ensures that tuples no longer derivable are deleted on deletes/updates.
- Example : Calculate 2 hop paths for given link set
  - Link={(a,b),(b,c),(b,e),(a,d),(d,c)}
  - Evaluates to {[(a,c)-2],[(a,e)-1]}
  - We keep count with both paths
  - If link (a,b) is deleted, algo uses stored count to reevaluate hop to {(a,c)}

Use Cases

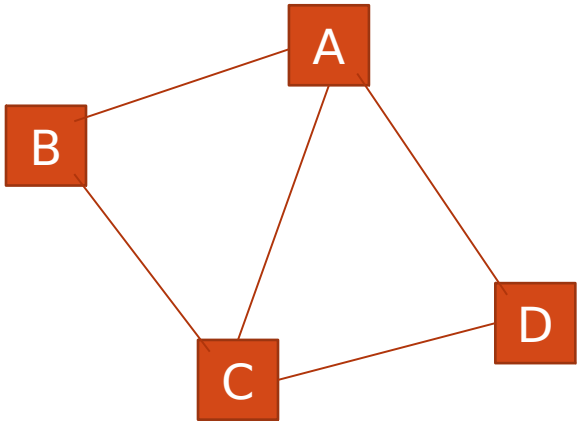
Use Cases of Declarative Networking

Declarative Routing

- Routing Protocol is implemented by writing query in NDlog
- Queries executed in distributed fashion on nodes
- Static analysis tests for termination of query in Ndlog (check for recursive defination and check for termination)
- NDlog can express variety of routing protocols such as distance vector, path vector ,dynamic source routing easily

Distance Vector Routing

- At each node, save next hop each destination with the cost
- Nodes exchange this information to get knowledge about complete network state



	A	C	D
1	1, A	1, C	--
2	1, A	1, C	2, A

	B	C	D
1	1, B	1, C	1, B

Distance Vector Routing

- DV1: path(@S,D,D,C) :- link(@S,D,C).
- DV2: path(@S,D,Z,C) :- link(@S,Z,C1),
- path(@Z,D,W,C2), C = C1 + C2
- DV3: shortestCost(@S,D,min<C>) :- path(@S,D,Z,C).
- DV4: nextHop(@S,D,Z,C) :- path(@S,D,Z,C), shortestCost(@S,D,C).
- Query: nextHop(S,D,Z,C).

Distance Vector Routing

- DV1: path(@S,D,D,C) :- link(@S,D,C).
- DV2: path(@S,D,Z,C) :- link(@S,Z,C1),
- path(@Z,D,W,C2), C = C1 + C2
- DV3: shortestCost(@S,D,min<C>) :- path(@S,D,Z,C).
- DV4: nextHop(@S,D,Z,C) :- path(@S,D,Z,C), shortestCost(@S,D,C).
- Query: nextHop(S,D,Z,C).

- Count to Infinity problem?
- DV2: path(@S,D,Z,C) :- link(@S,Z,C1),
- path(@Z,D,W,C2), C = C1 + C2, W != S
- DV5: path(@S,D,Z,∞) :- link(@S,Z,C1), path(@Z,D,S,C2)

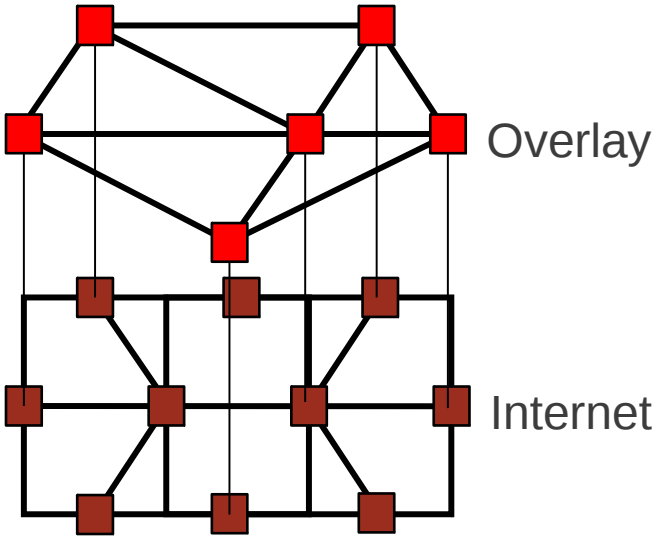
Policy Based Routing

- Some of the policies
  - Node Utilization
  - Link Utilization
  - Security issues
  - Transit- Peer relationships

- PBR1: invalidPath(@S,D,P,C) :- path(@S,D,P,C), excludeNode(@S,W), f_inPath(P,W) = true.
- PBR2: permitPath(@S,D,P,C) :- path(@S,D,P,C), not invalidPath(@S,D,P,C)
- Query: permitPath(@S,D,P,C).
- excludeNode(@S,W) : this says that node W can not carry any traffic for node S

Declarative Overlays

- **Overlay network** : Virtual nodes built on top of internet
- To provide services that are not available in existing network.



- Overlog declarative language an extension on NDlog.
- Soft-state is introduced on Overlog

Soft State

- Stored data has associated Time-to-live(TTL)
- Soft state dataum needs to be periodically refreshed
- If more time than TTL passes without datum being refreshed, that datum is deleted.
- Soft State is favored in networking implementations since it provides eventual consistency.
- Eventual values are obtained in case of transient errors such as reordered messages, node disconnection.
- In case of persistent failure no coordination is required, since any data provided by failed node would be forgotten.

Soft State

- In Overlog use materialized keyword
  - materialized(link, {1,2}, 10) , this specifies link tuple has a life time of ten seconds

- If TTL set to infinity : Hard State
- If predicate has no materialized keyword it is treated as event predicate (TTL =0)
- Event predicates are transient tables and use to trigger rules periodically or case of network events
- ping(@Y, X) :- periodic(@X, 10) , link(@X, Y)

Conclusion

- Ease of programming:
  - Compact and high-level representation of protocols
  - Orders of magnitude reduction in code size
  - Easy customization
- Safety:
  - Queries are “sandboxed” within query processor
  - Potential for static analysis techniques on safety
- What about efficiency?
  - No fundamental overhead when executing standard routing protocols
  - Application of well-studied query optimizations

References

Declarative Networking, Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica

1. CACM 52(11), Nov 2009
2. Declarative Networking: Language, Execution and Optimization Boon Thau Loo , Tyson Condie , Minos Garofalakis , David E. Gay , Joseph M. Hellerstein , Petros Maniatis , Raghu Ramakrishnan , Timothy Roscoe , Ion Stoica, SIGMOD-2006
3. Declarative Routing:Extensible Routing with Declarative Queries, Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, Raghu Ramakrishnan, SIGCOMM-2005
4. Database Systems and Concepts, Avi Silberschatz, Henry F. Korth, S. Sudarshan, 4th Edition
5. Design and Implementaion of Declarative Networks, Boon Thau Loo, powerpoint presentation, available online, URL : *www.cis.upenn.edu/~boonloo/research/talks/dn-sigmod07.**ppt***

Thank you