DECLARATIVE NETWORKING:

WHAT IS NEXT

JOSEPH M. HELLERSTEIN

UC BERKELEY
DECLARATIVE NETWORKING: WHAT IS NEXT.

JOSEPH M. HELLERSTEIN

UC BERKELEY
JOINT WORK

- Ashima ATUL Kuang CHEN David CHU Tyson CONDIE Lucian POPA Arsalan TAVAKOLI Scott SHENKER Ion STOICA Berkeley
- Boon Thau LOO UPenn
- David GAY Petros MANIATIS intel
- Timothy ROSCOE ETH Zurich
- Raghu RAMAKRISHNAN Minos GAROFALAKIS Yahoo!
- Stanislav FUNIAK Carlos GUESTRIN CMU
- Philip LEVIS Stanford
TWO SOURCES OF FLUX
TWO SOURCES OF FLUX

internet evolution revolution
TWO SOURCES OF FLUX

internet evolution revolution
GENI, wireless sensors, overlay nets, datacenters
TWO SOURCES OF FLUX

- internet evolution
- GENI, wireless sensors, overlay nets, datacenters

- industrial revolution of data
HOW DO WE HARNESS THE FLUX?
IN THIS TIME OF FLUX, WHAT CAN HARNES
AND ACCELERATE THE ENERGY AND INNOVATION.
WHAT IS THE FUTURE: DECLARATIVE NETWORKING
WHAT IS THE FUTURE: DECLARATIVE NETWORKING

- **WHAT:** beyond the network *how*
  - topology specification. routing constraints. addressing by content
WHAT IS THE FUTURE:
DECLARATIVE NETWORKING

▪ **WHAT:** beyond the network *how*
  - topology specification. routing constraints. addressing by content

▪ **who** where **why** when
  - authentication. geolocation. consensus. forensics.
WHAT IS THE FUTURE: DECLARATIVE NETWORKING

- **WHAT**: beyond the network *how*
  - topology specification. routing constraints. addressing by content

- **who** where **why** when
  - authentication. geolocation. consensus. forensics.

- **NW data**, reasoning and **control**
  - search. query. inference. movement.
THE EVOLUTION OF WHAT

query (in) the network
networks VIA queries
queries, networks and uncertainty

SYNTHESIS
• WHY WHAT?
• SAY WHAT
• WHAT: HOW
• WHAT FOR
• WHAT IS NEXT?
• WHAT’S IT TO YOU
WHY WHAT?

SAY WHAT

WHAT: HOW

WHAT FOR

WHAT IS NEXT?

WHAT’S IT TO YOU
WHY WHAT?
ease, insight:

- rapid prototyping & customization
- fitness to many distributed tasks
- uplevel ideas and their synergies
WHY WHAT?

- ease, insight:
  - rapid prototyping & customization
  - fitness to many distributed tasks
  - uplevel ideas and their synergies

- towards optimization & safety
  - state management and rendezvous
  - static and dynamic checks
WHAT FIRST

- textbook routing protocols
  - internet-style and wireless  SIGCOMM 05, Berkeley/Wisconsin
- distributed hash tables
  - chord overlay network  SOSP 05, Berkeley/Intel
- distributed debugging
  - watchpoints, snapshots  EuroSys 06, Intel/Rice/MPI
- consensus
  - paxos  44 lines, Harvard 2006
textbook routing protocols
  - internet-style and wireless  SIGCOMM 05, Berkeley/Wisconsin

distributed hash tables
  - chord overlay network  SOSP 05, Berkeley/Intel

distributed debugging
  - watchpoints, snapshots  EuroSys 06, Intel/Rice/MPI

consensus
  - paxos  44 lines, Harvard 2006
**WHAT NOW**

- **wireless sensornets**
  - radio link estimation. geo routing. data collection. code dissemination. object tracking. localization.
  - SenSys 07, Berkeley

- **secure networking**
  - SeNDLog. NetDB07, MSR/Penn

- **flexible data replication**
  - PADRE
  - SOSP07 poster, Texas

- **mobile networks**
  - MobiArch07, Penn

- **modular robotics**
  - MELD IROS 07, CMU
WHAT NEXT

- metacompilation
  - declarative compilers for declarative languages Berkeley/Intel

- distributed inference
  - junction trees, belief propagation Berkeley/CMU

- automatic optimization
  - protocol synthesis for state and rendezvous Berkeley
WHY WHAT?
SAY WHAT
WHAT: HOW
WHAT FOR
WHAT IS NEXT?
WHAT’S IT TO YOU
TODAY

● WHY WHAT?
● SAY WHAT
● WHAT: HOW
● WHAT FOR
● WHAT IS NEXT?
● WHAT’S IT TO YOU
P2 @ 10,000 FEET
parent(X,Y).

anc(X,Y) :- parent(X,Y).

anc(X,Z) :- parent(X,Y),
            anc(Y,Z).

anc(X,s)
parent(X,Y).

anc(X,Y) :- parent(X,Y).

anc(X,Z) :- parent(X,Y), anc(Y,Z).

anc(X, s)?
THE INTERNET CHANGES EVERYTHING?

\[\text{link}(X,Y).\]

\[\text{path}(X,Y) :\text{- link}(X,Y).\]

\[\text{path}(X,Z) :\text{- link}(X,Y), \text{ path}(Y,Z).\]

\[\text{path}(X, s)?\]
FORMING PATHS

- \texttt{link}(X,Y,C)
- \texttt{path}(X,Y,Y,C) :- \texttt{link}(X,Y,C)
- \texttt{path}(X,Z,Y,C+D) :- \texttt{link}(X,Y,C), \texttt{path}(Y,Z,N,D)
FORMING PATHS

link(X,Y,C)

path(X,Y,Y,C) :- link(X,Y,C)

path(X,Z,Y,C+D)
    :- link(X,Y,C), path(Y,Z,N,D)
FORMING PATHS

- \text{link}(X,Y,C) \leftarrow \text{COST}
- \text{path}(X,Y,Y,C) :\text{link}(X,Y,C)
- \text{path}(X,Z,Y,C+D)
  :\text{link}(X,Y,C), \text{path}(Y,Z,N,D)
link(X, Y, C)

path(X, Y, Y, C) :- link(X, Y, C)

path(X, Z, Y, C + D)
  :- link(X, Y, C), path(Y, Z, N, D)
FORMING PATHS

- \texttt{link}(X,Y,C)
- \texttt{path}(X,Y,Z,C) :- \texttt{link}(X,Y,C)
- \texttt{path}(X,Z,Y,C+D) :- \texttt{link}(X,Y,C), \texttt{path}(Y,Z,N,D)

\textit{Next Hop}
link(X,Y,C)

path(X,Y,Y,C) :- link(X,Y,C)

path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)
FORMING PATHS

- `link(X, Y, C)`
- `path(X, Y, Y, C) :- link(X, Y, C)`
- `path(X, Z, Y, C+D) :- link(X, Y, C), path(Y, Z, N, D)`
FORMING PATHS

link(X, Y, C)

path(X, Y, Y, C) :- link(X, Y, C)

path(X, Z, Y, C + D) :- link(X, Y, C), path(Y, Z, N, D)

COST
FORMING PATHS

\textbf{link}(X,Y,C)

\textbf{path}(X,Y,Y,C) :- \textbf{link}(X,Y,C)

\textbf{path}(X,Z,Y,C+D) :- \textbf{link}(X,Y,C), \textbf{path}(Y,Z,N,D)
FORMING PATHS

\texttt{link(X,Y,C)}

\texttt{path(X,Y,Y,C) :- link(X,Y,C)}

\texttt{path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)}
link(X,Y)

BEST PATHS
link(X,Y)

path(X,Y,Y,C) :- link(X,Y,C)
BEST PATHS

\[ \text{link}(X,Y) \]

\[ \text{path}(X, Y, Y, C) :\text{-} \text{link}(X, Y, C) \]

\[ \text{path}(X, Z, Y, C+D) :\text{-} \text{link}(X, Y, C), \text{path}(Y, Z, N, D) \]
BEST PATHS

link(X,Y)

path(X,Y,Y,C) :- link(X,Y,C)

path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)

mincost(X,Z,min<C>) :- path(X,Z,Y,C)
link(X,Y)

path(X,Y,Y,C) :- link(X,Y,C)

path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)

mincost(X,Z,min<C>) :- path(X,Z,Y,C)

bestpath(X,Z,Y,C) :- path(X,Z,Y,C), mincost(X,Z,C)
link(X,Y)

path(X,Y,Y,C) :- link(X,Y,C)

path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)

mincost(X,Z,min<C>) :- path(X,Z,Y,C)

bestpath(X,Z,Y,C) :- path(X,Z,Y,C), mincost(X,Z,C)

bestpath(src,D,Y,C)?
SO FAR...

- logic for path-finding
- on the link DB in the sky

- but can this lead to protocols?
TOWARD DISTRIBUTION:
DATA PARTITIONING

- logically global tables
- horizontally partitioned
- an address field per table
  - location specifier: @
- data placement based on loc.spec.
PARTITION SPECS INDUCE COMMUNICATION

\[ \text{link}(\text{X}, \text{Y}, \text{C}) \]
\[ \text{path}(\text{X}, \text{Y}, \text{Y}, \text{C}) \leftarrow \text{link}(\text{X}, \text{Y}, \text{C}) \]
\[ \text{path}(\text{X}, \text{Z}, \text{Y}, \text{C} + \text{D}) \leftarrow \text{link}(\text{X}, \text{Y}, \text{C}), \text{path}(\text{Y}, \text{Z}, \text{N}, \text{D}) \]
\textbf{PARTITION SPECS INDUCE COMMUNICATION}

\begin{align*}
\text{link}(@X,Y,C) \\
\text{path}(@X,Y,Y,C) & : \text{link}(@X,Y,C) \\
\text{path}(@X,Z,Y,C+D) & : \text{link}(@X,Y,C), \text{path}(@Y,Z,N,D)
\end{align*}
PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y,C)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link(@X,Y,C), path(@Y,Z,N,D)
PARTITION SPECS INDUCE COMMUNICATION

\[ \text{link}(\text{@x}, \text{y}, \text{c}) \]
\[ \text{path}(\text{@x}, \text{y}, \text{y}, \text{c}) :- \text{link}(\text{@x}, \text{y}, \text{c}) \]
\[ \text{path}(\text{@x}, \text{z}, \text{y}, \text{c+d}) :- \text{link}(\text{@x}, \text{y}, \text{c}), \text{path}(\text{@y}, \text{z}, \text{n}, \text{d}) \]
PARTITION SPECS INDUCE COMMUNICATION

\[ \text{link}(@X,Y,C) \]

\[ \text{path}(@X,Y,Y,C) : - \text{link}(@X,Y,C) \]

\[ \text{path}(@X,Z,Y,C+D) : - \text{link}(@X,Y,C), \text{path}(@Y,Z,N,D) \]
link(@X,Y,C)

path(@X,Y,Y,C) :- link(@X,Y,C)

path(@X,Z,Y,C+D) :- link(@X,Y,C), path(@Y,Z,N,D)
PARTITION SPECS INDUCE COMMUNICATION

\[ \text{link}(\alpha X, \alpha Y, \alpha C) \]

\[ \text{path}(\alpha X, \alpha Y, \alpha Y, \alpha C) :\sim \text{link}(\alpha X, \alpha Y, \alpha C) \]

\[ \text{path}(\alpha X, \alpha Z, \alpha Y, \alpha C + \alpha D) :\sim \text{link}(\alpha X, \alpha Y, \alpha C), \text{path}(\alpha Y, \alpha Z, \alpha N, \alpha D) \]
PARTITION SPECS INDUCE COMMUNICATION

\[\text{link}(\text{@X}, \text{Y}, \text{C})\]

\[\text{path}(\text{@X}, \text{Y}, \text{Y}, \text{C}) :\text{ link}(\text{@X}, \text{Y}, \text{C})\]

\[\text{path}(\text{@X}, \text{Z}, \text{Y}, \text{C}+\text{D}) :\text{ link}(\text{@X}, \text{Y}, \text{C}), \text{path}(\text{@Y}, \text{Z}, \text{N}, \text{D})\]
PARTITION SPECS INDUCE COMMUNICATION

link(@X,Y)

path(@X,Y,Y,C) :- link(@X,Y,C)

link_d(X,@Y,C) :- link(@X,Y,C)

path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)

Localization Rewrite

link_d:

path:

link:
PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- link_d(X,@Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)

Localization Rewrite
PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- link_d(X,@Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)

Localization Rewrite
PARTITION SPECS INDUCE COMMUNICATION

\[ \text{link}(X,Y) \]
\[ \text{path}(X,Y,Z,C) :- \text{link}(X,Y,C) \]
\[ \text{link}_d(X,Y,C) :- \text{link}(X,Y,C) \]
\[ \text{path}(X,Z,Y,C+D) :- \text{link}_d(X,Y,C), \text{path}(Y,Z,N,D) \]

Localization Rewrite
link(@X,Y)

path(@X,Y,Y,C) :- link(@X,Y,C)

link_d(X,@Y,C) :- link(@X,Y,C)

path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)
PARTITION SPECS INDUCE COMMUNICATION

link(@X,Y)

path(@X,Y,Y,C) :- link(@X,Y,C)

link_d(X,@Y,C) :- link(@X,Y,C)

path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)
PARTITION SPECS INDUCE COMMUNICATION

\[\text{link}(\text{@X,Y})\]
\[\text{path}(\text{@X,Y,Y,C}) :\text{ link}(\text{@X,Y,C})\]
\[\text{link}_{-d}(\text{X,}@\text{Y,C}) :\text{ link}(\text{@X,Y,C})\]
\[\text{path}(\text{@X,Z,Y,C+D}) :\text{ link}_{-d}(\text{X,}@\text{Y,C}), \text{path}(\text{@Y,Z,N,D})\]
PARTITION SPECS INDUCE COMMUNICATION

\[
\begin{align*}
\text{link}(@X,Y) \\
\text{path}(@X,Y,Y,C) & \text{: link}(@X,Y,C) \\
\text{link}_d(X,@Y,C) & \text{: link}(@X,Y,C) \\
\text{path}(@X,Z,Y,C+D) & \text{: link}_d(X,@Y,C), \text{path}(@Y,Z,N,D)
\end{align*}
\]
TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT FOR
- WHAT IS NEXT?
- WHAT’S IT TO YOU
TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT FOR
- WHAT IS NEXT?
- WHAT’S IT TO YOU
P2 @ 10,000 FEET
L1 lookupResults(@R,K,S,SI,E) :- node(@NI,N), lookup(@NI,K,R,E),
    bestSucc(@NI,S,SI),
    K in (N, S].

L2 bestLookupDist(@NI,K,R,E,min<D>) :- node(@NI,N),
    lookup(@NI,K,R,E),
    finger(@NI,I,B,BI),
    D:=K-B-1, B in (N,K)

L3 lookup(@min<BI>,K,R,E) :- node(@NI,N),
    bestLookupDist(@NI,K,R,E,D),
    finger(@NI,I,B,BI), D==K-B-1, B in (N,K).
DATAFLOW EXAMPLE IN P2
DATAFLOW EXAMPLE IN P2

```
L1
Join lookup.NI == node.NI
TimedPullPush 0
Join lookup.NI == bestSucc.NI
Select K in (N, S)
Project lookupRes

L2
Join lookup.NI == node.NI
TimedPullPush 0
Agg min<D> on finger
D := K-B-1, B in (N,K)

L3
Join bestLookupDist.NI == node.NI
TimedPullPush 0
Agg min<BI> on finger
D := K-B-1, B in (N,K)

Materializations
Insert node
Insert bestSucc
Insert finger

Network In
Demux (@local?)
Queue
TimedPullPush 0

Network Out
Queue
TimedPullPush 0
RoundRobin
```
DATAFLOW EXAMPLE IN P2
macro _TCP(ip, port) {
    let q := Queue("Source Data Q", size);
    let udp := Udp("Test Udp", port);
    let cct := CCT("Congestion Control Transmit", 1, 2048);
    let ccr := CCR("Congestion Control Receive", 2048);
    let order := Order("Ordered delivery");

    input q; output order;

    q[0] -> Route(ip) -> Sequence("Sequence", 1) -> RDelivery("Reliable Delivery") ->
    cct -> [1]MarshalField("marshal data", 1)[0] -> udp;

    udp -> UnmarshalField("Unpack", 1) -> ccr -> order;
}

let b := TimedPushSource("Data Generator", .01);
let tcp := _TCP(129.0.0.1, 80);

b[0] -> tcp -> Queue("Q", 1);
TODAY

• WHY WHAT?
• SAY WHAT
• WHAT: HOW
• WHAT FOR
• WHAT IS NEXT?
• WHAT’S IT TO YOU
TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT FOR
- WHAT IS NEXT?
- WHAT’S IT TO YOU
DSN-TRICKLE

Levis, et al., Sensys 2004

Chu, et al., Sensys 2007

Event | Action
---|---
\(\tau\) Expires | Double \(\tau\), up to \(\tau_H\). Reset \(c\), pick a new \(T\).
t Expires | If \(c < k\), transmit.
Receive same metadata | Increment \(c\).
Receive newer metadata | Set \(\tau\) to \(\tau_L\). Reset \(c\), pick a new \(T\).
Receive newer code | Set \(\tau\) to \(\tau_L\). Reset \(c\), pick a new \(T\).
Receive older metadata | Send updates.

\(t\) is picked from the range \([\frac{\tau}{2}, \tau]\)

Figure 12: Trickle Pseudocode.

Listing 3. Trickle Version Coherency
DSN-TRICKLE

Levis, et al., Sensys 2004

Chu, et al., Sensys 2007

Figure 12: Trickle Pseudocode.

Listing 3. Trickle Version Coherency
# DSN vs NATIVE TRICKLE

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>DSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>560 (NesC)</td>
<td>13 rules, 25 lines</td>
</tr>
<tr>
<td>Code Sz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.3KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Sz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.4KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph showing comparison between DSN and native trickle](image-url)
# DSN vs NATIVE TRICKLE

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>DSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>560 (NesC)</td>
<td>13 rules, 25 lines</td>
</tr>
<tr>
<td>Code Sz</td>
<td>12.3KB</td>
<td>24.4KB</td>
</tr>
<tr>
<td>Data Sz</td>
<td>0.4KB</td>
<td>4.1KB</td>
</tr>
</tbody>
</table>
P2-CHORD

- chord *distributed hash table*
  - Internet overlay for content-based routing
- high-function implementation
  - multiple successors
  - stabilization
  - optimized finger maintenance
  - failure detection
- 48 rules
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succCount, infinity, 1, keys(1)).
materialize(landmark, infinity, 1, keys(1)).
materialize(fFix, infinity, 160, keys(2)).
materialize(nextFingerFix, infinity, 1, keys(1)).
materialize(pingNode, 10, infinity, keys(2)).
materialize(pendingPing, 10, infinity, keys(2)).

/** Lookups */
watch(lookupResults).
watch(lookup).
l1 lookupResults@R(R,K,S,SI,E) :- node@NI(NI,N),
lookup@NI(NI,K,R,E), bestSucc@NI(NI,S,SI),
K in (N,S].
l2 bestLookupDist@NI(NI,K,R,E,min<D>) :- node@NI(NI,N),
lookup@NI(NI,K,R,E), finger@NI(NI,I,B,BI),
D:=K - B - 1, B in (N,K).
l3 lookup@BI(min<BI>,K,R,E) :- node@NI(NI,N),
bestLookupDist@NI(NI,K,R,E,D),
finger@NI(NI,I,B,BI), D == K - B - 1,
B in (N,K).

/** Neighbor Selection */
succEvent@NI(NI,S,SI) :- succ@NI(NI,S,SI).
succDist@NI(NI,S,D) :- node@NI(NI,N),
succEvent@NI(NI,S,SI), D:=S - N - 1.
bestSucc@NI(NI,S,SI) :- succ@NI(NI,S,SI),
bestSuccDist@NI(NI,D), node@NI(NI,N),
D == S - N - 1.
finger@NI(NI,0,S,SI) :- bestSucc@NI(NI,S,SI).

/** Successor eviction */
succCount@NI(NI,count<*>) :- succ@NI(NI,S,SI).
evictSucc@NI(NI,S,SI) :- succ@NI(NI,S,SI),
maxSuccDist@NI(NI,D), node@NI(NI,N),
evictSucc@NI(NI,N), succ@NI(NI,S,SI),
D = S - N - 1.

/** Finger fixing */
fFix@NI(NI,E,I) :- periodic@NI(NI,E,10),
nextFingerFix@NI(NI,I).

/** Churn Handling */
joinEvent@NI(NI,E) :- join@NI(NI,E).
joinReq@LI(LI,N,NI,E) :- joinEvent@NI(NI,E),
node@NI(NI,N), landmark@NI(NI,LI), LI != "-".
succ@NI(NI,N,NI) :- landmark@NI(NI,LI),
joinEvent@NI(NI,E), node@NI(NI,N), LI == "-".
lookup@LI(LI,N,NI,E) :- joinEvent@NI(LI,N,NI,E).
succ@NI(NI,S,SI) :- join@NI(NI,E),
lookupResults@NI(NI,K,S,SI,E).

/** Stabilization */
stab@NI(NI,E) :- periodic@NI(NI,E,15).
stabRequest@SI(SI,NI) :- stab@NI(NI,E),
bestSucc@NI(NI,S,SI).
sendPredecessor@PI1(PI1,P,PI) :- stabRequest@NI(NI,PI1),
pred@NI(NI,P,PI), PI != "-".
succ@NI(NI,P,PI) :- node@NI(NI,N),
sendPredecessor@NI(NI,P,PI),
bestSucc@NI(NI,S,SI), P in (N,S).
s succeeds@SI(SI,NI) :- stab@NI(NI,E),
succ@NI(NI,N,SI).
s succ@NI(NI,S,SI) :- succ@NI(NI,S,SI),
returnSuccessor@PI(PI,S,SI) :- sendSuccessors@SI(SI,NI),
succ@NI(NI,S,SI).
s succ@NI(NI,S,SI) :- succ@NI(NI,S,SI),
notifyPredecessor@SI(SI,NI) :- stab@NI(NI,E),
succ@NI(NI,S,SI).
s succ@NI(NI,P,PI) :- node@NI(NI,N),
notifyPredecessor@NI(NI,P,PI),
pred@NI(NI,P,PI), (PI == "-" || (PI in (P,NI))).

/** Connectivity Monitoring */
pongEvent@NI(NI,E) :- periodic@NI(NI,E,5).
pendingPong@NI(NI,P,PI,E) :- pongEvent@NI(NI,E),
pingNode@NI(NI,P,PI).
pingReq@PI1(PI1,P,PI) :- pendingPong@NI(NI,P,PI,E),
pingResp@NI(NI,P,PI), P in (N,P).
pingResp@PI1(PI1,P,PI) :- pendingPong@NI(NI,P,PI,E),
pingNode@NI(NI,P,PI),
pingResp@NI(NI,P,PI), P in (P,NI).
pongNode@NI(NI,SI) :- succ@NI(NI,S,SI),
notifyPredecessor@NI(NI,SI),
node@NI(NI,N), succ@NI(NI,S,SI).

I:=11 + 1, K:=II << I + N, K in (B,N),
NI != BI.
/* The base tuples */
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity, 1, keys(1)).
materialize(landmark, infinity, 1, keys(1)).
materialize(fFix, infinity, 160, keys(2)).
materialize(nextFingerFix, infinity, 1, keys(1)).
materialize(pingNode, 10, infinity, keys(2)).
materialize(pendingPing, 10, infinity, keys(2)).
/** Lookups */
watch(lookupResults).
watch(lookup).
l1 lookupResults@R(R,K,S,SI,E) :- node@NI(NI,N),
lookup@NI(NI,K,R,E), bestSucc@NI(NI,S,SI),
K in (N,S].
l2 bestLookupDist@NI(NI,K,R,E,min<D>) :- node@NI(NI,N),
lookup@NI(NI,K,R,E), finger@NI(NI,I,B,BI),
D:=K - B - 1, B in (N,K).
l3 lookup@BI(min<BI>,K,R,E) :- node@NI(NI,N),
bestLookupDist@NI(NI,K,R,E,D),
finger@NI(NI,I,B,BI), D == K - B - 1,
B in (N,K).
/** Neighbor Selection */
n1 succEvent@NI(NI,S,SI) :- succ@NI(NI,S,SI).
n2 succDist@NI(NI,S,D) := node@NI(NI,N),
succEvent@NI(NI,S,SI), D:=S - N - 1.
n3 bestSucc@NI(NI,S,SI) := succ@NI(NI,S,SI),
bestSuccDist@NI(NI,D), node@NI(NI,N),
D == S - N - 1.
n5 finger@NI(NI,0,S,SI) := bestSucc@NI(NI,S,SI).
/** Successor eviction */
s1 succCount@NI(NI,count<*>) := succ@NI(NI,N),
succCount@NI(NI,C), C > 2.
s2 maxSuccDist@NI(NI,max<D>) := succ@NI(NI,N),
node@NI(NI,N),
evictSucc@NI(NI), D:=S - N - 1.
s4 delete succ@NI(NI,N),
succCount@NI(NI,N),,
succCount@NI(NI,C),
maxSuccDist@NI(NI,D),
D == S - N - 1.
/** Finger fixing */
f1 fFix@NI(NI,E,I) := periodic@NI(NI,E,I),
nextFingerFix@NI(NI,I).
f2 fFixEvent@NI(NI,E,I) := fFixEvent@NI(NI,E,I),
node@NI(NI,N),
fFix@NI(NI,E,I).
f3 lookup@NI(K,R,E) := fFixEvent@NI(NI,E,I),
node@NI(NI,N),
K:=I + 1, K:=I + N.
f4 eagerFinger@NI(NI,I,B,BI) := fFix@NI(NI,E,I),
lookupResults@NI(NI,K,B,BI),
eagerFinger@NI(NI,K,B,BI).
f5 finger@NI(NI,I,B,BI) := eagerFinger@NI(NI,I,B,BI).
f6 eagerFinger@NI(NI,I,B,BI) := node@NI(NI,N),
eagerFinger@NI(NI,I,B,BI),
eagerFinger@NI(NI,I,B,BI),
eagerFinger@NI(NI,I,B,BI),
eagerFinger@NI(NI,I,B,BI),
I:=I + 1, K:=I + N.
f7 delete fFix@NI(NI,E,I) := fFix@NI(NI,E,I),
nextFingerFix@NI(NI,E,I),
K:=I + 1, K:=I + N.
f8 nextFingerFix@NI(NI,0) := eagerFinger@NI(NI,0,1,B),
nextFingerFix@NI(NI,0,1,B),
nextFingerFix@NI(NI,0,1,B).
P2-CHORD EVALUATION

- P2 nodes running Chord on 100 Emulab nodes:
  - Logarithmic lookup hop-count and state ("correct")
  - Median lookup latency: 1-1.5s
  - BW-efficient: 300 bytes/s/node
CHURN PERFORMANCE

- **P2-Chord:**
  - P2-Chord@90mins: 99% consistency
  - P2-Chord@47mins: 96% consistency
  - P2-Chord@16min: 95% consistency
  - P2-Chord@8min: 79% consistency

- **C++ Chord:**
  - MIT-Chord@47mins: 99.9% consistency
CHURN PERFORMANCE

- **P2-Chord:**
  - P2-Chord@90mins: 99% consistency
  - P2-Chord@47mins: 96% consistency
  - P2-Chord@16min: 95% consistency
  - P2-Chord@8min: 79% consistency

- **C++ Chord:**
  - MIT-Chord@47mins: 99.9% consistency
MOVING CATOMS IN MELD

**Rule 1:** \( \text{Dist}(S,D):= \text{At}(S,P), \)
\( P_d = \text{destination}(), \)
\( D = |P - P_d|, \)
\( D > \text{robot radius}. \)

**Rule 2:** \( \text{Farther}(S,T):= \text{Neighbor}(S,T), \)
\( \text{Dist}(S,D_s), \)
\( \text{Dist}(T,D_t), \)
\( D_s \geq D_t. \)

**Rule 3:** \( \text{MoveAround}(S,T,U):= \text{Farther}(S,T), \)
\( \text{Farther}(S,U), \)
\( U \neq T. \)
TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT FOR
- WHAT IS NEXT?
- WHAT’S IT TO YOU
WHY WHAT?
SAY WHAT
WHAT: HOW
WHAT FOR
WHAT IS NEXT?
WHAT’S IT TO YOU
DISTRIBUTED INFERENCE

- industrial revolution in data
  - data, networks, uncertainty.

- challenge: real-time info
  - despite uncertainty and acquisition cost

- applications
  - internet security, building control, disaster response, robotics. ANY distributed query.
DISTRIBUTED INFERENCE

- industrial revolution in data
  - data, networks, uncertainty.

- challenge: real-time info
  - despite uncertainty and acquisition cost

- applications
  - internet security, building control, disaster response, robotics. ANY distributed query.
given:
- a graphical model
- node: random variables
- edge: correlation
- evidence (data)

find probabilities

tactic: belief propagation

a “message passing” algorithm
DISTRIBUTED INFERENCE

- graphs upon graphs
- each can be easy to build
- opportunity for rich cross-layer optimization
DISTRIBUTED INFERENC

• graphs upon graphs
• each can be easy to build
• opportunity for rich cross-layer optimization
DISTRIBUTED INFERENCE

- graphs upon graphs
  - each can be easy to build
  - opportunity for rich cross-layer optimization
even fancy belief propagation is not bad

- robust distributed junction tree 22 rules
- 10x smaller than Paskin’s Lisp
- + identified a race condition
- various approximate algorithms Friday!
RESEARCH ISSUES

- optimization at each layer.
  - custom Inference Overlay Networks (IONs)
  - network-aware approximate inference algorithms (NAIAs)
- optimization across layers?
  - co-design to balance NW cost and approximation quality
optimization at each layer.
- custom Inference Overlay Networks (IONs)
- network-aware approximate inference algorithms (NAIAs)

optimization across layers?
- co-design to balance NW cost and approximation quality
EVITA RACED: OVERLOG METACOMPILER

DECLARATIVE
EVITA RACED: OVERLOG METACOMPILER
EVITA RACED: OVERLOG METACOMpiler

- represent:
  - overlog as data
  - optimizations as overlog
  - optimization scheduling as a lattice -- i.e. data

- needs just a little bootstrapping
- optimization as “hand-wired” dataflow
WHY METACOMPILATION?

- datalog a good fit to datalog optimizations
  - dynamic programming = recursive table-building
  - magic sets: traversal of “rule/goal graph”
  - statistics-gathering = query processing & inference
- extensibility required
  - platforms: P2, DSN. MapReduce? Manycore?
  - apps: networking, inference, security, robotics, ...
OVERLOG AS DATA
OPTIMIZER AS OVERLOG

- System R’s Dynamic Programming
  - 38 rules
- Magic Sets Rewriting
  - 68 rules
  - close translation of Ullman’s course notes
THE EVITA RACED DATAFLOW

Demux

<stage>::programEvent

program

Stage Scheduler

Parser

Physical Planner

Installer

update stream

program

insertion

program
TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT IS NEXT?
- WHAT FOR
- WHAT’S IT TO YOU
TODAY

• WHY WHAT?
• SAY WHAT
• WHAT: HOW
• WHAT IS NEXT?
• WHAT FOR
• WHAT’S IT TO YOU
This is not a talk about databases and networking
VISION

automatic programming
one domain at a time

parallel dataflow runtimes
sweet spot between embarrassing and intractable

distributed intelligence
bring ML program(mer)s into the network
QUERIES

http://www.declarativity.net
OVERLAY NETWORKS

- distributed apps on the network
- the game: track...
  - subset of participating nodes
  - names for participating nodes
  - multi-hop routing via other nodes
- many examples
  - VPNs, P2P, MS Exchange, Distributed Hash Tables...
OVERLAY NETWORKS

- distributed apps on the network
  - the game: track...
    - subset of participating nodes
    - names for participating nodes
    - multi-hop routing via other nodes

- many examples
  - VPNs, P2P, MS Exchange, Distributed Hash Tables...
DECLARATIVE OVERLAYS

- more challenging than simple routing
  - must generate/maintain overlay topology
  - message delivery, acks, failure detection, timeouts, periodic probes, etc...

- timer-based “built-in” event predicates:

  ping(@D,S) :- periodic(@S,10), link(@S,D)