JOINT WORK

Peter ALVARO
Tyson CONDIE
Neil CONWAY
Bill MARCZAK

Khaled ELMELEEGY
Rusty SEARS
• data-centric cloud programming
• datalog and overlog
• a look at BOOM
• a whiff of Bloom
• directions
THE FUTURE’S SO CLOUDY

- a new software dev/deploy platform
  - shared, dynamic, evolving
  - spanning sets of machines over time
  - data and session-centric

http://www.flickr.com/photos/kky/704056791/
WHAT DRIVES A NEW PLATFORM?

http://en.wikipedia.org/wiki/IBM_PC
http://en.wikipedia.org/wiki/Macintosh
http://en.wikipedia.org/wiki/Facebook
http://en.wikipedia.org/wiki/iPhone
http://en.wikipedia.org/wiki/Wii
DEVELOPERS!

Squeals of derision rang through the room. “You, program a computer?” someone asked incredulously. “Now I’ve heard everything!”

“Enjoy your laugh, beetleface,” I thought. “You won’t be chuckling for long.” Little did they know I had MICROSOFT BASIC II, the powerful programming language that uses simple English commands.

I slipped the potent little cartridge into my ATARI Home Computer and closed the door with a confident slap. In a very short time, my friends were astounded at my programming prowess. Information, sounds, colors — even player-missile graphics — leapt across the screen. True, at one point I did have a little bug in a program, but MICROSOFT BASIC II’s debugging features helped me correct it easily. I finished my tour de force by typing in a program written in another computer’s MICROSOFT BASIC dialect.

Oohs and ahs filled the air. “Top drawer,” snapped the Colonel. “What a man,” Mimi cooed. MICROSOFT BASIC II and I had won the day.

http://www.flickr.com/photos/nicoll/150272557/
the ultimate challenge?

- parallel
- distributed
- elastic
- minimally managed
WHO’S THE BOSS

- it’s all about the (distributed) state
  - session state
  - coordination state
  - system state
  - protocol state
  - permissions state
- .. and the mission critical stuff

- and deriving/updating/communicating that state!

http://www.flickr.com/photos/face_it/2178362181/
**reify state as data**

- system state is 1st-class data.
- model. react. evolve.

**data-centric programming**

- declarative specs for event handling, state safety and transitions
- reduces hard problems to easy ones
  - e.g. concurrent programming => data parallelism
  - e.g. synchronize only for counting
decades of theory

logic programming, dataflow

but: recent groundswell of applied research

networking, distributed computing, statistical machine learning, multiplayer games, 3-tier services, robotics, natural language processing, compiler analysis, security...

see http://declarativity.net/related

and CCC Blog:

http://www.cccblog.org/2008/10/20/the-data-centric-gambit/
GRAND ENOUGH FOR YOU?

- automatic programming ... Gray’s Turing lecture

- “the problem is too hard ... Perhaps the domain can be limited ... In some domains, declarative programming works.” (Lampson, JACM 50’th)

- can cloud be one of those domains?

- how many before we emend Lampson?
data-centric cloud programming

Datalog and overlog

A look at BOOM

A whiff of Bloom

Directions
Data (stored).

Logic: what we can deduce from the data

\[ p \quad \text{:-} \quad q. \]

SQL “Views” (stored/named queries)

This is all of computing

Really!

But until recently, it helped to be European.
parent(X,Y).

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

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

anc(X, s)?

Notes: unification, vars in caps, head vars must be in body.
Set semantics (no dups).
DUSTY OLD DATALOG

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

Notes: *unification*, vars in caps, head vars must be in body.
Set semantics (no dups).
THE INTERNET CHANGES EVERYTHING?

- `link(X,Y).
- `path(X,Y) :- link(X,Y).
- `path(X,Z) :- link(X,Y), path(Y,Z).
- `path(X,s)?

Notes: unification, vars in caps, head vars must be in body. Set semantics (no dups).
DATALOG SEMANTICS

- **minimal model**
  - i.e. smallest derived DB consistent with stored DB

- **Lemma**: datalog programs have a unique minimal model
  - "least model"

- **Lemma**: natural recursive join strategy computes this model
  - "semi-naive" evaluation

- \( \text{link}(X, Y) \)
- \( \text{path}(X, Y) :- \text{link}(X, Y) \)
- \( \text{path}(X, Z) :- \text{link}(X, Y), \text{path}(Y, Z) \)
- \( \text{path}(X, s) ? \)
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

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

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

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

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

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

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

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

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

\begin{align*}
\text{link}(X, Y, C) \\
\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) \\
\end{align*}
FORMING PATHS

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

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

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

Note: we just extended Datalog with functions, which are infinite relations. E.g. `sum(C, D, E)`. Need to be careful that programs are still “safe” (finite model).
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)`

Note: we just extended Datalog with functions, which are infinite relations. E.g. `sum(C, D, E)`. Need to be careful that programs are still “safe” (finite model).
FORMING PATHS

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

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

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

Note: we just extended Datalog with functions, which are infinite relations. E.g. \text{sum}(C, D, E). Need to be careful that programs are still “safe” (finite model).
BEST PATHS
BEST PATHS

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

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

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

míncost(X,Z,mín<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)

míncost(X, Z, mín<C>) :- path(X, Z, Y, C)

bestpath(X, Z, Y, C) :- path(X, Z, Y, C), míncost(X, Z, C)
**BEST PATHS**

- \texttt{link(X,Y)}
- \texttt{path(X,Y,Z,C) :- link(X,Y,C)}
- \texttt{path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)}
- \texttt{mincost(X,Z,min<C>) :- path(X,Z,Y,C)}
- \texttt{bestpath(X,Z,Y,C) :- path(X,Z,Y,C), mincost(X,Z,C)}
- \texttt{bestpath(src,D,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)

\text{mincost}(X,Z,\text{min}\langle C \rangle) :- \text{path}(X,Z,Y,C)

\text{bestpath}(X,Z,Y,C) :- \text{path}(X,Z,Y,C), \text{mincost}(X,Z,C)

\text{bestpath}(\text{src},D,Y,C)\ ?

Note: we just extended Datalog with aggregation. You can’t compute an aggregate until you fully compute its inputs (stratification).
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.
LOCATION SPECS INDUCE COMMUNICATION

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

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

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

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

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

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

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

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

---

**Diagram:**

- **Path:**
  - a b b l
  - b a l
  - c b b l
  - d c c l

- **Link:**
  - a b l
  - b a l
  - c b l
  - d c l
LOCATION SPECS INDUCE COMMUNICATION

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

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

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

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

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

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

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

- $\text{link}(@X,Y)$
- $\text{path}(@X,Y,Y,C) :- \text{link}(@X,Y,C)$
- $\text{link}_d(X,@Y,C) :- \text{link}(@X,Y,C)$
- $\text{path}(@X,Z,Y,C+D) :- \text{link}_d(X,@Y,C), \text{path}(@Y,Z,N,D)$
**LOCATION SPECS INDUCE COMMUNICATION**

- \texttt{link}(\texttt{@X,Y})
- \texttt{path}(\texttt{@X,Y,Y,C}) :- \texttt{link}(\texttt{@X,Y,C})
- \texttt{link\_d}(\texttt{X,Y,C}) :- \texttt{link}(\texttt{@X,Y,C})
- \texttt{path}(\texttt{@X,Z,Y,C+D}) :- \texttt{link\_d}(\texttt{X,Y,C}), \texttt{path}(\texttt{Y,Z,N,D})
LOCATION SPECS INDUCE COMMUNICATION

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

Localization Rewrite

\[
\begin{array}{cccc}
\text{link}_d: & a & b & l \\
& c & b & l \\
& d & c & l \\
\text{path:} & a & b & b & l \\
& b & a & a & l \\
& c & d & d & l \\
& d & c & c & l \\
\text{link:} & a & b & l \\
& b & a & l \\
& c & l \\
& d & l \\
\end{array}
\]
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)
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(@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

THIS IS DISTANCE VECTOR

link:

path:

link_d:
OVERLOG IS...

- Datalog w/aggregation & function symbols
- + Horizontally partitioned tables (data, not messages!)
  - “Event” tables for clock/net/host (data again!)
- + iterated (single-machine) fixpoints
  - “state update” happens atomically between fixpoints
- formal temporal logic treatment in *Dedalus* (foundation of *Bloom*)
**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 $\tau_I$. Reset $c$, pick a new $t$.
Receive newer code | Set $\tau$ to $\tau_I$. Reset $c$, pick a new $t$.
Receive older metadata | Send updates.

$t$ is picked from the range $\left[\frac{\tau}{2}, \tau\right]$

**Figure 12: Trickle Pseudocode.**

```plaintext
1 % Tau expires:
2  % Double Tau up to tauHi. Reset C, pick a new T.
3  tauVal[@X,Tau*2] := timer[@X,tauTimer,Tau], Tau*2 < tauHi.
4  tauVal[@X,TauHi] := timer[@X,tauTimer,Tau], Tau*2 >= tauHi.
5  timer[@X,tTimer,T] := tauVal[@X,TauVal], T =
6  rand(TauVal/2,TauVal).
7  timer[@X,tauTimer,TauVal] := tauVal[@X,TauVal].
8
9 % T expires; If C < k, transmit.
10  msgCnt[@X,0] := tauVal[@X,TauVal].
11
12 % Receive same metadata: Increment C.
13  msgCnt[@X,C+i] := msgVer[@X,Y,Old_CurVer], ver[@X,Old,CurVer],
14       msgCnt[@X,C].
15
16 % Receive newer metadata:
17  % Set Tau to tauLow. Reset C, pick a new T.
18  tauVal[@X,tauLow] := msgVer[@X,Y,Old_NewVer],
19       ver[@X,Old,OldVer], NewVer > OldVer.
20
21 % Receive newer data:
22  % Set Tau to tauLow. Reset C, pick a new T.
23  tauVal[@X,tauLow] := msgStore[@X,Y,Old_NewVer,Obj],
24       ver[@X,Old,OldVer], NewVer > OldVer.
25
26  % Receive older metadata: Send updates.
27  msgStore[@X,Y,Old_NewVer,Obj] := msgVer[@X,Y,Old,OldVer],
28    ver[@X,Old,NewVer], NewVer > OldVer,
29    store[@X,Old,NewVer,Obj],
30
31 % Update version upon successfully receiving store
32  store[@X,Y,Old_NewVer,Obj] := msgStore[@X,Y,Old_NewVer,Obj],
33     store[@X,Old,OldVer,Obj], NewVer > OldVer.
34
35 % Send updates upon successfully receiving store
36  ver[@X,Old,NewVer,Obj] := store[@X,Old,NewVer,Obj].
```

**Listing 3. Trickle Version Coherency**

---

Thursday, November 19, 2009
### Figure 12: Trickle Pseudocode.

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$ Expires</td>
<td>Double $\tau$, up to $\tau_R$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>$t$ Expires</td>
<td>If $c &lt; k$, transmit.</td>
</tr>
<tr>
<td>Receive same metadata</td>
<td>Increment $c$.</td>
</tr>
<tr>
<td>Receive newer metadata</td>
<td>Set $\tau$ to $\tau_I$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>Receive newer code</td>
<td>Set $\tau$ to $\tau_I$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>Receive older metadata</td>
<td>Send updates.</td>
</tr>
</tbody>
</table>

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

### Listing 3. Trickle Version Coherency

1. Tau expires:
   2. Double Tau up to tauHi. Reset C, pick a new T.
   4. tauVal[@X, tauHi] := timer[@X, tauTimer, Tau], TauStar2 >= tauHi.
   7. msgCnt[@X, 0] := tauVal[@X, TauVal].

8. T expires:
   9. If $C < k$, transmit.
   10. msgVer[@X, Y, Old, Ver] := ver[@Y, Old, Ver], timer[@X, tTimer, ..].
   11. msgCnt[@X, C], $C < k$.

12. Receive same metadata:
   13. Increment C.
   14. msgCnt[@X, C+1] := msgVer[@X, Y, Old, CurVer], ver[@X, Old, CurVer], msgCnt[@X, C].

15. Receive newer metadata:
   16. Set Tau to tauLow. Reset C, pick a new T.
   17. tauVal[@X, tauLow] := msgStore[@X, Y, Old, NewVer], ver[@X, Old, OldVer], NewVer > OldVer.

18. Receive newer data:
   19. Set Tau to tauLow. Reset C, pick a new T.
   20. tauVal[@X, tauLow] := msgStore[@X, Y, Old, NewVer, Obj], ver[@X, Old, OldVer], NewVer > OldVer.

21. Receive older metadata:
   22. Send updates.
   23. msgStore[@X, Y, Old, NewVer, Obj] := msgVer[@X, Y, Old, OldVer], ver[@X, Old, NewVer], NewVer > OldVer,
   24. store[@X, Old, NewVer, Obj].

25. Update version upon successfully receiving store
   26. store[@X, Old, NewVer, Obj] := msgStore[@X, Y, Old, NewVer, Obj], store[@X, Old, OldVer, Obj], NewVer > OldVer.
   27. ver[@X, Old, NewVer, Obj] := store[@X, Old, NewVer, Obj].

Thursday, November 19, 2009
chord distributed hash table

Internet overlay for content-based routing

high-function implementation

all the research bells and whistles

48 rules, 13 table definitions
P2-CHORD

- chord distributed hash table
- Internet overlay for content-based routing
- high-function implementation
- all the research bells and whistles
- 48 rules, 13 table definitions
### Chord Distributed Hash Table

**Internet Overlay for Content-Based Routing**

**High-Function Implementation**

**All the Research Bells and Whistles**

### 48 Rules, 13 Table Definitions
/* The base tuples */
materialize(node, infinity, keys(1)).
materialize(finger, 10, keys(2)).
materialize(bestSucc, infinity, keys(1)).
materialize(succDist, 10, keys(2)).
materialize(pred, infinity, keys(1)).
materialize(succCount, infinity, keys(1)).
materialize(join, 10, keys(1)).
materialize(landmark, infinity, keys(1)).
materialize(fFix, 160, keys(2)).
materialize(nextFingerFix, infinity, keys(1)).
materialize(pingNode, 10, keys(2)).
materialize(pendingPing, 10, 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),
  succDist@NI(NI,S,D), node@NI(NI,N),
  D == S - N - 1.
n4 maxSuccDist@NI(NI,D) :- succ@NI(NI,S,SI),
  node@NI(NI,N), succ@NI(NI,S,SI),
  D := max<IN(N)(S,D), node@NI(NI,N),
  D = S - N - 1.
/** Successor eviction */
s1 succCount@NI(NI,count<*>) :- succ@NI(NI,S,SI).
s2 evictSucc@NI(NI) :- succCount@NI(NI,C), C > 2.
s3 maxSuccDist@NI(NI,max<D>) :- succ@NI(NI,S,SI),
  node@NI(NI,N),
  evictSucc@NI(NI,SI),
  D := S - N - 1.
s4 delete succ@NI(NI,S,SI) :- node@NI(NI,N),
  succ@NI(NI,S,SI),
  maxSuccDist@NI(NI,D), node@NI(NI,N),
  D := S - N - 1.
/** Finger fixing */
f1 fFix@NI(NI,E,I) :- periodic@NI(NI,E,10),
  nextFingerFix@NI(NI,I).
f2 fFixEvent@NI(NI,E,I) :- fFix@NI(NI,E,I),
  nextFingerFix@NI(NI,I).
f3 fFix@NI(NI,E,I) :- fFixEvent@NI(NI,E,I),
  node@NI(NI,N), k:=II + 1,
  K:=II << I + N, K in (N,K),
  I:=I + 1, I in (N),
  fFix@NI(NI,E,I).
f6 eagerFinger@NI(NI,L,BI) :- node@NI(NI,N),
  eagerFinger@NI(NI,L,BI),
  I:=I + 1, K:=II << I + N, K in (N,BI),
  I in (N),
  efinger@NI(NI,L,BI).
f7 delete fFix@NI(NI,E,I) :- eagerFinger@NI(NI,L,BI),
  fFix@NI(NI,E,I), I > 0, II := I - 1.
f8 nextFingerFix@NI(NI,E,I) :- eagerFinger@NI(NI,L,BI),
  (I := 159 | (BI == NI)).
100x LESS CODE THAN MIT CHORD

Thursday, November 19, 2009
TODAY

- data-centric cloud programming
- datalog and overlog
- a look at BOOM
- a whiff of Bloom
- directions
Berkeley Orders Of Magnitude

- OOM bigger systems
- OOM less code

- we did it for network protocols, time to generalize
- and make attractive to developers

*Bloom* is the language for BOOM

Thursday, November 19, 2009
experiment: build a Big Data cloud stack in Overlog

- goal 1: convince ourselves we’re on track
- goal 2: inform the design of a better language (Bloom)
  - for cloud ... and multicore?
- goal 3: pull off some feats of derring-do

metrics
- not just LOCs
- flexibility, malleability, performance
Prototype: basic Hadoop functionality

Subsequent revisions (prototype Hadoop’s future)
- Availability rev: hot-standby masters
- Scalability rev: scale out master state
- Monitoring rev: invariant checking, logging

9 months, 4 grad student developers
- Most work in a 3-month span
### Current Code Base

#### Filesystem

<table>
<thead>
<tr>
<th></th>
<th>Lines of Java</th>
<th>Lines of Overlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>21,700</td>
<td>0</td>
</tr>
<tr>
<td>BOOM-FS</td>
<td>1,431</td>
<td>469</td>
</tr>
</tbody>
</table>

#### MapReduce

<table>
<thead>
<tr>
<th></th>
<th>Lines of Java</th>
<th>Lines of Overlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>88,864</td>
<td>0</td>
</tr>
<tr>
<td>BOOM-MR</td>
<td>82,291</td>
<td>396</td>
</tr>
</tbody>
</table>
THE $3,500 SLIDE
HDFS has single point of failure at master

- we found JIRA proposals for warm-standby

- but we went for a hot-standby scheme

- had wanted to do serious Paxos all along as a stress test

- Paxos: 50 Overlog rules (Stasis for persistence)

- basic Paxos vs. serious multiPaxos
1. Priest p chooses a new ballot number b greater than lastTried [p], sets lastTried [p] to b, and sends a NextBallot (b) message to some set of priests.

2. Upon receipt of a NextBallot (b) message from p with b > nextBal [q], priest q sets nextBal [q] to b and sends a LastVote (b, v) message to p, where v equals prevVote [q]. (A NextBallot (b) message is ignored if b < nextBal [q].)

3. After receiving a LastVote (b, v) message from every priest in some majority set Q, where b = lastTried [p], priest p initiates a new ballot with number b, quorum Q, and decree d, where d is chosen to satisfy B3. He then sends a BeginBallot (b, d) message to every priest in Q.

4. Upon receipt of a BeginBallot (b,d) message with b = nextBal [q], priest q casts his vote in ballot number b, sets prevVote [q] to this vote, and sends a Voted (b, q) message to p. (A BeginBallot (b, d) message is ignored if b = nextBal [q].)

5. If p has received a Voted (b, q) message from every priest q in Q (the quorum for ballot number b), where b = lastTried [p], then he writes d (the decree of that ballot) in his ledger and sends a Success (d) message to every priest.
1. Priest p chooses a new ballot number b greater than lastTried[p], sets lastTried[p] to b, and sends a NextBallot(b) message to some set of priests.

2. Upon receipt of a NextBallot(b) message from p with b > nextBal[q], priest q sets nextBal[q] to b and sends a LastVote(b,v) message to p, where v equals prevVote[q]. (A NextBallot(b) message is ignored if b < nextBal[q].)

3. After receiving a LastVote(b,v) message from every priest in some majority set Q, where b = lastTried[p], priest p initiates a new ballot with number b, quorum Q, and decree d, where d is chosen to satisfy B3. He then sends a BeginBallot(b,d) message to every priest in Q.

4. Upon receipt of a BeginBallot(b,d) message with b = nextBal[q], priest q casts his vote in ballot number b, sets prevVote[q] to this vote, and sends a Voted(b,q) message to p. (A BeginBallot(b,d) message is ignored if b = nextBal[q].)

5. If p has received a Voted(b,q) message from every priest q in Q (the quorum for ballot number b), where b = lastTried[p], then he writes d (the decree of that ballot) in his ledger and sends a Success(d) message to every priest.
MULTIPAXOS IN OVERLOG

“I Do Declare…”, Alvaro, et al. NetDB 09

master scaling woes? buy a bigger box!

- a real problem at Yahoo

- “scale out” master to multiple machines?
  
  - massive rewrite in HDFS. trivial in BOOM-FS!
  
  - hash-partition metadata tables as you would in a DB

- lookups by unicast or broadcast

- task completed in one day

- by Rusty Sears, the “OS guy” on the team
monitoring rev

- invariant checking easy to add
  - messages are data; just query that messages match protocol
  - we validated Paxos message counts

- tracing/logging via metaprogramming
  - code is data: can write “queries” to generate more code
  - we built a code coverage tool in a day (17 rules + a java driver)

- system telemetry, logging/querying
  - sampled /proc into tuples
  - easily wrote real-time in-network monitoring in Overlog
because everything is data...
- easy to design scale-out
- *interposition* (classic OS goal) easy via dataflow
- concurrency simplified
  - data derivation (stratification) vs. locks on object updates
  - simple dataflow analysis vs. state/event combinatorics

all this applies to dataflow programming
- e.g. mapreduce++
- potentially sacrifice code analysis
LESSONS 2

- **overlog limitations**
  - datalog syntax: hard to write, *really* hard to read
  - partitioned tables are a *lie*, so we don’t use them
    - except as a layer above Paxos/2PC etc.
  - state update is “illogical”
    - as noted in recent papers on operational semantics of P2
data-centric cloud programming

datalog and overlog

a look at BOOM

a whiff of Bloom

directions
there is no space. only time.

now.

next.

later.

machine boundaries induce unpredictable delays

otherwise space is irrelevant

time is a fiction

Dedalus: a temporal logic capturing state update, atomicity/visibility, and delays
Batch
- what to deQ when. defines a “trace”

Logic
- “now”: derivations, assertions, invariants

Operations
- “next”: local state modification, side effects

Orthography
- i.e., acronym enforcement

Messages
- “later”: network xmission, asynchronous calls
A NOTE TO READERS OF THE SLIDES

- the following slide is a ruby-ish “mockup” of what Bloom might look like.
- Bloom itself was not specified at the time of this talk
- Hence “v. -1”
SHORTEST PATHS: BLOOM v. -1

**BATCH:**

each path or every 1 second;

**LOGIC:**

table link [String from, String to] [integer cost];

define path [String from, String to] [String nexthop, integer cost] {
    link.each |l| :
    yield { [l.from, l.to] => [l.to, l.cost] };

    (path.to->link.from).each |p,l| :
    yield { [p.from, l.to] => [p.nexthop, p.cost + l.cost] };
}

define shortest_paths [String from, String to] [integer cost] {
    least = path.reduce([from,to] => [min(cost)]);
    (path.[from,to]->least[from,to]).each |p,l| :
    yield { [p.from, p.to] => [p.nexthop, l.cost] }
}

**OPS:**

**MSGS:**

path.each |p| { send(p.from, p) if p.from != localhost }
data-centric cloud programming

datalog and overlog

a look at BOOM

a whiff of Bloom

directions
continue pushing Hadoop community
  e.g. HOP for streams and online agg
from analytics to interactive apps
  C4: a low-latency (explosive) runtime
towards a more complete Cloudstack
  multifaceted/ambitious look at storage consistency
  cloud operator/service management
  monitoring/prediction/control (w/Guestrin@CMU)
  secure analytics, nets (w/DawnSong, Mitchell@Stanford, Feamster@GTU)
BLOOM AGENDA

Syntax & Semantics
- nail down *Dedalus*
- integral syntax for time
  - now/next/later
- logic made approachable
  - list comprehensions

Static analysis
- parallelism/concurrency
- redistribution
- concurrency

Debugging
- static checks
- message provenance
- distributed checkpt

Complexity?
- resources are free
- coordination is expensive
- “coordination surfaces”
- randomization & approximation

Thursday, November 19, 2009
QUERIES?

http://www.declarativity.net
remaining slides are backup
DECLARATIVE NETWORKING @ BERKELEY/INTEL, ETC.

- 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
- metacompilation  Evita Raced  VLDB 08, Berkeley/Intel
- wireless sensornets  DSN
  - link estimation, geo routing, data collection, code dissemination, object tracking, localization  SenSys 07, IPSN 09, Berkeley
DECLARATIVE NETS: EXTERNAL

- simple paxos in overlog 44 lines, Harvard, 2006
- secure networking SeNDLog. NetDB07, MSR/Penn
- flexible replication in overlog PADRE/PADS SOSP07, NSDI09, Texas
- overlog semantics & analysis MPII 09
- distributed ML inference CMU/Berkeley 08
OTHERS

- video games (sgl) Cornell
- 3-tier apps (hilda, xquery) Cornell, ETH, Oracle
- compiler analysis (bddbddd) Stanford
- nlp (dyna) Johns Hopkins
- modular robotics (meld) CMU
- trust management (lbtrust) Penn/LogicBlox
- security protocols (pcl) Stanford
- ... see http://declarativity.net/related
“BOTTOM-UP” EXECUTION

- link(X,Y).
- path(X,Y) :- link(X,Y).
- path(X,Z) :- link(X,Y), path(Y,Z).
- path(X,s)?

- Akin to RDBMS with recursion
- join/project body predicates to derive new head facts.
- repeat until fixpoint
- Optimization: avoid rederiving known facts
- semi-naive evaluation
BOTTOM-UP EXECUTION

Dataflow plan

Event Handler

= net shuffle
= join

link

lookup

path

link

path
enrolled (N,A) :- student (N,A),
    average (B), A > B.

average (avg <A>) :-
    enrolled (N,A).

student (Carlos, 30).

student (Joey, 20).

enrolled (Carlos, 30)?
STRATIFICATION

- no recursion through negation/aggregation
- lemna: evaluating strata in order of the dependency graph produces a (natural) minimal model!

- local stratification: similar lemma if no facts can ever recurse through negation/aggregation
Asynch Service:

\[
\text{msg}(\text{Client}, \text{@Server}, \text{Svc}, X) : - \\
\quad \text{request}(\text{@Client}, \text{Server}, \text{Svc}, X).
\]

\[
\text{response}(\text{@Client}, \text{Server}, \text{Svc}, X, Y) : - \\
\quad \text{msg}(\text{Client}, \text{@Server}, \text{Svc}, X), \\
\quad \text{service}(\text{@Server}, \text{Svc}, X, Y).
\]
SOME SIMPLE OVERLOG

Asynch Service:

\[
\text{msg}(@C, @S, Svc, X) :- \\
\quad \text{request}(@C, @S, Svc, X).
\]

\[
\text{response}(@C, @S, Svc, X, Y) :- \\
\quad \text{msg}(@C, @S, Svc, X), \\
\quad \text{service}(@S, Svc, X, Y).
\]

Timeout:

\[
\text{timer}(t, \text{physical}, 1000, \text{infinity}, 0).
\]

\[
\text{waits}(@C, S, Svc, X, \text{cnt}<f_{\text{rand}}()>, \text{X}) :- t(_, _, _), \\
\quad \text{request}(@C, S, Svc, X), \\
\quad !\text{response}(@C, S, Svc, X, _).
\]

\[
\text{late}(@C, S, Svc, X) :- \text{waits}(@C, S, Svc, X, \text{Delay}), \text{Delay} > 1.
\]
Multicast:

```
msg(@Dest, Payload) :- xmission(@Src, Payload),
                 group(@Src, Dest).
```

NW Routes:

```
path(@Src, Dest, Dest, Cost) :-
  link(@Src, Dest, Cost).
path(@Src, Dest, Hop, C1+C2) :-
  link(@Src, Hop, C1),
  path(@Hop, Dest, N, C2).
bestcost(@Src, Dest, min<Cost>) :-
  path(@Src, Dest, Hop, Cost).
bestpath(@Src, Dest, Hop, Cost) :-
  path(@Src, Dest, Hop, Cost),
  bestcost(@Src, Dest, Cost).
```
OVERLOG EXECUTION

Network

Clock

Java

Datalog

Local, atomic computation

Events

Events

Network

Machine Boundary

Java
KEY CONCEPTS IN DEDALUS

- `link@4(1,2).
- `link@next(F,T) :- link(F, T).
- `path(F,T) :- link(F,N),  
  path(N,T).
- `msg@later(T,F,M)  
  :- link(F,T),  
  M = “howdy, neighbor”.

- facts @constant.
- head predicates have timespecs  
  N, N+1, N+r()
- body predicates implicitly @N.
STATE UPDATE IN DEDALUS

- **persistence:**
  - \( r@next(X) :- r(X) \text{ and } !\text{del}_r(X). \)

- **deletion:**
  - \( \text{del}_r(X) :- \text{msg}(X). \)

- **key update:**
  - \( \text{del}_s(K,W) :- s(K,W), \text{new}(K,V). \)
  - \( s@next(K,V) :- \text{new}(K,V). \)

- "deferred" delete and update
- there’s a gotcha here we’re still ironing out...
Konwinski/Zaharia’s LATE protocol:

- 3 lines pseudocode, 5 rules in Overlog
- vs. 800-line patchfile
  - ~200 lines implement LATE
  - other ~600 lines modify 42 Java files
- comparable results
aggregation = stratification = "wait"

natural analogy to counting semaphores

this is the *only* reason for parallel barriers

delay iff data dependencies depend on parallelism

or even cheat: approximate aggregates, speculation.
the “trace” of a system

mapping between external sequence (msg queue) and system time

“entanglement” of 2 systems

relationship between msgs in their traces
chains of inference on independent data can be “rescheduled”

prove two “traces” equivalent.
“causal” ordering
“happens before”
our “cause” is data dependency.
what else “happens”?! 
captured faithfully (statically and dynamically) via logic.
P2 @ 10,000 FEET

- Overlog
- Parser
- AST
- Planner
- Net
- Tables
- Dataflow
- Scheduler
P2 @ 10,000 FEET

java, ruby

Overlog

Parser

Net

Tables

Dataflow

 java, ruby

Thursday, November 19, 2009
P2 @ 10,000 FEET

java, ruby

Net

Parser

Overlog

Tables

Dataflow
DATAFLOW EXAMPLE IN P2


DATAFLOW EXAMPLE IN P2
flow runs at multiple nodes
data partitioned by locspec
this is SPMD parallel dataflow

a la database engines, MapReduce

locspecs can be hash functions via content routing
unlike MapReduce, finer-grained operators that pipeline
### DSN vs NATIVE TRICKLE

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

**Diagram:**

- **Nodes Updated** vs **Seconds**
- Graphs show performance comparison between DSN and Native Trickle.
- Different lines represent different configurations of Trickle, indicating the rate of node updates over time.
## 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 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
Dedalus is really Datalog

- with negation/aggs, a *successor* relation for time, and a non-deterministic function (for *later*)
- time an attribute of each table
- rewrite rule bodies to include “now predicates”.

Dedalus semantics: minimal model

- with “don’t-care” semantics on non-deterministic values
- some details to work out here
given a fixed input DB, can just run semi-naive eval.

- assertion: “now predicate” locally stratifies on (monotonically increasing) time
- challenge: “implement” minimal model of a Dedalus program via “traditional” persistence
- I.e. store, don’t re-derive.
EVITA RACED: OVERLOG METACOMPILER
EVITA RACED:
OVERLOG METACOMPILER
EVITA RACED:
OVERLOG METACOMPILER

- represent:
  - overlog as data
  - optimizations as overlog
  - optimizer stage schedule as a lattice -- i.e. data

- needs just a little bootstrapping
  - optimization as “hand-wired” dataflow
OVERLOG AS DATA
OPTIMIZER AS OVERLOG

- System R’s Dynamic Programming
  - 38 rules
- Magic Sets Rewriting
  - 68 rules
  - close translation to Ullman’s course notes
- VLDB Feedback story
  - replaced System R with Cascades Branch-and-Bound search
  - 33 rules, 24 hours
  - paper accepted
SOME LESSONS

dynamic programming & search
- another nice fit for declarative programming

extensible optimizer really required
- e.g. protocol optimization not like a DBMS
- graph algorithms vs. search-space enumeration
MOVING CATOMS IN MELD

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

RULE 2: Farther\( (S, T) \) :- Neighbor\( (S, T) \),
\[
\text{Dist}(S, D_S),
\text{Dist}(T, D_T),
D_S \geq D_T.
\]

RULE 3: MoveAround\( (S, T, U) \) :- Farther\( (S, T) \),
\[
\text{Farther}(S, U),
U \neq T.
\]

[Ashley-Rollman, et al. IROS ’07]
**challenge**: real-time distributed info

- despite uncertainty and acquisition cost

**applications**

- internet security, building control, disaster response, robotics

- really ANY distributed query.
given:
- a graphical model
- node: random variable
- edge: correlation
- evidence (data)

find probabilities for RVs

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 INFERENCE

- 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 39 rules
- 5x smaller than Paskin’s Lisp
- + identified a race condition
- also variants of Loopy Belief Propagation

[Funiak, Atul, Chen, Guestrin, Hellerstein, 2008]
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
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