THE CLOUD GOES BOOM
DATA-CENTRIC PROGRAMMING FOR DATA CENTERS
JOSEPH M HELLERSTEIN
UC BERKELEY
data-centric cloud programming
peek at BOOM
towards lincoln
directions
a new software dev/deploy platform

- shared, dynamic, evolving
- spanning multiple machines over time
- data and session-centric
WHAT DRIVES A NEW PLATFORM?

http://en.wikipedia.org/wiki/IBM_PC
http://en.wikipedia.org/wiki/Macintosh
http://en.wikipedia.org/wiki/Iphone
http://en.wikipedia.org/wiki/Facebook
http://www.flickr.com/photos/kky/704056791/

THEY LAUGHED WHEN I SAT DOWN AT THE KEYBOARD

Squeals of derision rang through the room. "You, program a computer?" someone asked incredulously. "Now I've heard everything!"
"Enjoy your laugh, beetface," 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.

DEVELOPERS!

http://www.flickr.com/photos/nicoll/150272557/
ARE DEVELOPERS DOING THIS IN THE CLOUD YET?

http://www.flickr.com/photos/fontplaydotcom/504443770/
CLOUD DEVELOPMENT

- 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 that state!

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

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

data-centric programming

- declarative specs for state transition and constraints
- machine/workload independent ... continuously optimizable
- data parallelism
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 LANGUAGES

- lots of academic experimentation
  - largely domain-specific
- substantial success
  - wide variety of domains
- still a largely disconnected field
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 Networks: 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
compiler analysis (bddd) Stanford

nlp (dyna) Johns Hopkins

modular robotics (meld) CMU

video games (sgl) Cornell

3-tier apps (hilda, xquery) Cornell, ETH, Oracle

trust management (lbtrust) Penn/LogicBlox

security protocols (pcl) Stanford
Figure 12: Trickle Pseudocode.

Listing 3. Trickle Version Coherency

```
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 = rand(TauVal/2, TauVal).
7 msgCnt[@X, 0] := tauVal[@X, TauVal].
8
9 % T expires: If C < k, transmit.
10 msgVer[@X, Old, Ver] := ver[@X, Old, Ver], timer[@X, tTimer, ...],
11 msgCnt[@X, C], C < k.
12
13 % Receive same metadata: Increment C.
14 msgCnt[@X, C+1] := msgVer[@X, Old, CurVer], ver[@X, Old, CurVer],
15 msgCnt[@X, C].
16 % Receive newer metadata:
17 % Set Tau to tauLow. Reset C, pick a new T.
18 tauVal[@X, tauLow] := msgStore[@X, Y, Old, NewVer],
19 ver[@X, Old, OldVer], NewVer > OldVer.
20 % Receive newer data:
21 % Set Tau to tauLow. Reset C, pick a new T.
22 tauVal[@X, tauLow] := msgStore[@X, Y, Old, NewVer, Obj],
23 ver[@X, Old, OldVer], NewVer > OldVer.
24 % Receive older metadata: Send updates.
25 msgStore[@X, X, Old, NewVer, Obj] := msgVer[@X, Old, OldVer],
26 ver[@X, Old, NewVer], NewVer > OldVer,
27 store[@X, Old, NewVer, Obj].
28
29 % Update version upon successfully receiving store
30 store[@X, Old, NewVer, Obj] := msgStore[@X, Y, Old, NewVer, Obj],
31 store[@X, Old, OldVer, Obj], NewVer > OldVer.
32
33 ver[@X, Old, NewVer, Obj] := store[@X, Old, NewVer, Obj].
```
<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$ Expires</td>
<td>Double $\tau$, up to $\tau_H$. 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 $\tau_H$. Reset $c$, pick a new $t$.</td>
</tr>
<tr>
<td>Receive newer code</td>
<td>Set $\tau$ to $\tau_L$. 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 $[\frac{\tau}{2}, \tau]$

**Figure 12: Trickle Pseudocode.**

Listing 3. Trickle Version Coherency

---

Levis, et al., Sensys 2004

Chu, et al., Sensys 2007
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, 100, 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, 1, keys(1)).
materialize(nextFingerFix, infinity, 1, keys(1)).

/** 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),
bestSucc@NI(NI,S,SI),
node@NI(NI,N), evictSucc@NI(NI,S,SI),
D:=S - N - 1.

eagerFinger@NI(NI,0,S,SI) :- node@NI(NI,N),
succCount@NI(NI,S,SI), maxSuccDist@NI(NI,D),
D == S - N - 1.

/** Finger fixing */
eagerFinger@NI(NI,0,S,SI) :- node@NI(NI,N),
succCount@NI(NI,S,SI), maxSuccDist@NI(NI,D),
D == S - N - 1.

/** Churn Handling */
joinEvent@NI(N,E,LI) :- joinRequest@NI(N,E),
node@NI(NI,N), landmark@NI(NI,LI), LI != "."
joinEvent@NI(N,E,LI) :- node@NI(NI,N),
landmark@NI(NI,LI), LI == "-".

/** Stabilization */
stabEvent@NI(NI,E) :- node@NI(NI,N),
stabRequest@NI(NI,E),
stab@NI(NI,N).

/** Connectivity Monitoring */
pingEvent@NI(N,E) :- periodic@NI(N,E,5),
pingNode@NI(NI,N),
node@NI(NI,N), succ@NI(NI,S,SI),
K in (B,N).
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materiaize(finger, 180, 160, keys(2)).
materiaize(bestSucc, infinity, 1, keys(1)).
materiaize(succDist, 10, 100, keys(2)).
materiaize(succ, 10, 100, keys(2)).
materiaize(pred, infinity, 100, keys(1)).
materiaize(succCount, infinity, 1, keys(1)).
materiaize(landmark, infinity, 1, keys(1)).
materiaize(fIX, infinity, 160, keys(2)).
materiaize(nextFingerFix, infinity, 1, keys(1)).
materiaize(pingNode, 10, infinity, keys(2)).
materiaize(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),
betweenDist@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.
succCount@NI(NI,S,SI) :- succ@NI(NI,S,SI),
betweenDist@NI(NI,S,SI,D), node@NI(NI,N),
D == S - N - 1.
finger@NI(NI,0,S,SI) :- succ@NI(NI,S,SI),
node@NI(NI,N),
maxSuccDist@NI(NI,N,S,SI), node@NI(NI,N),
evictSucc@NI(NI,N), S in (N,B).
succ@NI(NI,N,SI) :- succ@NI(NI,N,SI),
maxSuccDist@NI(NI,N,S,SI), node@NI(NI,N),
evictSucc@NI(NI,N), D:=S - N - 1.

/** Successor eviction */
select@NI(NI,E,10),
nextFingerFix@NI(NI,I).
fixEvent@NI(NI,E,10),
lookupResults@NI(NI,K,B,BI).
finger@NI(NI,I,B,BI) :- eagerFinger@NI(NI,I,B,BI),
node@NI(NI,N),
succ@NI(NI,N,SI),
succ@NI(NI,N,SI),
maxSuccDist@NI(NI,N,SI),
node@NI(NI,N),
evictSucc@NI(NI,N),
I:=I1 + 1, K:=I1 << I + N, K in (B,N).

/** Churn Handling */
joinEvent@NI(NI,E) :- join@NI(NI,E).
joinReq@NI(LI,NI,E) :- joinEvent@NI(NI,E),
ode@NI(NI,N), landmark@NI(NI,L), LI != "-".
succ@NI(NI,N,NI) :- landmark@NI(NI,L),
joinEvent@NI(NI,E), node@NI(NI,N),
LI == "-".
llookup@LI(LN,NI,E) :- joinReg@LI(NI,LN,NI,E),
succ@NI(NI,N,NI) :- joinEvent@NI(NI,E),
lookup@NI(NI,N,NI,E), node@NI(NI,N),
LI == "-".

/** Stabilization */
stable@NI(NI,E) :- periodic@NI(NI,E,5),
stableRequest@SI(SI,NI) :- stabilize@NI(NI,E),
bestSucc@NI(NI,N,SI). sendPredecessor@PI(LI,NI,P,PI) :- stabilize@NI(NI,E),
bestSucc@NI(NI,N,SI), sendPredecessor@NI(NI,P,PI),
bestSucc@NI(NI,N,SI),
prev@NI(NI,P,PI).

/** Connectivity Monitoring */
pingEvent@NI(NI,E) :- periodic@NI(NI,E,5),
pingNode@NI(NI,P,PI).
delete pending@NI(NI,P,PI),
pingResp@NI(NI,R). pingNode@NI(NI,P,PI),
node@NI(NI,N),
succ@NI(NI,N,SI).
delete pending@NI(NI,P,PI),
pingResp@NI(NI,R).
node@NI(NI,N),
succ@NI(NI,N,SI).
delete pending@NI(NI,P,PI),
pingResp@NI(NI,R,PI).
node@NI(NI,N),
succ@NI(NI,N,SI).
delete pending@NI(NI,P,PI),
pingResp@NI(NI,R,PI).
node@NI(NI,N),
succ@NI(NI,N,SI).
data-centric cloud programming
peek at BOOM
towards lincoln
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

lincoln
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 for the cloud and multicore?
- goal 3: pull off some feats of derring-do

metrics

- not just LOCs
- flexibility, malleability, performance
EVOLUTION SCENARIO

- prototype: basic Hadoop functionality
- subsequent revisions (Hadoop fast-forward)
  - 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
JOL
  - Java-based OverLog interpreter

BOOM-MR
  - Hadoop MapReduce with Overlog “brain transplant”

BOOM-FS
  - Hadoop Filesystem (HDFS) rewrite in Overlog
  - API-compliant Java skin
Example Data Flow

```
response/@Client, true, DirContents) :-
  request/@Master, Client, ReqType, DirName),
  ReqType == "ls",
  directory/@Master, DirName, DirContents);
```

---

**Request Table**

<table>
<thead>
<tr>
<th>@Master</th>
<th>Client</th>
<th>ReqType</th>
<th>DirName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>ls</td>
<td>/home</td>
</tr>
</tbody>
</table>

**Directory Table**

<table>
<thead>
<tr>
<th>@Master</th>
<th>Dirname</th>
<th>DirContents</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>/</td>
<td>{ home, tmp }</td>
</tr>
<tr>
<td>10.0.0.1</td>
<td>/home</td>
<td>{ foo, bar }</td>
</tr>
</tbody>
</table>

**Response Table**

<table>
<thead>
<tr>
<th>@Client</th>
<th>Success</th>
<th>DirContents</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.2</td>
<td>TRUE</td>
<td>{ foo, bar }</td>
</tr>
</tbody>
</table>
File System State

- `file(Master, FileId, FName, FParentId, IsDir)`
- `chunk(Master, ChunkId, FileId)`
- `fqpath(Master, Path, FileId)`
Example: “ls”

response(@Client, true, DirContents) :-
    request(@Master, Client, ReqType, DirName),
    ReqType == “ls”,
    directory(@Master, DirName, DirContents);

response(@Client, false, null) :-
    request(@Master, Client, ReqType, DirName),
    ReqType == “ls”,
   notin directory(@Master, DirName, _);

directory(@Master, DirName, set<FileName>) :-
    fqpath(@Master, DirName, DirId),
    file(@Master, FileId, DirId, FileName, _);
Heartbeats and Timers

timer(hb_clock, 1000);

heartbeat(@Master, Datanode, Tstamp, ChunkId) :-
    dn_chunk(@Datanode, ChunkId),
    master(@Datanode, Master),
    hb_clock(@Datanode, Tstamp);
Heartbeats and Timers

timer(hb_clock, 1000);

heartbeat(@Master, Datanode, Tstamp, ChunkId) :-
    dn_chunk(@Datanode, ChunkId),
    master(@Datanode, Master),
    hb_clock(@Datanode, Tstamp);

hb_cache(@Master, Datanode, Tstamp) :-
    heartbeat(@Master, Datanode, Tstamp, _);

hb_chunk(@Master, Datanode, ChunkId) :-
    heartbeat(@Master, Datanode, _, ChunkId);
Invariant Maintenance

clone_chunk(@Source, Target, ChunkId) :-
exemplar(@Master, ChunkId, Source),
candidate(@Master, ChunkId, Target),
chunk_cnt(@Master, ChunkId, Cnt),
Cnt < 3;

chunk_cnt(@Master, ChunkId, count<Datanode>) :-
heartbeat_cache(@Master, Datanode, _),
dn_chunk(@Master, Datanode, ChunkId);

delete
hb_cache(@Master, Datanode, Tstamp) :-
hb_cache(@Master, Datanode, Tstamp),
time() - Tstamp > 5000;
Does It Work?
Does It Work?

- How fast is it?
  - Within ~20% of HDFS for Hadoop
Does It Work?

- How fast is it?
  - Within ~20% of HDFS for Hadoop

- How complex is it?
  - BoomFS: 469 LOC Overlog, 1431 LOC Java
  - HDFS: 21,700 LOC Java
Does It Work?

- How fast is it?
  - Within ~20% of HDFS for Hadoop
- How complex is it?
  - BoomFS: 469 LOC Overlog, 1431 LOC Java
  - HDFS: 21,700 LOC Java
- How easy is it to change?
BOOM-MR SCHEMA
Task status aggregates attempt status

Task status = best attempt status

task(JobId, TaskId, Type, Partition, ..., max<Status>) :-
  task(JobId, TaskId, Type, Partition, ...),
  taskAttempt(..., AttemptId, Progress, State, Phase, ...),
  AttemptId.getTaskID() == TaskId,
  Status := new TaskStatus(Progress, State, Phase);

Job status aggregates task status

Job status = average task status

job(JobId, ..., avg<TaskStatus>) :-
  job(JobId, ...),
  task(JobId, TaskId, ..., TaskStatus);
Tracker actions triggered from state updates

New task attempt => launch task action

```
taskTrackerAction(JobTracker, @TaskTracker, TaskTrackerAction.ActionType.LAUNCH_TASK, Action) :-
  taskAttempt(@JobTracker, TaskTracker, AttemptId, _, TaskState.RUNNING, TaskPhase.STARTING),
  job(JobId, JobName, JobFile),
  task(JobId, TaskId, Type, Partition, FileInput, MapCount, _),
  JobId == AttemptId.getJobID(), TaskId == AttemptId.getTaskID(),
  Action := Type == Constants.TaskType.MAP ?
    launchMap(FileInput, JobFile, AttemptId, Partition) :
    launchReduce(JobFile, AttemptId, Partition, MapCount);
```

Job/task state set to “kill” => kill job/task action

```
taskTrackerAction(JobTracker, @TaskTracker, TaskTrackerAction.ActionType.KILL_TASK, Action) :-
  taskAttempt(@JobTracker, TaskTracker, AttemptId, _, TaskState.RUNNING, _, _, _, _, _),
  job(JobId, ... JobStatus),
  JobStatus.state() == Constants.JobState.KILLED ||
  JobStatus.state() == Constants.JobState.FAILED,
  Action := new KillTaskAction(AttemptId);
```
SIMPLE TASK SCHEDULER
Best job priority with unscheduled maps

```prolog
bestJobPriorityMap(JobId, min<Priority>) :-
    job(JobId, JobPriority, JobStatus, Conf),
    task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
    Priority := [JobPriority, JobStatus.getSubmitTime()];
```
Best job priority with unscheduled maps

bestJobPriorityMap(JobId, min\(<\text{Priority}\>) ) :-
  job(JobId, JobPriority, JobStatus, Conf),
  task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
  Priority := [JobPriority, JobStatus.getSubmitTime()];

Tracker workload

trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
  taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
  MapID := AttemptId.isMap() ? AttemptId : null,
  ReduceID := AttemptId.isMap() ? null : AttemptId;
### Best job priority with unscheduled maps

```
bestJobPriorityMap(JobId, min<Priority>) :-
  job(JobId, JobPriority, JobStatus, Conf),
  task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
  Priority := [JobPriority, JobStatus.getSubmitTime()];
```

### Tracker workload

```
trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
  taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
  MapID := AttemptId.isMap() ? AttemptId : null,
  ReduceID := AttemptId.isMap() ? null : AttemptId;
```

### Schedule a new map attempt

```
taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
```
Best job priority with unscheduled maps

\[
\text{bestJobPriorityMap(JobId, min(Priority)) :-}
\]

\[
\text{job(JobId, JobPriority, JobStatus, Conf),}
\]

\[
\text{task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),}
\]

\[
\text{Priority := [JobPriority, JobStatus.getSubmitTime()];}
\]

Tracker workload

\[
\text{trackerWorkload(@JobTracker, TaskTracker, count(MapID), count(ReduceID)) :-}
\]

\[
\text{taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),}
\]

\[
\text{MapID := AttemptId.isMap() ? AttemptId : null,}
\]

\[
\text{ReduceID := AttemptId.isMap() ? null : AttemptId;}
\]

Schedule a new map attempt

\[
\text{taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-}
\]

\[
\text{task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),}
\]
Best job priority with unscheduled maps

```prolog
bestJobPriorityMap(JobId, min<Priority>) :-
   job(JobId, JobPriority, JobStatus, Conf),
   task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
   Priority := [JobPriority, JobStatus.getSubmitTime()].
```

Tracker workload

```prolog
trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
   taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
   MapID := AttemptId.isMap() ? AttemptId : null,
   ReduceID := AttemptId.isMap() ? null : AttemptId.
```

Schedule a new map attempt

```prolog
taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
   task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),
   bestJobPriorityMap(JobId, _),
```
**Best job priority with unscheduled maps**

```prolog
bestJobPriorityMap(JobId, min<Priority>) :-
    job(JobId, JobPriority, JobStatus, Conf),
    task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
    Priority := [JobPriority, JobStatus.getSubmitTime()];
```

**Tracker workload**

```prolog
trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
    taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
    MapID := AttemptId.isMap() ? AttemptId : null,
    ReduceID := AttemptId.isMap() ? null : AttemptId;
```

**Schedule a new map attempt**

```prolog
taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
    task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),
    bestJobPriorityMap(JobId, _),
    trackerWorkload(@JobTracker, TaskTracker, MapCount, ReduceCount),
```
Best job priority with unscheduled maps

\[
\text{bestJobPriorityMap(JobId, min\langle Priority\rangle)} :- \\
\text{job(JobId, JobPriority, JobStatus, Conf)}, \\
\text{task(JobId, TaskId, TaskType.MAP, \ldots, TaskState.UNASSIGNED)}, \\
\text{Priority := [JobPriority, JobStatus.getSubmitTime()];}
\]

Tracker workload

\[
\text{trackerWorkload(@JobTracker, TaskTracker, count\langle MapID\rangle, count\langle ReduceID\rangle)} :- \\
\text{taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, \ldots)}, \\
\text{MapID := AttemptId.isMap() ? AttemptId : null,} \\
\text{ReduceID := AttemptId.isMap() ? null : AttemptId;}
\]

Schedule a new map attempt

\[
\text{taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING)} :- \\
\text{task(JobId, TaskId, TaskType.MAP, \_, InputLocation, \_, TaskState.UNASSIGNED)}, \\
\text{bestJobPriorityMap(JobId, \_),} \\
\text{trackerWorkload(@JobTracker, TaskTracker, MapCount, ReduceCount),} \\
\text{taskTracker(@JobTracker, TaskTracker, Hostname, \ldots, MaxMaps, MaxReduces, \ldots),}
\]
Best job priority with unscheduled maps

bestJobPriorityMap(JobId, min<Priority>) :-
    job(JobId, JobPriority, JobStatus, Conf),
    task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
    Priority := [JobPriority, JobStatus.getSubmitTime()];

Tracker workload

trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
    taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
    MapID := AttemptId.isMap() ? AttemptId : null,
    ReduceID := AttemptId.isMap() ? null : AttemptId;

Schedule a new map attempt

taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
    task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),
    bestJobPriorityMap(JobId, _),
    trackerWorkload(@JobTracker, TaskTracker, MapCount, ReduceCount),
    taskTracker(@JobTracker, TaskTracker, Hostname, ..., MaxMaps, MaxReduces, ...),
    MapCount < MaxMaps;
Best job priority with unscheduled maps

```prolog
bestJobPriorityMap(JobId, min<Priority>) :-
    job(JobId, JobPriority, JobStatus, Conf),
    task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
    Priority := [JobPriority, JobStatus.getSubmitTime()];
```

Tracker workload

```
trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
    taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
    MapID := AttemptId.isMap() ? AttemptId : null,
    ReduceID := AttemptId.isMap() ? null : AttemptId;
```

Schedule a new map attempt

```
taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
    task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),
    bestJobPriorityMap(JobId, _),
    trackerWorkload(@JobTracker, TaskTracker, MapCount, ReduceCount),
    taskTracker(@JobTracker, TaskTracker, Hostname, ..., MaxMaps, MaxReduces, ...),
    MapCount < MaxMaps;
```
Best job priority with unscheduled maps

```prolog
bestJobPriorityMap(JobId, min<Priority>) :-
  job(JobId, JobPriority, JobStatus, Conf),
  task(JobId, TaskId, TaskType.MAP, ..., TaskState.UNASSIGNED),
  Priority := [JobPriority, JobStatus.getSubmitTime()];
```

Tracker workload

```prolog
trackerWorkload(@JobTracker, TaskTracker, count<MapID>, count<ReduceID>) :-
  taskAttempt(@JobTracker, TaskTracker, AttemptId, TaskState.RUNNING, ...),
  MapID := AttemptId.isMap() ? AttemptId : null,
  ReduceID := AttemptId.isMap() ? null : AttemptId;
```

Schedule a new map attempt

```prolog
taskAttempt(@JobTracker, TaskTracker, new TaskAttemptID(JobId, TaskId, 0), TaskState.RUNNING, TaskPhase.STARTING) :-
task(JobId, TaskId, TaskType.MAP, _, InputLocation, _, TaskState.UNASSIGNED),
bestJobPriorityMap(JobId, _),
trackerWorkload(@JobTracker, TaskTracker, MapCount, ReduceCount),
taskTracker(@JobTracker, TaskTracker, Hostname, ..., MaxMaps, MaxReduces, ...),
MapCount < MaxMaps;
```
Task runners execute task attempts

```prolog
taskRunner(@TaskTracker, JobTracker, Id, Type, 0.0f, State, Phase, Runner) :-
    taskTrackerAction(@TaskTracker, JobTracker, TaskTrackerAction.ActionType.LAUNCH_TASK, Action),
    configuration(@TaskTracker, Host, HttpPort, MaxMaps, MaxReduces, TT)
{    
    Id := ((LaunchTaskAction) Action).getTask().getTaskID();
    Type := ((LaunchTaskAction) Action).getTask().isMapTask() ? TaskType.MAP : TaskType.REDUCE;
    State := TaskState.RUNNING;
    Phase := Type == TaskType.MAP ? TaskPhase.MAP : TaskPhase.SHUFFLE;
}
```

... and periodically update task attempt status

```prolog
taskAttempt(@JobTracker, TaskTracker, AttemptId, Progress, State, Phase, FileLocation, StartTime, FinishTime) :-
    heartbeat(@TaskTracker, ...),
    taskRunner(@TaskTracker, JobTracker, AttemptId, Type, Progress, State, Phase, Runner),
    configuration(@TaskTracker, Host, HttpPort, ...)
{    
    FileLocation := "http://" + Host + ":" + HttpPort.toString();
    StartTime := Runner.startTime();
    FinishTime := Runner.finishTime();
}
```
Figure 4: CDF of map and reduce task completion time (secs) running Hadoop and BOOM-MR over HDFS and BOOM-FS.

erations and BOOMFS protocol commands1 The resulting
BOOMFS implementation works with either vanilla Hadoop
MapReduce or BOOMMR1

Like GFSfi HDFS maintains an elegant separation of confl
\pagebreak

\noindent\textbf{3.2.2 File System State}

The first step of our rewrite was to represent file system
metadata as a collection of relations1 Expressing file system
\pagebreak

\noindent\textbf{Name} \hspace{2cm} \textbf{Description} \hspace{2cm} \textbf{Relevant attributes}

\begin{tabular}{|l|l|l|}
\hline
\textbf{file} & Files & fileid, parentfileid, name, isDir \\
\hline
\textbf{fchunk} & Chunks per file & chunkid, fileid \\
\hline
\textbf{datanode} & DataNode heartbeats & address, lastHeartbeatTime \\
\hline
\textbf{hb_chunk} & Chunk heartbeats & nodeAddr, chunkid, length \\
\hline
\end{tabular}

\noindent Table 2: Relations defining file system metadata.

The order in which the chunks appear in a file must also
be specifiedfi because relations are unordered1 In the current
systemfi we assign chunk IDs in a monotonically increasing
fashion and only support append operationsfi so the client can
\pagebreak

\noindent\textbf{The NameNode must ensure that the file system metadata}
is durablefi and restored to a consistent state after a failure1
This was easy to implement using Overlogfi because of the
natural atomicity boundaries provided by fixpoints1 We used
the Stasis storage library \cite{7} to achieve durabilityfi by writing
the durable state modifications to disk as an atomic transaction
at the end of each fixpoint1 We return to a discussion of Stasis
in Section 91

Since a file system is naturally hierarchicalfi it is a good
fit for a recursive query language like Overlog1 For exam-
\pagebreak

\noindent\textbf{Name} \hspace{2cm} \textbf{Description} \hspace{2cm} \textbf{Relevant attributes}

\begin{tabular}{|l|l|l|}
\hline
\textbf{ assortive} & assortive voter & assVote \\
\hline
\end{tabular}

\noindent\textbf{assortive} \hspace{2cm} \textbf{assortive voter} \hspace{2cm} \textbf{assVote}

\noindent\textbf{assortive voter} \hspace{2cm} \textbf{assortive voter} \hspace{2cm} \textbf{assVote}
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
- 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 $b > \text{nextBal}[q]$, 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 $b < \text{nextBal}[q].$)

3. After receiving a LastVote($b, v$) message from every priest in some majority set $Q$, where $b = \text{lastTried}[p]$, 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 $b = \text{nextBal}[q]$, 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 $b = \text{nextBal}[q].$)

5. If $p$ has received a Voted($b, q$) message from every priest $q$ in $Q$ (the quorum for ballot number $b$), where $b = \text{lastTried}[p]$, 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.
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
everything is data

- persistent stuff (e.g. FS metadata)
- runtime state (e.g. Hadoop bookkeeping)
- summary stats (e.g. LATE metrics)
- in-flight msgs and system events
- even parsed code
because everything is data...
  easy to design scale-out
  *interposition* (classic OS goal) easy via dataflow
  simpler concurrency?
  *data derivation vs. locks on object updates*
  *dataflow typing vs. state/event combinatorics*

all this applies to dataflow programming
  *e.g. mapreduce++*
  potentially sacrifice code analysis
LESSONS 3

🎯 overlog woes
  ● datalog syntax: hard to write, *really* hard to read
  ● operational semantics of unique keys and updates tricky

🎯 thematic issues
  ● distributed systems pseudocode often written as automata
    ● hence a lot of our Overlog looks like automata, rather than invariants
    ● fixable? automatically?
  ● java/overlog boundaries hard to pick, affect performance
data-centric cloud programming
peek at BOOM
towards lincoln
directions
WHY LINCOLN
WHY LINCOLN

Gettysburg:
WHY LINCOLN

Gettysburg:
  ✦ Edward Everett spoke for 2 hours
WHY LINCOLN

Gettysburg:

- Edward Everett spoke for 2 hours
- Abraham Lincoln spoke for 2 minutes
Gettysburg:

- Edward Everett spoke for 2 hours
- Abraham Lincoln spoke for 2 minutes
  - Everett praised Lincoln’s address as superior to his own
WHY LINCOLN
WHY LINCOLN

Prolog
WHY LINCOLN

Prolog

Datalog
WHY LINCOLN

Prolog
Datalog
Overlog
WHY LINCOLN

Prolog
Datalog
Overlog

Lincoln:
The Log that’s fun to play with!
OVERLOG’S STRENGTHS

- concise
- surprisingly broad suitability
  - graph algorithms and recursion (NW, Compilers, etc.)
  - asynchronous communication/rendezvous
  - distributed invariant assertion (PODC)
  - machine learning workhorses
    - dynamic programming, message-passing
- malleable
  - metaprogramming
  - reflection
  - interposition
ISSUES WITH OVERLOG/P2

- **syntax**
  - unfamiliar code structure (not like any popular language)
  - positional notation is clumsy, esp. with wide tables
    - have to write down variables/positions even when they’re unused
    - readability: hard to do variable-matching by eyeball

- **sloppy semantic definitions**
  - wishing globally, acting locally
  - state modification, time and atomicity still have corner cases
  - network delays and timeout
  - non-monotonicity (NOT, aggregates)
  - persistence, soft state, events
LINCOLN IDEAS

- underpinnings
- atomic deduction automata (ADAs)
- distributed ADAs (DADAs)
  - ADA+2PC
  - sometimes just ADA! analyze for this?
- bundles
  - reify stratification, transactions
  - logic equivalent of barriers
- decision rules
  - how to end a round of deduction (proceed past barriers)
  - timeout as a building block, leading to 2PC, etc
- host-language embedding?
data-centric cloud programming
peek at BOOM
towards lincoln
directions
DIRECTIONS

- lincoln
- exploit the advantage
  - use BOOM for speculative Hadoop development
    - energy, QP, cross-layer optimization (Hive/PIG), FS functionality
  - towards a more complete Cloudstack
    - multifaceted/ambitious look at storage consistency
    - cloud operator management
    - cloud service management
    - monitoring/prediction/control (w/Guestrin@CMU)
    - secure analytics, nets (w/DawnSong, Mitchell@Stanford, Feamster@GTU)
QUERIES?

http://www.declarativity.net
Data (stored).

Logic: what we can deduce from the data

p :- 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)?
DUSTY OLD DATALOG

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?

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

- \texttt{link}(X,Y,C) \leftarrow \texttt{COST}
- \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) <- COST

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

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

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

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

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

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

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

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

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

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

\begin{itemize}
  \item \texttt{link(X,Y)}
  \item \texttt{path(X,Y,Y,C) :- link(X,Y,C)}
  \item \texttt{path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)}
  \item \texttt{mincost(X,Z,min<C>) :- path(X,Z,Y,C)}
\end{itemize}
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

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

- \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)}
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)
\textbf{PARTITION SPECS INDUCE COMMUNICATION}

\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)
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

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

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

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

- 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

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

Localization Rewrite

<table>
<thead>
<tr>
<th>link_d:</th>
<th>a b l</th>
<th>a b l</th>
<th>b c l</th>
<th>c d l</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c b l</td>
<td></td>
<td>d c l</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>path:</th>
<th>a b b l</th>
<th>b a a l</th>
<th>c d d l</th>
<th>d c c l</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b c c l</td>
<td></td>
<td>d c c l</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>link:</th>
<th>a b l</th>
<th>b a l</th>
<th>c d l</th>
<th>d c l</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b l</td>
<td></td>
<td>c d l</td>
<td></td>
</tr>
</tbody>
</table>

Localization Rewrite
PARTITION SPECS INDUCE COMMUNICATION

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

Localization Rewrite

```
link_d:  b a l  a b l  a b l  b c l  b c l  c d l  c d l  d c l  d c l
path:   a b b l  b a a l  c d d l  c d c l  d c c l  d c c l  d c c l
link:   a b l  a b l  a b l  b c l  b c l  c d l  c d l  d c l  d c l
```
PARTITION SPECS INDUCE COMMUNICATION

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

<table>
<thead>
<tr>
<th>link(@X,Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>path(@X,Y,Y,C) :- link(@X,Y,C)</td>
</tr>
<tr>
<td>link_d(X,@Y,C) :- link(@X,Y,C)</td>
</tr>
<tr>
<td>path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)</td>
</tr>
</tbody>
</table>

Localization Rewrite

<table>
<thead>
<tr>
<th>link_d:</th>
<th>b</th>
<th>a</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>b</td>
<td>l</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>c</td>
<td>l</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>d</td>
<td>l</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>path:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b b l</td>
</tr>
<tr>
<td>a c b 2</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>link:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b l</td>
</tr>
<tr>
<td>b c l</td>
</tr>
<tr>
<td>c d l</td>
</tr>
</tbody>
</table>

THIS IS DISTANCE VECTOR

a — b — c — d
OVERLOG EXECUTION

Network → Clock → Events → Datalog → Events → Network

Datalog
Local, atomic computation

Network Boundary

Java → Network → Java
P2 @ 10,000 FEET

Overlog

Parser

AST

Planner

Tables

Dataflow

Scheduler

Net
Parser

Net

Tables

Dataflow

Overlog

java, ruby

P2 @ 10,000 FEET
P2 @ 10,000 FEET

java, ruby

Parser

Overlog

Net

Tables

Dataflow


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
Insert
Insert

Network In
Mux
Queue
TimedPullPush 0
Demux (tuple name)

Dup

Network Out
RoundRobin
TimedPullPush 0
Queue

finger
node
bestSucc

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>LOC</td>
<td>560 (NesC)</td>
<td>13 rules, 25 lines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Sz</th>
<th>Data Sz</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.3KB</td>
<td>24.4KB</td>
</tr>
<tr>
<td>0.4KB</td>
<td>4.1KB</td>
</tr>
</tbody>
</table>
## 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
EVITA RACED:
OVERLOG METACOMPILER

DECLARATIVE
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
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

RULE1: Dist(S,D):- At(S,P),
         P_d = destination(),
         D = |P - P_d|,
         D > robot radius.

RULE2: Farther(S,T):- Neighbor(S,T),
         Dist(S,D_S),
         Dist(T,D_T),
         D_S ≥ D_T.

RULE3: MoveAround(S,T,U):- Farther(S,T),
         Farther(S,U),
         U ≠ T.

(i) starting location at origin
(ii) robot a moves around b
(iii) robot a finishes moving
(iv) robot c now moves around a
(v) robot c finishes moving

[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