Dedalus: Datalog in Time and Space

Peter Alvaro, Ras Bodík, *Neil Conway*, Joe Hellerstein, David Maier (PSU), Bill Marczak, and Rusty Sears (Yahoo! Research)

2010 Berkeley OSQ Retreat

Dedalus

- Dedalus is a declarative language for distributed programming
- Grounded in our experiences using declarative languages to build distributed systems
  - Declarative Networking (2003-2008)
  - The BOOM Project (2008-Present)

Outline

1. Context and Motivation
   - Declarative Networking
   - Declarative Systems: BOOM
   - A taste of Overlog
2. Dedalus: Datalog in Time and Space
3. Future Directions and Open Problems

Declarative Networking

- Networking is about moving data from one location to another
- Can we view networking as a distributed data management problem?
  - E.g., can we express a routing protocol as a distributed query in a declarative language?
- Yes: transport protocols, routing protocols, sensor networks, DHTs, replication policies, distributed snapshots, consensus protocols, ...
  - Typically 10x reduction in code size

B.T. Loo, T. Condie, M. Garofalakis, D.E. Gay, J.M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, I. Stoica. ***Declarative Networking***. CACM, 2009

Declarative Systems

- Focus has turned from **protocols** toward distributed **systems**
  - Larger programs
  - More complex algorithms
- BOOM: **B**erkeley **O**rders **O**f **M**agnitude
  - OOM more scale, OOM less code
  - Goal: A complete cloud computing stack built using declarative languages
  - Could we build Google in 10 kLOC?

Overlog: Distributed Datalog

- Datalog: a recursive query language from the deductive database community
  - Defined over a static database
- Add state update, distributed queries (communication)

Datalog Example

Rule Body
(conjunction of terms)

Rule Head

```
path(X, Y, C) :- link(X, Y, C);
path(X, Z, C1 + C2) :- link(X, Y, C1),
                       path(Y, Z, C2);
mincost(X, Z, min<C>) :- path(X, Z, C);
```

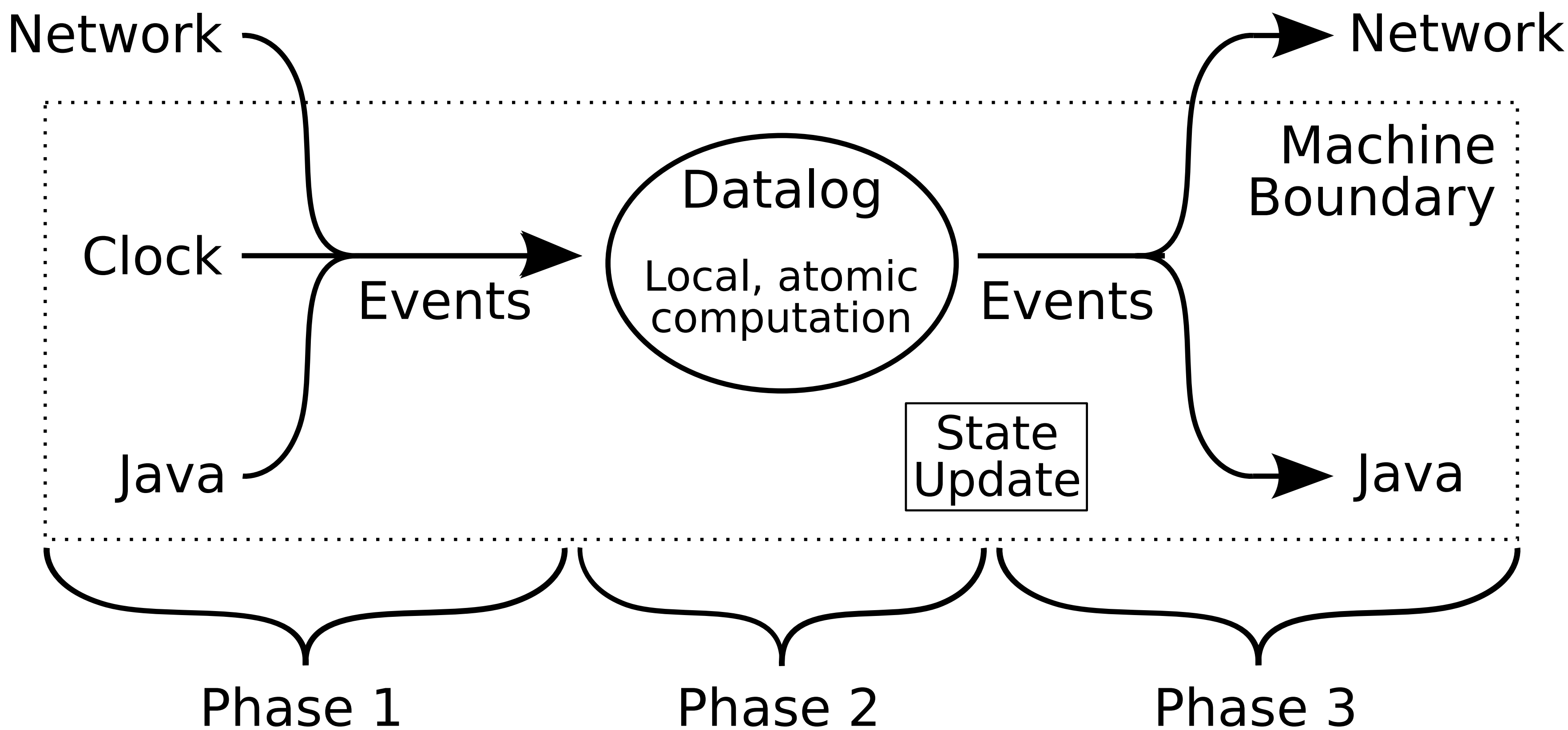
Transitive
Closure

Overlog Example

Distributed join

```
path(@X, Y, C) :- link(@X, Y, C);
path(@X, Z, C1 + C2) :- link(@X, Y, C1),
                        path(@Y, Z, C2);
mincost(@X, Z, min<C>) :- path(@X, Z, C);
```

Overlog Timestep Model



Outline

1. Context and Motivation
   – Declarative Networking
   – Declarative Systems: BOOM
   – A taste of Overlog
2. Dedalus: Datalog in Time and Space
3. Future Directions and Open Problems

Dedalus

- Datalog = The Good Stuff
  - Precise semantics, established techniques for optimization and evaluation
- In Overlog, the Hard Stuff happens **between** time steps
  - State update
  - Asynchronous messaging
- Can we talk about the Hard Stuff with logic?

State Update

```
sequence(A, Val + 1) :- sequence(A, Val),
                        event(A);
```

- How do we interpret this?
  - Datalog: infinite database
  - Overlog: runtime deletes old version of tuple
- Overlog: **ugly**, "outside" of logic, ambiguous
  - Semantics defined by the implementation
- Hence, difficult to express common patterns
  - Queues, sequencing
- Order doesn't matter ... except when it does!

Asynchronous Messaging

Logical interpretation unclear:

```
p(@A, B) :- q(@B, A);
```

Asynchronous Messaging

Logical interpretation unclear:

p(@A, B) :- **q**(@B, A);

- Overlog "@" notation describes **space**
- Upon reflection, ***time*** is more fundamental
  - Model failure with arbitrary delay

Dedalus: Datalog in Time

(1) Deductive rule: (Pure Datalog)

```
p(A, B) :- q(A, B);
```

(2) Inductive rule: (Constraint across "next" timestep)

```
p(A, B)@next :- q(A, B);
```

(3) Async rule: (Constraint across arbitrary timesteps)

```
p(A, B)@async:- q(A, B);
```

Dedalus: Datalog in Time

(1) Deductive rule: (Pure Datalog)

All terms in body have same time

p(A, B, S) :- **q**(A, B, T), T = S;

(2) Inductive rule: (Constraint across "next" timestep)

p(A, B, S) :- **q**(A, B, T), **successor**(T, S);

(3) Async rule: (Constraint across arbitrary timesteps)

p(A, B, S) :- **q**(A, B, T), **time**(S),
choose((A, B, T), (S));

State Update in Dedalus

p(A, B)@next :- **p**(A, B), notin **p_del**(A, B);

Example Trace:

p(1, 2)@101;

p(1, 3)@102;

p_del(1, 3)@300;

Time	p(1, 2)	p(1, 3)	p_del(1, 3)
101			
102			
…			
300			
301			

Sequences in Dedalus

sequence(A, Val + 1)@next :-
 sequence(A, Val),
 event(A);

sequence(A, Val)@next :-
 sequence(A, Val),
 notin **event**(A);

Asynchrony in Dedalus

Unreliable Broadcast in Dedalus:

```
sbcast(#Target, Sender, Message)@async :-
  new_message(#Sender, Message),
  members(#Sender, Target);
```

- More satisfactory logical interpretation
- Can build Lamport clocks, reliable broadcast, etc.
- What about "space"?
  - Space is the unit of atomic deduction w/o partial failure

Asynchrony in Dedalus

Unreliable Broadcast in Dedalus:

Include sender's local time

```
sbcast(#Target, Sender, T, Message)@async :-
  new_message(#Sender, Message)@T,
  members(#Sender, Target)@T;
```

- More satisfactory logical interpretation
- Can build Lamport clocks, reliable broadcast, etc.
- What about "space"?
  - Space is the unit of atomic deduction w/o partial failure

Dedalus Summary

- Logical, model-theoretic semantics for two key features of distributed systems
  1. Mutable state
  2. Asynchronous communication
- All facts are transient
  - Persistence and state update are explicit
- Has been successful in clarifying the semantics of our programs

Outline

1. Context and Motivation
   – Declarative Networking
   – Declarative Systems: BOOM
   – A taste of Overlog
2. Dedalus: Datalog in Time and Space
3. Future Directions and Open Problems

Big Picture Agenda

1. Language
   - **Overlog**: concise code
   - **Dedalus**: precise semantics
   - **C4**: efficient execution (new language runtime)
   - **Bloom**: friendly syntax, “mainstream” appeal
2. BOOM Project
   - Build more systems using logic (e.g., Cassandra)
   - Move up the stack? (Business logic, GUIs, ...)

Verification of Dedalus programs

- Premises:
  - Program expressed as a set of logical implications
  - All asynchrony/non-determinism is explicit
  - “Close to the specification” but still executable
- Conclusion: easier verification?
  - Programmer does (some) of the abstraction for us
- Can we integrate formal verification into the development process?

Network-Oriented Optimization

- Traditional compiler optimization is node-oriented
- The big wins are in network-oriented optimizations
  - Given program for **n** nodes, execute using **m** nodes
    - Given $100, what is the best cluster configuration?
  - Automatically colocate code and data
  - Co-optimize application logic and network protocols
    - E.g., if program transitions are commutative, consensus is cheaper
- As cloud computing environments become more complex and unpredictable, automatic optimization will be crucial

Questions?

Thank you! http://boom.cs.berkeley.edu

Initial Publications:

- *BOOM Analytics*: EuroSys'10, Alvaro et al.
- *Paxos in Overlog*: NetDB'09, Alvaro et al.
- *Dedalus*: UCB TR #2009-173, Alvaro et al.