Cloud Programming: From Doom and Gloom to BOOM and Bloom

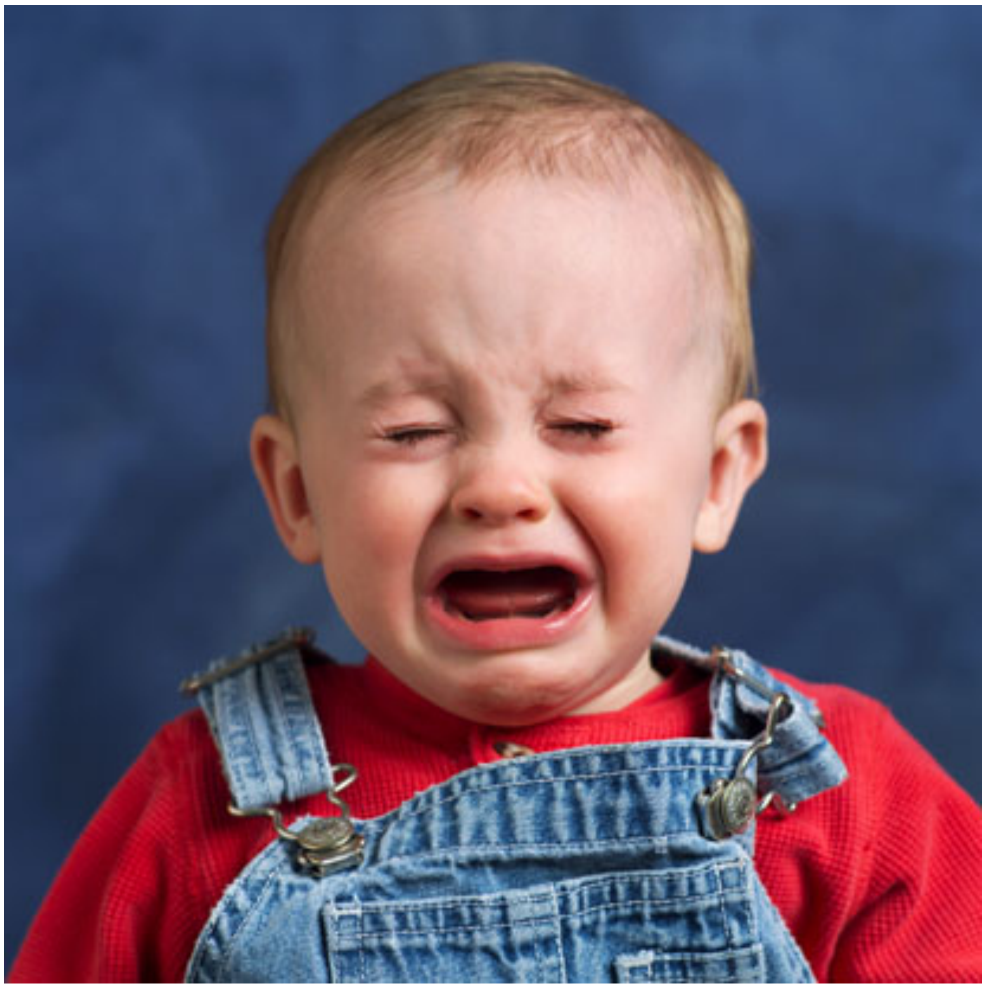
Neil Conway
UC Berkeley

Joint work with Peter Alvaro, Ras Bodik, Tyson Condie, Joseph M. Hellerstein, David Maier (PSU), William R. Marczak, and Russell Sears (Yahoo! Research)

Datalog 2.0 Workshop

Cloud Computing: The Next Great Computing Platform

The Problem



Writing reliable, scalable distributed software remains **extremely difficult**.

Doom and Gloom!

"...when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced .... I would be **panicked** if I were in industry."

-- John Hennessey, Stanford

A Ray of Light

- We understand **data-parallel** computing
  - MapReduce, parallel DBs, etc.
- Can we take a hard problem and transform it into an easy one?

Everything is Data

- Distributed computing is all about **state**
  - System state
  - Session state
  - Protocol state
  - User and security-related state
  - ... and of course, the actual “data”
- Computing = Creating, updating, and communicating that state

Datalog to the Rescue!

1. Data-centric programming
   – Explicit, uniform state representation: relations
2. High-level declarative queries
   – Datalog + asynchrony + state update

Outline

1. The **BOOM** Project
   - Cloud Computing stack built w/ distributed logic
   - BOOM Analytics: MapReduce and DFS in Overlog
2. **Dedalus**: Datalog in Time (and Space)
3. (Toward) The Bloom Language
   - Distributed Logic for Joe the Programmer

The BOOM Project

- **B**erkeley **O**rders **O**f **M**agnitude
  - OOM more scale,
    OOM less code
  - Can we build Google in 10k LOC?
- Build "Real Systems" in distributed logic
  - Begin w/ an existing variant of Datalog ("Overlog")
  - Inform the design of a new language for distributed computing (Bloom)

BOOM Analytics

- Typical "Big Data" stack: MapReduce (Hadoop) + distributed file system (HDFS)
- We tried two approaches:
  - HDFS: *clean-slate* rewrite
  - Hadoop: replace job scheduling logic w/ Overlog
- Goals
  1. Replicate existing functionality
  2. Add Hard Stuff (and make it look easy!)

Overlog: Distributed Datalog

- Originally designed for routing protocols and overlay networks (Loo *et al.*, SIGMOD'06)
  - Routing = recursive query over distributed DB
- Datalog w/ aggregation, negation, functions
- Distribution = horizontal partitioning of tables
  - Data placement *induces* communication

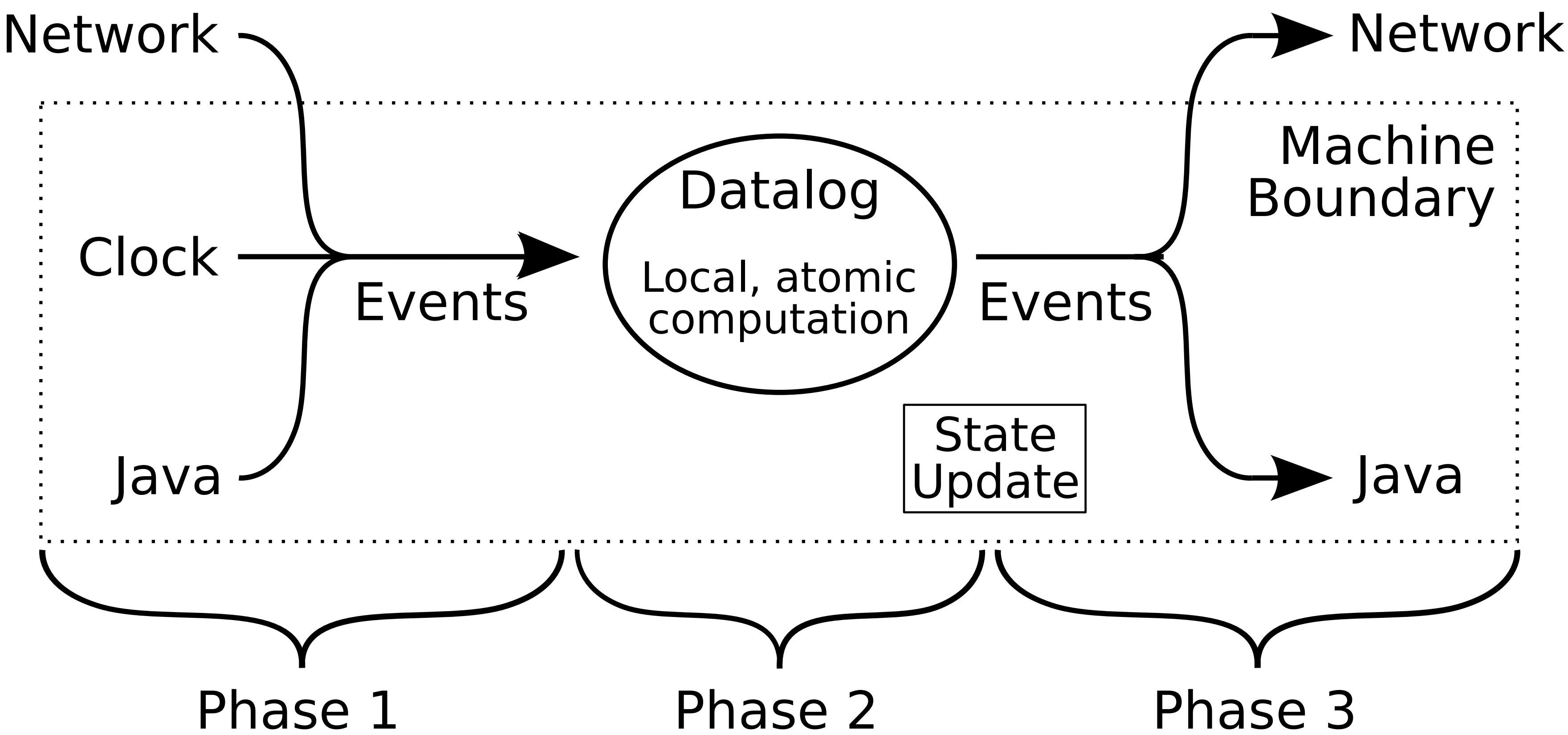
Yet Another Transitive Closure

```
link(X, Y, C);
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 Example

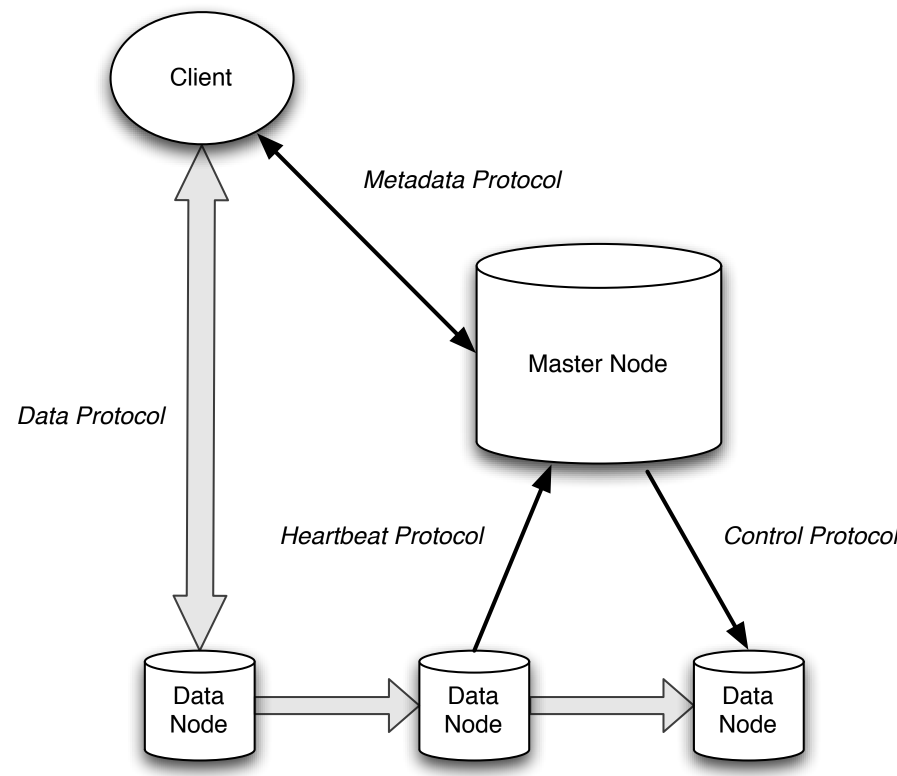
```
link(@X, Y, C);
path(@X, Y, C) :- link(@X, Y, C);
path(@X, Z, C1 + C2) :- link(@X, Y, C1),
                        path(@Y, Z, C2);
mincost(@X, Z, min<C>) :- path(@X, Z, C);
```

Overlog Timestep Model



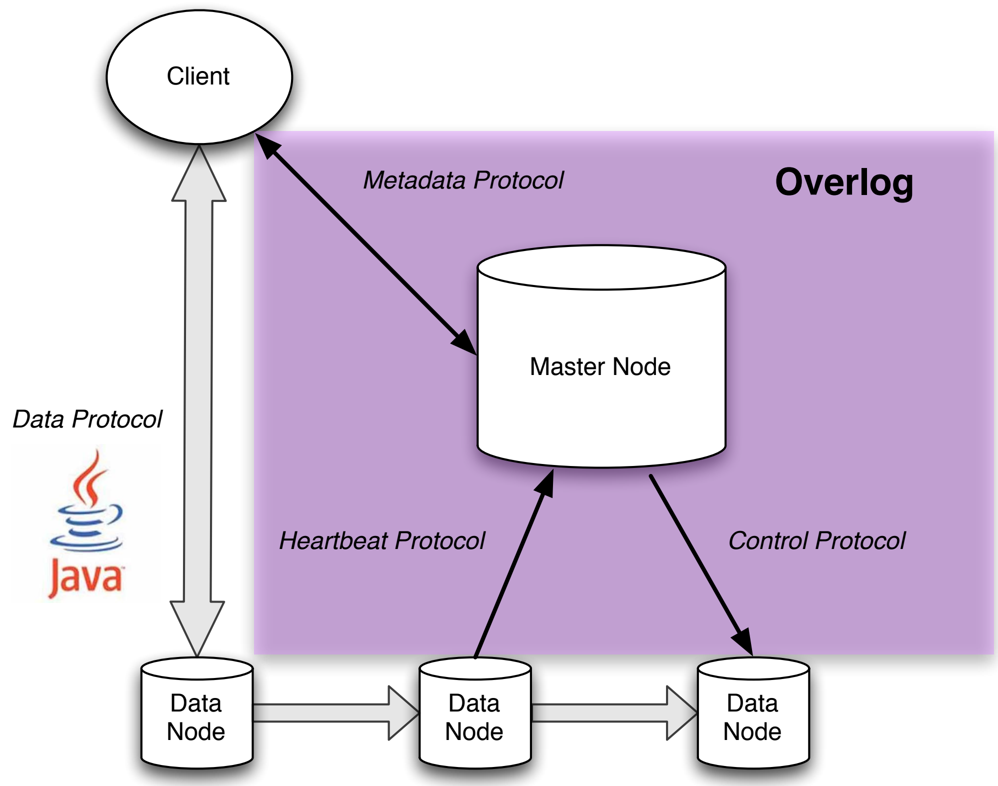
Hadoop Distributed File System

- Based on the Google File System (*SOSP'03)*
  - Large files, sequential workloads, append-only
  - Used by Yahoo!, Facebook, etc.
- Chunks 3x replicated at data nodes for fault tolerance



BOOM-FS

- Hybrid system
  - Complex logic: Overlog
  - Performance-critical (but simple!): Java
- Clean separation between policy and mechanism



BOOM-FS Example: State

Represent file system metadata with relations.

Name	Description	Attributes
file	Files	fileID, parentID, name, isDir
fqpath	Fully-qualified path names	fileID, path
fchunk	Chunks per file	chunkID, fileID
datanode	DataNode heartbeats	nodeAddr, lastHbTime
hb_chunk	Chunk heartbeats	nodeAddr, chunkID, length

BOOM-FS Example: Query

Represent file system metadata with relations.

```
// Base case: root directory has null parent
fqpath(FileId, Path) :-
    file(FileId, FParentId, FName, IsDir),
    IsDir = true, FParentId = null, Path = "/";

fqpath(FileId, Path) :-
    file(FileId, FParentId, FName, _),
    fqpath(FParentId, ParentPath),
    // Do not add extra slash if parent is root dir
    PathSep = (ParentPath = "/" ? "" : "/"),
    Path = ParentPath + PathSep + FName;
```

BOOM-FS Example: Query

Distributed protocols: join between event stream and local database

```
// "ls" for extant path => return listing for path
response(@Source, RequestId, true, DirListing) :-
    request(@Master, RequestId, Source, "Ls", Path),
    fqpath(@Master, FileId, Path),
    directory_listing(@Master, FileId, DirListing);

// "ls" for nonexistent path => error
response(@Source, RequestId, false, null) :-
    request(@Master, RequestId, Source, "Ls", Path),
    notin fqpath(@Master, _, Path);
```

Comparison with Hadoop

Competitive performance (~20%)

	Lines of Java	Lines of Overlog
HDFS	~21,700	0
BOOM-FS	1,431	469

New Features:

1. Hot Standby for FS master nodes using Paxos
2. Partitioned FS master nodes for scalability
   – ~1 day!
3. Monitoring, tracing, and invariant checking

Lessons from BOOM Analytics

- Overall, Overlog was a good fit for the task
  - Concise programs for real features, easy evolution
- Data-centric design: language-independent
  - Replication, partitioning, monitoring all involve data management
  - Node-local invariants are "cross-cutting" queries
    - Specification *is* enforcement
- Policy vs. mechanism ⇔ Datalog vs. Java

Challenges from BOOM Analytics

- Poor perf, cryptic syntax, little/no tool support
  - Easy to fix!
- Many bugs related to updating state
  - Ambiguous semantics (in Overlog)
- We avoided distributed queries
  - "The global database is a lie!"
  - Hand-coding protocols vs. stating distributed invariants

Outline

1. The **BOOM** Project
   – Cloud Computing stack built w/ distributed logic
   – BOOM Analytics: MapReduce and DFS in Overlog
2. **Dedalus**: Datalog in Time (and Space)
3. (Toward) The Bloom Language
   – Distributed Logic for Joe the Programmer

Dedalus

- Dedalus: a theoretical foundation for Bloom
- In Overlog, the Hard Stuff happens **between** time steps
  - State update
  - Asynchronous messaging
- Can we talk about the Hard Stuff with logic?

State Update

- Updates in Overlog: **ugly**, “outside” of logic
- Difficult to express common patterns
  - Queues, sequencing
- Order doesn’t matter ... except when it does!

```
counter(@A, Val + 1) :- counter(@A, Val),
                        event(@A, _);
```

Asynchronous Messaging

- Overlog "@" notation describes **space**
- Logical interpretation unclear:

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

- Upon reflection, ***time*** is more fundamental
  - Model failure with arbitrary delay
- Discard illusion of global DB

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_neg(A, B);

p (1, 2)@101;

p (1, 3)@102;

p_neg(1, 2)@300;

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

Counters in Dedalus

```
counter(A, Val + 1)@next :-
    counter(A, Val),
    event(A, _);

counter(A, Val)@next :-
   counter(A, Val),
   notin event(A, _);
```

Asynchrony in Dedalus

Unreliable Broadcast:

```
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:

Project sender's local time

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

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
- Initial correctness checks
  1. Temporal stratifiability ("modular stratification in time")
  2. Temporal safety ("eventual quiescence")

Directions: Bloom

1. Bloom: Logic for Joe the Programmer
   – Expose sets, map/reduce, and callbacks?
   – Translation to Dedalus
2. Verification of Dedalus programs
3. Network-oriented optimization
4. Finding the right abstractions for Distributed Computing
   – Hand-coding protocols vs. stating distributed invariants
5. Parallelism and monotonicity?

Questions?

Thank you! http://declarativity.net

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.

Temporal Stratifiability

Reduction to Datalog not syntactically stratifiable:

```
p(X)@next :- p(X), notin p_neg(X);
p_neg(X) :- event(X, _), p(X);
```