BOOM Analytics: Exploring Data-Centric, Declarative Programming for the Cloud

Neil Conway*
http://boom.cs.berkeley.edu

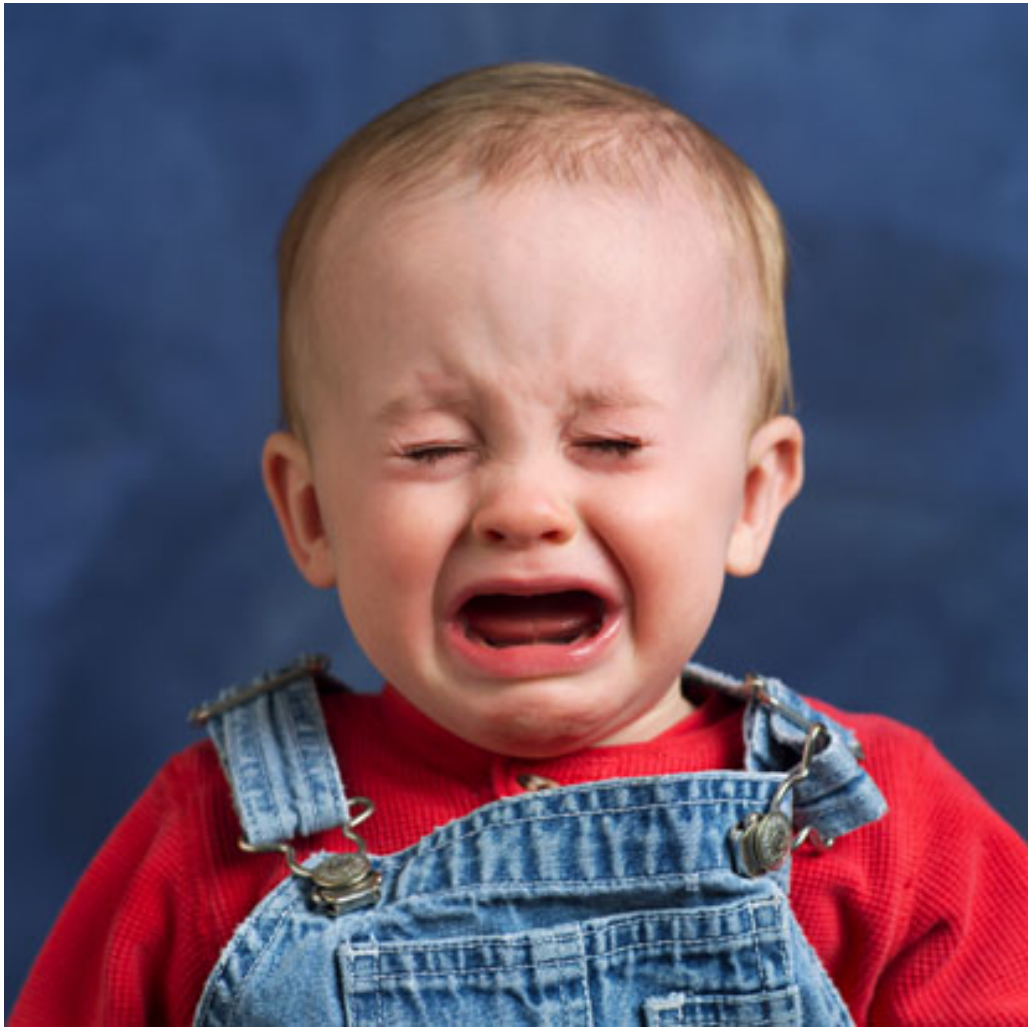
Joint work with Peter Alvaro*, Tyson Condie*,
Khaled Elmeleegy†, Joseph M. Hellerstein*, Russell Sears†

* UC Berkeley † Yahoo! Research

Cloud Computing: The Next Major Computing Platform

amazon web services™ Windows Azure Google™ App Engine force.com™ platform as a service

The Challenge



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

Raising The Level of Abstraction

- Currently, distributed programming is both **difficult** and **tedious**
  - Essence of a distributed algorithm is swamped by mundane details: fine-grained locking, messaging, serialization/deserialization, event loops, ...
- A good language should let the programmer focus on the difficult stuff
  - (... and handle the tedious stuff automatically)

Everything is Data

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

Two Design Principles

1. Data-centric programming
   - Explicit, uniform state representation
     - We chose relations; could use XML, graphs, etc.
   - Language independent, to some extent
2. High-level declarative queries
   - Start with a recursive query language (*Datalog*)
   - Add communication and state update

Agenda

- **Long-term agenda**: Build a broad range of cloud software using data-centric programming and declarative languages
  - BOOM Project (Berkeley Orders of Magnitude)
- **This talk**: Our experience using this design style to implement a “Big Data” analytics stack
  - BOOM Analytics: Hadoop + HDFS rebuilt using a declarative language

Which language to use?

- Eventual goal: design a new distributed logic language for cloud computing
  - For now, we used the language we had in-house
- Overlog is a declarative language for writing routing protocols and overlay networks
  - P2 Project: *SIGCOMM'*05, *SOSP*'05, *SIGMOD*'06
- Support for recursion, aggregation, negation, distributed queries (network communication)

Evaluation Criteria

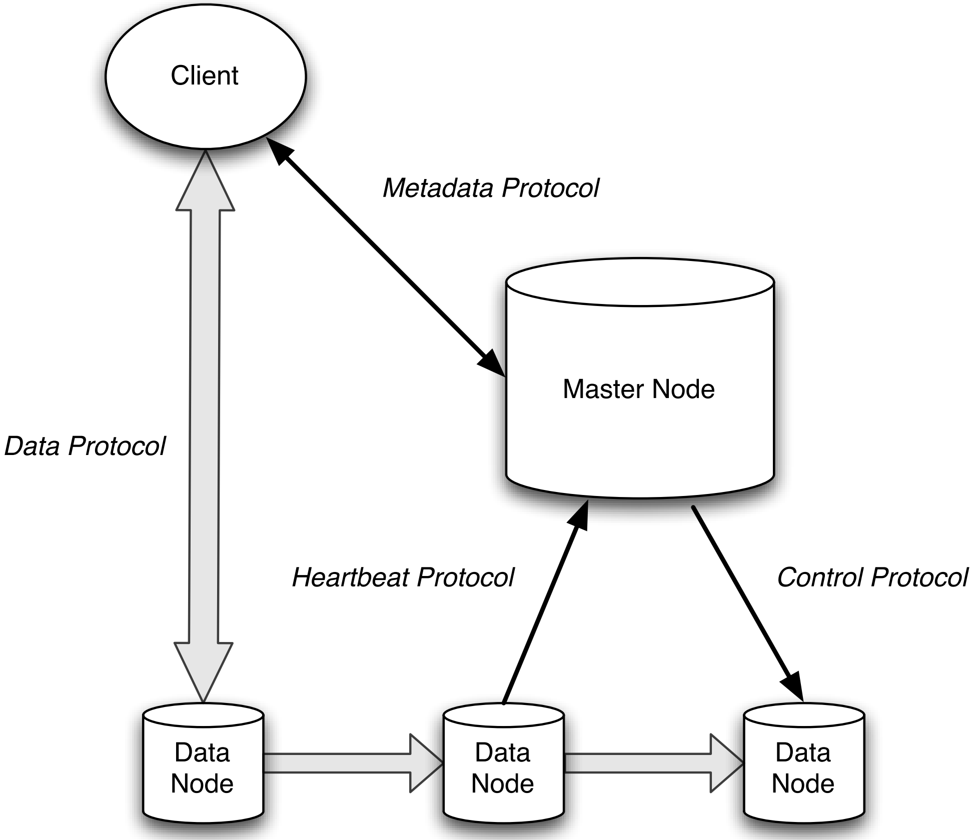
- How to compare the systems we built with traditional implementations?
- We use three metrics:
  1. **Performance**
     - Goal: rough performance parity
  2. **Code size** (lines of source code)
  3. **Ease of evolution**
     - Can we quickly evolve our software to add complex new distributed features?

Outline

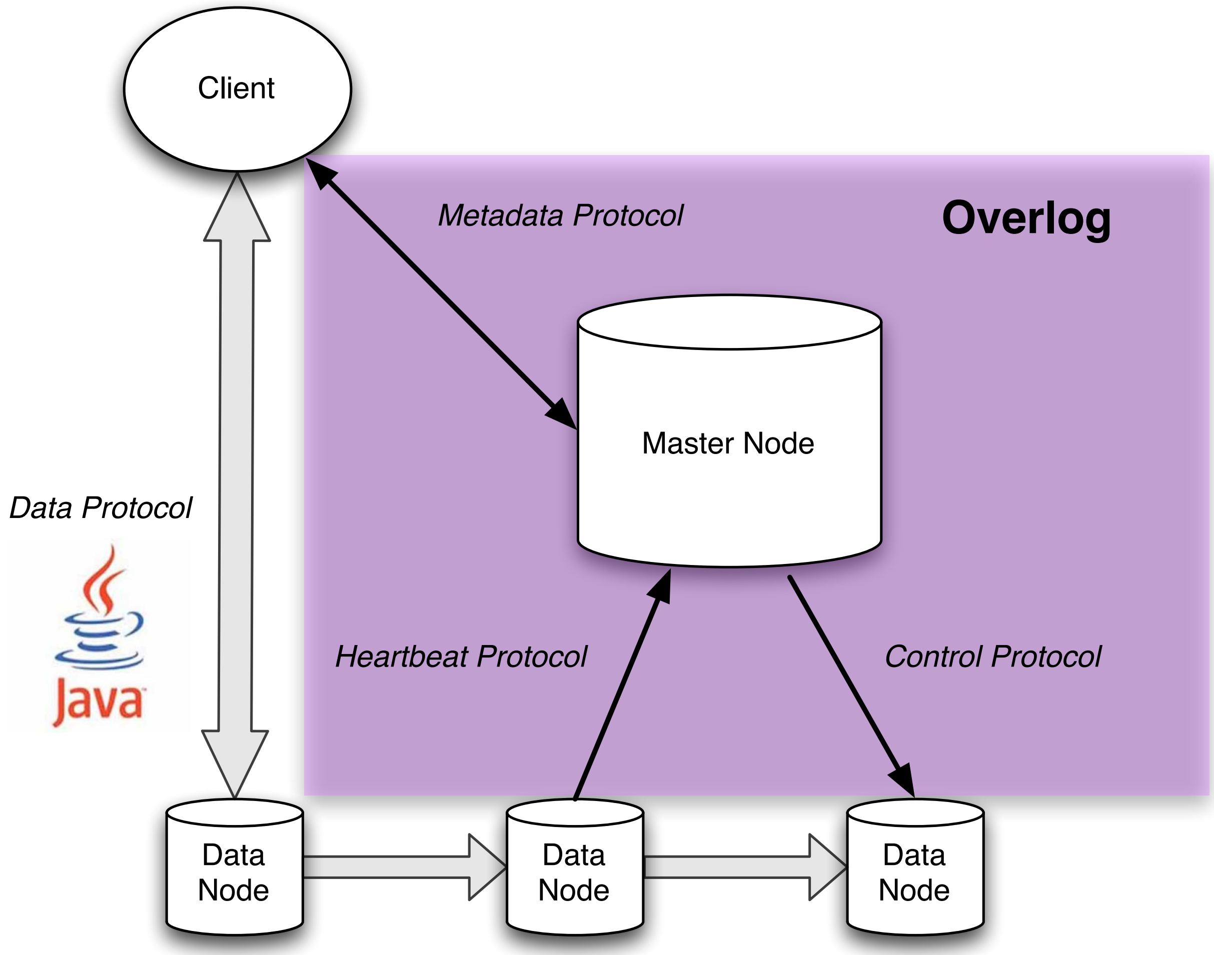
1. BOOM-FS: HDFS in distributed logic
2. Evolving BOOM-FS
3. BOOM-MR: MapReduce scheduling in logic
4. Lessons Learned

HDFS Architecture

- Based on the Google File System (*SOSP'03)*
  - Large files, sequential workloads, append-only
- Chunks replicated at data nodes for fault tolerance
  - Each chunk ≥ 64MB



BOOM-FS Architecture



BOOM-FS Example: State

Represent file system metadata with relations.

Relation 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, time
hb_chunk	Chunk heartbeats	nodeAddr, chunkID, length

BOOM-FS Example: Query

File system behavior = queries over relations.

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

Rule head

Rule body

BOOM-FS Example: Query

File system behavior = queries over relations.

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

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

Transitive Closure

fqpath Example

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

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

file			
FileId	**ParentId**	**Name**	**IsDir**

fqpath	
FileId	**Path**

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true

fqpath

FileId	Path

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true

fqpath

FileId	Path
1	/

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
2	**1**	**foo**	**true**

fqpath

FileId	Path
1	/

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
2	**1**	**foo**	**true**

fqpath

FileId	Path
1	/
2	**/foo**

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
2	1	foo	true
3	**2**	**bar**	**false**

fqpath

FileId	Path
1	/
2	/foo

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
2	1	foo	true
3	**2**	**bar**	**false**

fqpath

FileId	Path
1	/
2	/foo
3	**/foo/bar**

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
~~2~~	~~1~~	~~foo~~	~~true~~
3	2	bar	false

fqpath

FileId	Path
1	/
2	/foo
3	/foo/bar

fqpath Example

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

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

file

FileId	ParentId	Name	IsDir
1	null	null	true
~~2~~	~~1~~	~~foo~~	~~true~~
3	2	bar	false

fqpath

FileId	Path
1	/
~~2~~	~~/foo~~
~~3~~	~~/foo/bar~~

“View maintenance”

Distributed Query Example

“Location specifier”
(node addr)

```
request(@Master, RequestId, Source, ReqType, Arg) :-
   client_request(@Source, RequestId, ReqType, Arg),
   master_addr(@Source, Master);
```

Distributed Query Example

```
request(@Master, RequestId, Source, ReqType, Arg) :-
   client_request(@Source, RequestId, ReqType, Arg),
   master_addr(@Source, Master);

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

Event
streams

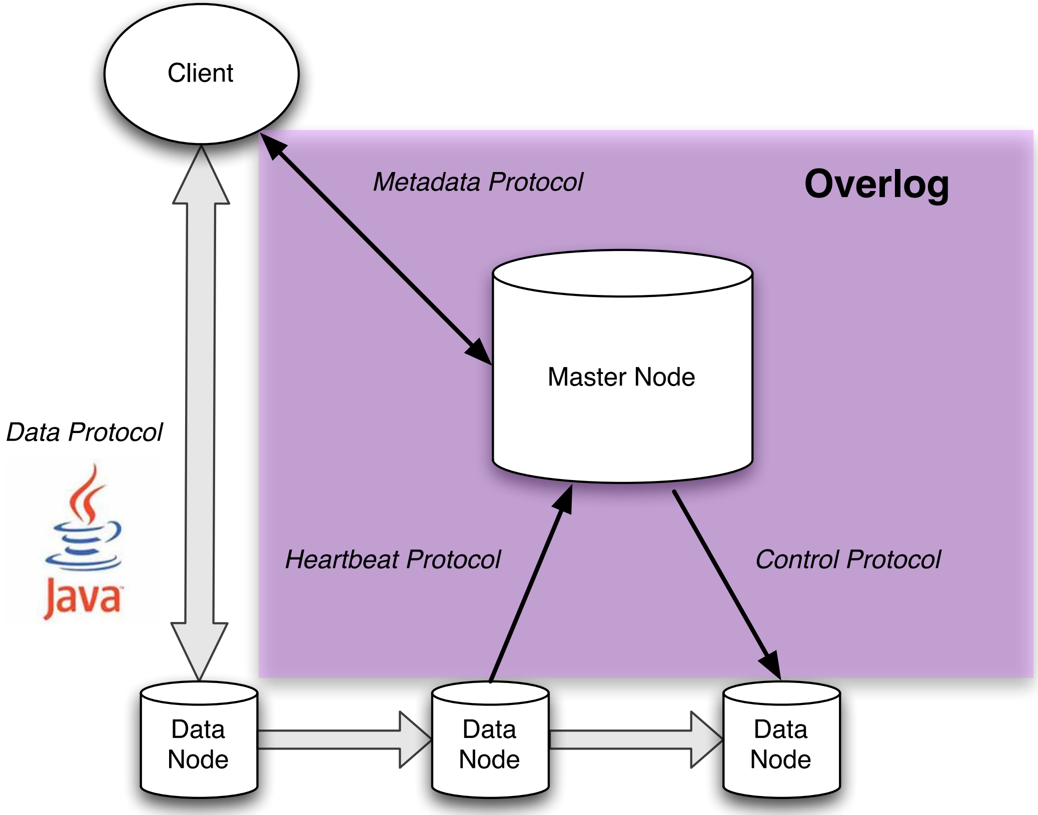
Distributed Query Example

```
request(@Master, RequestId, Source, ReqType, Arg) :-
   client_request(@Source, RequestId, ReqType, Arg),
   master_addr(@Source, Master);

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

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

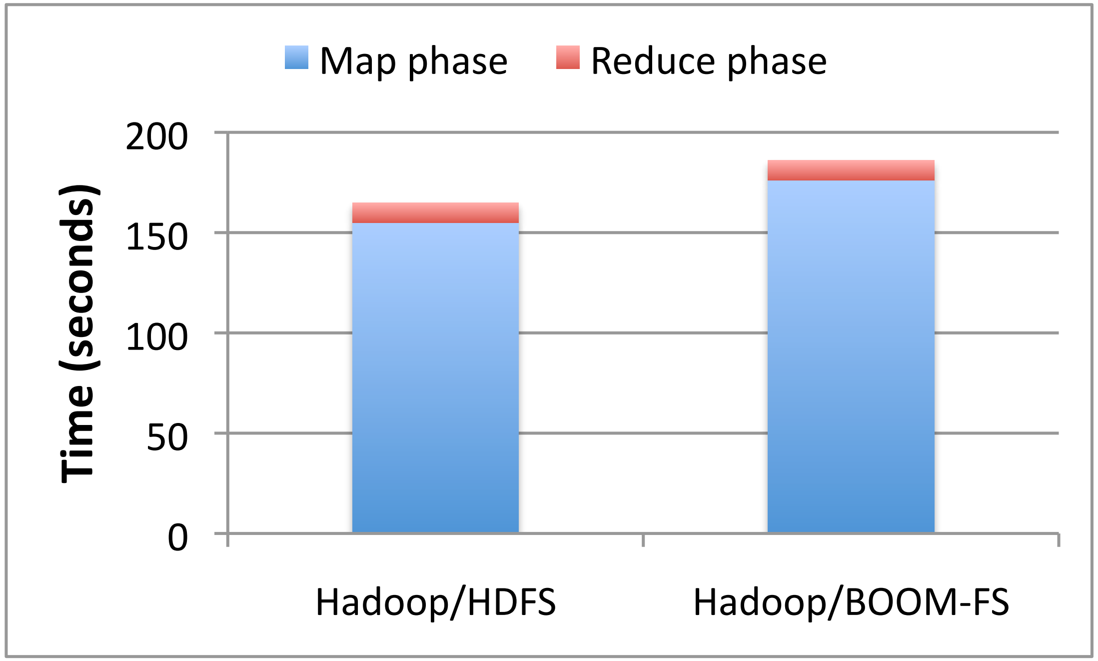
Disjunction

BOOM-FS Architecture

- Hybrid system
  - Complex logic: Overlog
  - Performance-critical (but simple!): Java
- Separation of policy and mechanism



Performance Comparison



- Workload: 30GB word count (average of 5 runs)
  - 481 map tasks, 100 reduce tasks
- 101 node cluster on Amazon EC2
- BOOM-FS is ~15% slower for map phase (read)
- Lots of room for improvement
  - E.g., overlapping of map function and network I/O

Code Size Comparison

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

- 9 months, 4 grad student developers
    - Most work in 3 month span

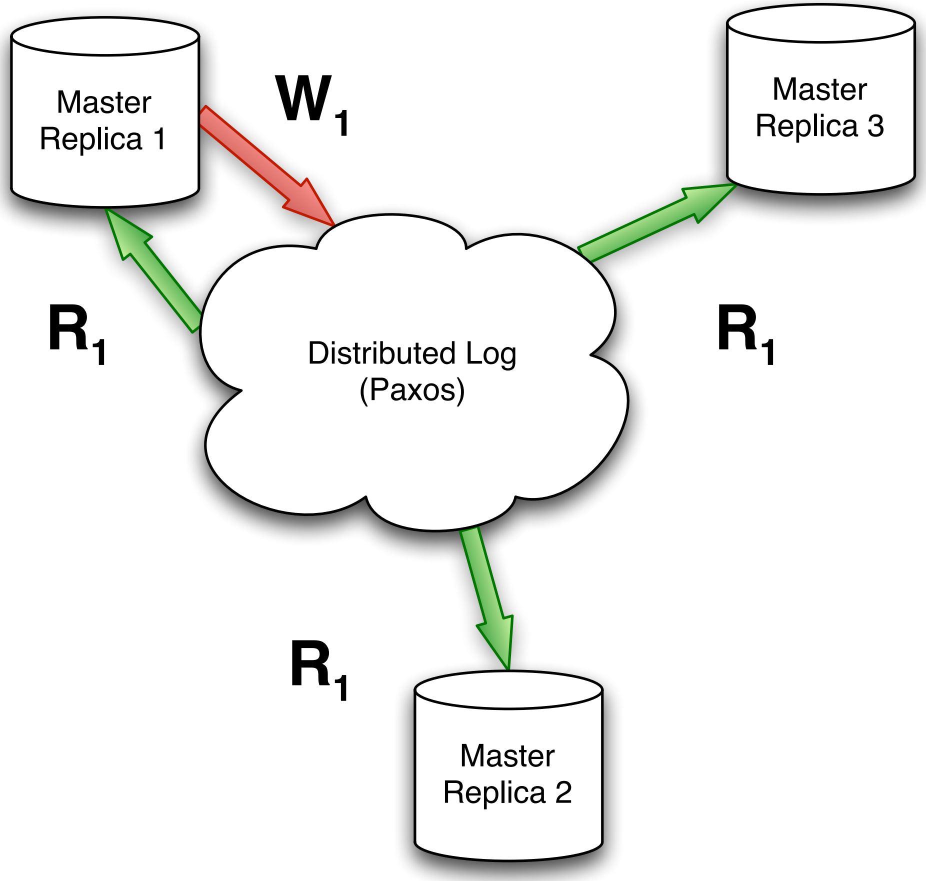
Rapid Evolution of BOOM-FS

We added 3 kinds of functionality to the base BOOM-FS design:

1. High availability for master nodes
2. Improved scalability for master nodes
3. Monitoring and debugging tools

High-Availability Rev

- HDFS: single point of failure at master
- We built hot standby using Paxos
  - **Invariant**: Before applying operation *i*, pass *i* through log



Scalability Rev

- HDFS: All FS metadata **in memory** at master
  - More metadata => buy more RAM!
- One solution: partition master state across multiple machines ("horizontal partitioning")
- Very hard in HDFS
- Took ~1 day in BOOM-FS
  - Hash partitioning for FS metadata
  - Broadcast or unicast FS operations, as appropriate
  - Composes naturally with high availability support

Monitoring Rev

- Overlog allows natural system introspection
  - "Everything is data": just query it!
- Logging and monitoring = distributed queries over a distributed database
  - E.g., record per-machine monitoring stats
  - E.g., record execution counts for each rule
- Invariant checking = query over local database
  - E.g., no file has ≠ 1 fqpath entries
  - No private state: rules are "cross-cutting"

BOOM-MR

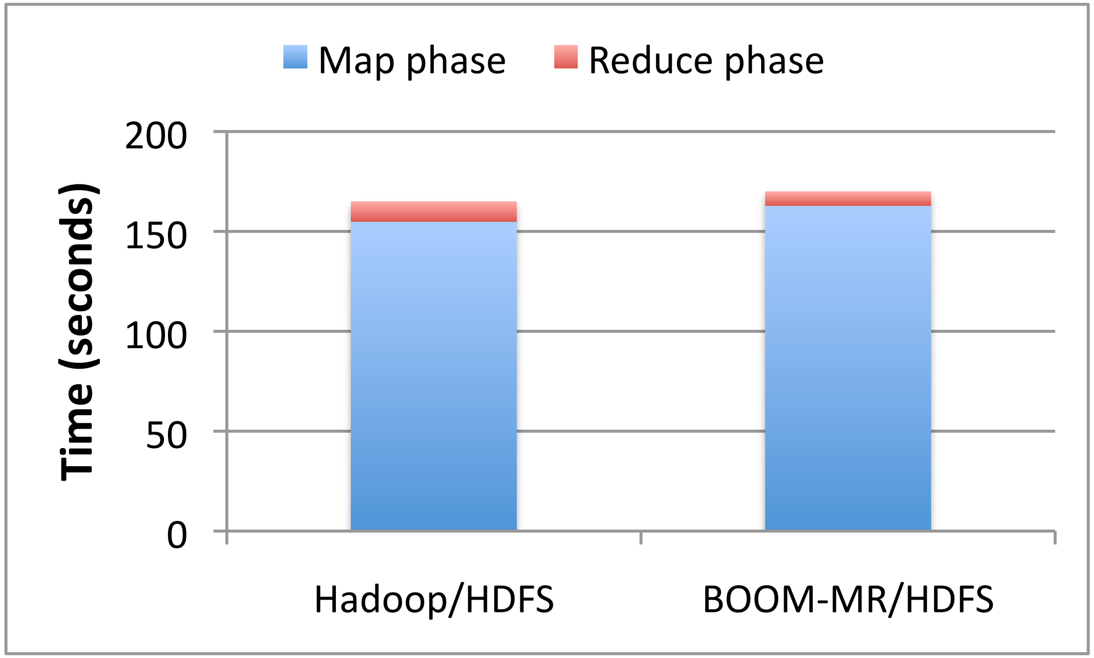
- MR scheduling is a popular research area
  - Hadoop JobTracker code is notoriously fragile
- Unlike with BOOM-FS, clean-slate rewrite is not practical
- Can data-centric programming and declarative languages simplify an existing system *in situ*?

BOOM-MR Example: State

Kept opaque state as Java objects

Relation Name	Description	Attributes
job	Job definitions	jobID, priority, status, jobConf
task	Task definitions	jobID, taskID, type, partition, status
task_attempt	Task execution attempts	jobID, taskID, attemptID, progress, state, phase, trackerID, input_loc, start_time, finish_time
task_tracker	TaskTracker descriptors	trackerID, hostname, state, map_count, reduce_count, map_max, reduce_max

Performance Comparison



- Workload: 30GB word count (average of 5 runs)
    - 481 map tasks, 100 reduce tasks
- 101 node cluster on Amazon EC2
- Very similar performance
    - BOOM-MR: higher CPU utilization at scheduler node

Comparison with Hadoop

	Lines of Java	Lines of Overlog
Hadoop	~89,000	0
BOOM-MR	~82,000	396

	Lines of Java	Lines of Overlog	# of files modified
LATE in Hadoop	~800	0	13
LATE in BOOM-MR	0	30	1

- BOOM-MR enables scheduling policies to be written concisely
  - E.g., LATE policy of Zaharia et al., *OSDI*'08
- Scheduling = statistics collection + rules for how to respond to state changes
  - Natural fit for a declarative query language

Lessons Learned

- Overall, Overlog was a good fit for the task
  - Concise programs for real problems
  - Agile system evolution
- Data-centric design: language-independent
  - Replication, partitioning, monitoring are state management problems
- Node-local invariants were convenient and useful
- Policy vs mechanism ⇔ Declarative vs Imperative

Lessons Learned

- Poor performance of language runtime, cryptic syntax, little/no tool support
  - Easy to fix!
- Encapsulation and modularity?
- Hand-coding protocols vs. stating distributed invariants

Future Work

1. BOOM stack
    - Interactive cloud storage (e.g., Cassandra)
    - High-performance Paxos
    - **C4**: New high-performance language runtime
2. Language
    - **Dedalus**: formalize language semantics
    - **Bloom**: “distributed logic for mere mortals”
3. Verification of distributed systems
    - Model checking of safety & liveness properties
    - **Blossom**: Network-oriented adaptive optimizer

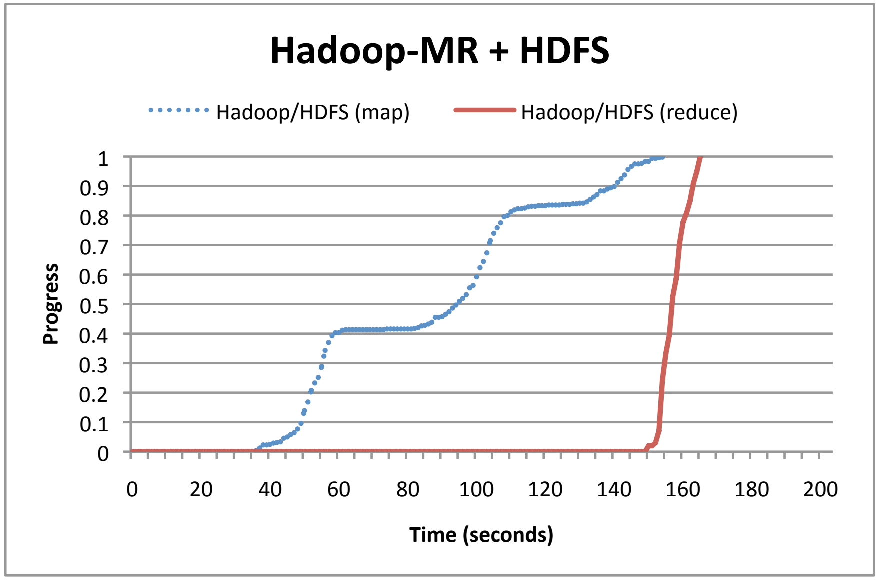
Questions?

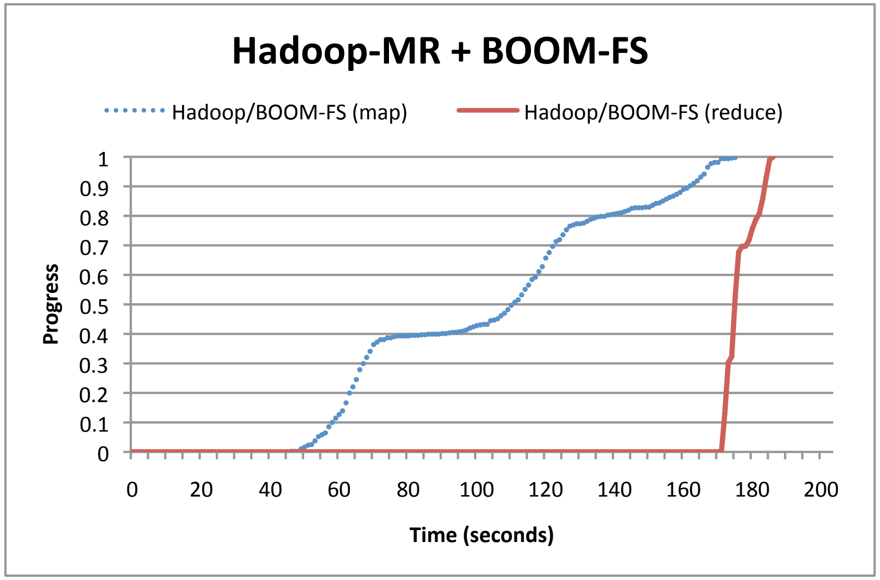
Thank you!

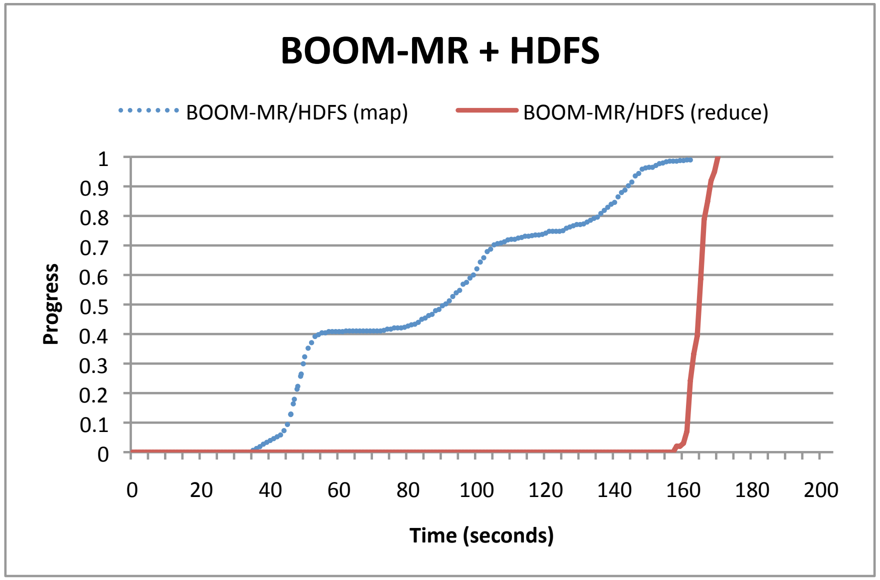
Papers and BOOM Analytics source code can be downloaded from:

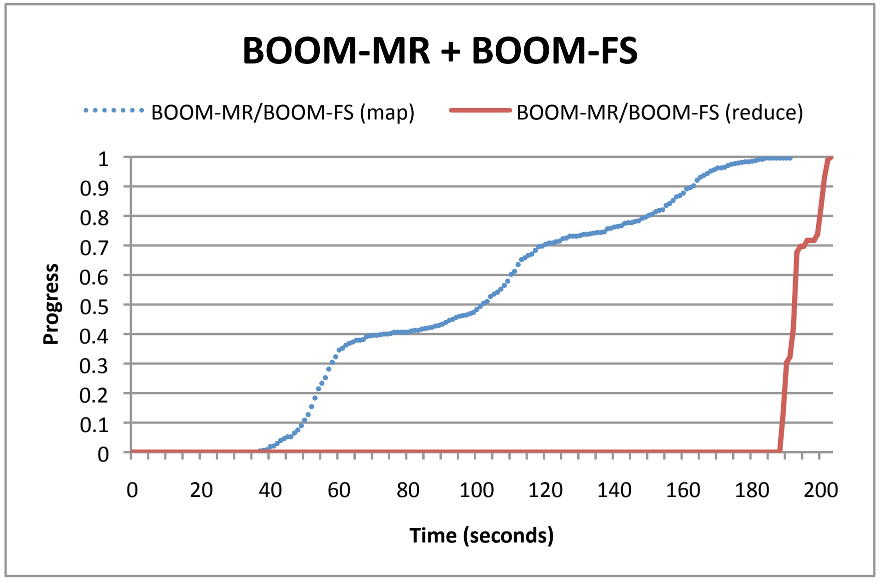
http://boom.cs.berkeley.edu

Performance Comparison


Hadoop-MR + HDFS


Hadoop-MR + BOOM-FS


BOOM-MR + HDFS


BOOM-MR + BOOM-FS