BOOM

Exploring Data-Centric, Declarative Programming for the Cloud

Neil Conway

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
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
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

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

- 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

Data Node
  - Master Node
  - Control Protocol
  - Heartbeat Protocol

Client
  - Metadata Protocol
  - Data Protocol

G

SOSP’03)
Example:

State

Relational	
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,	
*me

hb_chunk
Chunk	
heartbeats	
nodeAddr,	
chunkID,	
length
Example:

Query

File system behavior = queries over relations.

// Base case: root directory has null parent

// Do not add extra slash if parent is root dir

PathSep = (ParentPath = "/" ? "" : "/")

Path = ParentPath + PathSep + FName;
Example:

File system behavior = queries over relations.

// Base case: root directory has null parent

// Do not add extra slash if parent is root dir
// Base case: root directory has null parent
F
F  F  D
  D
F
F  F
F  F

// Do not add extra slash if parent is root
F
// Base case: root directory has null parent
   F
   F   F   D
   D
   F
   F   F

// Do not add extra slash if parent is root
   F
// Base case: root directory has null parent

F
F
F
D
D
F
F

// Do not add extra slash if parent is root

F
// Base case: root directory has null parent

// Do not add extra slash if parent is root
// Base case: root directory has null parent

F
F
F
D
D

// Do not add extra slash if parent is root

F
// Base case: root directory has null parent

```
F
F  F  D
D
F
F  F
```

// Do not add extra slash if parent is root

```
F
```
// Base case: root directory has null parent

// Do not add extra slash if parent is root
// Base case: root directory has null parent

F
F  F  D
 D
F
F  F

// Do not add extra slash if parent is root

F
// Base case: root directory has null parent

// Do not add extra slash if parent is root
Distributed Query Example

client_request(@Source, RequestId, ReqType, Arg),
master_addr(@Source, Master);

// "ls" for extant path => return dir list for path
request(@Master, RequestId, Source, ReqType, Path),
ReqType = "Ls",
fqpath(@Master, FileId, Path),
directory_listing(@Master, FileId, DirListing);

// "ls" for nonexistent path => return error response
request(@Master, RequestId, Source, "Ls", Path),
no*fqpath(@Master, _, Path);

Loca*on specifier (node addr)
// "ls" for extant path => return dir listing for path
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", no* fqpath (@Master, _, Path);
BOOM-FS

• Hybrid system
  – Complex logic: Overlog
  – Performance-critical (but simple!): Java

• Separation of policy and mechanism

Data Node

Client

Metadata Protocol

Overlog

Data Protocol

Master Node

Data Node

Data Node

Heartbeat Protocol

Control Protocol

Java

B

– C

–
• 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
<table>
<thead>
<tr>
<th>DF</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

- 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

- HDFS: single point of failure at master
- We built hot standby using Paxos

\[\text{Master Replica 1} \rightarrow \text{Distributed Log (Paxos)} \rightarrow \text{Master Replica 2} \rightarrow \text{Master Replica 3}\]
- DF  A  F
  - A
- DF
- B  F
  - F
  - B  F
  - C
Monitoring

- Overlog allows natural system introspection – “Everything is data”: just query it!

- Logging and monitoring = distributed queries
  - 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”
• 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?
<table>
<thead>
<tr>
<th>B</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Kept opaque state as Java objects.
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

• 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

<table>
<thead>
<tr>
<th>Lines of Java</th>
<th>Lines of Overlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOM-MR</td>
<td>~82,000</td>
</tr>
<tr>
<td>Hadoop</td>
<td>~89,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># of files modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATE in Hadoop</td>
</tr>
<tr>
<td>LATE in BOOM-MR</td>
</tr>
</tbody>
</table>

• C

<table>
<thead>
<tr>
<th>B</th>
<th>E</th>
<th>A</th>
<th>E</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>E</th>
<th>B</th>
</tr>
</thead>
</table>

OSDI
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, parrying, monitoring are state management problems

• Node-local invariants were convenient and useful

• Policy vs mechanism
  – Declarative vs Imperative

• C
• A
• D
• D ↔ D
Lessons Learned

• Poor performance of language run*me, cryp*c syntax, liEle/no tool support – Easy to fix!

• Encapsulation and modularity?

• Hand-coding protocols vs. sta*ng 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
Analy*cs
source code can be downloaded from:
http://boom.cs.berkeley.edu