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

Ştefan Istrate

University of Cambridge

February 10, 2011

Outline

1. Introduction
   - The problem
   - The solution
2. Background
   - Overlog
3. BOOM Analytics
   - HDFS Rewrite (BOOM-FS)
   - The Availability Rev
   - The Scalability Rev
   - The Monitoring Rev
   - MapReduce Port (BOOM-MR)
4. Performance
5. Conclusions
6. Questions / Comments

The problem

Building and debugging distributed software is extremely difficult.

The developer spends time on:

- orchestrating concurrent computation and communication across machines
- minimize the delays
- handle failures

instead of

- being creative

The solution

A broad range of distributed software can be recast in a data-parallel programming model.

Solution:

- adopt a data-centric approach to system design
- switch to declarative programming languages

Advantages:

- raised level of abstraction for programmers
- improved code simplicity
- better speed of development
- ease of software evolution
- program correctness

BOOM Analytics

BOOM = Berkeley Orders Of Magnitude

BOOM Analytics = reimplementation of HDFS and Hadoop MapReduce in Overlog

Why Hadoop?

1. It shows the distributed power of a cluster.
2. Significant distributed features are missing => It can be extended.

Overlog

- declarative language (logic of computation, not the control flow)
- based on Datalog
  - defined over relational tables
  - query language that makes no changes to the stored tables
  - rules:

$$r_{head}(\langle col - list \rangle) \vdash r_1(\langle col - list \rangle), \ldots, r_n(\langle col - list \rangle)$$

- extends Datalog
  - can specify location of data
  - primary keys and aggregation
  - defines a model for processing and generating changes to tables
- relational tables may be partitioned across a set of machines
- implementations: P2, JOL (Java-based Overlog)

HDFS

- files system metadata stored at a centralized NameNode
- file data distributed across DataNodes
- by default, data chunks of 64MB replicated three times
- DataNodes send heartbeat messages to the NameNode
- clients only contact the NameNode

BOOM-FS

- represent file system metadata as a collection of relations

Name	*Description*	*Relevant attributes*
file	Files	fileid, parentfileid, name, isDir
fqpath	Fully-qualified pathnames	path, fileid
fchunk	Chunks per file	chunkid, fileid
datanode	DataNode heartbeats	nodeAddr, lastHeartbeatTime
hb_chunk	Chunk heartbeats	nodeAddr, chunkid, length

- metadata and heartbeat protocols implemented with Overlog rules
- data protocol implemented in Java
- 4 person-months of work

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

The Availability Rev

Goal:

- hot standby replication for NameNodes

Solution: Paxos algorithm

- solves consensus in the network
- is a collection of logical invariants
- messages and disk writes $\rightarrow$ insertions into tables
- invariants $\rightarrow$ rules

Results:

- 400 lines of code
- 6 person-weeks of development time

The Scalability Rev

Goal:

- scale out the NameNode across multiple partitions

Solution:

- add a 'partition' column to tables to split them across nodes

Results:

- 8 hours of development time

The Monitoring Rev

Goal:

- develop performance monitoring and debugging tools

Solution:

- replicate the body of each rule and send it to a log table
- add a relation called *"die"* to JOL
- when *"die"* is added throw a Java exception

Results:

- performance monitoring: 64 lines of code, less than 1 day
- debugging: 60 lines of code, 8 person-hours

Hadoop MapReduce

- single master node (JobTracker)
- many worker nodes (TaskTrackers)
- job is divided in *maps* and *reduces*
- *map*: reads an input chunk, runs a function, partition the output into buckets
- *reduce*: fetch hash buckets, sort by key, runs a function, writes to distributed file system
- fixed number of slots for every TaskTracker
- heartbeat protocol between each TaskTracker and JobTracker

BOOM-MR

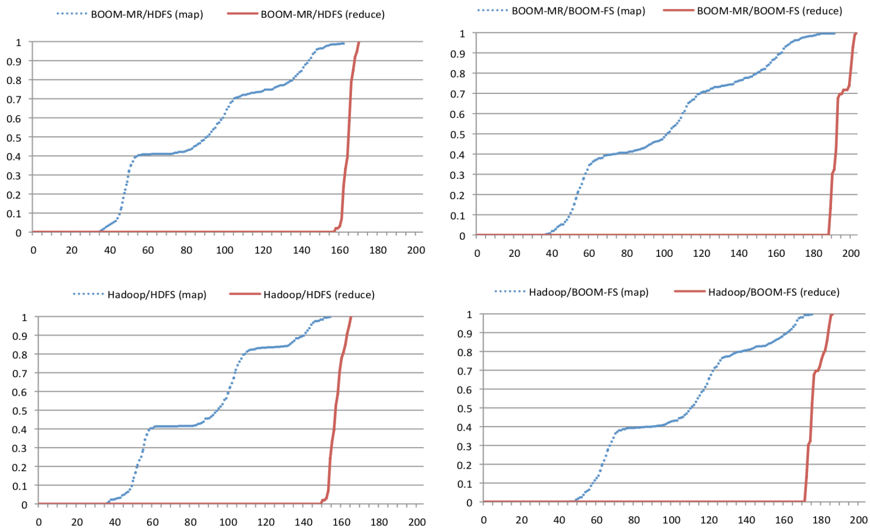
Name	*Description*	*Revelant attributes*
job	Job definitions	jobid, priority, submit_time, status, jobConf
task	Task definitions	jobid, taskid, type, partition, status
taskAttempt	Task attempts	jobid, taskid, attemptid, progress, state, phase, tracker, input_loc, start, finish
taskTracker	TaskTracker definitions	name, hostname, state, map_count, reduce_count, max_map, max_reduce

- evaluation on Hadoop's default First-Come-First-Serve (FCFS) policy and the LATE (Longest Approximation Time to End) policy
- better results for LATE

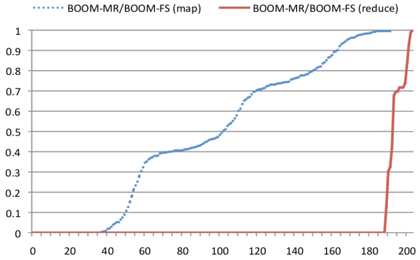
Results:

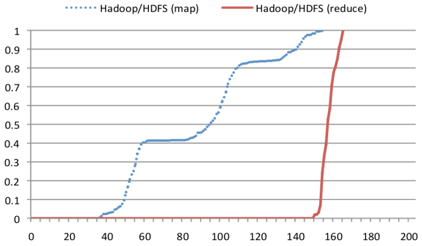
- initial version: one person-month
- debugging and tuning: two person-months
- 55 Overlog rules
- 6573 lines removed from Hadoop

Performance



Performance (cont.)





Conclusions

Good things:

- focus on what, not on how
- simplified code
- faster development
- program correctness

Bad things:

- system load averages higher with BOOM Analytics
- Overlog needs some other features
- difficult and time-consuming to read the code
- hard for programmers to switch to declarative programming

Questions / Comments?