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

Jadwiga Kańska

21 grudnia 2011

Introduction

- Data-centric approach to system design and employing declarative programming languages can significantly raise the level of abstraction for programmers, improve code simplicity, speed of development, ease of software evolution, and program correctness.
- Experiment includes rewriting and extending Hadoop MapReduce engine and HDFS.

Data-centric approach

In data-centric approach:

- The primary function is the management and manipulation of data.
- Applications are expressed in terms of high-level operations on data.
- The runtime system transparently controls the scheduling, execution, load balancing, communications, and movement of programs and data across the computing cluster.
- Such abstraction and focusing on the data makes problems much simpler to express.
- In distributed systems programmer's attention is focused on carefully capturing all the important state of the system as a family of collections (sets, relations, streams, etc.). Given such a model, the state of the system can be distributed naturally and flexibly across nodes via familiar mechanisms like partitioning and replication.

Declarative programming languages

Declarative programming languages:

- Express the logic of a computation without describing its control flow (specify what the program should accomplish, rather than describe how to accomplishing it).
- The key behaviors of mentioned systems can be naturally implemented using declarative programming languages that manipulate these collections, abstracting the programmer from both the physical layout of the data and the fine-grained orchestration of data manipulation.

Datalog

- Overlog is based on Datalog - the basic language for deductive databases.
- It is defined over relational tables, so facts in Datalog are represented in the form of relations $name(arg_1, ..., arg_k)$, where name is a name of a relation and $arg_1, ..., arg_k$ are constants (e.g. likes(John, Marc)).
- Atomic queries are of the form $name(arg_1, ..., arg_k)$, where $arg_1, ..., arg_k$ are constants or variables (e.g. likes(John, Marc) – does John like Marc? or likes(X , Marc) – who likes Marc? (compute X's satisfying likes(X , Marc)) or likes(X , Y) – compute all pairs X , Y such that likes(X , Y)holds).

Datalog - rules

- A Datalog program is a set of rules or named queries, in the spirit of SQL's views.
- Rules in Datalog are expressed in the form of $r_{head}(< col - list >) : r_1(< col - list >), ..., r_k(< col - list >)$, where:
- Each term r_i represents a relation, either stored (a database table) or derived (the result of other rules).

Datalog - rules

- Relations' columns are listed as a comma-separated list of variable names or constants symbols such that any variable appearing on the lefthand side of ':' (called the head of the rule - corresponding to the SELECT clause in SQL) appears also on the righthand side of the rule (called the body of the rule - corresponding to the FROM and WHERE clauses in SQL).
- Each rule is a logical assertion that the head relation contains those tuples that can be generated from the body relations.
- Tables in the body are joined together based on the positions of the repeated variables in the column lists of the body terms.

Example Overlog for computing all paths from links, along with an SQL translation

```
path(@From, To, To, Cost)
        :- link(@From, To, Cost);
path(@From, End, To, Cost1 + Cost2)
        :- link(@From, To, Cost1),
           path(@To, End, NextHop, Cost2);
```

```
WITH RECURSIVE path(Start, End, NextHop, Cost) AS
(   SELECT From, To, To, Cost FROM link
    UNION
    SELECT link.From, path.End, link.To,
           link.Cost + path.Cost
      FROM link, path
     WHERE link.To = path.Start );
```

Overlog extensions

Overlog extends Datalog in three main ways:

- It adds notation to specify the location of data.
- Provides some SQL-style extensions such as primary keys and aggregation.
- Defines a model for processing and generating changes to tables

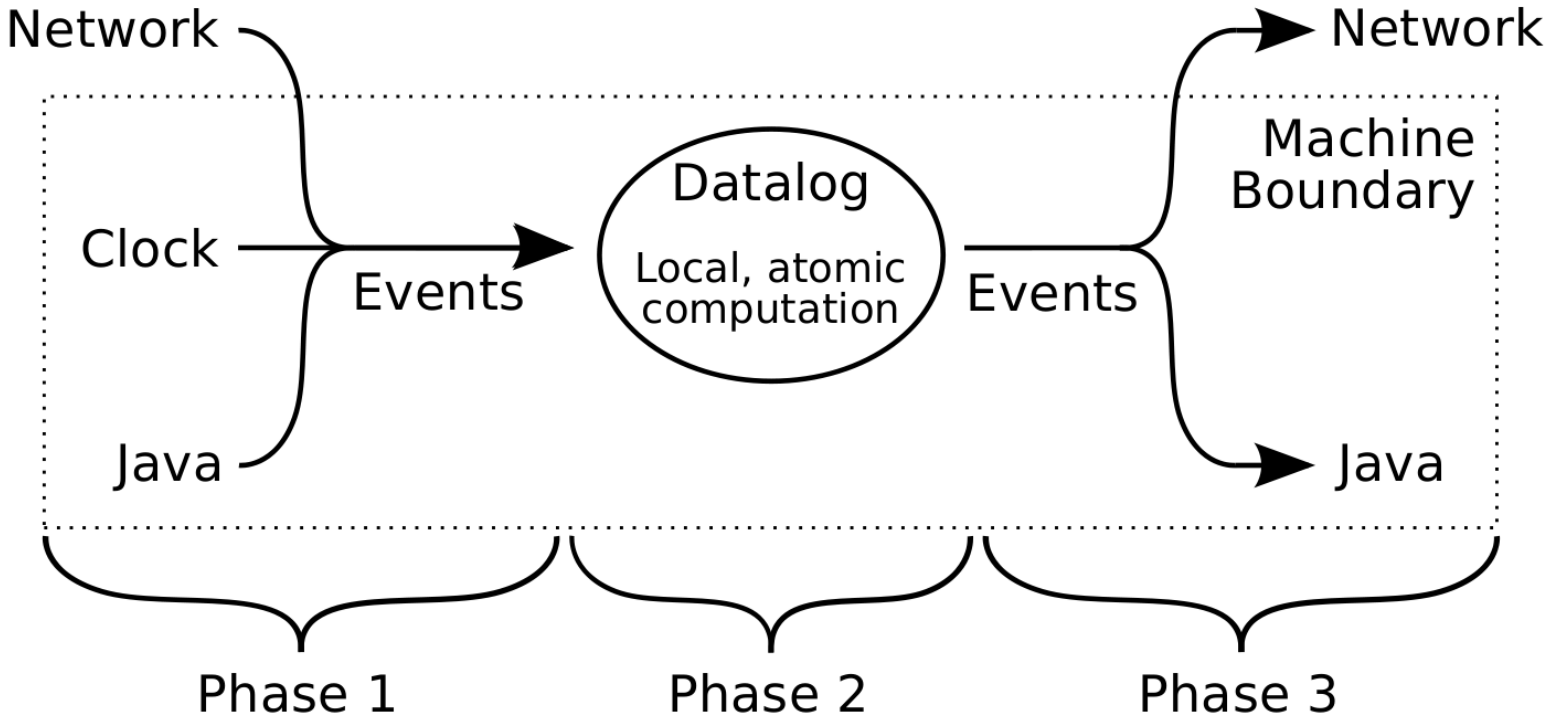
  Overlog supports relational tables that may optionally be "horizontally" partitioned row-wise across a set of machines based on a column called the location specifier, which is denoted by the symbol @.

Overlog events

- Communication between Datalog and the rest of the system (Java code, networks, and clocks) is modeled using events corresponding to insertions or deletions of tuples in Datalog tables.
- When Overlog tuples arrive at a node either through rule evaluation or external events, they are handled in an atomic local Datalog "timestep."
- Each timestep consists of three phases.

Overlog timestep

An Overlog timestep at a participating node: incoming events are applied to local state, the local Datalog program is run to fixpoint, and outgoing events are emitted.



JOL

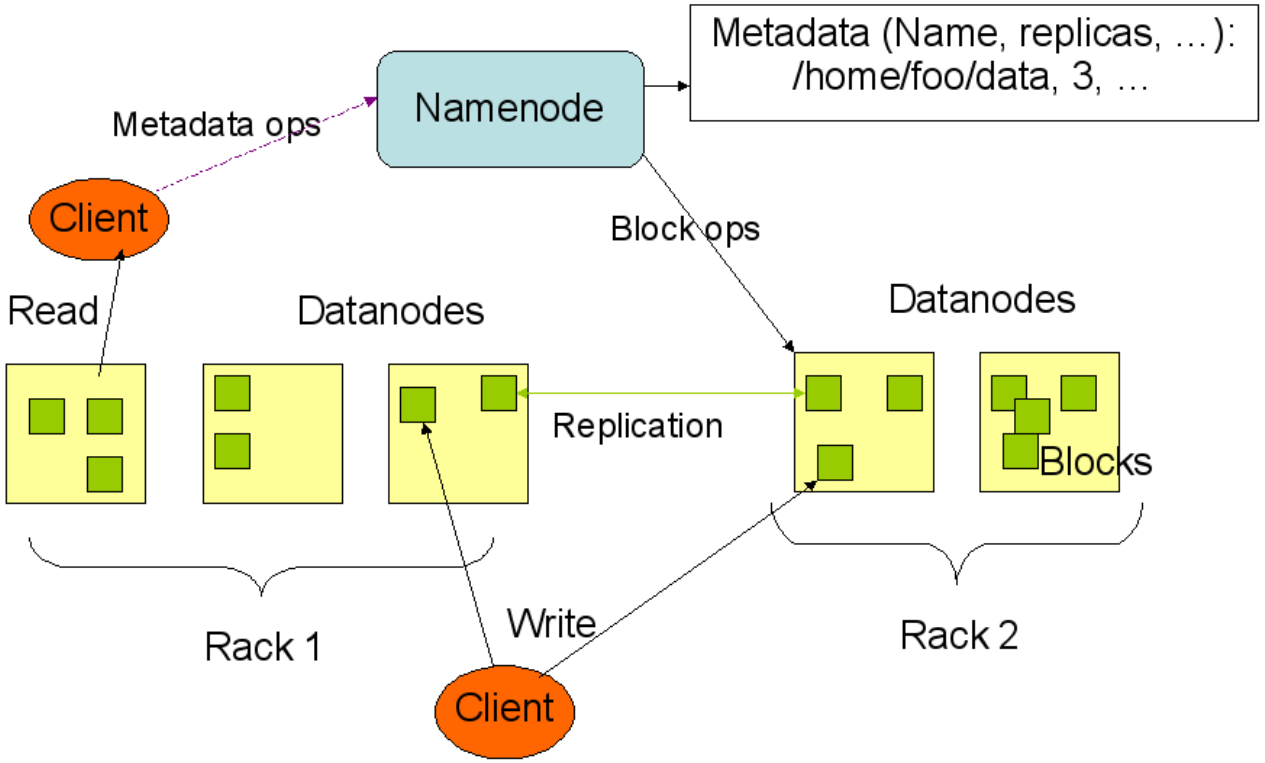
The original Overlog implementation (P2) is aging and targeted at network protocols so authors of experiment developed JOL - a new Java-based Overlog runtime.

HDFS

- HDFS is targeted at storing large files for full-scan workloads.
- File system metadata is stored at centralized NameNode.
- File data is partitioned into chunks and distributed across a set of DataNodes.
- By default, each chunk is 64MB and is replicated at three DataNodes to provide fault tolerance.
- DataNodes periodically send heartbeat messages to NameNode containing the set of chunks stored at the DataNode.
- HDFS only supports file read and append operations - chunks cannot be modified once they have been written.

HDFS

HDFS Architecture



BOOM-FS relations defining file system metadata

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

Features

- Easily ensured that file system metadata is durable and restored to a consistent state after a failure.
- Natural recursive queries.
- The materialization views can be changed via simple Overlog table definition statements without altering the semantics of the program.

Example Overlog for deriving fully-qualified path-names from the base file system metadata in BOOM-FS

```
// fqpath: Fully-qualified paths.
// Base case: root directory has null parent
fqpath(Path, FileId) :-
    file(FileId, FParentId, _, true),
    FParentId = null, Path = "/";

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

Communication Protocols

Both HDFS and BOOM-FS use three different protocols:

- The metadata protocol that clients and NameNodes use to exchange file metadata.
- The heartbeat protocol that DataNodes use to notify the NameNode about chunk locations and DataNode liveness.
- The data protocol that clients and DataNodes use to exchange chunks.

Quantities

BOOM-FS contains an order of magnitude less code than HDFS.

- The DataNode implementation accounts for 414 lines of the Java in BOOM-FS.
- The remainder is devoted to system configuration, bootstrapping, and a client library.
- Adding support for accessing BOOM-FS via Hadoop's API required an additional 400 lines of Java.

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

Notions

- The main benefit of our data-centric approach was to expose the simplicity of HDFS's core state.
- Overlog's declarativity was useful to express paths as simple recursive queries over parent links, and flexibly decide when to maintain materialized views (i.e., cached or precomputed results) of those paths separate from their specification.

Hot standby

- Attempt to retrofit BOOM-FS with high availability failover via "hot standby" NameNodes.
- Usage of globally-consistent distributed log, which guarantess a total ordering over events affecting a replicated state.
- Lamport's Paxos algorithm as the canonical mechanism for this feature.

Paxos algorithm

- Lamport's description of basic Paxos is given in terms of "ballots" and "leadgers", which corresponds to network messages and stable storage.
- The consensus algorithm is given as a collection of logical invariants when agents cast ballots and commit writes to their ledgers.
- In Overlog, messages and disk writes are represented as insertions into tables, while invariants are represented as Overlog rules.

Integration

- All state-altering actions are represented in the revised BOOM-FS as Paxos decrees.
- Tentative actions are intercepted and placed into a table that is joined with Paxos rules.
- Each action is considered complete at a given site when it is "read back" from the Paxos log.
- In the absence of failure, replication has negligible performance impact, but when the primary NameNode fails, a backup NameNode takes over reasonably quickly.

Quantities

- Paxos implementation constituted roughly 400 lines of code and required six person-weeks of development time
- Adding Paxos support to BOOM-FS took two person-days and required making mechanical changes to ten BOOM-FS rules.

Notions

- Lamport's original paper describes Paxos as a set of logical invariants.
- This specification naturally lent itself to a data-centric design in which "ballots," "ledgers," internal counters and vote-counting logic are represented uniformly as tables.
- The principal benefit of our approach came directly from our use of a rule-based declarative language to encode Lamport's invariants.
- Authors found that they were able to capture the design patterns frequently encountered in consensus protocols (e.g., multicast, voting) via the composition of language constructs like aggregation, selection and join.

Notions

- In initial implementation of basic Paxos each rule covered a large portion of the state space, avoiding the case-by-case transitions that would need to be specified in a state machine-based implementation.
- However, choosing an invariant-based approach made it harder to adopt optimizations from the literature as the code evolved, in part because these optimizations were often described using state machines.
- Authors had to choose between translating the optimizations "up" to a higher-level while preserving their intent, or directly "encoding" the state machine into logic, resulting in a lower-level implementation. In the end, they adopted both approaches, giving sections of the code a hybrid feel.

NameNode-partitions

- HDFS NameNodes manage large amounts of file system metadata, which are kept in memory to ensure good performance, so it's not scalable.
- Given the data-centric nature of BOOM-FS it was easy to scale out the NameNode across multiple NameNode-partitions.
- Having exposed the system state in tables, this was straight-forward: it involved adding a "partition" column to various tables to split them across nodes in a simple way.
- Files in a directory tree were partitioned based on the hash of the fully-qualified pathname of each file.
- For most BOOM-FS operations, clients have enough local information to determine the correct NameNode-partition.

Notions

- Primarily due to the data-centric nature of the design, scaling out the NameNodes turned out to be a very easy task (it took 8 hours of developer time).
- It was independent of any declarative features of Overlog.
- It composed with previous availability implementation: each NameNode-partition can be deployed either as a single node or a Paxos group.

Strategy

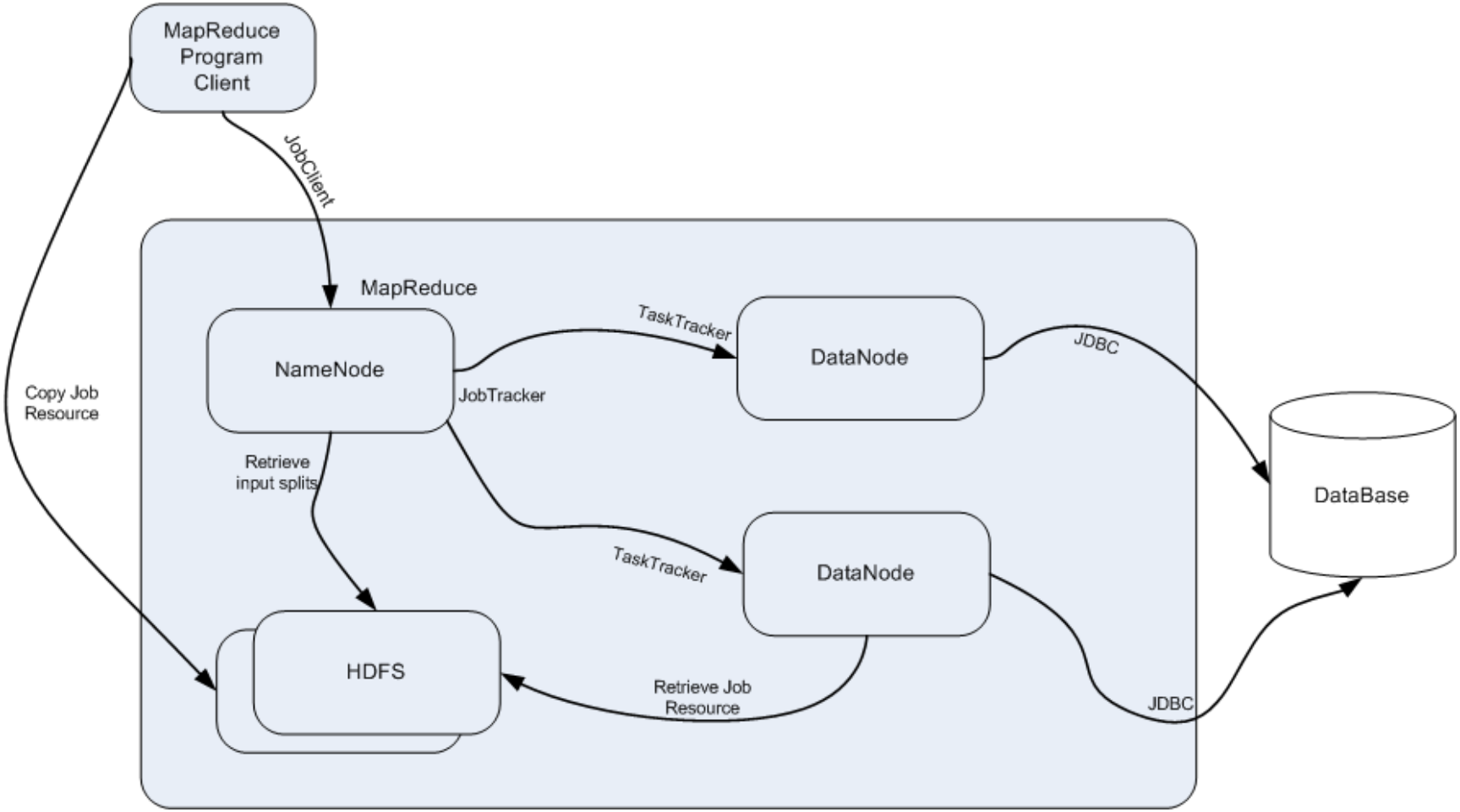
- BOOM-MR is not a clean-slate rewrite of Hadoop's MapReduce.
- Just Hadoop's core scheduling logic is replaced with Overlog.
- Hadoop's MapReduce codebase is mapped into a relational representation.
- And there are written Overlog rules to manage that state in the face of new messages delivered by the existing Java APIs.

Hadoop MapReduce

- There is a single master node called the JobTracker which manages a number of worker nodes called TaskTrackers.
- A job is divided into a set of map and reduce tasks.
- The JobTracker assigns tasks to worker nodes.
- Each map task reads an input chunk from the DFS, runs a map function, and partitions output key/value pairs into hash buckets on the local disk.
- Reduce tasks are created for each hash bucket. Each reduce task fetches the corresponding hash buckets from all mappers, sorts locally by key, runs a reduce function and writes the results to the DFS.

Hadoop MapReduce

- Each TaskTracker has a fixed number of slots for executing tasks (two maps and two reduces by default).
- A heartbeat protocol is used to update the JobTracker's knowledge of the state of running tasks.
- Hadoop will attempt to schedule speculative tasks to reduce a job's response time if it detects "straggler" nodes.


MapReduce Program Client
JobClient
MapReduce
NameNode
JobTracker
TaskTracker
DataNode
JDBC
Copy Job Resource
Retrieve input splits
TaskTracker
DataNode
DataBase
HDFS
Retrieve Job Resource
JDBC

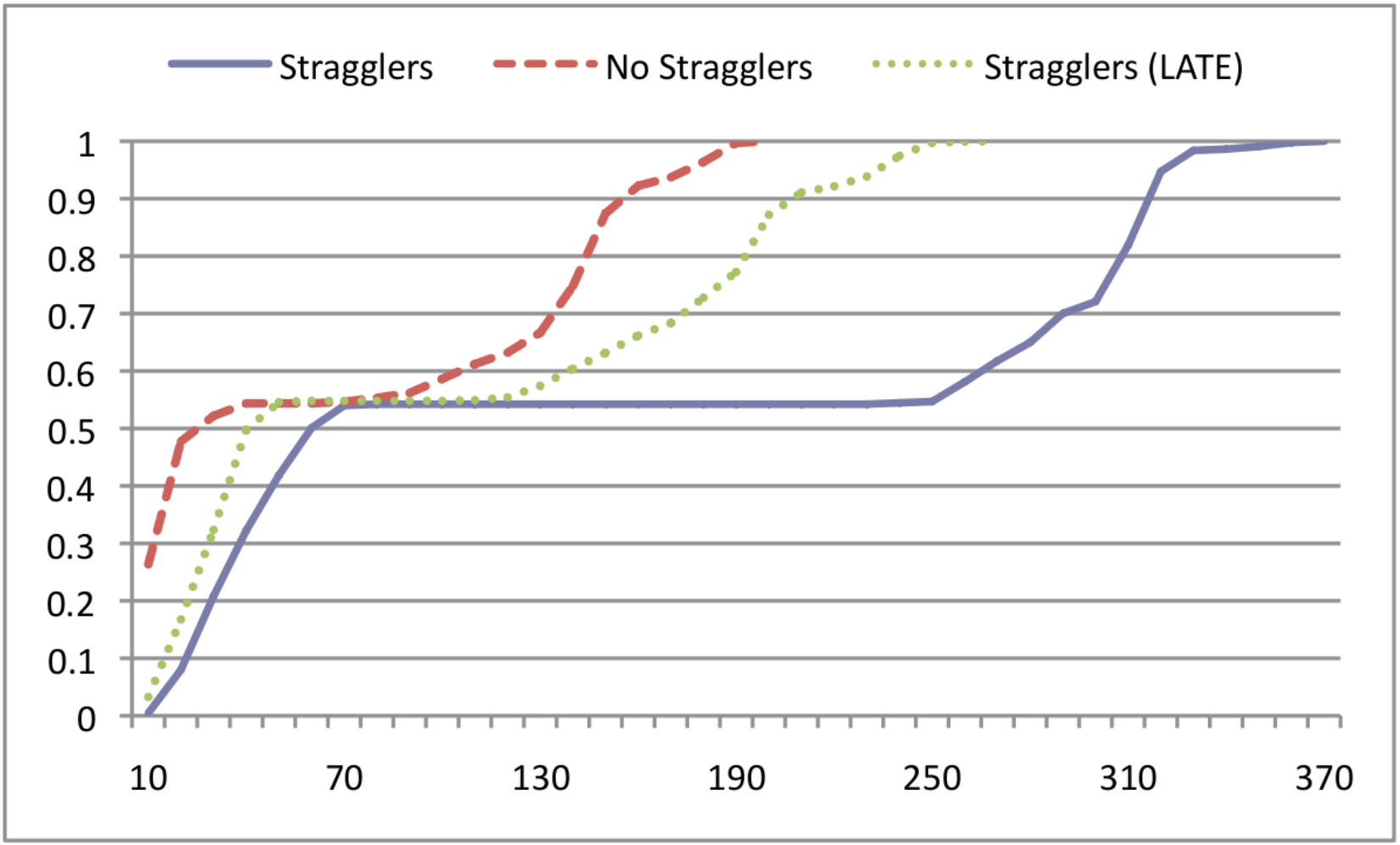
BOOM-MR relations defining JobTracker state

Name	*Description*	*Relevant 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

BOOM-MR relations defining JobTracker state

- Overlog rules are used to update the JobTracker's tables by converting inbound messages into job, taskAttempt and taskTracker tuples.
- Scheduling decisions are encoded in the taskAttempt table, which assigns tasks to TaskTrackers.
- A scheduling policy is simply a set of rules that join against the taskTracker relation to find TaskTrackers with unassigned slots, and schedules tasks by inserting tuples into taskAttempt.
- This architecture makes it easy for new scheduling policies to be defined.

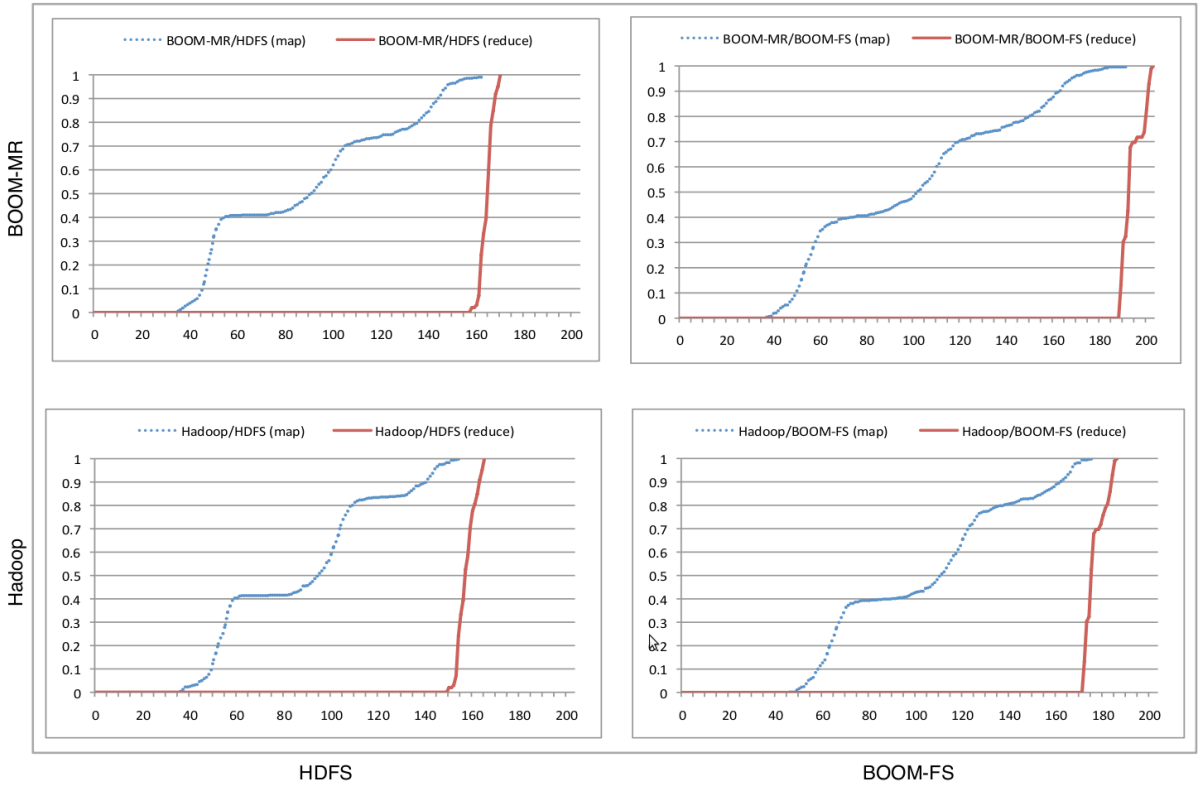
CDF of reduce task duration in seconds



Notions

- Scheduling policies are a good fit for a declarative language.
- This is because scheduling can be decomposed into two tasks: monitoring the state of a system and applying policies for how to react to changes to that state.
- And both of these things are well-handled by Overlog.

CDFs representing the elapsed time between job startup and task completion



Notions

- Improved performance was not a goal of experiment.
- But it turned out that map and reduce task durations under BOOM-MR are nearly identical to Hadoop 18.1.
- And that BOOM-FS performance is slightly slower than HDFS, but remains competitive.

Strategy

- Overall experience with BOOM Analytics has been quite positive (nine months of part-time work of four developers).
- This experience is not universal.
- But sheds light on common patterns that occur in many distributed systems: the coordination of multiple nodes toward common goals, replicated state for high-availability, state partitioning for scalability.
- Much of productivity came from using a data-centric design philosophy, which exposed the simplicity of the undertaken tasks.
- Overlog imposed this data-centric discipline throughout the development process: no private state was registered "on the side" to achieve specific tasks.