Disco: Beyond MapReduce

Prashanth Mundkur

Nokia

Mar 22, 2013

Outline

- BigData/MapReduce
- Disco
- Disco Pipeline Model
- Disco Roadmap

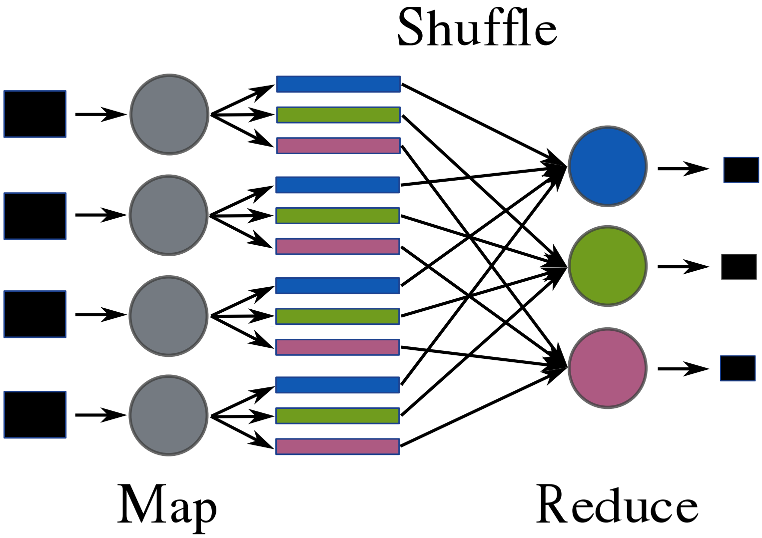
BigData/MapReduce

- Data too big to fit in RAM/disk of any single machine

$\longrightarrow$

- Analyze chunks of data in parallel (maps)
- Collect intermediate results into a final result (reduce)
- Use a cluster of machines

MapReduce TaskGraph

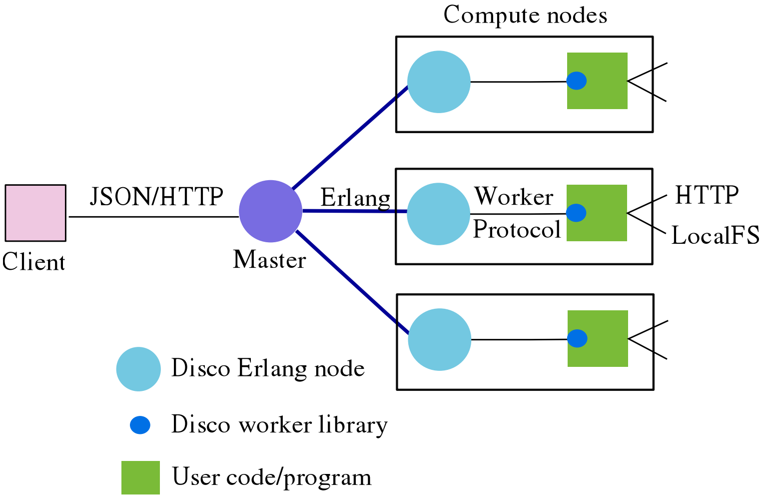


Disco Origins

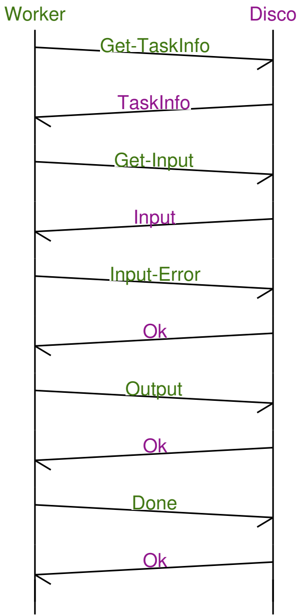
```
commit 1aa76c1eda8081317f66afbaf872c0f92dfc46f7
Author: Ville Tuulos <tuulos@parvus.pp.htv.fi>
Date:   Mon Jan 14 02:04:34 2008 -0800

    Initial commit
```

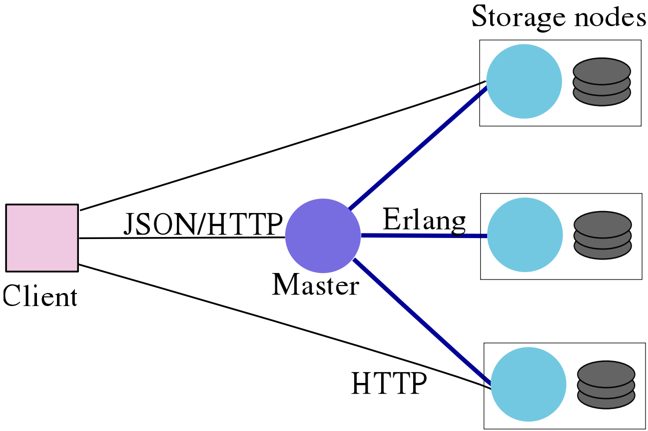
Disco Architecture



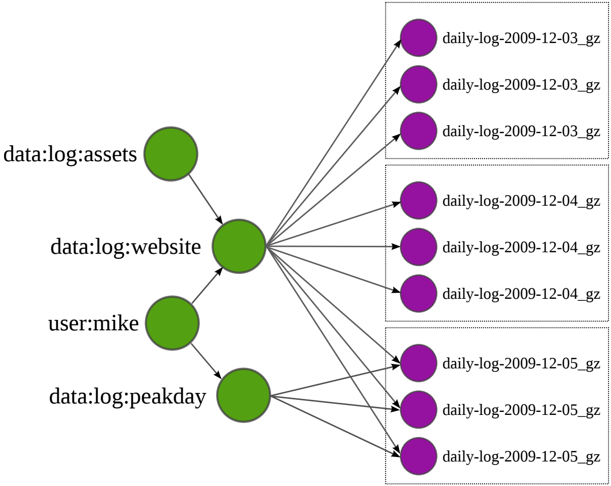
Worker Protocol



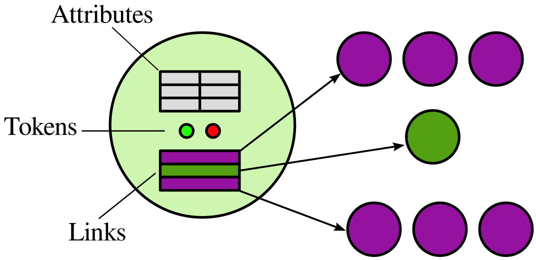
Disco DFS



Data in DDFS


data:log:assets
data:log:website
user:mike
data:log:peakday
daily-log-2009-12-03_gz
daily-log-2009-12-03_gz
daily-log-2009-12-03_gz
daily-log-2009-12-04_gz
daily-log-2009-12-04_gz
daily-log-2009-12-04_gz
daily-log-2009-12-05_gz
daily-log-2009-12-05_gz
daily-log-2009-12-05_gz

Metadata (tags) in DDFS



Code size

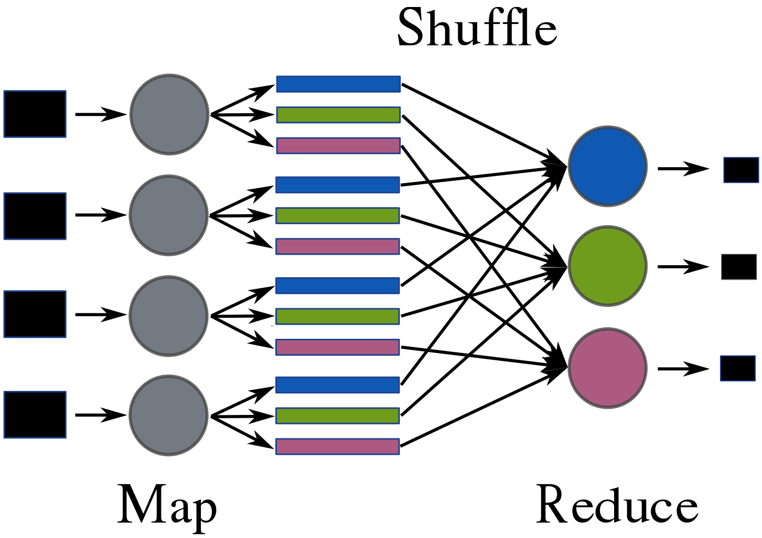
	Hadoop 1.0	Disco (dev)
Map-reduce	53333⋆ (Java)	8053† (Erlang)
		3276⋆ (Python)
		1724⋆ (OCaml)
DFS	34301⋆ (Java)	4600† (Erlang)

⋆ David A. Wheeler's 'SLOCCount'

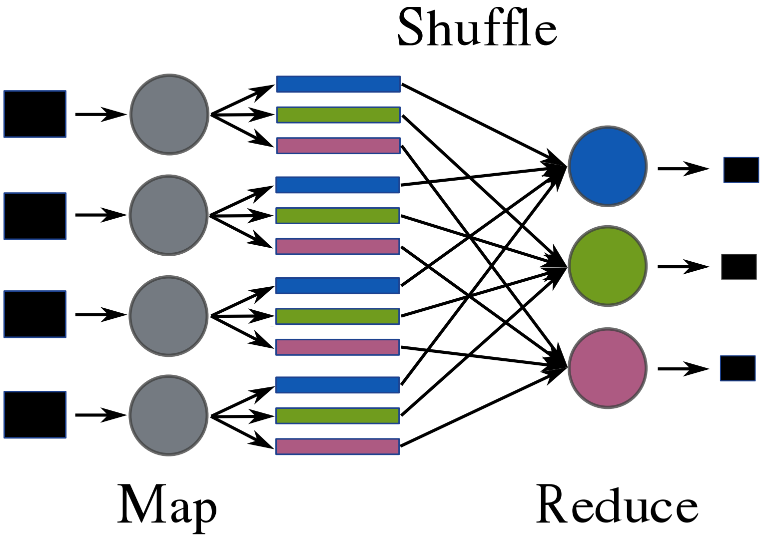
† `wc -l`

- Disco's external dependencies:
  - HTTP library (`mochiweb`, 12.5kLOC)
  - Logging library (`lager`, 4.3kLOC)
  - Erlang/Python standard libraries

Disco scheduler bug: no backtracking



Shuffle in Disco: bulk user data through Erlang



Rethink

Limitations of MapReduce

- Job computation is performed in three fixed stages

Rethink

Limitations of MapReduce

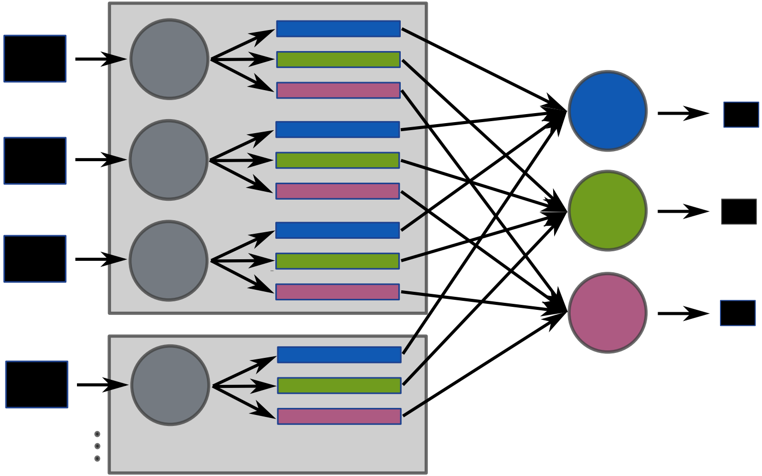
- Job computation is performed in three fixed stages
- Processing model is tied to content (key-value pairs)

Rethink

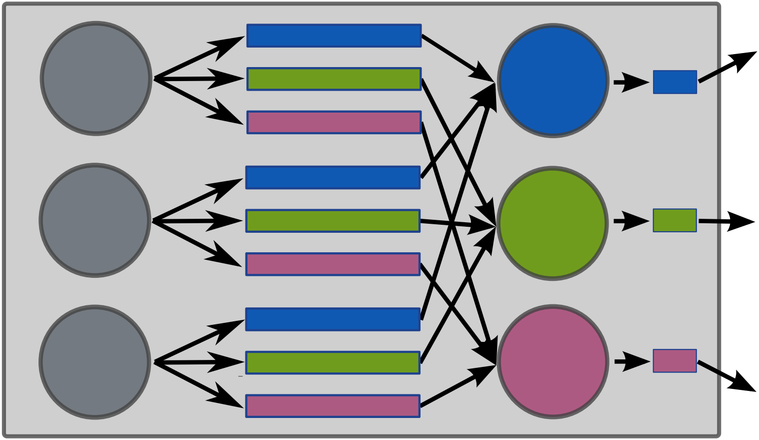
Limitations of MapReduce

- Job computation is performed in three fixed stages
- Processing model is tied to content (key-value pairs)
- No inter-task optimization of network resources (crucial for shuffle/reduce)

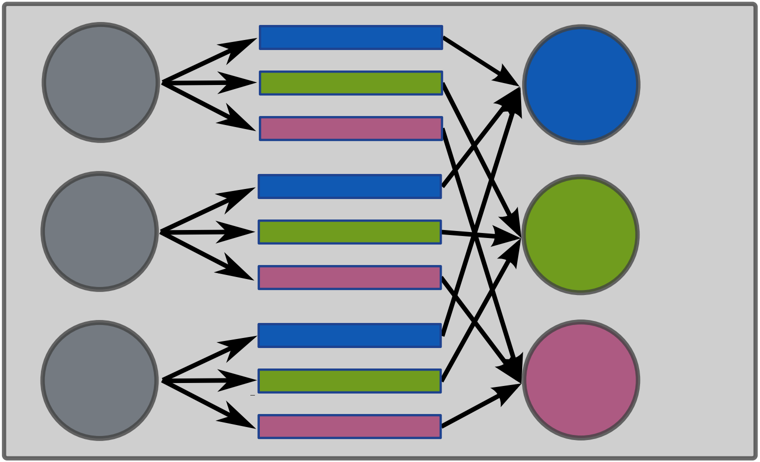
Node-locality of Tasks in MapReduce



Optimizing network-use based on Node-locality

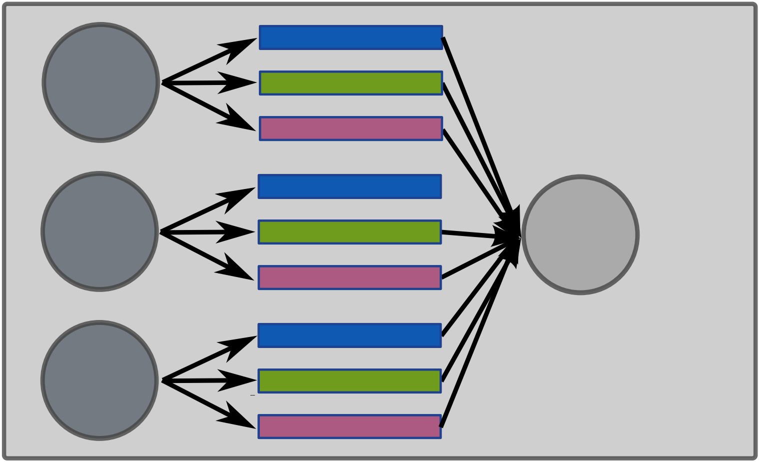


Output grouping



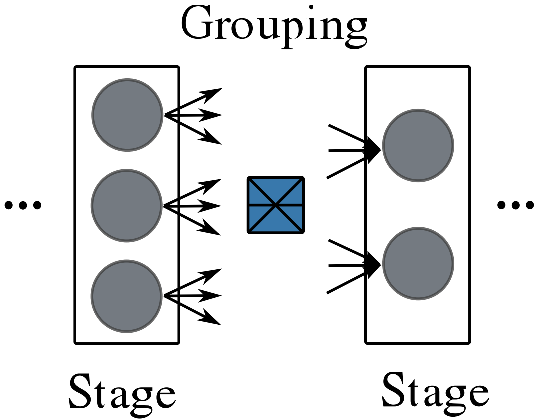
Grouping by label per node (`group_node_label`)

Output grouping



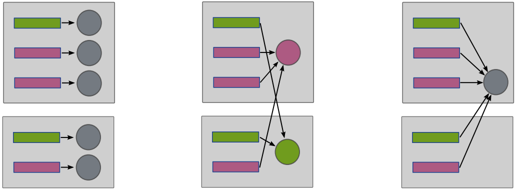
Grouping per node (`group_node`)

Pipelined Stages of Tasks



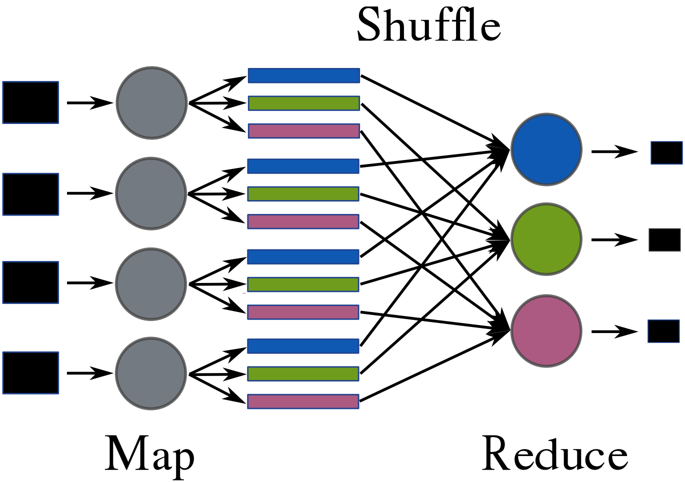
$$\begin{aligned} \texttt{pipeline} &::= \texttt{stage}+ \\ \texttt{stage} &::= \{\texttt{grouping}, \texttt{task}\} \end{aligned}$$

Other grouping options



Split Group_label Group_all

MapReduce as a Pipeline



$$\texttt{map-reduce} = \{\texttt{split}, \texttt{map}\}, \{\texttt{group\_label}, \texttt{reduce}\}$$

Disco Pipeline Model

- Fixes existing issues
  - backtracking scheduler
  - no bulk data passes through Erlang

Disco Pipeline Model

- Fixes existing issues
  - backtracking scheduler
  - no bulk data passes through Erlang
- Adds a flexible compute model
  - Allows multiple user-defined stages, as opposed to just map-(shuffle)-reduce
  - Exposes shuffle to user-code
  - Exposes node-locality to tasks, exploitable via user-selectable grouping options

Disco Pipeline Model

- Conservative extension
  - linear pipeline simpler than DAG
  - no need for a graph DSL as in Dryad
- More flexible platform for higher-level tools like Pig/FlumeJava/etc.

Pipeline Limitations

- no forks/joins in dataflow
- no iteration or recursion

New Task API for Pipelines

- no “map” or “reduce” tasks

New Task API for Pipelines

- no “map” or “reduce” tasks

- user pulls data from input via iterators (`process`)
- simpler handling of processing state (`init`, `done`)
- control iteration over input labels (`input_hook`)

Disco Roadmap

- Disco 0.5 coming soon
  - backtracking job coordinator
  - pipelines
  - alternative task API
  - *plus* support* for existing map-reduce API
- Evolve pipeline model / API
- Network-topology-aware task scheduler

Disco and Hadoop

- DDFS/HDFS storage are different
- Disco Pipelines/YARN compute models are different

Questions?

http://discoproject.org

DDFS

Design choices

- optimized for log-file storage (bulk immutable data files)
- data is not modified but stored as submitted (e.g. no chunking by default)
- replication of data *and metadata* (unlike Hadoop/HDFS)
- only metadata is mutable
- DAG structure as opposed to tree
  → DAG design imposes garbage collection

Implementation choices

- all metadata in readable JSON
- all data access over HTTP or local file
  - metadata/data can be recovered using scripts without needing a running DDFS

Hadoop vs Disco Benchmarks

8-node physical cluster of Cisco UCS M2
each node with 4 Xeon, 128GB RAM, 512GB disk
from 2011

Job Latency

Wordcount on a 1 byte file

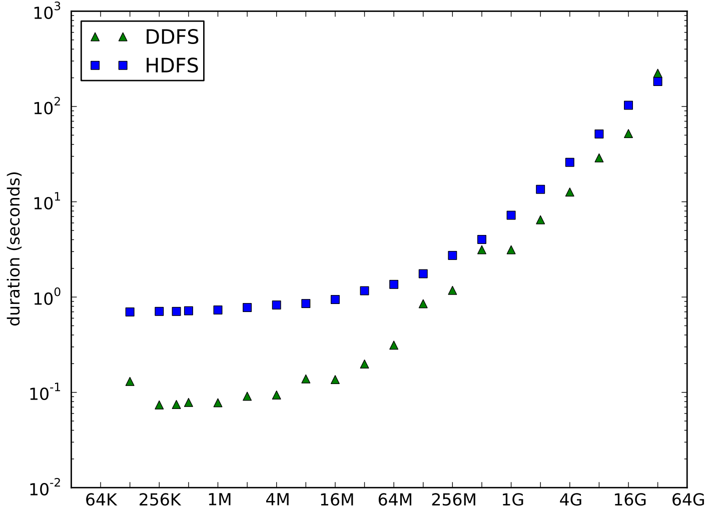
	Completion time (ms)
Hadoop	12324
PDisco	359
ODisco	35

DFS Latencies

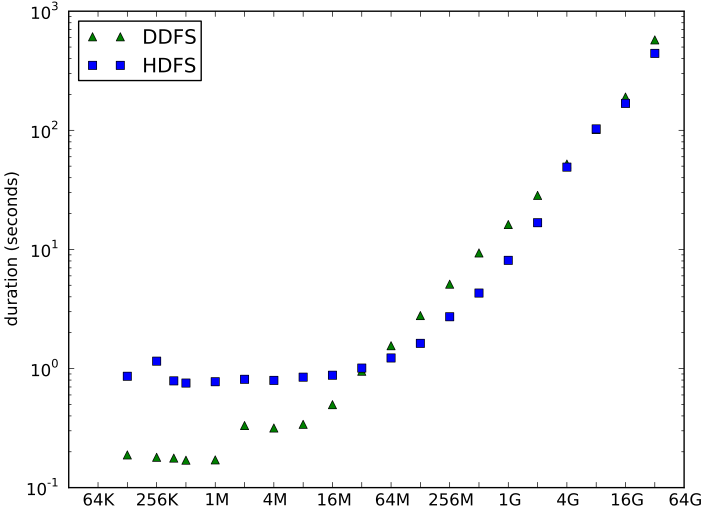
Read / Write a 1 byte file (avg in msecs)

	HDFS	DDFS
Reads	670	70
Writes	720	136

DFS Read Throughput



DFS Write Throughput



Job Performance

Wordcount of English Wikipedia (33GB)