Incoop: MapReduce for Incremental Computation

Pramod Bhatotia, Alexander Wieder,
Rodrigo Rodrigues, Umat A. Acar,
Rafael Pasquini

Presented by Albert Kim

Background

- MapReduce revolutionized bulk data processing
  - Highly scalable and simple
- Many datasets are constantly changing
  - Examples: web index, log processing
- Need to deal with incremental changes

Goal

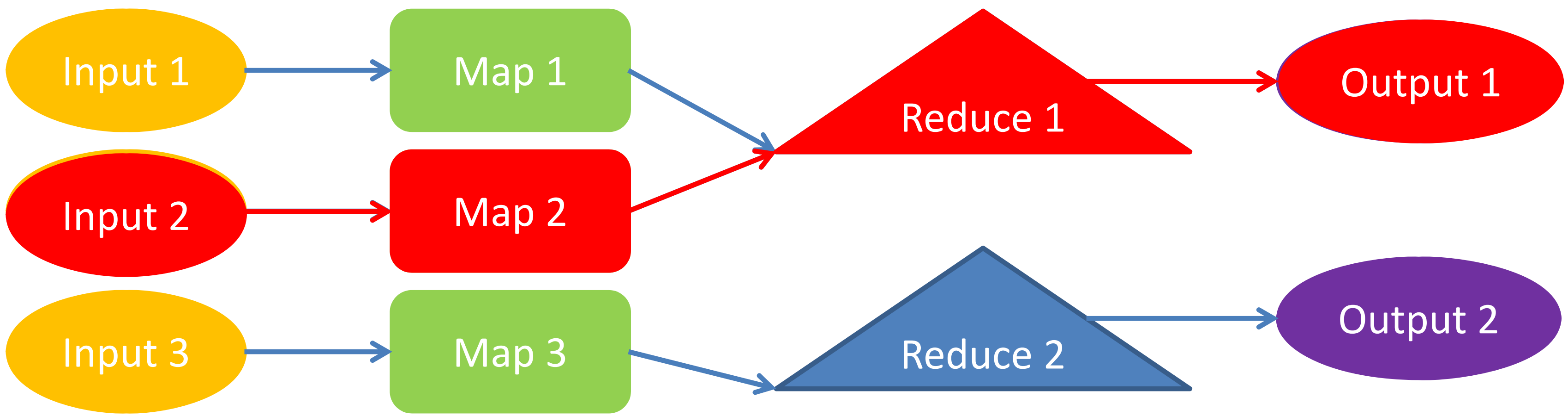
MapReduce-like framework that can deal with incremental changes to the input transparently and efficiently

3 Key Ideas:

- Transparency
- Efficiency
- MapReduce-like

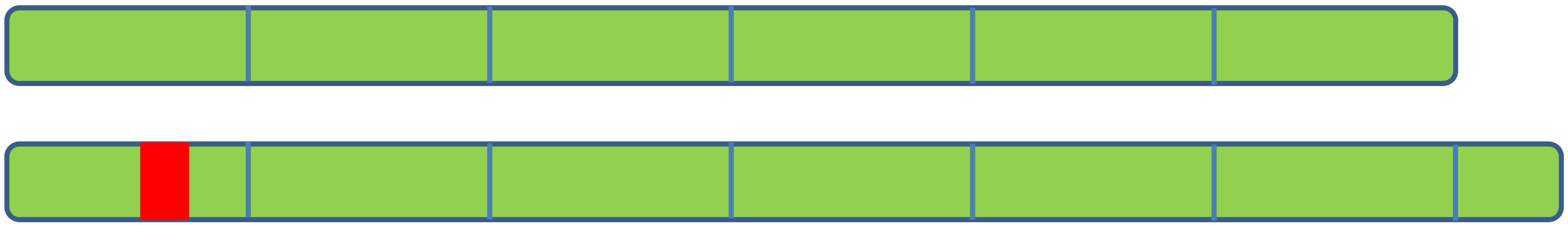
Overview

- **Memoization**
- Record each input/output for every map and reduce task (memoization server)
- In future iterations, only run map and reduce tasks if their input has changed



Incremental Map

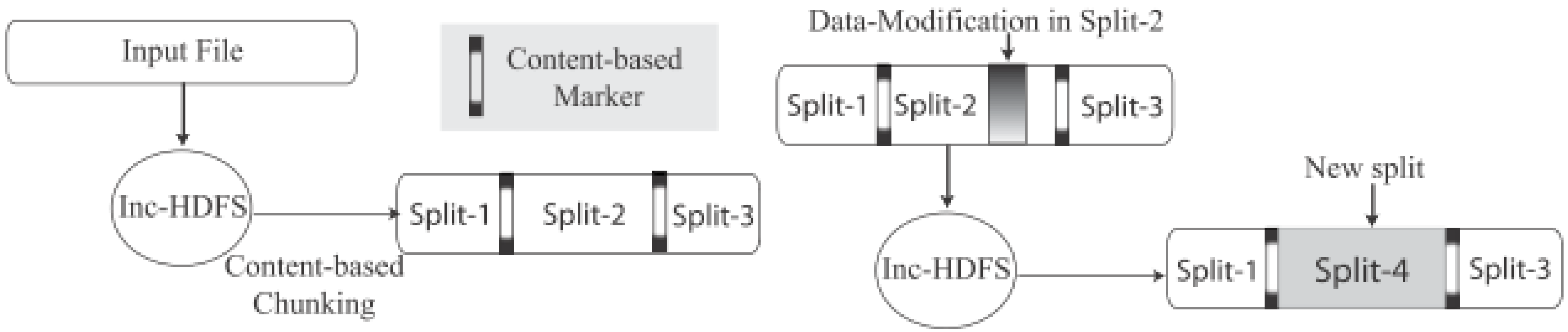
- Easy for in-place modification, but what about insertions or deletions? (*stability*)



- Instead of using, fixed-offset partitioning, use content-based partitioning
- Content-based partitioning: decides partition boundaries based on local input content
  - Same content = same boundaries

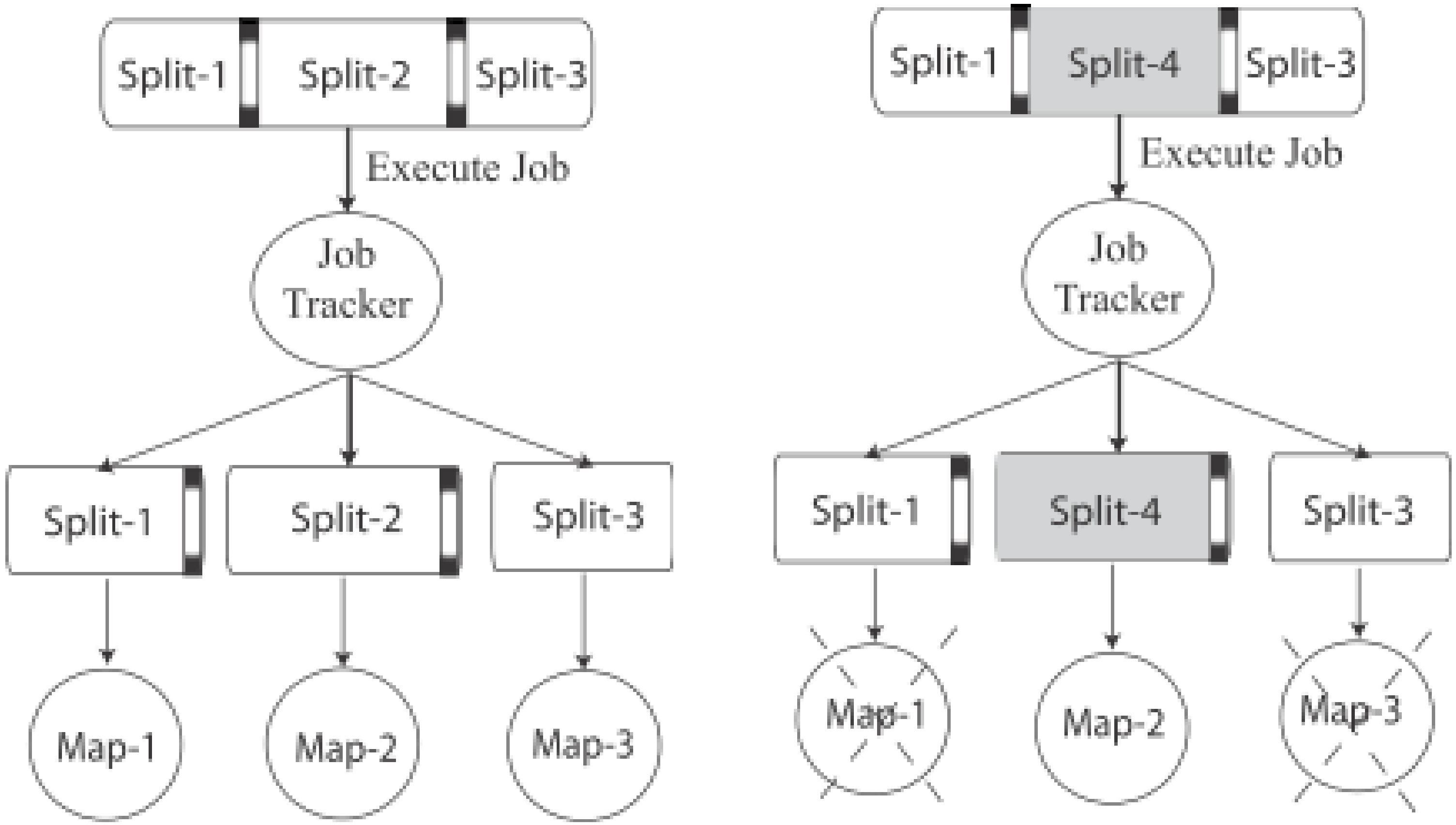
Incremental Map

- Scan file using sliding window and compute fingerprint for each window
- If fingerprint matches *marker* pattern, it is a partition boundary



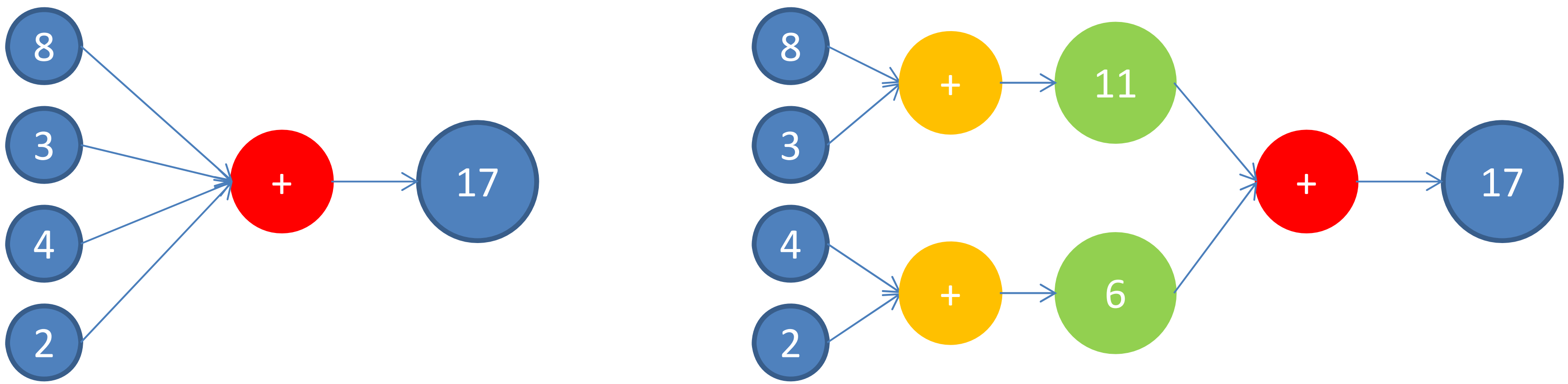
- Can have min/max offsets to make sure partitions aren't too small/big

Incremental Map



Incremental Reduce

- Reduce tasks can be large, and changing one input will force the task to rerun (*granularity*)
- Need a way to split up reduce tasks: **Combiners**
- New *Contraction Phase* which groups input into chunks

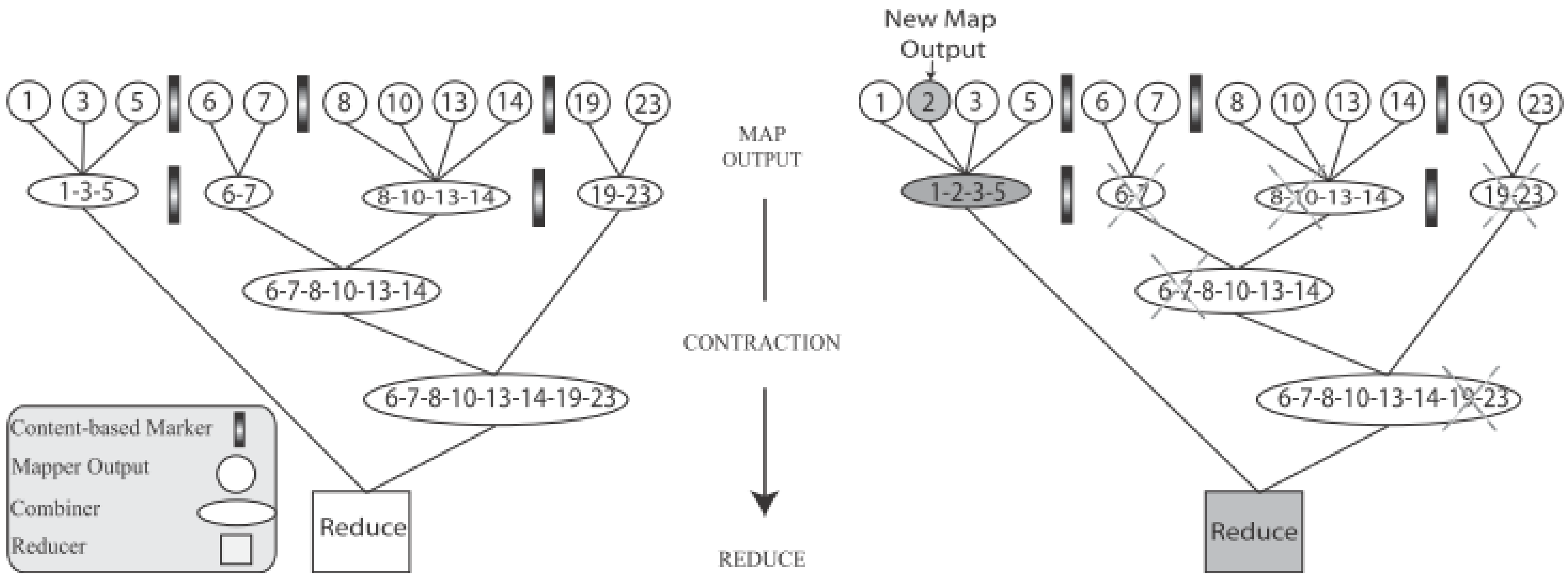


Incremental Reduce

- Now we can memoize input/output to combiner tasks and reduce tasks

- How do we partition reduce tasks into combiner groups?
  - Use content-based partitioning again!

Incremental Reduce



Memoization-Aware Scheduler

- Augment scheduler to take into account memoization locality while still flexible enough to deal with stragglers
- Simple work-stealing algorithm
  - Each node has queue of tasks
  - Tasks are assigned to queue based on memoization locality
  - Nodes steal work from largest queues with mininmum memoization locality

Evaluation

Version	Skip Offset [MB]	Throughput [MB/s]
HDFS	-	34.41
Incremental HDFS	20	32.67
	40	34.19
	60	32.04

- 20 MB generates too many fingerprints
- 60 MB means not enough parallelization within one file
  - Increase file size?
  - Process more than one file at a time?

Evaluation

- Work – total computation done by system
- Time – end-to-end time taken to finish job

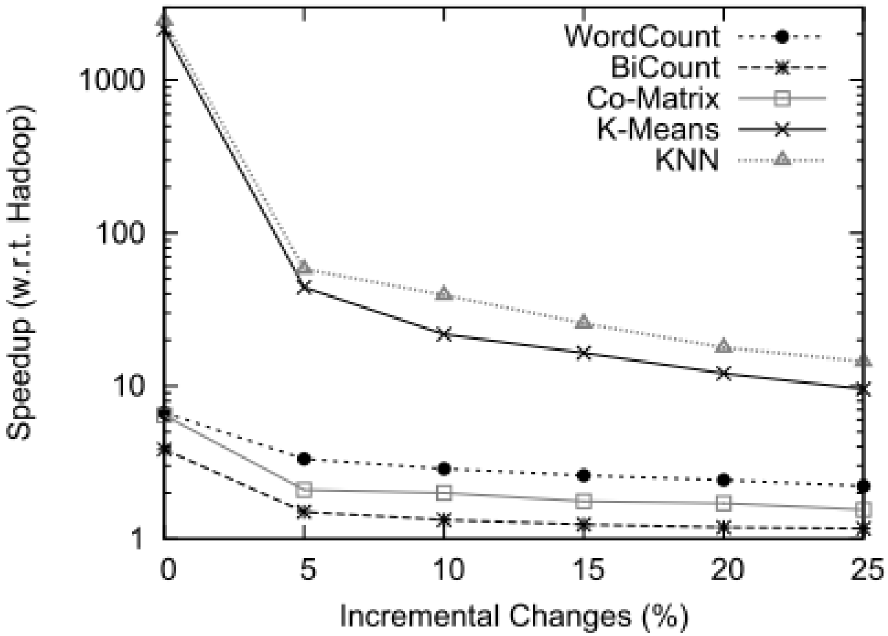


Figure 5: Work speedups versus change size.

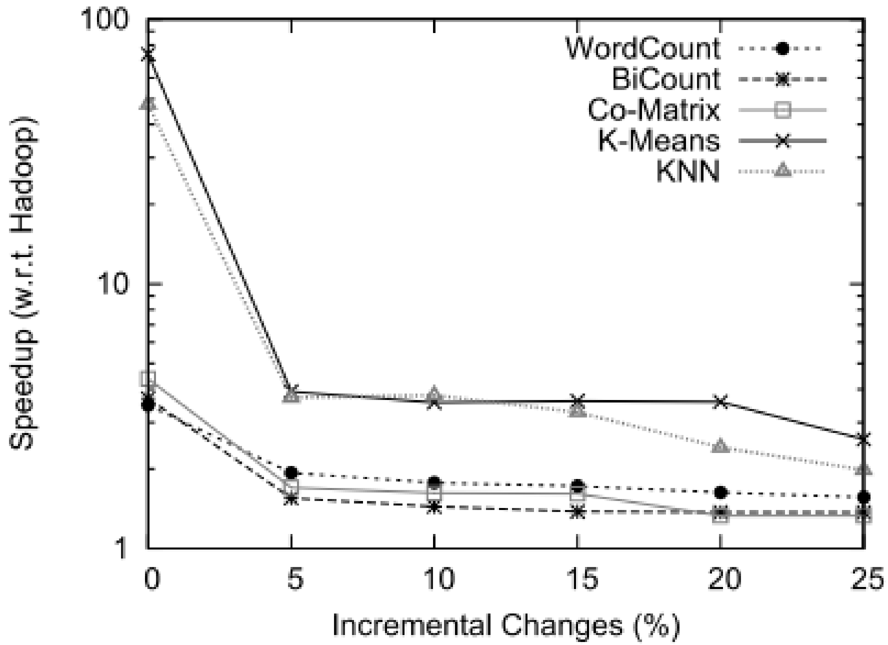
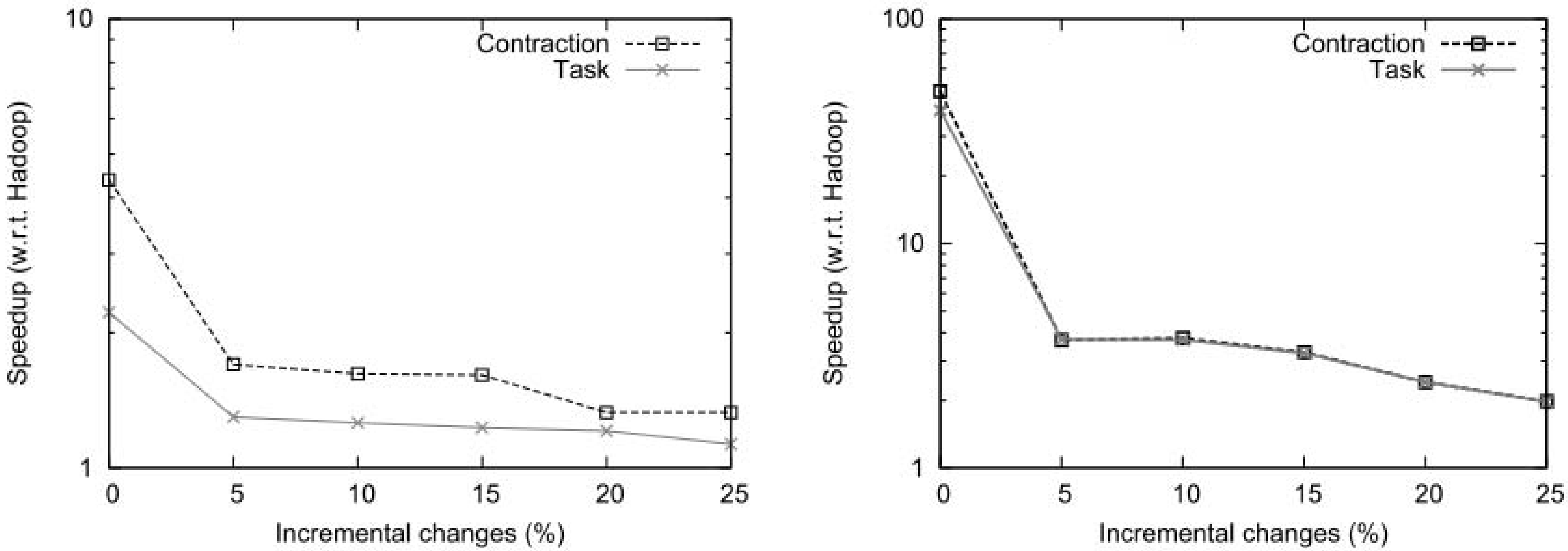


Figure 6: Time speedups versus change size.

Evaluation



(a) Co-occurrence Matrix

(b) k-NN Classifier

Figure 8: Performance gains comparison between `Contraction` and `task` variants

Evaluation

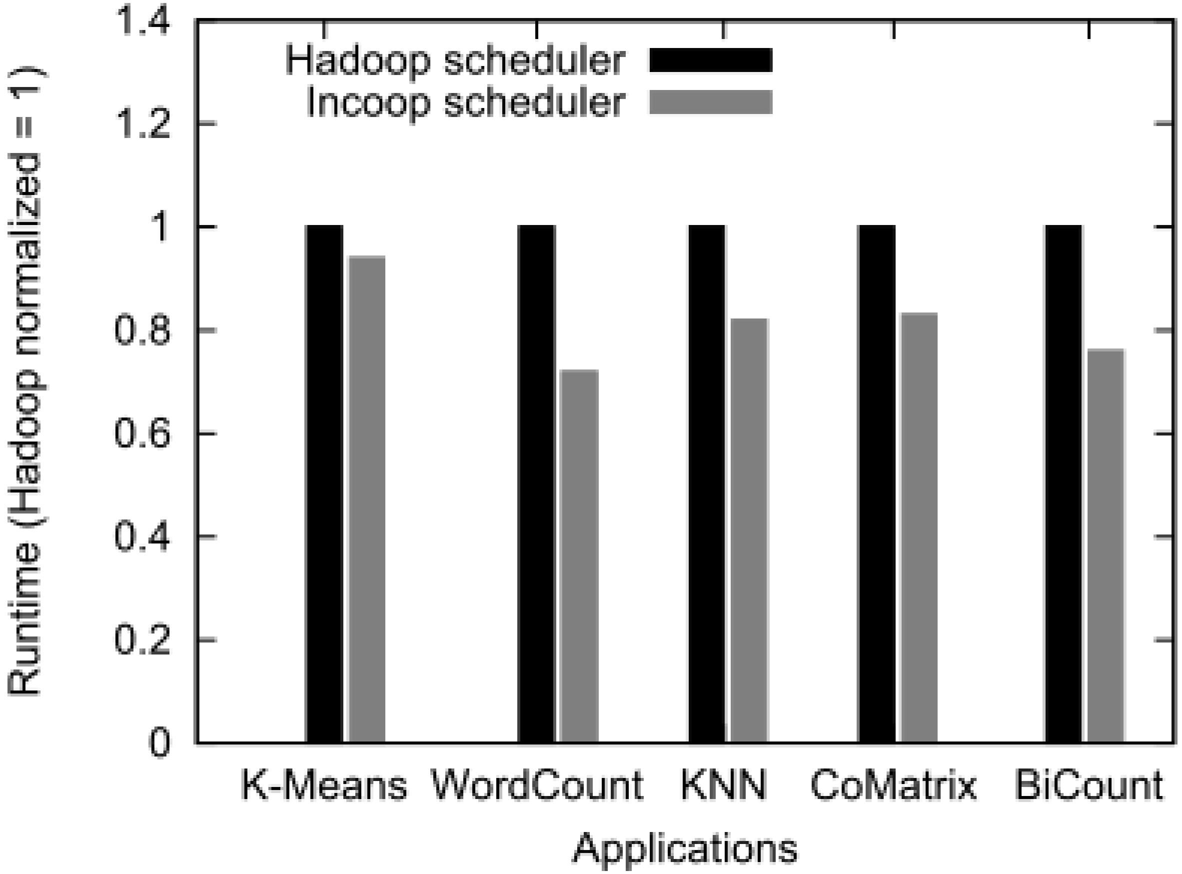
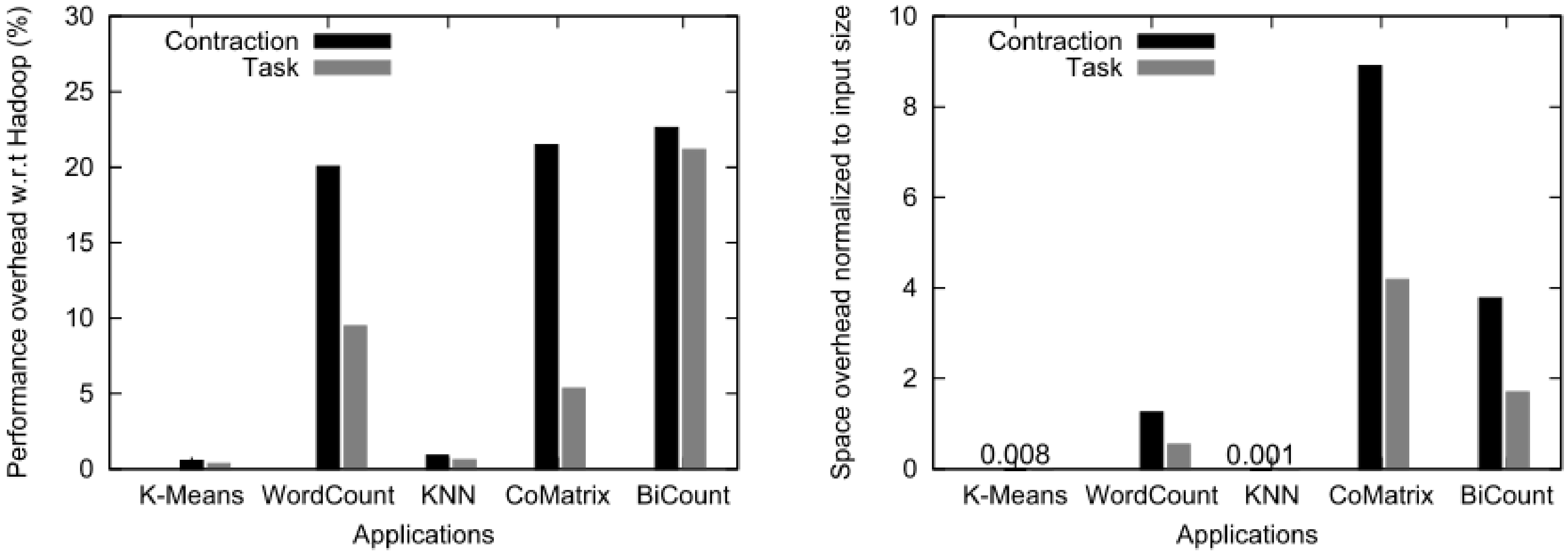


Figure 9: Effectiveness of scheduler optimizations.

Evaluation



(a) Performance overhead for the first job run

(b) Space overhead

Figure 10: Overheads imposed by Incoop in comparison to Hadoop

Related Work

- Programming language-based approaches
  - Assumes sequential, non-distributed, uniprocessor model
- Google's Percolator, Yahoo!'s CBP
  - Not transparent to programmer
- DryadInc, Nectar, Haloop
  - Incoops uses effective content-based stability partitioning
  - Incoop has MapReduce-like framework

Comments

- Overall nice work!
- Transparency
  - Need to write combiners
- How do you get a good marker pattern?
  - Preprocess the data?
- What granularity did they change data for evaluation
- Graph on end-to-end time for initial run + update run would be nice
- Would be nice to compare with Percolator