Haloop

Efficient Iterative Data Processing on Large Clusters

University of Washington, 2010
Yingyi Bu / Bill Howe / Magdalena Balazinska / Michael D. Ernst

2014104124 Lee kyu Tae, 2014104155 Chae Myung Jun

Background

1. An opensource implemented base on Mapreduce
2. For performing large-scale data processing using "commodity computer" clusters
3. Can scale to thousands of nodes in a fault-tolerant manner

※ Because of version gap, There is some difference in Hadoop what you knew.

Motivation

1. limit of Hadoop

- Not Support Iterative Data Processing, (like PageRank, K-means)

2. Related work

Framework	Mahout	Twister	Pregel
Concept	Machine learning libraries on top of Hadoop.	Stream-based MapReduce framework	Processing graph data
Iteration	Not Support	Support	Not Support
Characteristic	Can use the iteration by injecting from user	Sensitive to failures	-

Haloop

For distribution processing big data focused on cached data

OutLine

1. Architecture

2. Program Model
   - Formula
   - API

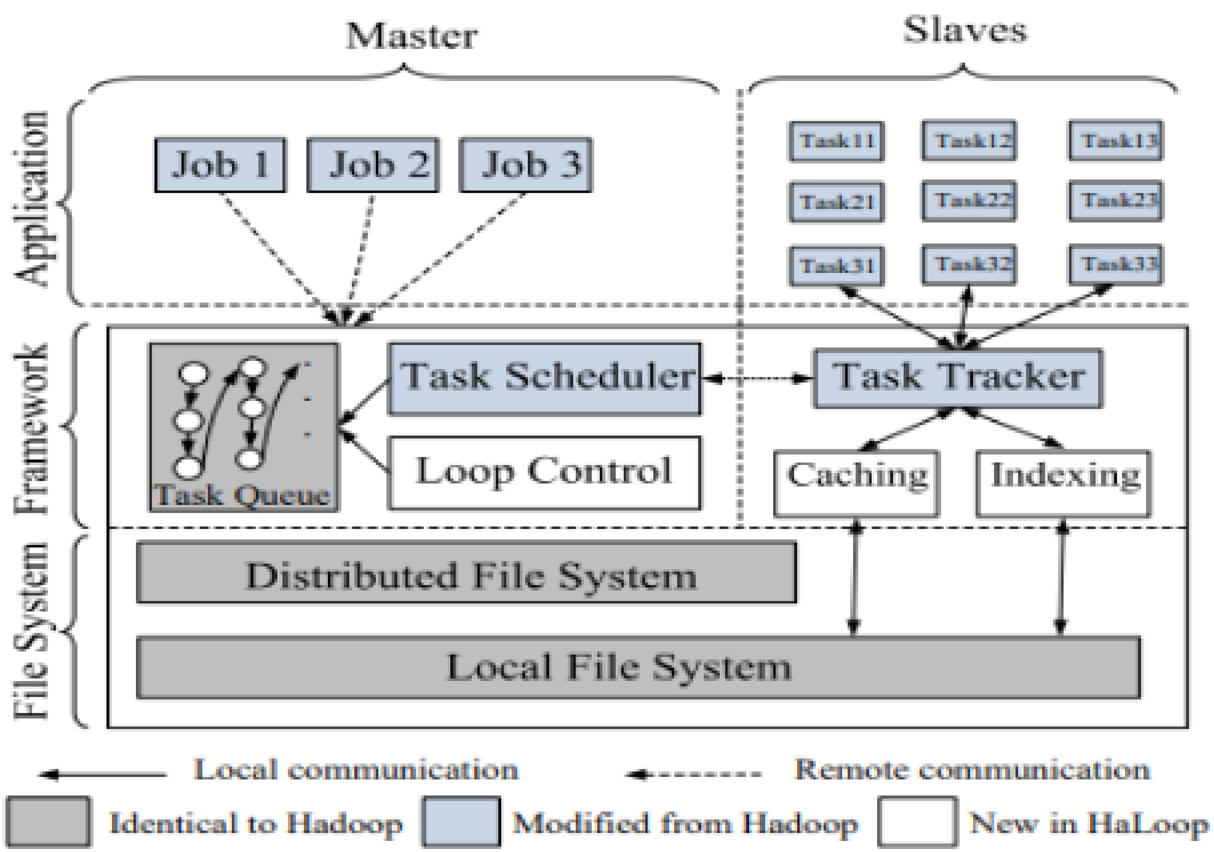
3. Loop Aware Task Scheduling
   - Inter Iteration Locality
   - Scheduling Algorithm

4. Caching & Indexing
   - Reducer Input Cache
   - Reducer Output Cache
   - Mapper Input Cache
   - Cache Reloading

5. Experiment Evaluation

6. Conclusion

Architecture



Based On : Hadoop Architecture
Modified : Task Scheduler / Task Tracker
New : Loop Control / Caching / Indexing

Programming Model

1. Suitable algorithms

Algorithms what using invariant data in each iterations

Because Haloop get better performance when using caching and indexing.

Ex) Page Rank, K-means, Descendant query

Programming Model

1. Formula

$$R_{i+1} = R_0 \cup (R_i \bowtie L)$$

2. Terminate Conditions

- specified maximum number of iteration
- convergence condition
  a) Default Fixpoint ($R_{i+1} = R_i$)
  b) Approximate Fixpoint ($R_{i+1} - R_i$ <= Specified Threshould)

Programming Model

3. API

: Haloop provides an efficient foundation API for Iteration

1) Loop Body ; constructed a multistep MapReduce Job

a) AddMap
b) AddReduce
※ when constructing Loop Body, you must indicate that order of MapReduce

2) Termination Condition

a) Set Fixed Point Threshold
b) Result Distance
c) Set Max Num Of Iterations

3) Control Input

a) Set Iteration Input
b) Add Step Input
c) Add Invariant Table

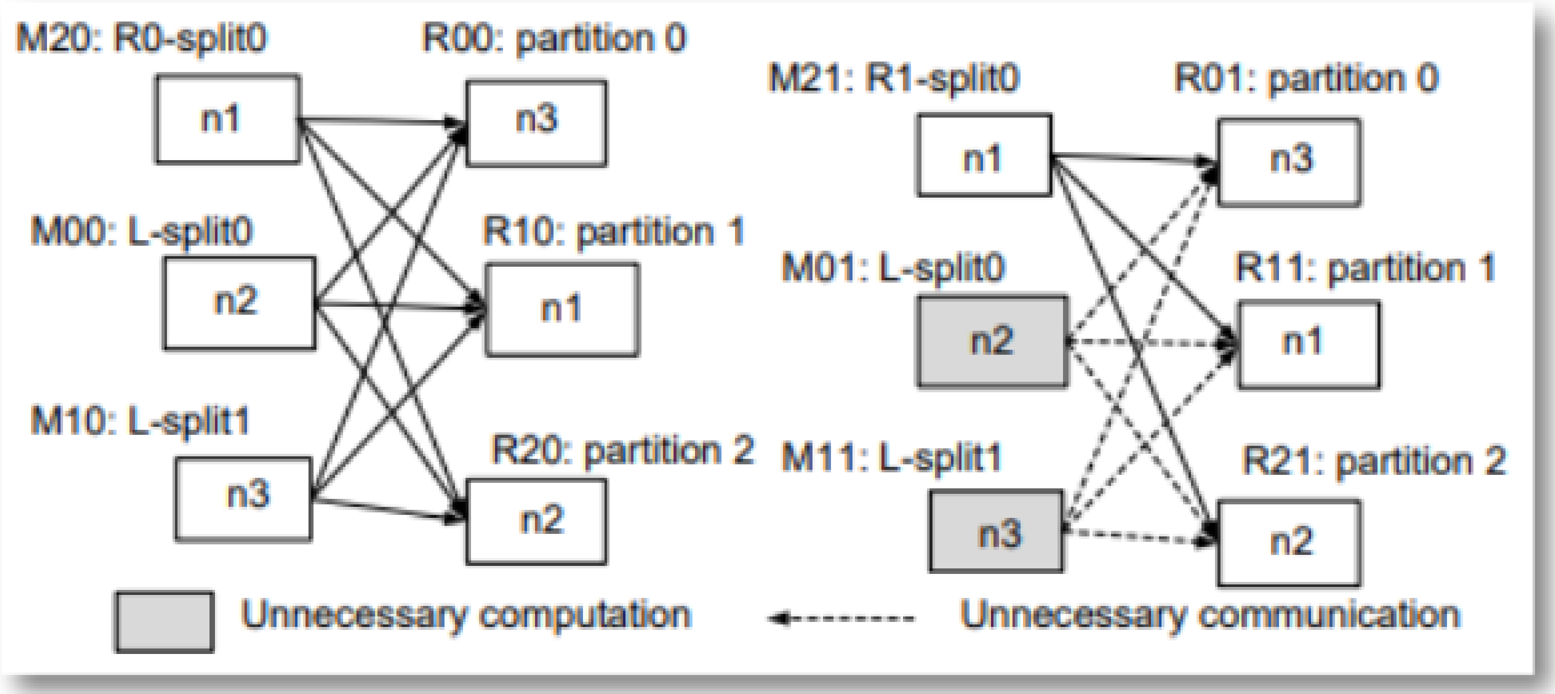
Loop-Aware Task Scheduling

1. Goal of Haloop's Scheduler

To place on the same physical machines
those map and reduce tasks that occur in different iterations
but access the same data

Loop-Aware Task Scheduling

2. Inter-Iteration Locality



※ Example of Inter-Iteration Locality

Another Iteration / Same Data / Same Node

Loop-Aware Task Scheduling

3. Scheduling Algorithm

Task Scheduling

Input: Node node

// The current iteration's schedule; initially empty

Global variable: Map⟨Node, List⟨Partition⟩⟩ current

// The previous iteration's schedule

Global variable: Map⟨Node, List⟨Partition⟩⟩ previous

```
if iteration == 0 then
    Partition part = hadoopSchedule(node);
    current.get(node).add(part);
else
    if node.hasFullLoad() then
        Node substitution = findNearestIdleNode(node);
        previous.get(substitution).addAll(previous.remove(node));
        return;
    end if
    if previous.get(node).size()>0 then
        Partition part = previous.get(node).get(0);
        schedule(part, node);
        current.get(node).add(part);
        previous.remove(part);
    end if
end if
```

Caching & Indexing

1. Overview

Thanks to the Inter-iteration locality, Only one physical node is needed for processing Loop invariant data

Types of caches

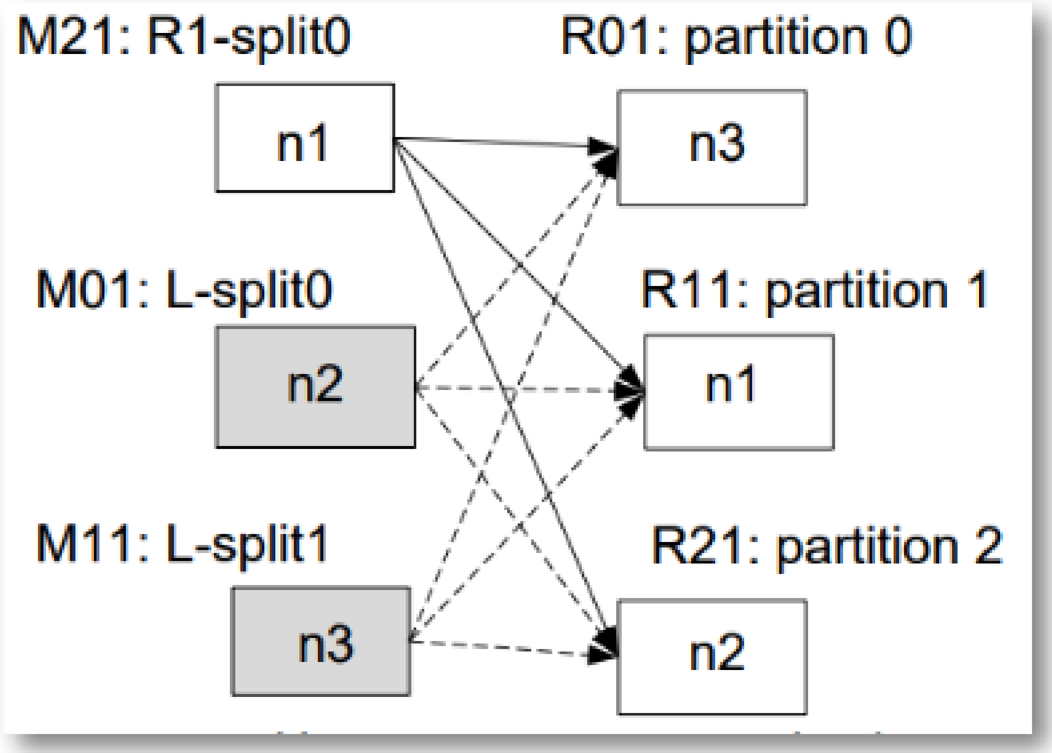
- Reducer Input Cache
- Reducer Ouput Cache
- Mapper Input Cache

To reduce I/O cost, Haloop caches those data partitions

To accelerate processing, Haloop indexes cached data

Caching & Indexing

2. Reducer Input Cache

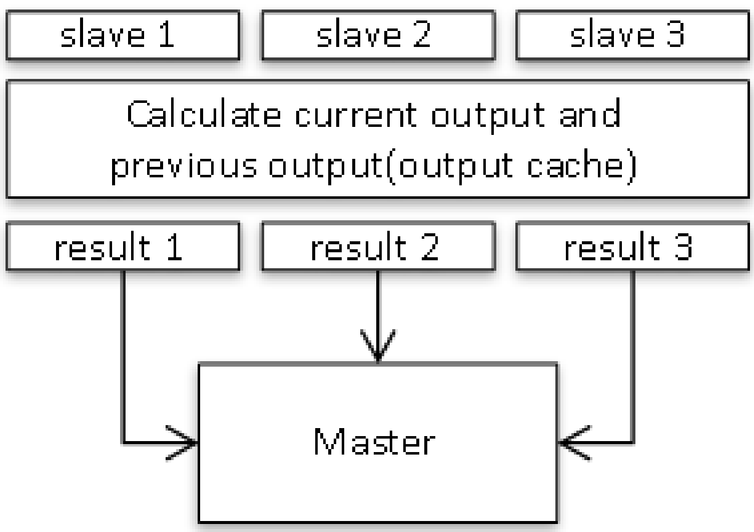


: There is no necessary to process M01 and M02 mapper, because reducers alreay have output of L-split0 and L-split1

Caching & Indexing

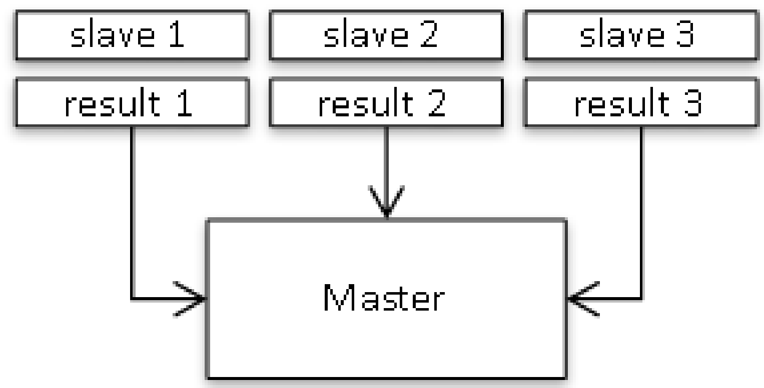
3. Reducer Output Cache

Using



1. compute those result
2. compare with computed result and threshold
3. decide to repeat next iteration

Non-Using



1. access previous result of iteration
2. group each result by key
3. Calculate current output and previous output
4. compute those result
5. compare with computedresult and threshold
6. decide to repeat next iteration

4. Mapper Input Cache

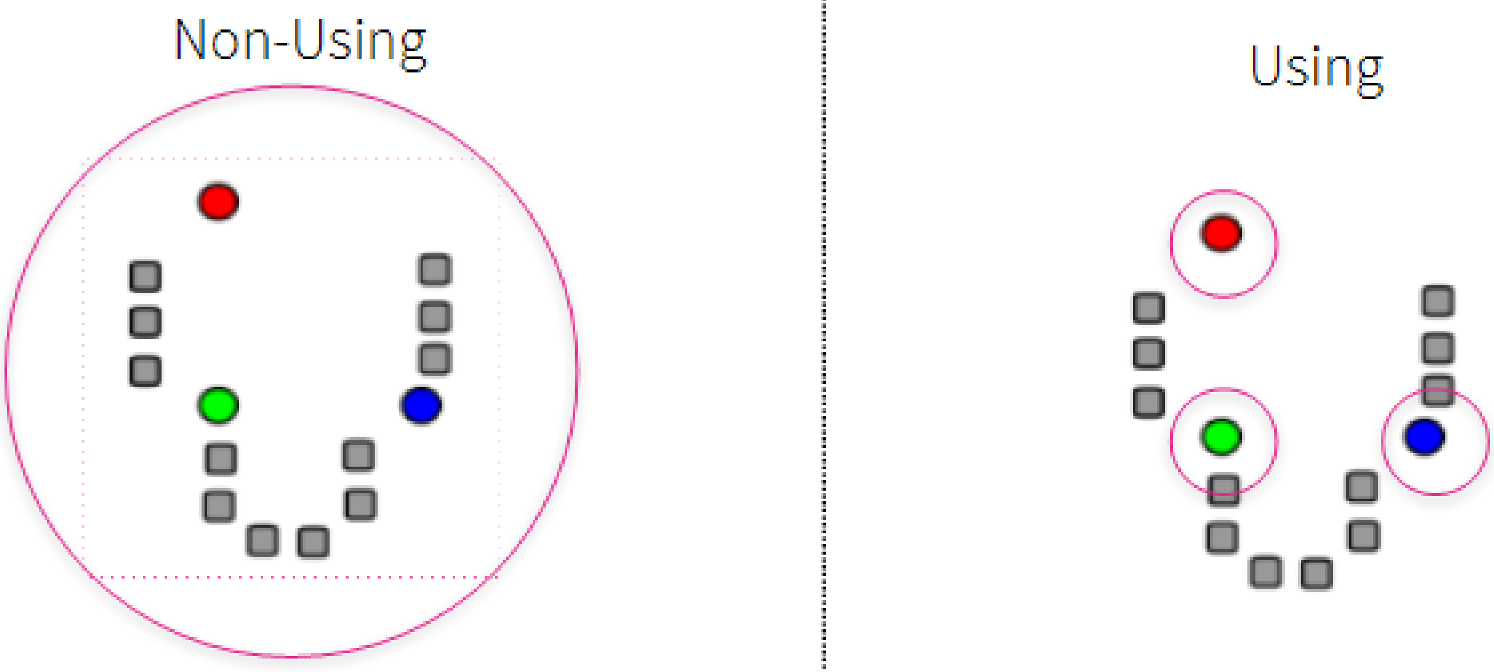
Because the first iteration get data from HDFS, local file system

and cache them

So there is no necessary to reload data next iterations

Caching & Indexing

4. Mapper Input Cache



- Datum point for clustering
- Data what will be clustered
- Data to read from HDFS or local file system

※K-means Algorithm

Caching & Indexing

4. Cache Reloading

: if node is out of order or full, Cache reloading is necessary

1) Reducer Input Cache Reloading

: Duplicate Initial Mapper output result

2) Reducer ouptut Cache & Mapper input Cache Reloading

: Reload from HDFS

EXPERIMENTAL EVALUATION

1) Compared preformance of iterative data analysis on HaLoop and Hadoop

  a) reducer input cache
  b) reducer output cache
  c) mapper input cache

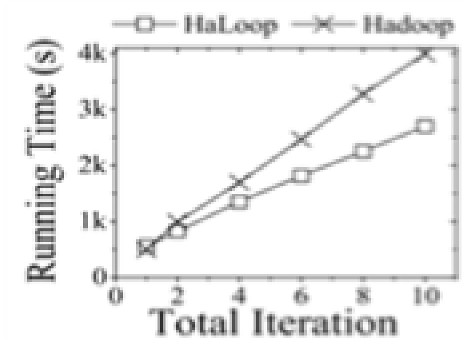
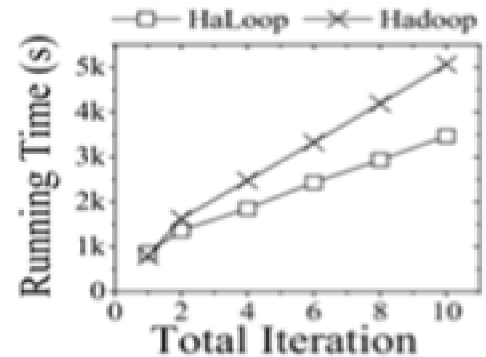
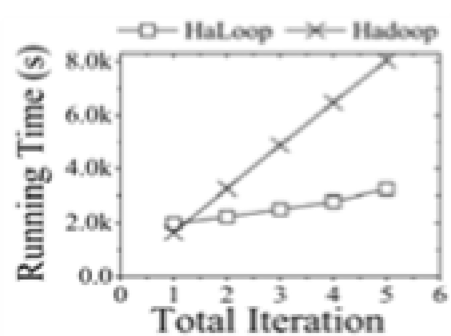
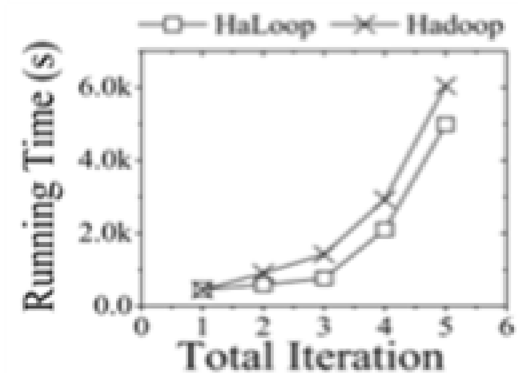
2) Using Data Set

Name	Data capactiy	Detail
Livejournal	18GB	social network data
Triples	120GB	semantic web data
Freebase	12GB	concept linkage graph

3) Environment setting

- virtual machine nodes: 50
- slave nodes(EC2) : 90
- master node : 1

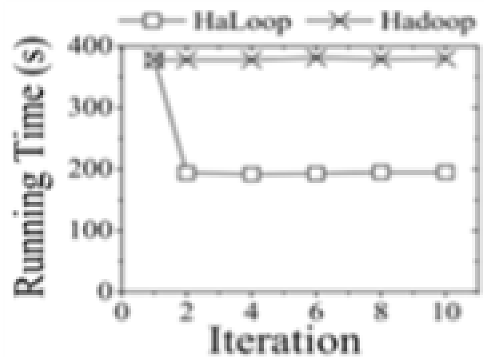
Evaluation of Reducer Input Cache

<table>
<tr><th colspan="2">Over all</th></tr>
<tr><th colspan="2">Page Rank</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Livejurnal Dataset, 50 nodes</td><td>Freebase Dataset, 90 nodes</td></tr>
<tr><th colspan="2">Descendant query</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Triples Dataset, 90 nodes</td><td>Livejurnal Dataset, 50 nodes</td></tr>
</table>

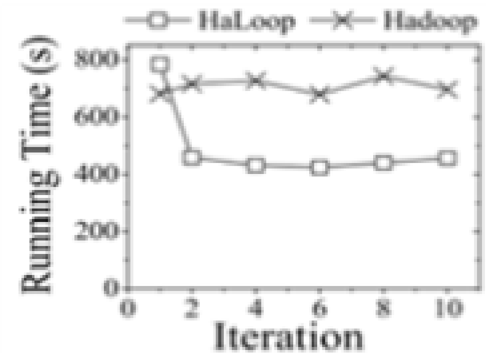
Evaluation of Reducer Input Cache

Join Step

Page Rank

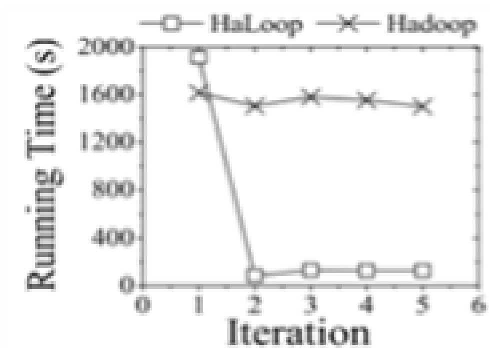


Livejurnal Dataset, 50 nodes

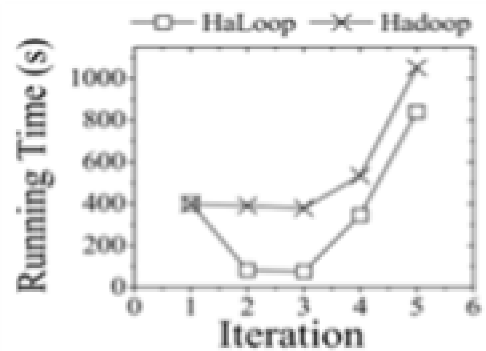


Freebase Dataset, 90 nodes

Descendant query



Triples Dataset, 90 nodes

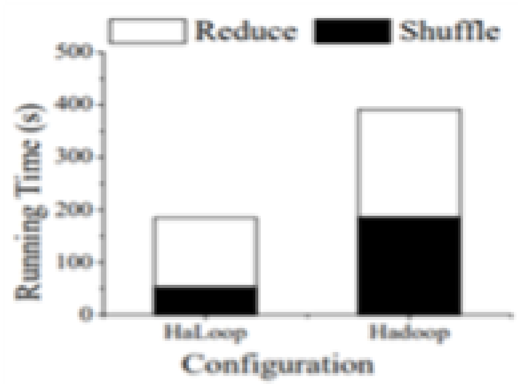


Livejurnal Dataset, 50 nodes

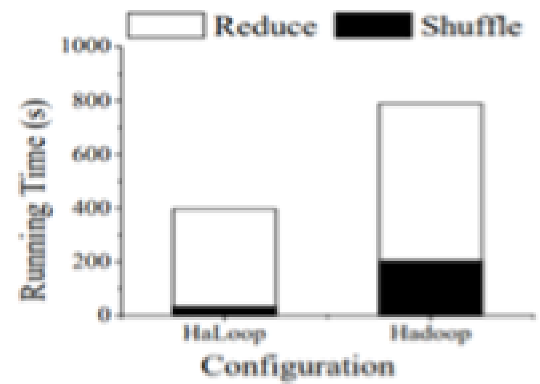
Evaluation of Reducer Input Cache

Cost Distribution for Join Step : iteration 3

Page Rank

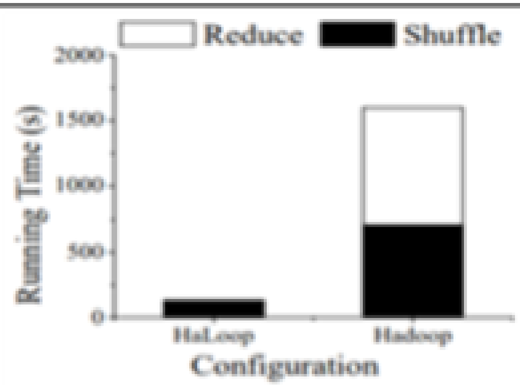


Livejurnal Dataset, 50 nodes

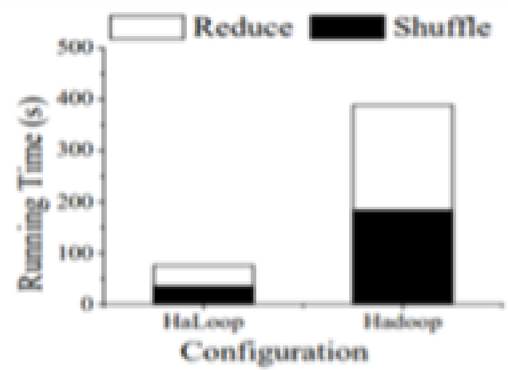


Freebase Dataset, 90 nodes

Descendant query



Triples Dataset, 90 nodes

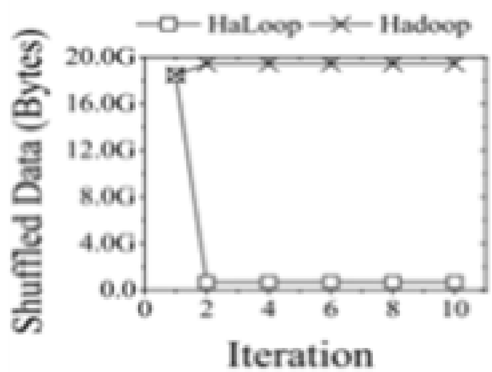


Livejurnal Dataset, 50 nodes

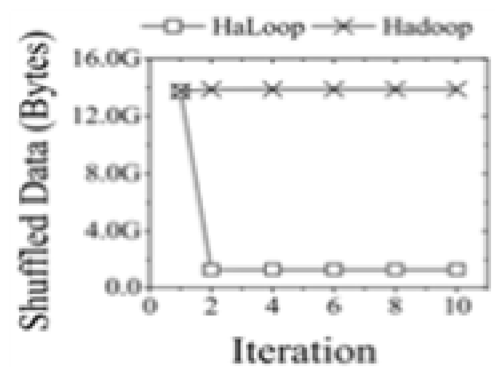
Evaluation of Reducer Input Cache

I/O in Shuffle Phase of Join Step

Page Rank

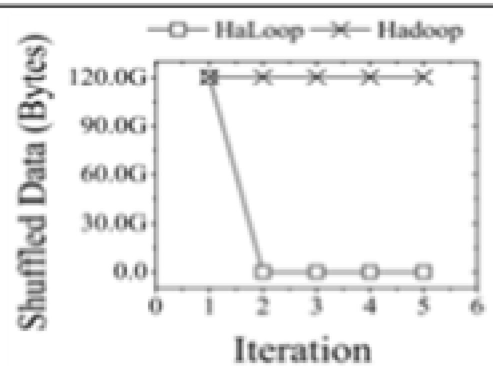


Livejurnal Dataset, 50 nodes

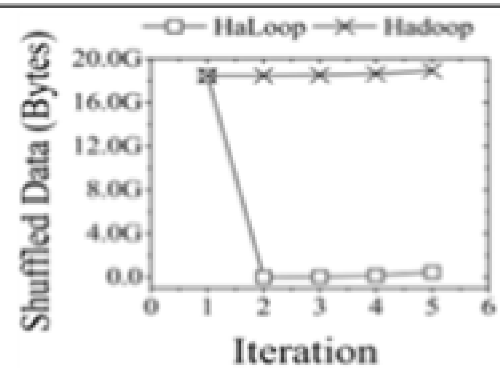


Freebase Dataset, 90 nodes

Descendant query



Triples Dataset, 90 nodes

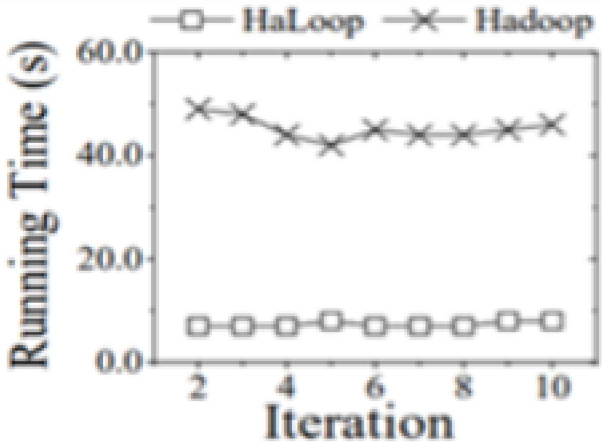


Livejurnal Dataset, 50 nodes

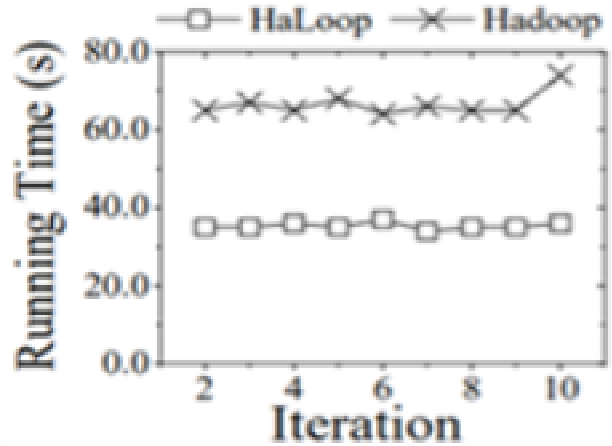
Evaluation of Reducer Output Cache

Compared the cost of fixpoint evaluation

Page Rank



Livejurnal Dataset, 50 nodes



Freebase Dataset, 90 nodes

Analysis

- **Haloop reduces the cost of this step to 40%**
- **Hadoop need an extra MapReduce job**

Evaluation of Mapper Intput Cache

1. Using Application

k-means clustering algorithm

2. Environment Setting

8-node physical machine cluster

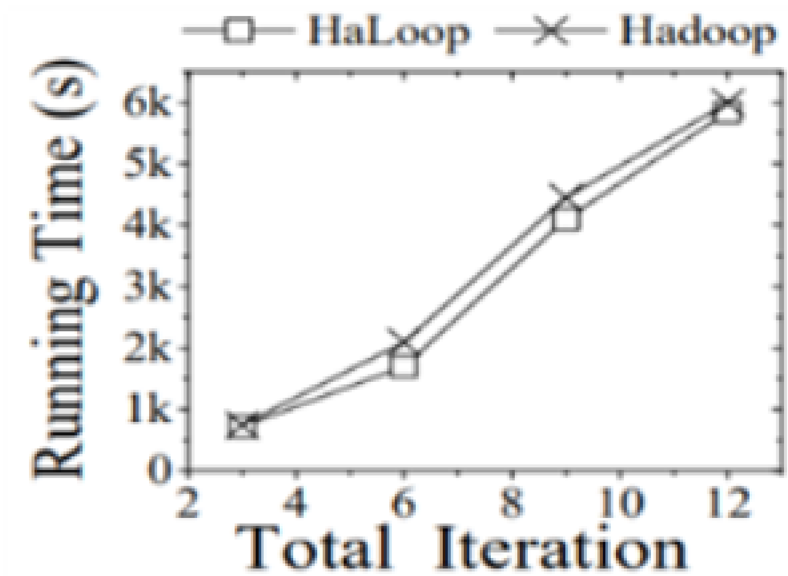
3. Data Set

Name	Data capactiy
cosmo-dark	46GB
cosmo-gas	54GB

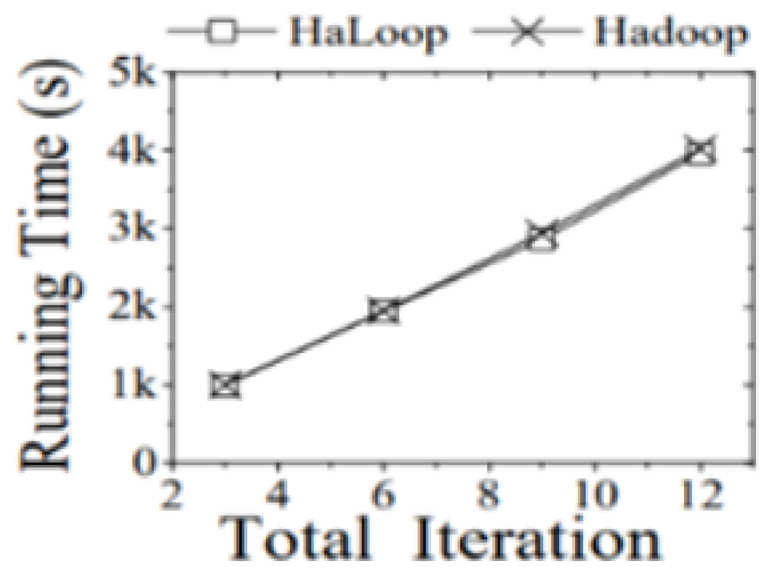
Evaluation of Mapper Intput Cache

Running Time

k-means



Cosmo-dark



Cosmo-gas

Conclusion

Haloop is

New Programming Model and Architecture for Iterative Programs

have advantages

a) Data can reuse across iterations by Loop-Aware Task Scheduler

b) Reduce the I/O cost in next iterations by Caching loop-invariant data

c) Reduce useless Mapreduce step for checking fixpoint or convergence by Caching Reducer's output

outperforms in Iterative Programs

a) Reduce query runtimes by 1.85

b) Shuffles only 4% of the data between mappers and reducers

Q&A