HaLoop

Efficient Iterative Data Processing on Large Clusters

Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst
University of Washington
Department of Computer Science & Engineering

Presented by Albert Haque
University of Texas at Austin
April 15, 2013

Many of the slides in this presentation were taken from the author's website, which can be found here: http://www.ics.uci.edu/~yingyib/

HaLoop: Efficient Iterative Data Processing on Large Clusters
In *VLDB'10*: The 36th International Conference on Very Large Data Bases, Singapore, 24-30 September, 2010.

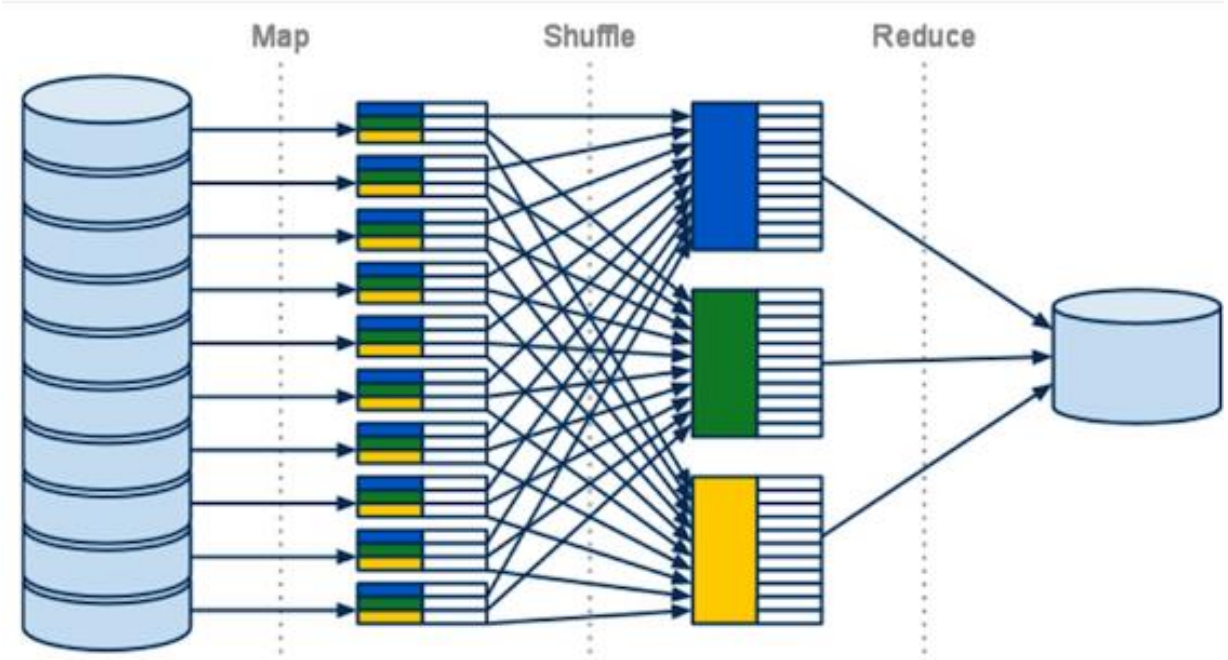
Agenda

I. Background
II. System Design
    A. Loop-Aware Scheduling
    B. Caching & Indexing
III. Performance
IV. Related Work
V. Conclusion

Background

Hadoop

- Used at Google, Facebook, Yahoo, eBay, etc.
- MapReduce, fault-tolerance, uses commodity hardware, and can scale to thousands of nodes



Iterative Computations

Data is processed iteratively until it converges or satisfies a stopping condition

- PageRank
- HITS (Hypertext-Induced Topic Search)
- Recursive Relational Queries
- Clustering
- Neural and Social Network Analysis

Problem: Static Data

- Chunks of data may be static
- But MapReduce reloads and processes it at each iteration!
  - Excess I/O
  - Wastes network bandwidth
  - Takes up CPU cycles

Problem: Fixpoint Detection

Fixpoint –output has not changed from last iteration

$$\left|V_{t_0} - V_{t_1}\right| < \epsilon$$

- May require an extra MapReduce job
  - Adds extra tasks
  - Requires additional disk I/O
  - More network usage

Problem: Fixpoint Detection

Fixpoint –output has not changed from last iteration

$$|V_{t_0} - V_{t_1}| < \epsilon$$

Key Problems with Hadoop:

1. Invariant Data
2. Fixpoint Detection

- May require an extra MapReduce job
  - Adds extra tasks
  - Requires additional disk I/O
  - More network usage

PageRank

Initial Rank Table R0	
url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

Linkage Table L	
url_source	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.d.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.com
www.a.com	www.d.com

PageRank

Initial Rank Table R0	
url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

Linkage Table L	
url_source	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.d.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.com
www.a.com	www.d.com

Linkage Table L is invariant across iterations

Termination

When to stop iterating?

- We stop when the values don't change
- Need an extra MapReduce step

Rank Table R3	
url	rank
www.a.com	2.13
www.b.com	3.89
www.c.com	2.60
www.d.com	2.60
www.e.com	2.13

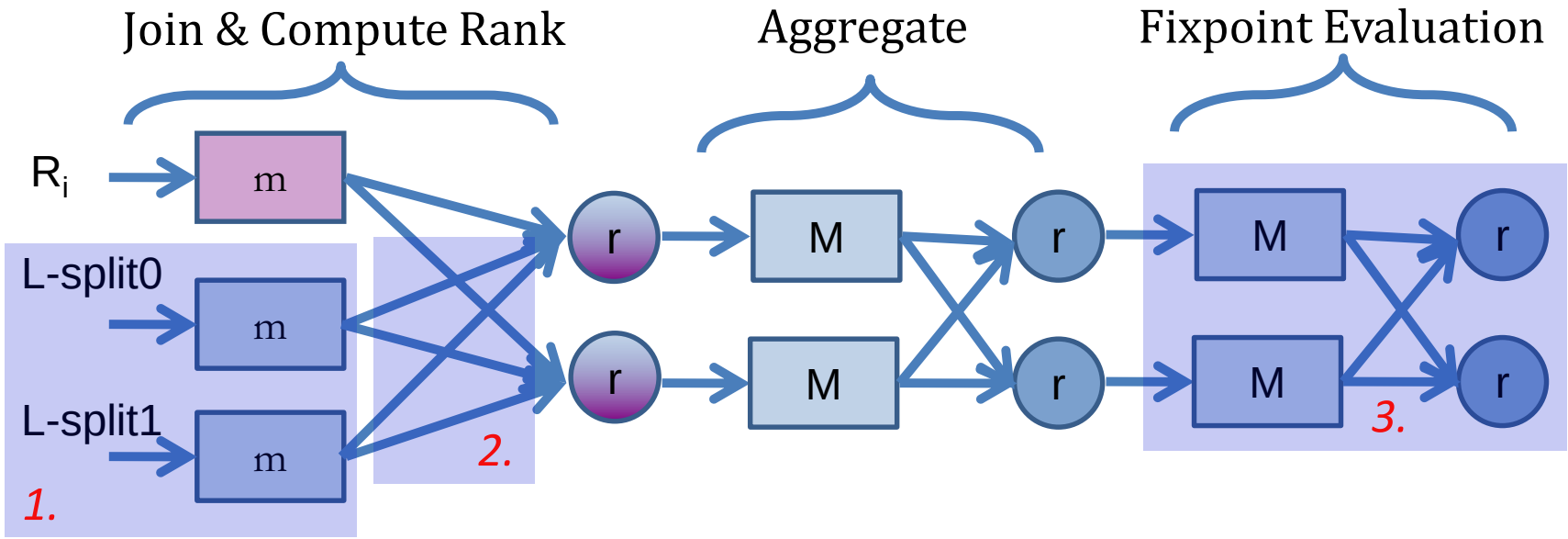
<u>MapReduce Job</u>

$\Delta rank \leftarrow rank_i - rank_{i-1}$

if $\Delta rank < \epsilon$ **then** terminate

else if iteration_num > max_iterations **then** terminate

What is the Problem?



L is loop invariant, but…

1. L is loaded on each iteration
2. L is shuffled on each iteration
3. Fixpoint evaluated as a separate MapReduce job per iteration

Descendant Query

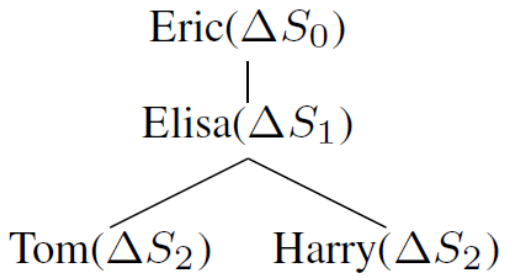
name1	name2
Tom	Bob
Tom	Alice
Elisa	Tom
Elisa	Harry
Sherry	Todd
Eric	Elisa
Todd	John
Robin	Edward

(a) Friend Table F

$$MR_1 \begin{cases} T_1 = \Delta S_i \bowtie_{\Delta S_i.\text{name2}=F.\text{name1}} F \\ T_2 = \pi_{\Delta S_i.\text{name1},F.\text{name2}}(T_1) \end{cases}$$

$$MR_2 \begin{cases} T_3 = \bigcup_{0 \le j \le (i-1)} \Delta S_j \\ \Delta S_{i+1} = \delta(T_2 - T_3) \end{cases}$$

(b) Loop Body



(c) Result Generating Trace

name1	name2
Eric	Elisa
Eric	Tom
Eric	Harry

(d) Result Table ΔS

Solution

- HaLoop offers solutions to these problems:
  1. A New Programming Model & Architecture for Iterative Programs
  2. Loop-Aware Task Scheduling
  3. Caching for Loop Invariant Data
  4. Caching for Fixpoint Evaluation

System Design

Loop-Aware Scheduling

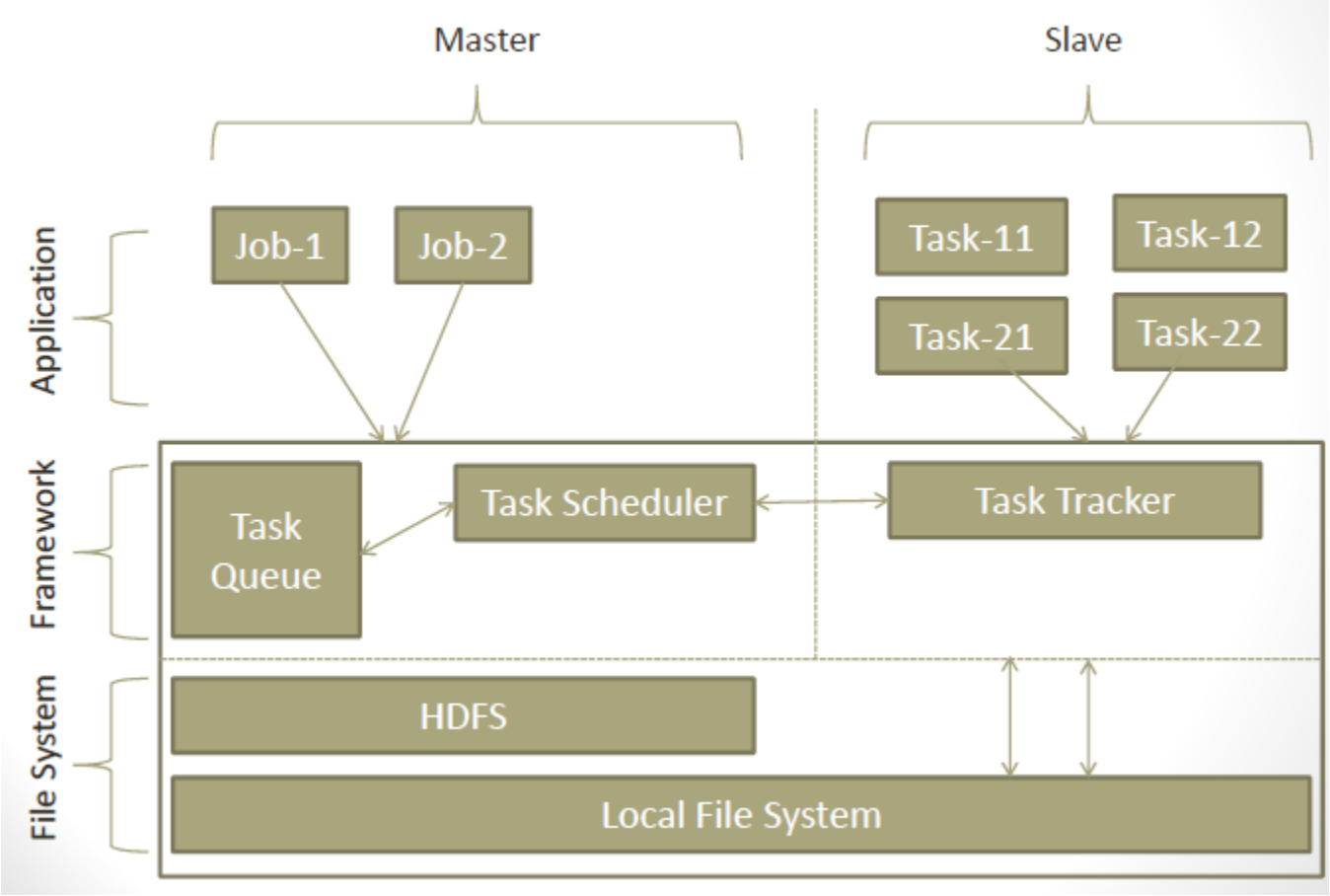
- Starts new MapReduce jobs for each loop
- Stops when stopping condition is met

HaLoop will work when the following is true:

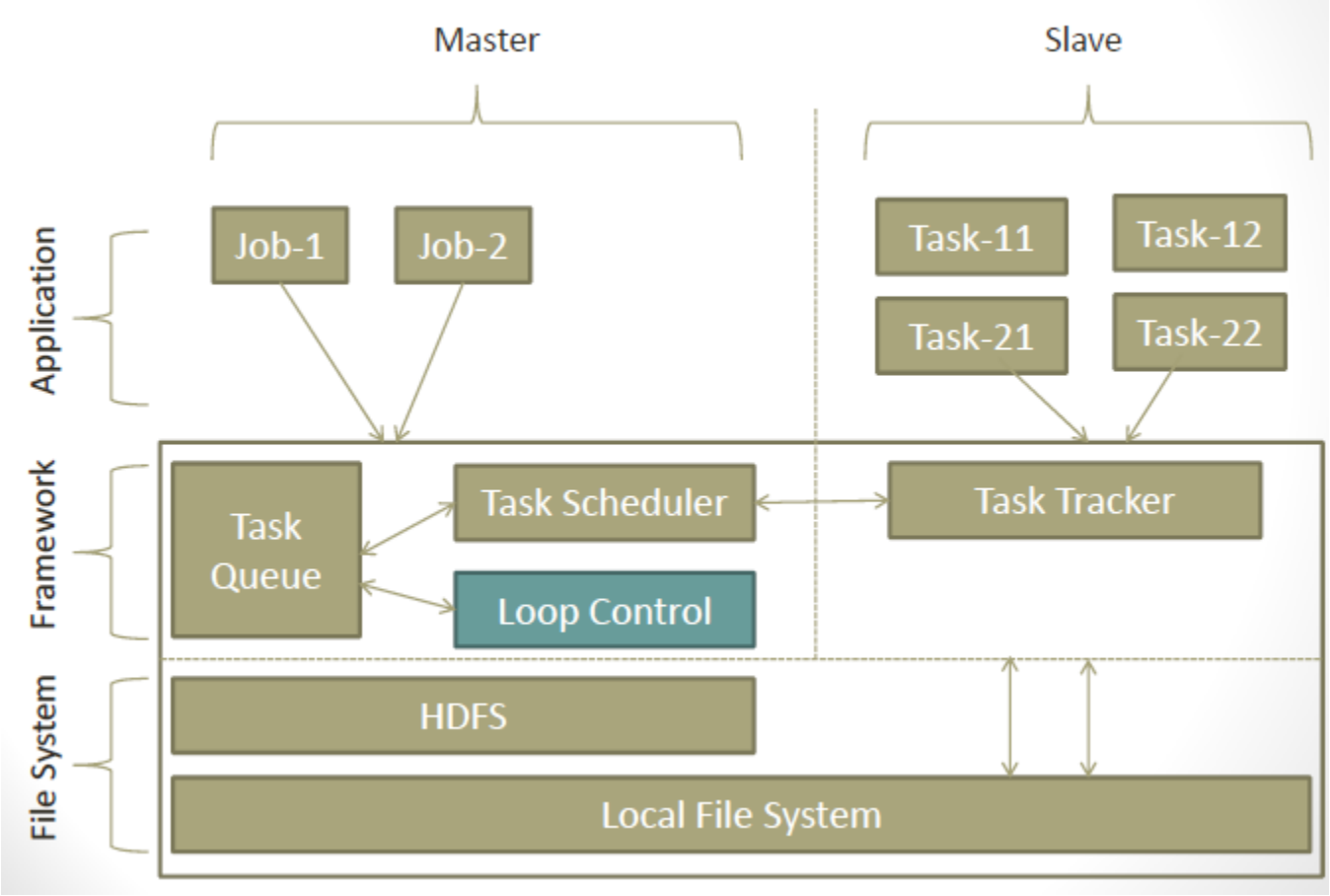
$$R_{i+1} = R_0 \cup (R_i \bowtie L)$$

- In other words, the next result is a join of the previous result ***and loop-invariant data L***.

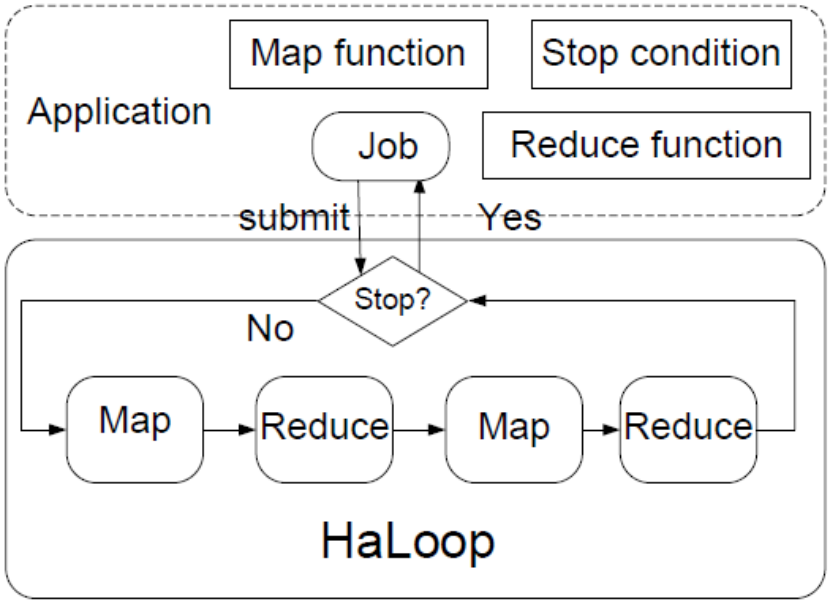
Loop Control

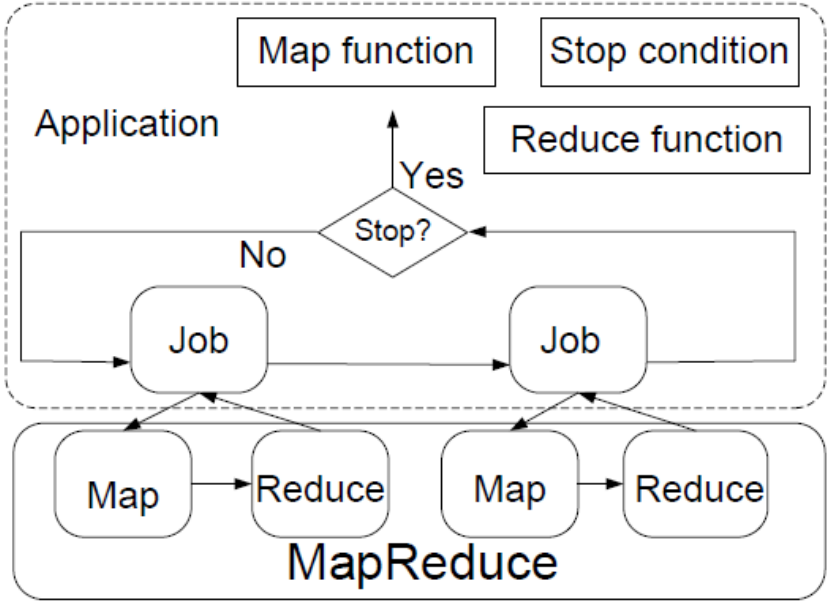


Loop Control

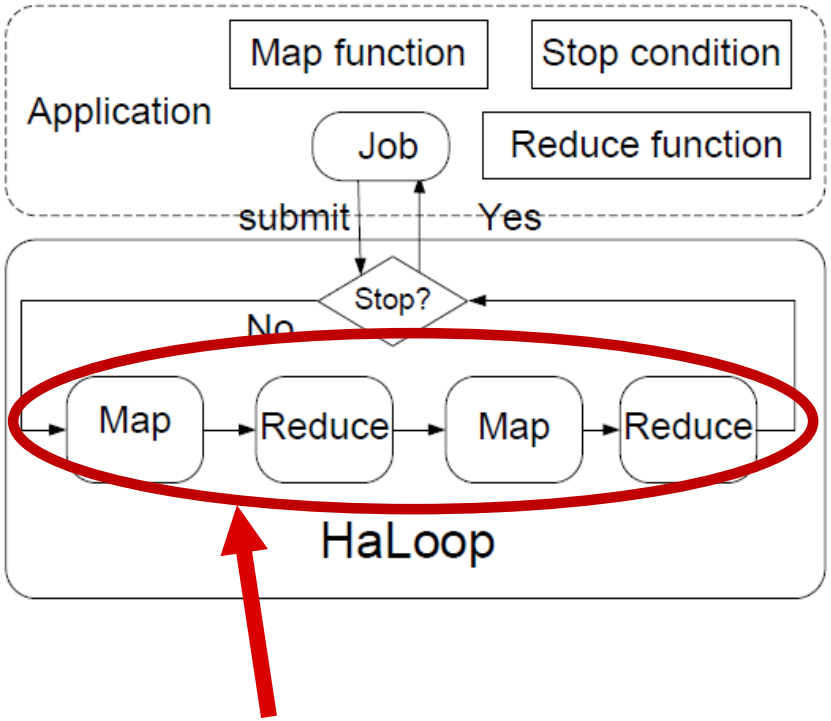


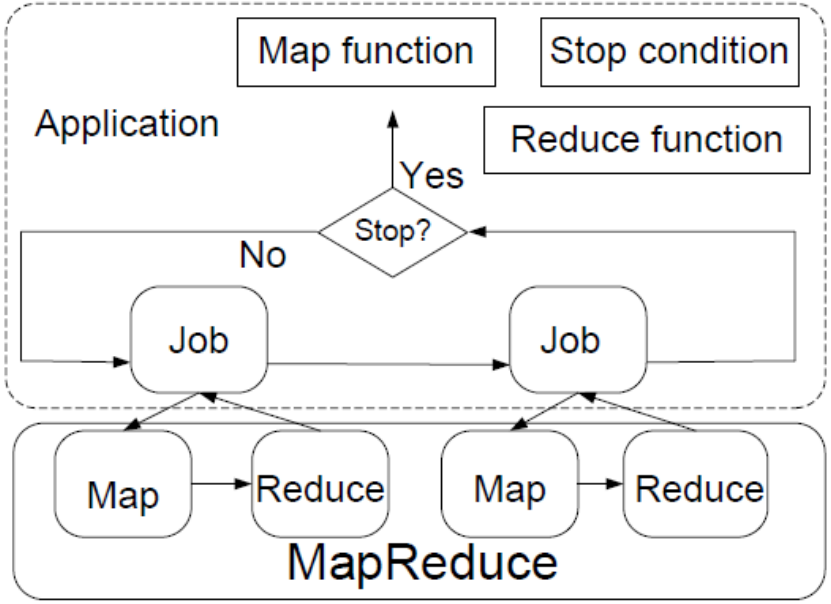
Loop Control





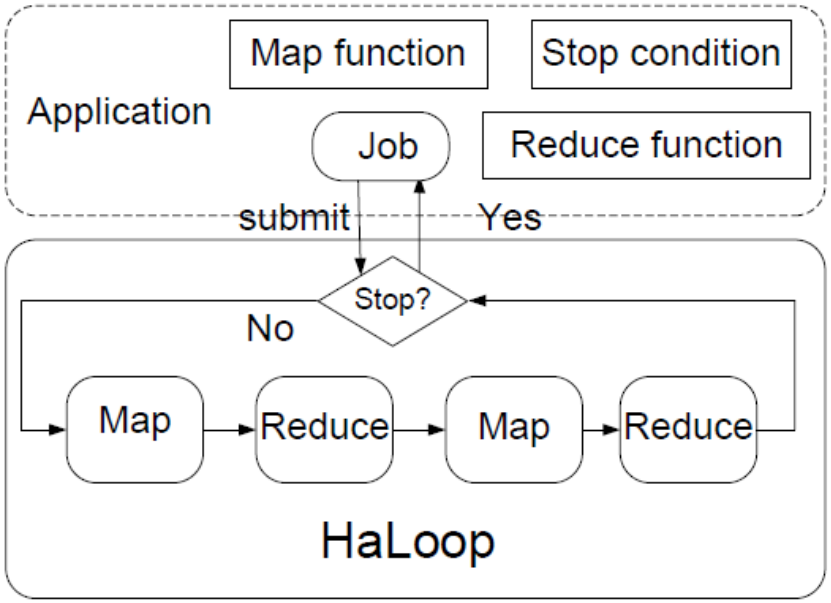
Loop Control

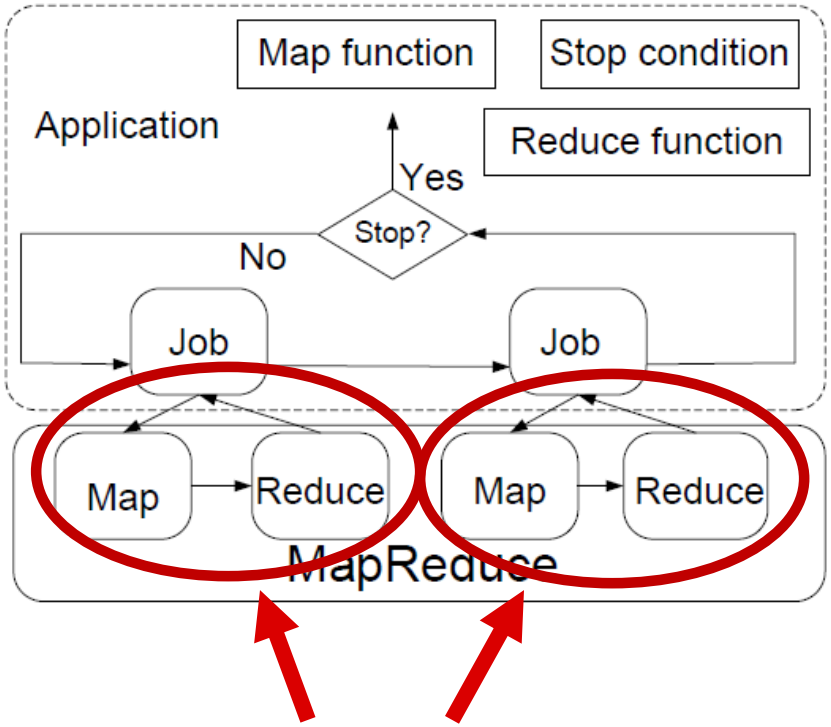




HaLoop controls the entire loop

Loop Control





Hadoop controls single job;
User must manage the loop

Inter-Iteration Locality

Goal of HaLoop's Scheduler:

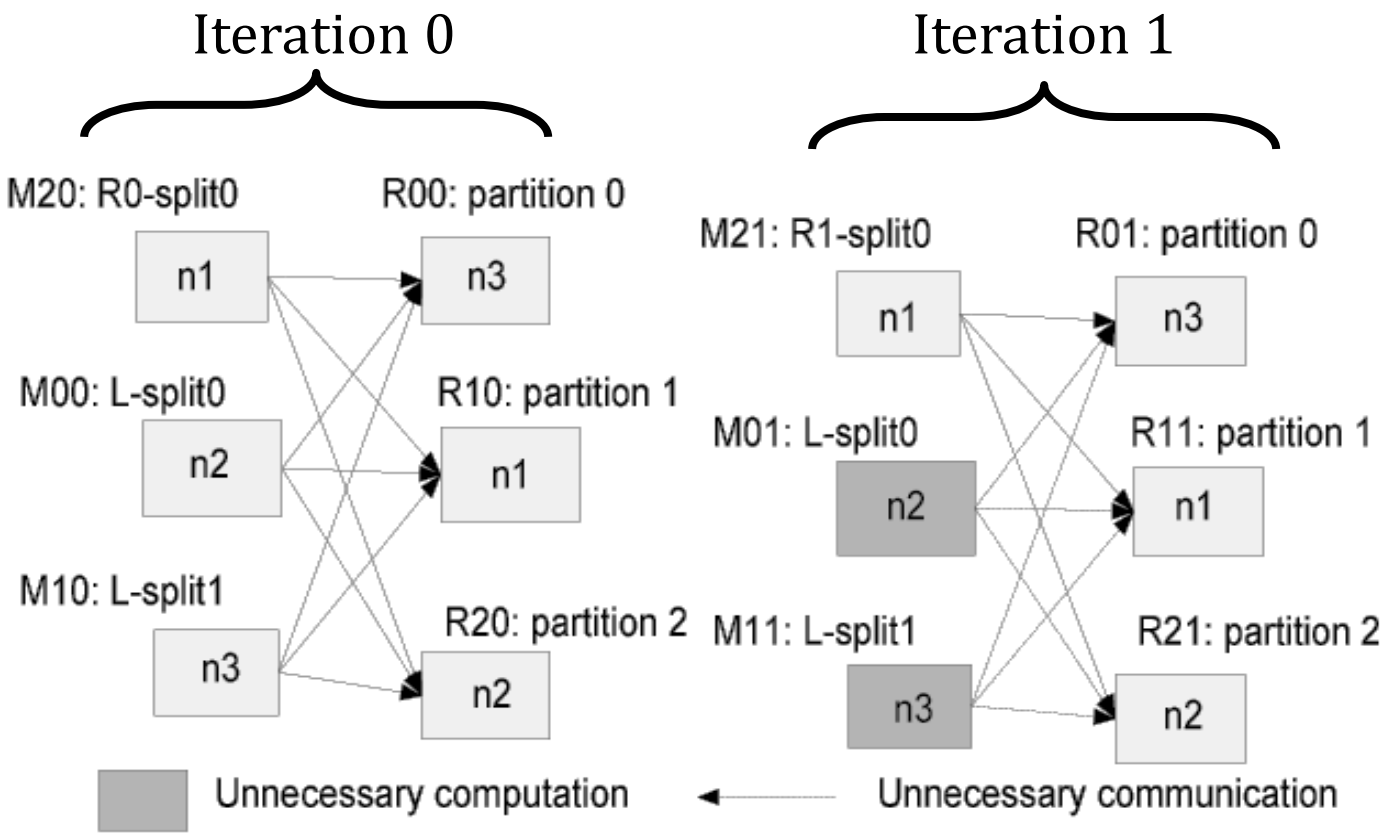
- Place, on the same physical machines, map/reduce tasks that occur in different iterations, but access the same data
- This is done by the master node
- Iteration 1 is the same as Hadoop

A schedule exhibits inter-iteration locality if…

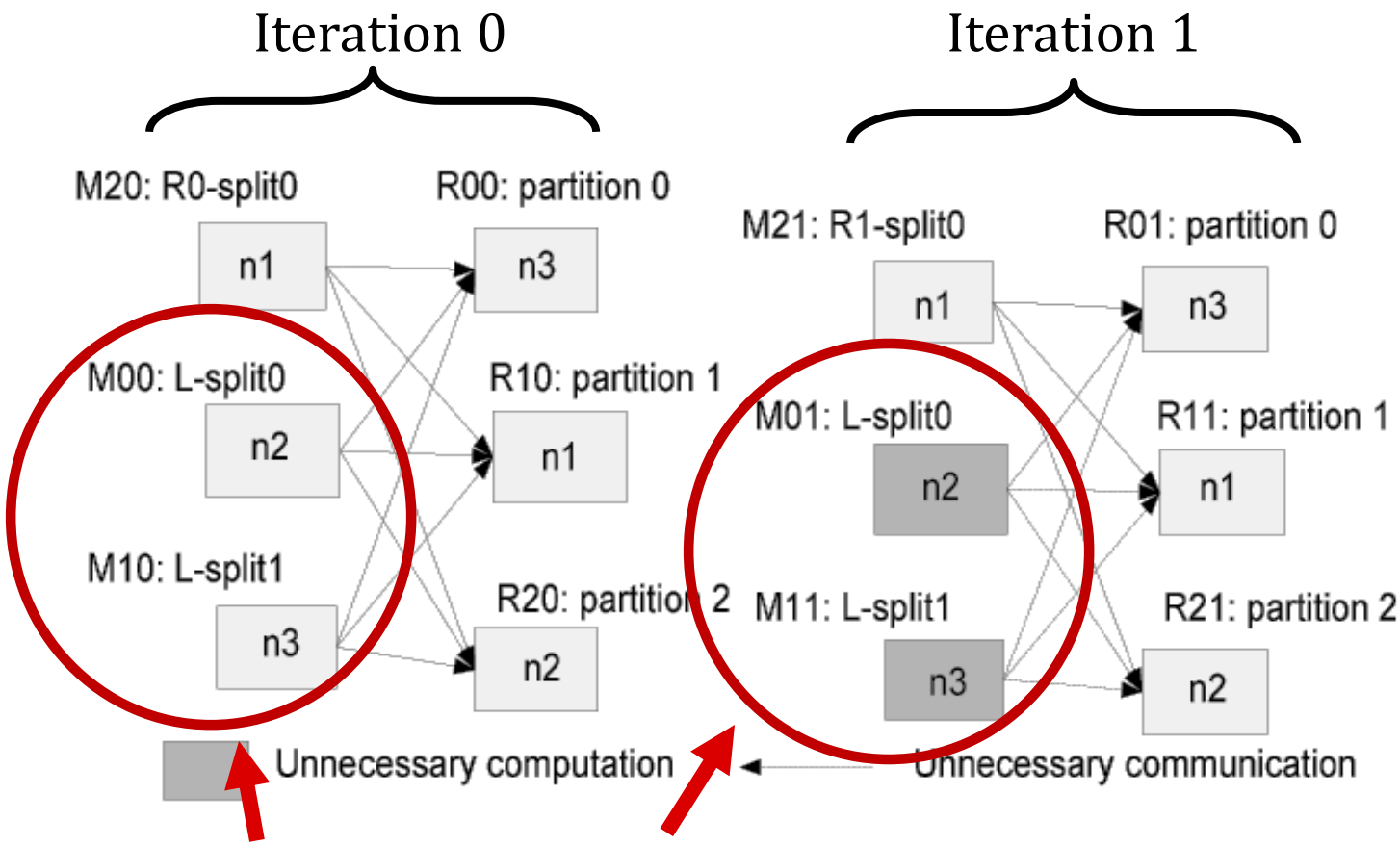
$\forall i > 1, T_d^i$ and T_d^{i-1} are assigned to the same physical node if T_d^{i-1} exists

where i is the iteration number and d is the data file

Inter-Iteration Locality

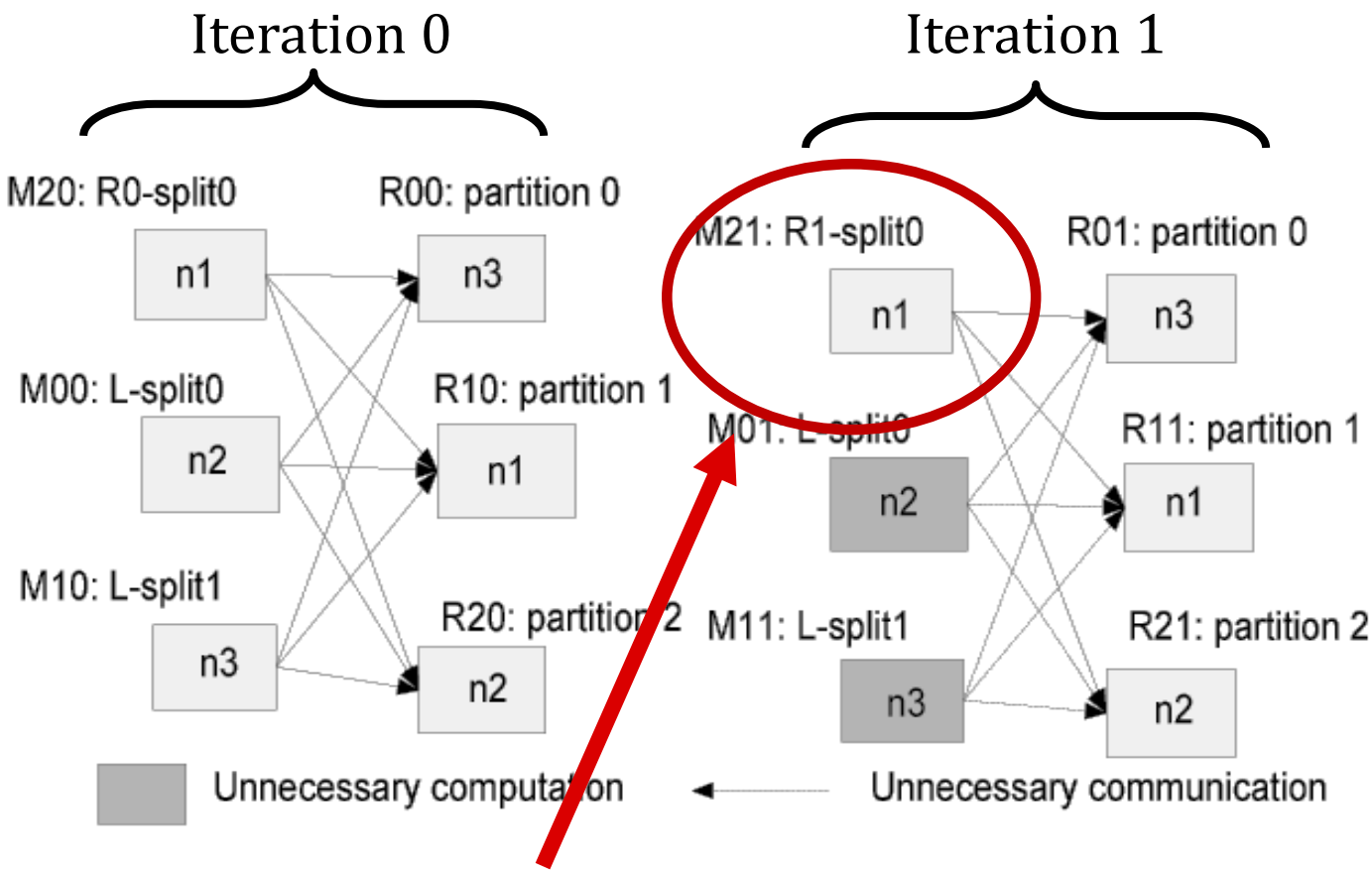


Inter-Iteration Locality



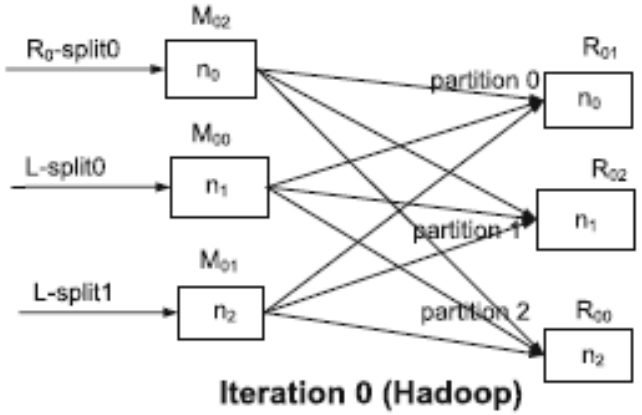
These will compute the same result!

Inter-Iteration Locality

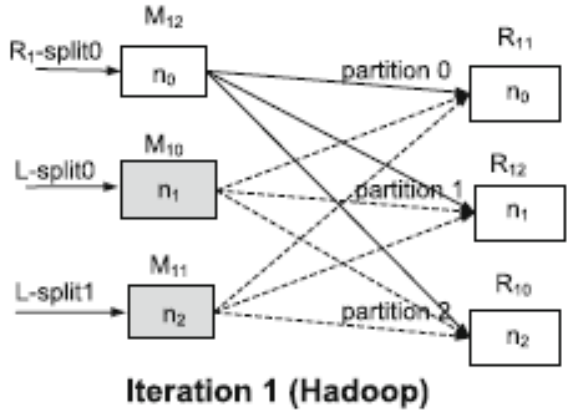


In iteration 1, only one mapper needs to be run

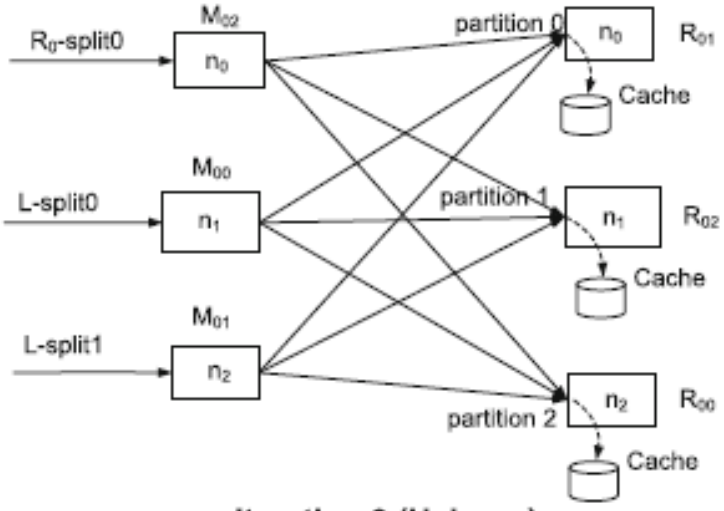
Inter-Iteration Locality



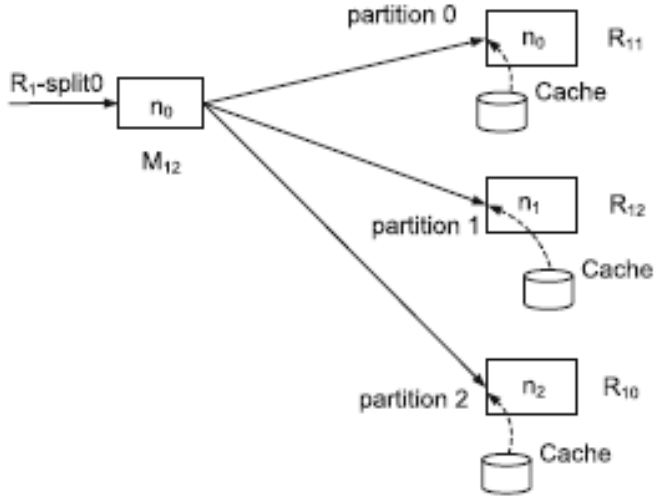
Iteration 0 (Hadoop)



Iteration 1 (Hadoop)

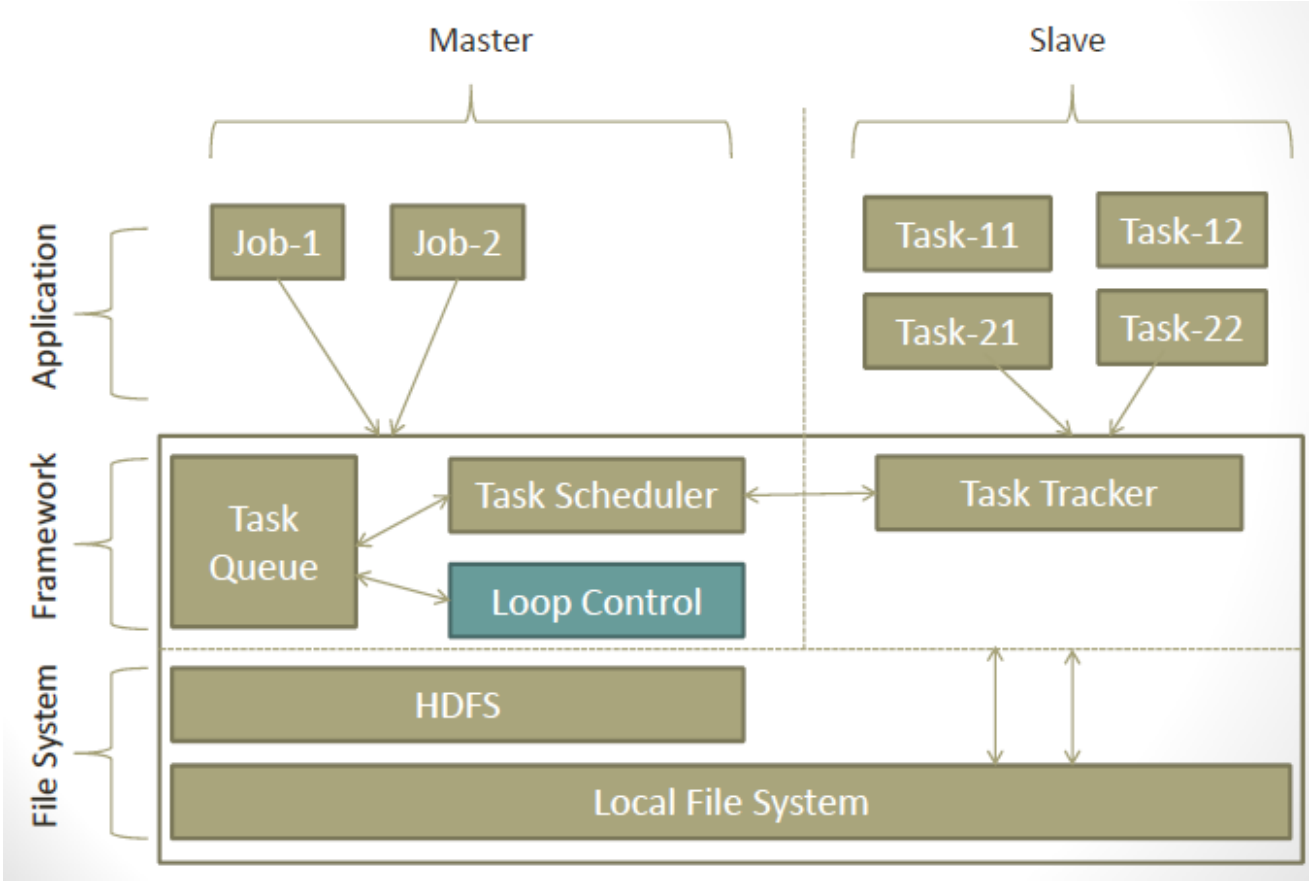


Iteration 0 (HaLoop)

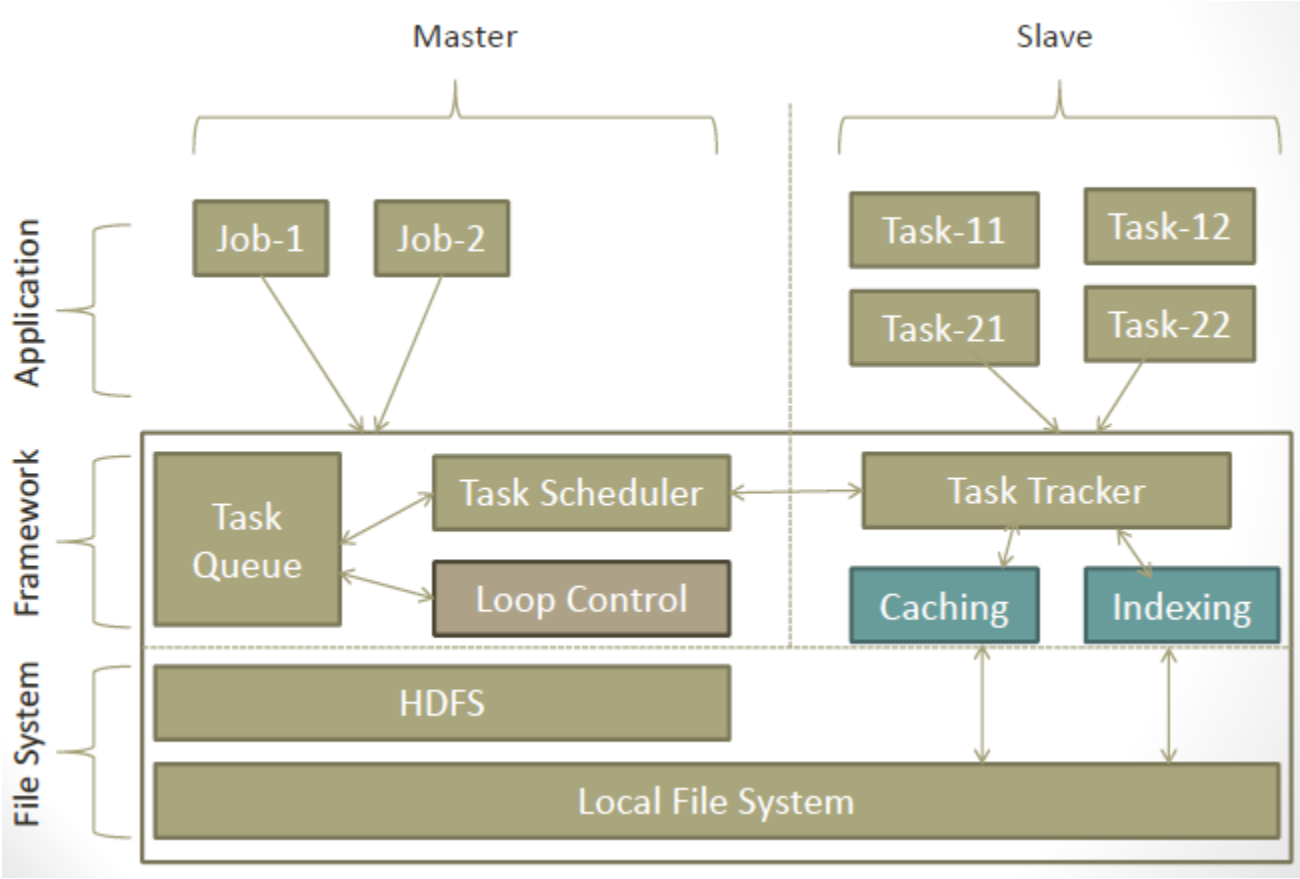


Iteration 1 (HaLoop)

Caching & Indexing

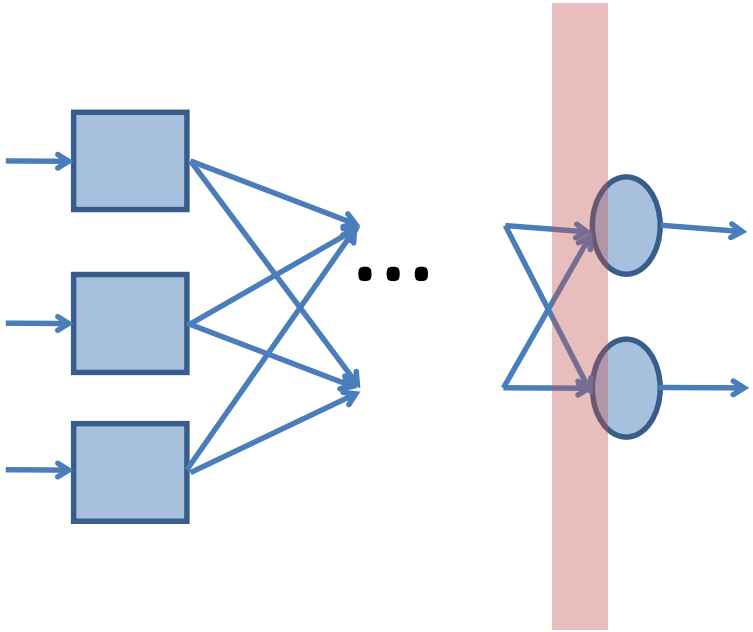


Caching & Indexing



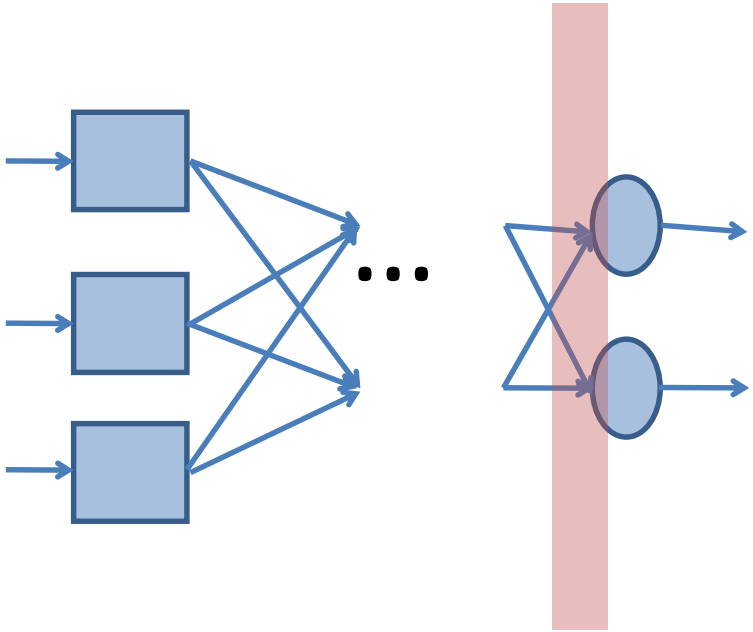
Reducer Input Cache

- Provides:
  - Access to loop invariant data without map/shuffle
- Used By:
  - Reducer function
- Assumes:
  1. Mapper output for a given table constant across iterations
  2. Static partitioning (implies: no new nodes)
- PageRank
  - Avoid shuffling the network at every step



Reducer Input Cache

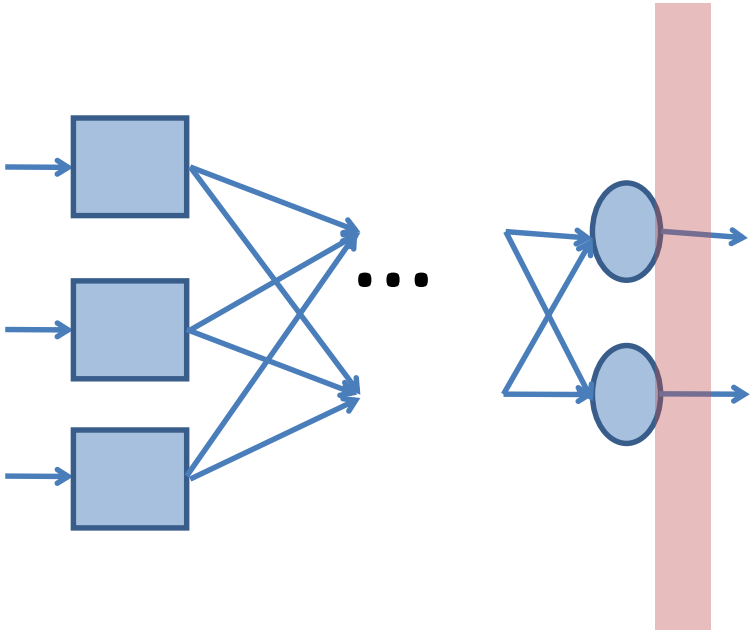
- Provides:
  - Access to loop invariant data without map/shuffle
- Used By:
  - Reducer function
- Assumes:
  1. Mapper output for a given table constant across iterations
  2. Static partitioning (implies: no new nodes)
- PageRank
  - Avoid shuffling the network at every step



Worst case scan: O(n)
where n is the size of local disk cache

Reducer Output Cache

- Provides:
  - Distributed access to output of previous iterations
- Used By:
  - Fixpoint evaluation
- Assumes:
  1. Partitioning constant across iterations
  2. Reducer output key functionally determines Reducer input key
- PageRank
  - Allows distributed fixpoint evaluation
  - Obviates extra MapReduce job

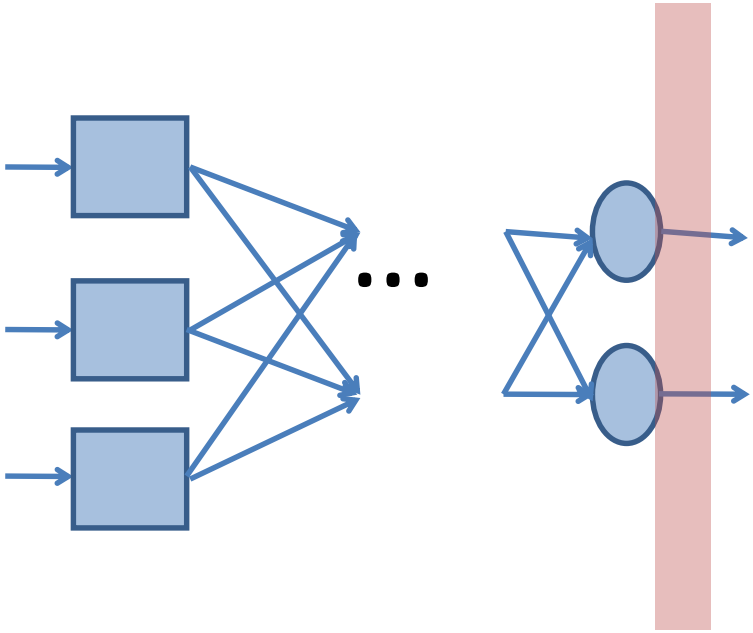


Reducer Output Cache

In the last map-reduce job of the loop, the partition function f must satisfy:

if $(k_{o1}, v_{o1}) \in reduce(k_i, V_i) \wedge$
$(k_{o2}, v_{o2}) \in reduce(k_j, V_j) \wedge$
$k_{o1} = k_{o2}$
then $f(k_i) = f(k_i)$

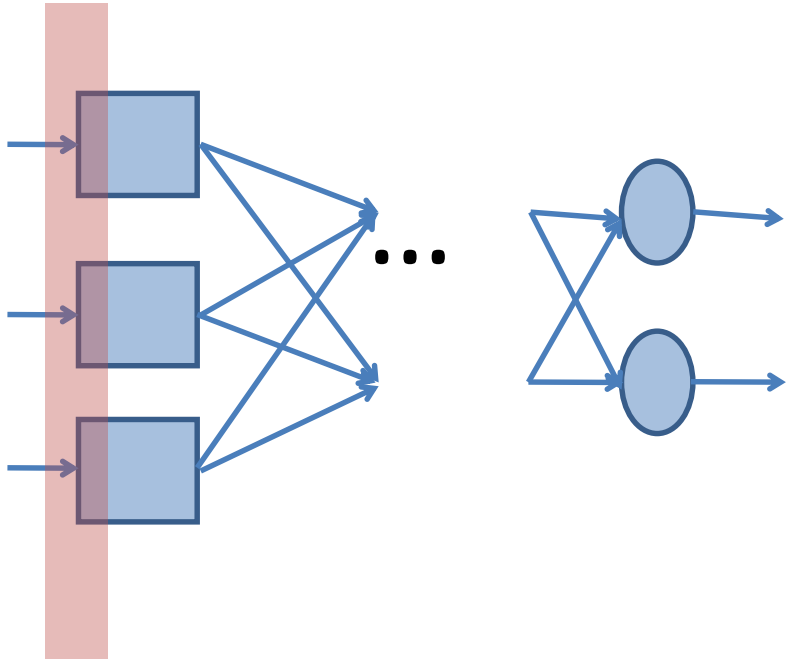
- Two different reducer input keys, but same output key, the reducer keys must be placed in same partition



Mapper Input Cache

- Provides:
  - Access to non-local mapper input on later iterations
- Used:
  - During scheduling of map tasks
- Assumes:
  1. Mapper input does not change

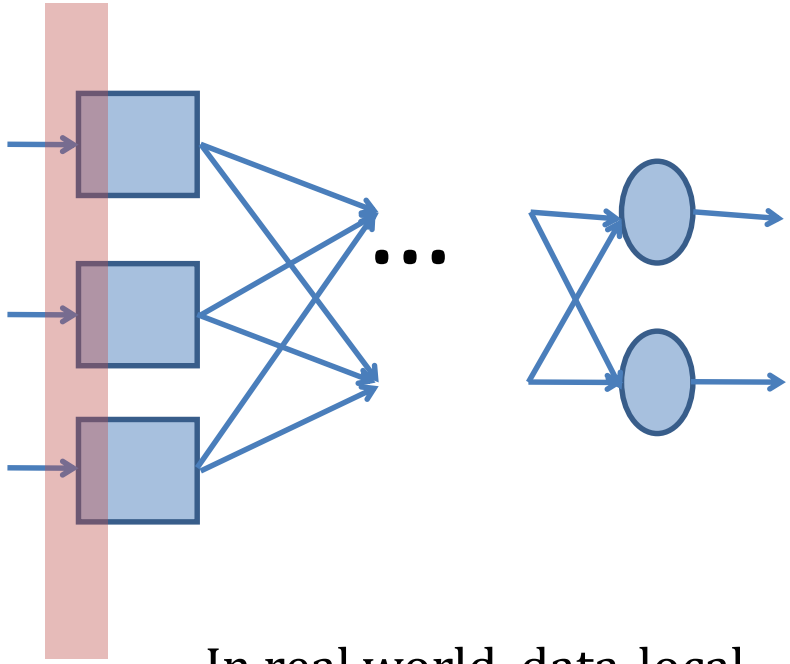
- PageRank
  - Subsumed by use of Reducer Input Cache



Mapper Input Cache

- Provides:
  - Access to non-local mapper input on later iterations
- Used:
  - During scheduling of map tasks
- Assumes:
  1. Mapper input does not change

- PageRank
  - Subsumed by use of Reducer Input Cache



In real world, data-local mapper rate is 70-95%

PageRank In HaLoop

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

PageRank In HaLoop

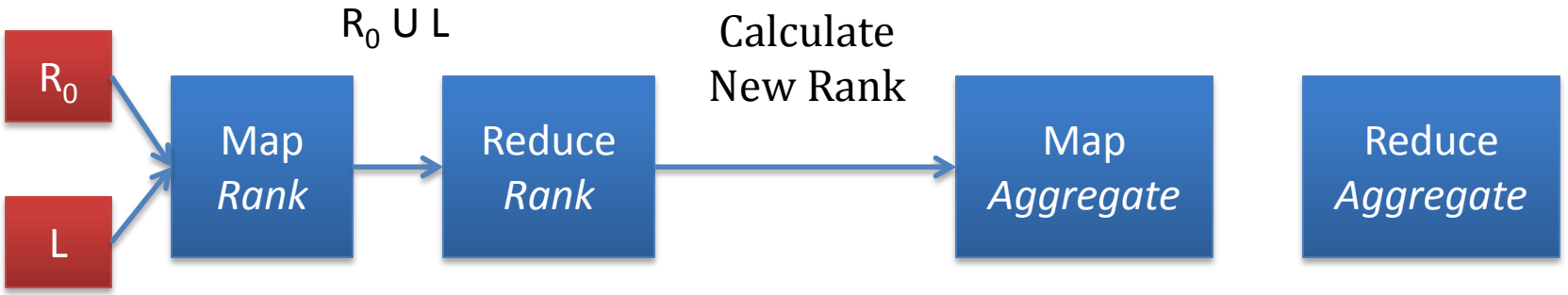
Source URL	Dest/Rank	Source File
a.com	1.0	#2
a.com	b.com,c.com, d.com	#1
b.com	1.0	#2
b.com		#1
c.com	1.0	#2
c.com	a.com, e.com	#1
d.com	1.0	#2

PageRank In HaLoop

Source URL	Dest/Rank	Source File
a.com	1.0	#2
a.com	b.com,c.com, d.com	#1
b.com	1.0	#2
b.com		#1
c.com	1.0	#2
c.com	a.com, e.com	#1
d.com	1.0	#2

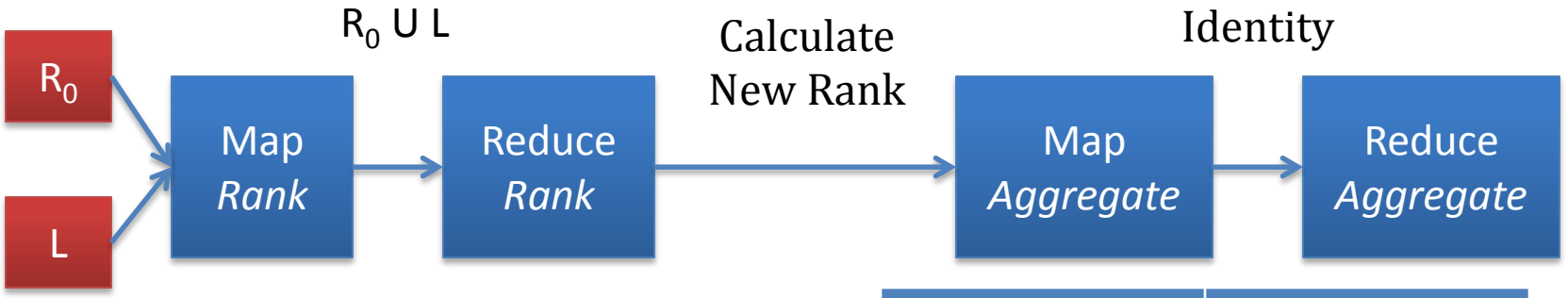
Only these values are given to reducer

PageRank In HaLoop



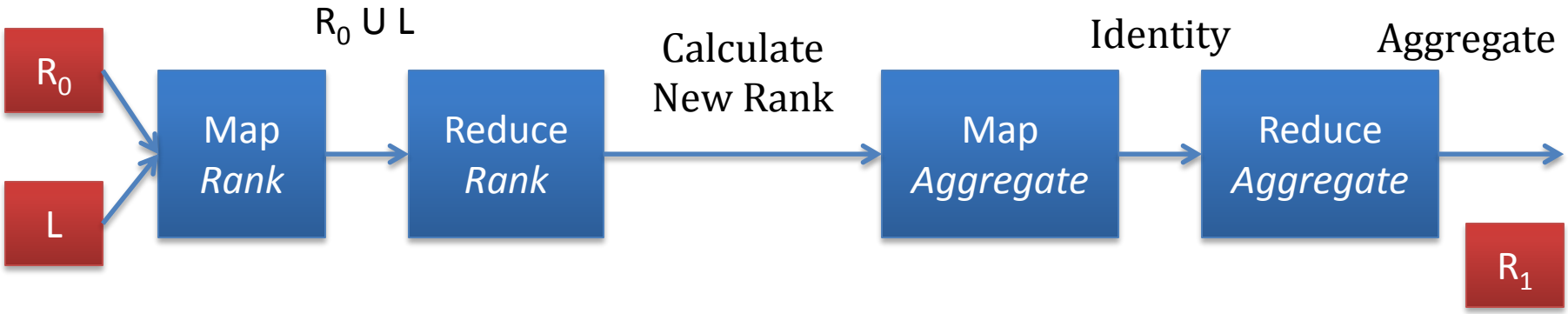
Destination	New Rank
b.com	1.5
c.com	1.5
d.com	1.5
a.com	1.5
e.com	1.5
b.com	1.5
d.com	1.5
c.com	1.5

PageRank In HaLoop



Destination	New Rank
b.com	1.5
c.com	1.5
d.com	1.5
a.com	1.5
e.com	1.5
b.com	1.5
d.com	1.5
c.com	1.5

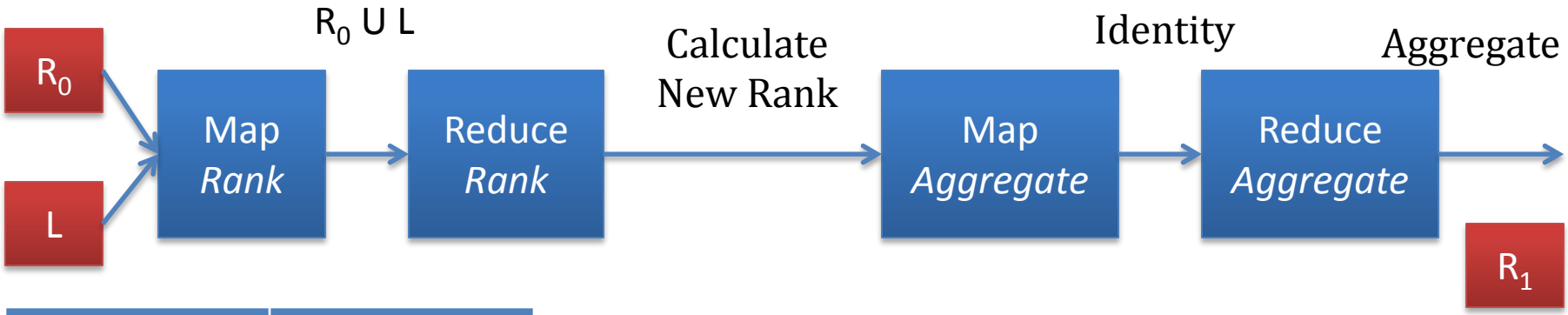
PageRank In HaLoop



$$R_{i+1} = \gamma_{urldest \rightarrow url, SUM(newrank) \rightarrow rank}$$

Destination	New Rank
a.com	1.5
b.com	3.0
c.com	3.0
d.com	3.0
e.com	1.5

PageRank In HaLoop

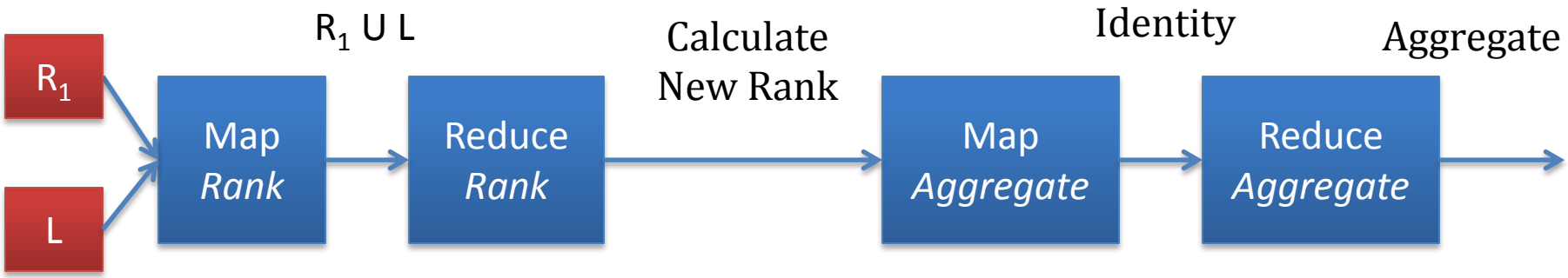


Destination	New Rank
a.com	1.0
b.com	1.0
c.com	1.0
d.com	1.0
e.com	1.0

Destination	New Rank
a.com	1.5
b.com	3.0
c.com	3.0
d.com	3.0
e.com	1.5

Compare R0 and R1. If not under threshold, repeat.

PageRank In HaLoop



Source URL	Dest/Rank	Source File
a.com	1.5	#2
a.com	b.com,c.com,d.com	#1
b.com	1.5	#2
b.com		#1
c.com	1.5	#2
c.com	a.com, e.com	#1
d.com	1.5	#2
…	…	…

HaLoop Review

- Loop Control – HaLoop manages scheduling
- Inter-Iteration Locality

- Reducer Input: invariant data
- Reducer Output: fixpoint evaluation
- Mapper Input: avoids non-local data reads

Performance

Experiments

Hardware

- Amazon EC2 small instances
- 50 and 90 slave nodes
  - +1 master node
- 1.7 GB memory
- 160 GB disk
- 32 bit OS

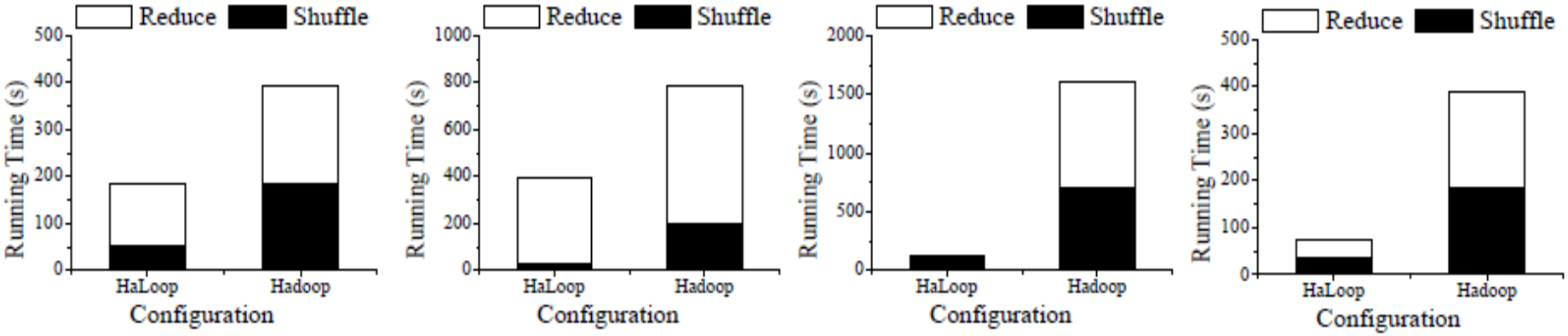
Name	Nodes	Edges	size
Livejournal	4,847,571	68, 993,773	18GB
Triples	1,464,829,200	1,649,506,981	120GB
Freebase	7,024,741	154,544,312	12GB

Dataset / Tests

- PageRank and descendant query tests
- HaLoop vs. Hadoop
- Livejournal
  - 18GB, social network data
- Triples
  - 120GB, semantic web data
- Freebase
  - 12GB, concept linkage graph

Reducer Input Cache

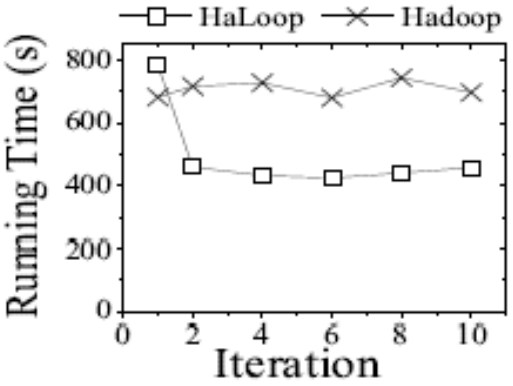
Cost Distribution



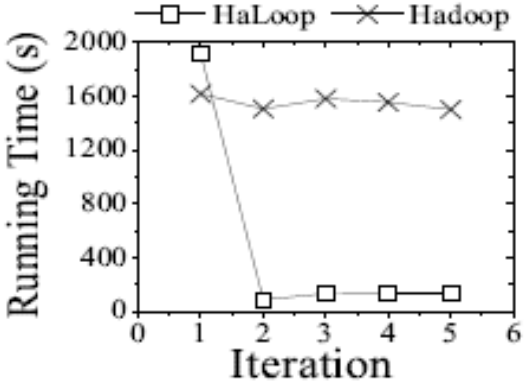
- HaLoop dramatically reduces shuffle time
- Outperforms Hadoop in both shuffle and reduce phases

Reducer Input Cache

Join Step



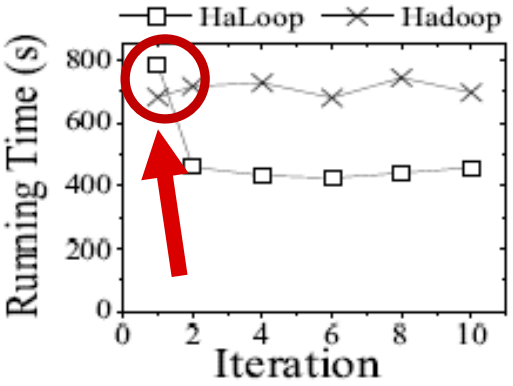
PageRank
Freebase Dataset
90 nodes



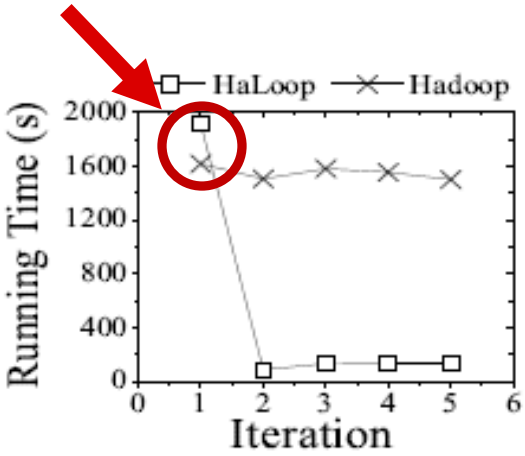
Descendant Query
Triples Dataset
90 nodes

Reducer Input Cache

Join Step



PageRank
Freebase Dataset
90 nodes

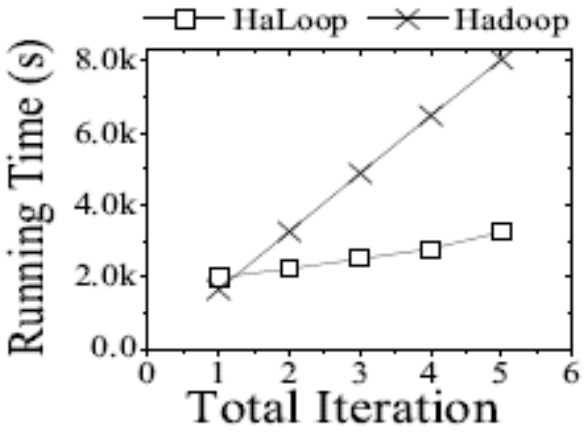


Descendant Query
Triples Dataset
90 nodes

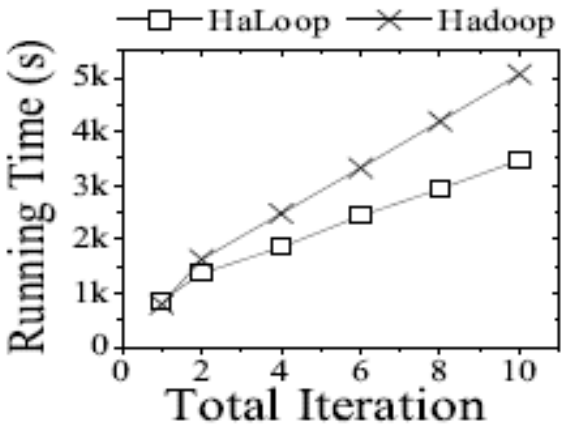
- Caches data on each reducer's local disk
- Creates an index for this cache
- Writes everything to disk

Reducer Input Cache

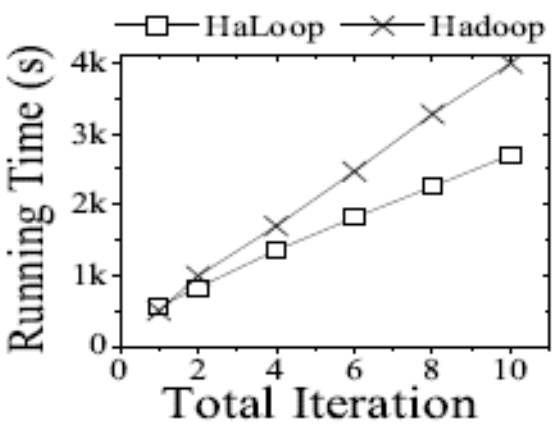
Overall Performance



PageRank
Freebase Dataset
90 nodes



Descendant Query
Triples Dataset
90 nodes

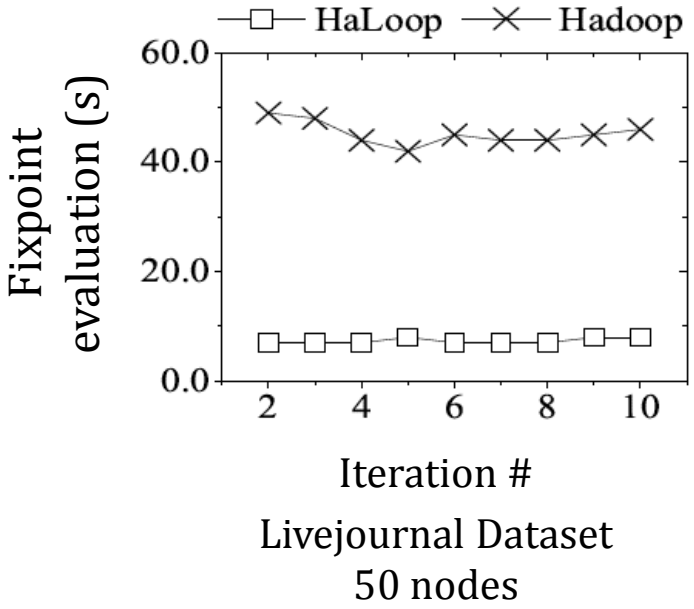


PageRank
LiveJournal Dataset
50 nodes

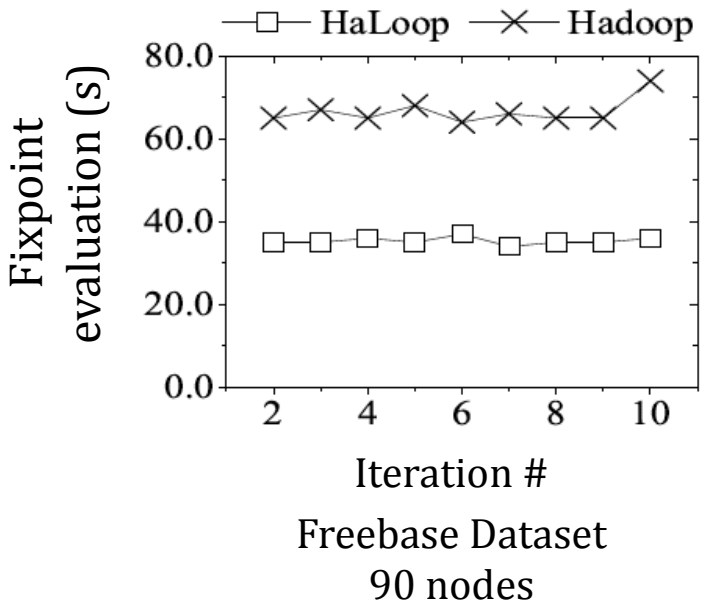
HaLoop lowers runtime by 1.85x on average (10-iteration jobs)

Reducer Output Cache

Fixpoint Evaluation



Livejournal Dataset
50 nodes



Freebase Dataset
90 nodes

- Hadoop performs an extra MapReduce job
- HaLoop looks at the reducer output cache → 40% savings

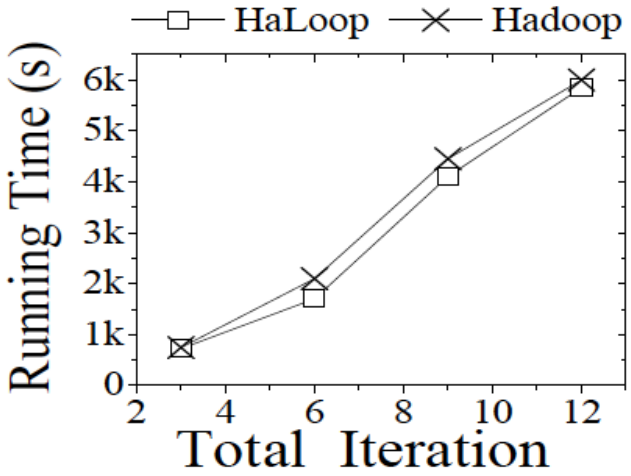
Mapper Input Cache

- PageRank and Descendant Query have dynamic inputs

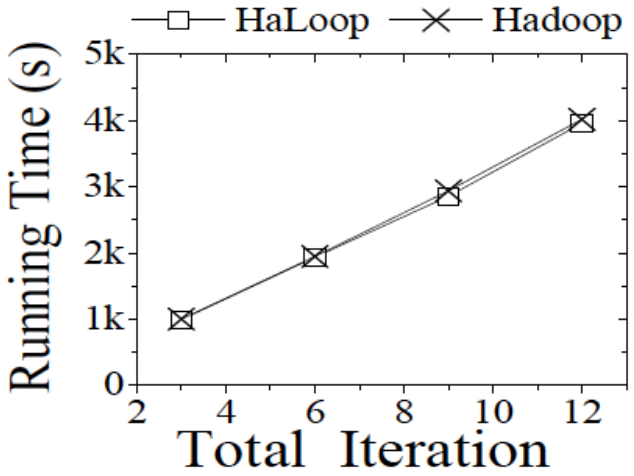
- *k*-means clustering algorithm
- Astronomy, Multi-dimensional tuple datasets
  - 46 GB cosmo-dark
  - 54 GB cosmo-gas
- 8 nodes

Mapper Input Cache

Running Time



Cosmo-Dark
8 nodes



Cosmo-Gas
8 nodes

- Only 5% non-local data reads → minimal improvement

Related Work

Mahout

- http://mahout.apache.org/
- Scalable machine learning libraries on Hadoop
- Model fitting applications – iterative
- Includes a loop control driver program

Twister

- http://www.iterativemapreduce.org/
- Supports iterative programs
- Mappers and reducers have distributed memory caches
  - Avoid repeating mapper data loads
- Sensitive to failures
- Commodity clusters do not have large memory

HaLoop (re-visited)

The HaLoop Approach to Large-Scale Iterative Data Analysis.
Yingyi Bu, Bill Howe, Magdalenda Balazinska, Michael D. Ernst.
The VLDB Journal (VLDBJ), Volume 21, Number 2, April 2012.

Node Failure and Recovery

- Task failure (programming errors, data format, etc.)
- Slave node failure
- Master node failure

Disk Cache vs. Memory Cache

- Use disk or memory? Answer: Disk

Conclusion

Summary

HaLoop

1. Loop-aware task scheduling
2. Loop-invariant data caching
3. Caching for efficient fixpoint verification

- Future work: Datalog engine

Overall Paper Rating

Grade: **A-**

- Significant performance improvements for iterative jobs
- Thorough explanation of system

<u>Suggestions</u>

- Needs better explanation of caching with node failures

Questions?

HaLoop

Efficient Iterative Data Processing on Large Clusters

Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst
University of Washington
Department of Computer Science & Engineering

Presented by Albert Haque
University of Texas at Austin
April 15, 2013