HaLoop

Efficient Itarative Data Processing On Large Clusters (2010)

Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Erns
Department of Computer Science and Engineering University of Washington, Seattle, WA, U.S.A.

2019.10.18

2016104115 김원규

2016104163 정의동

Index

01 Introduction

02 Overview

03 Scheduler

04 Cache & Index

05 Experiment

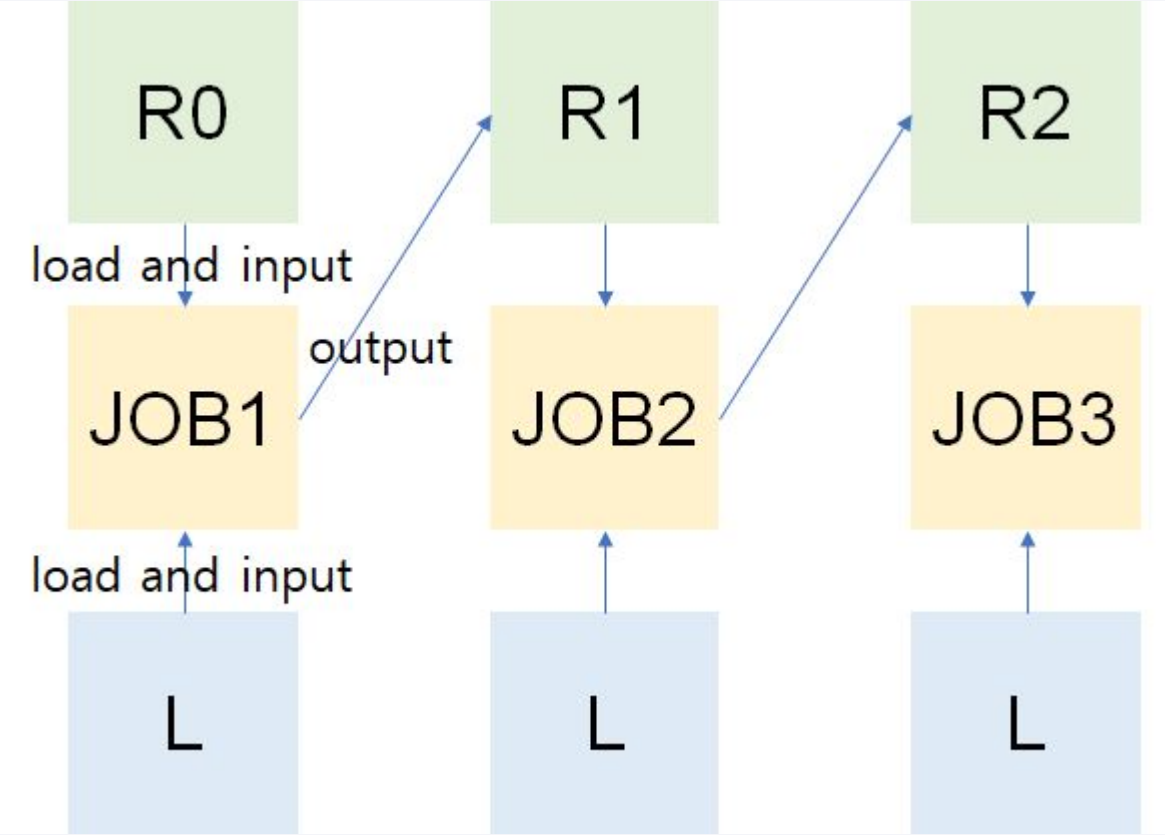
06 Overall

Introduction

A strong platform Haloop for iterative programs

01

Introduction



iterative program

1. using invariant table for every iterations(L)
2. re-using output for next iteration(R1, R2)

01

PageRank

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

(a) Initial Rank Table R_0

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

(b) Linkage Table L

$$MR_1 \begin{cases} T_1 = R_i \bowtie_{url=url\_source} L \\ T_2 = \gamma_{url, rank, \frac{rank}{\texttt{COUNT}(url\_dest)} \rightarrow new\_rank}(T_1) \\ T_3 = T_2 \bowtie_{url=url\_source} L \end{cases}$$

$$MR_2 \left\{ R_{i+1} = \gamma_{url\_dest \rightarrow url, \texttt{SUM}(new\_rank) \rightarrow rank}(T_3) \right.$$

(c) Loop Body

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

(d) Rank Table R_3

Figure 1: PageRank example

Work flow

1. every url has rank and other url links(R0, L)
2. compute weights which url_dest will gain using initial table R0, invariant table L
3. compute Ri+1 table using map-reduce pairs(loop body) , Ri(previous iteration table) and L

01

Descendant Query

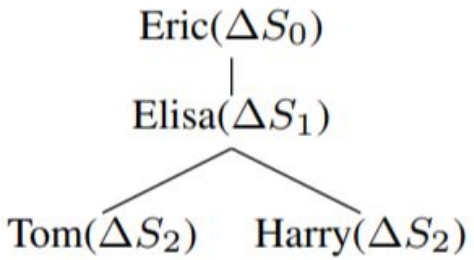
name1	name2
Tom	Bob
Tom	Alice
Elisa	Tom
Elisa	Harry
Sherry	Todd
Eric	Elisa
Todd	John
Robin	Edward

(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

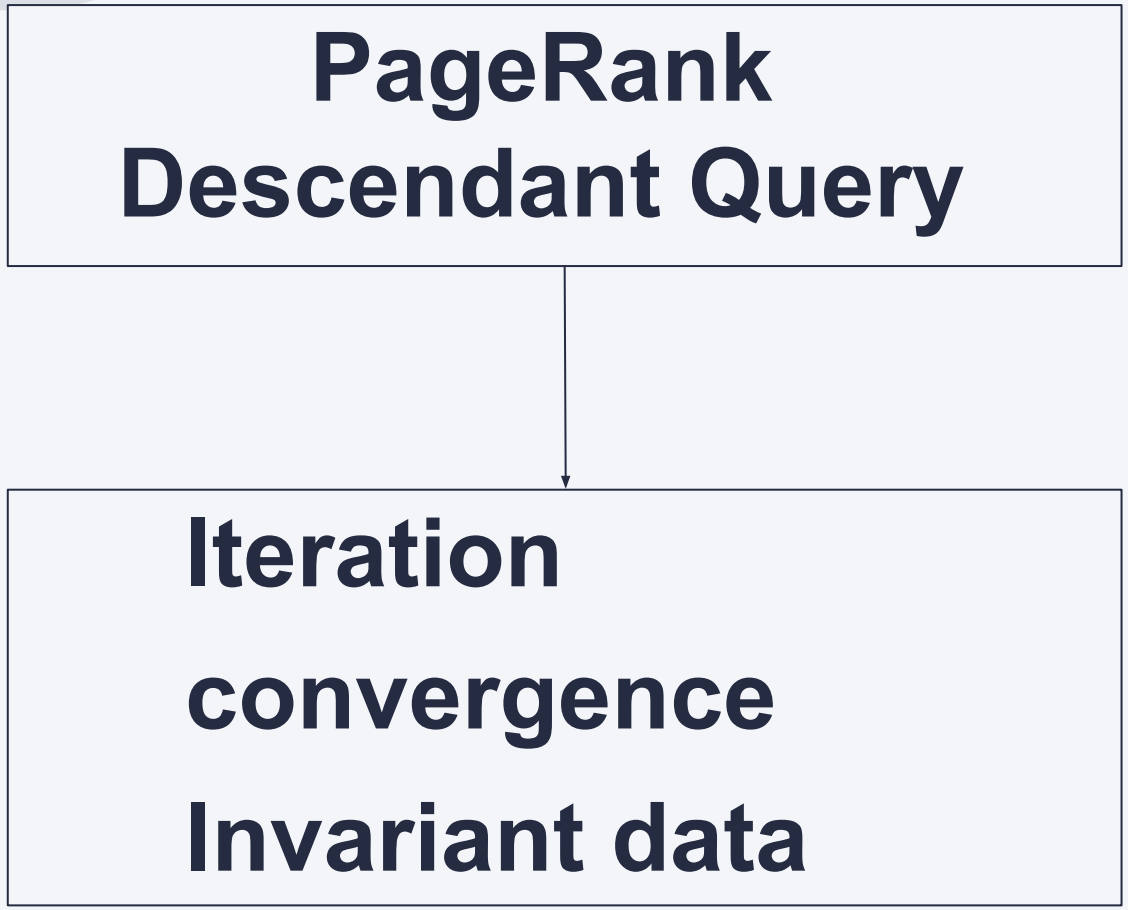
Figure 2: Descendant query example

Work flow

1. table F has relationships about person to person.
2. compute friend's relationship for every iteration, (S0 = (Eric, Eric))
3. compute Si+1 table using map-reduce pairs(loop body), Si(0<=i<iteration) and F
4. when computing Si+1,
   remove duplicates previous relationships
   [Si(0<=i<iteration)] from new relationships

01

Haloop



Haloop

1. don’t need to re-load iterative data.
   (save re-loading cost)
2. don’t need to proceed additional application anymore.
   (early stop)

Overview

Haloop Overview

02

Haloop Architecture

How Haloop works

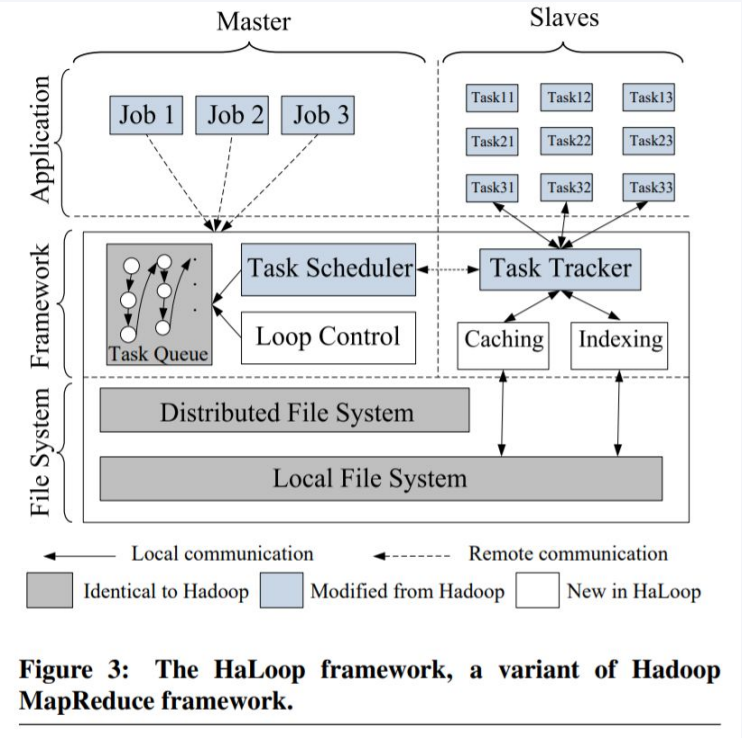


Figure 3: The HaLoop framework, a variant of Hadoop MapReduce framework.

1. offer programming interface to make a program
2. master node has a ‘ **loop control** ’ which composes map-reduce pairs(loop body) until meeting fixpoint.
3. new task scheduler leverages data locality. let reducers get tasks which they handled during before iteration.
4. Haloop cahces and indexes invariant data, output result and data from other nodes.

- **‘fixpoint’** means the state which satisfys terminating conditions.
  ex) current iteration’s result - previous iteration’s result < threshold(ϵ)
  ex) haloop reaches the maximum number of iterations

02

Haloop Architecture

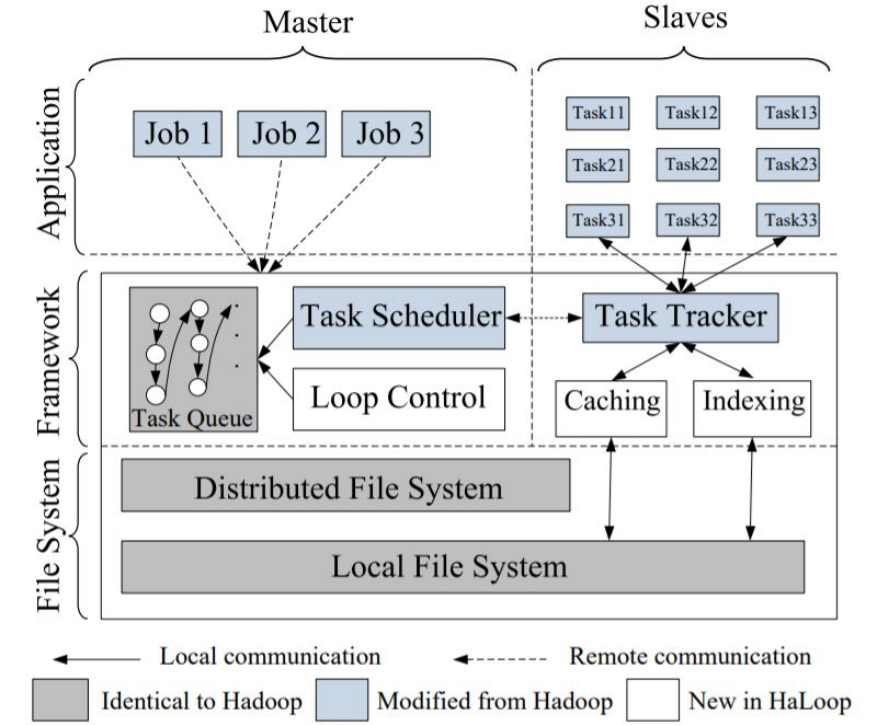
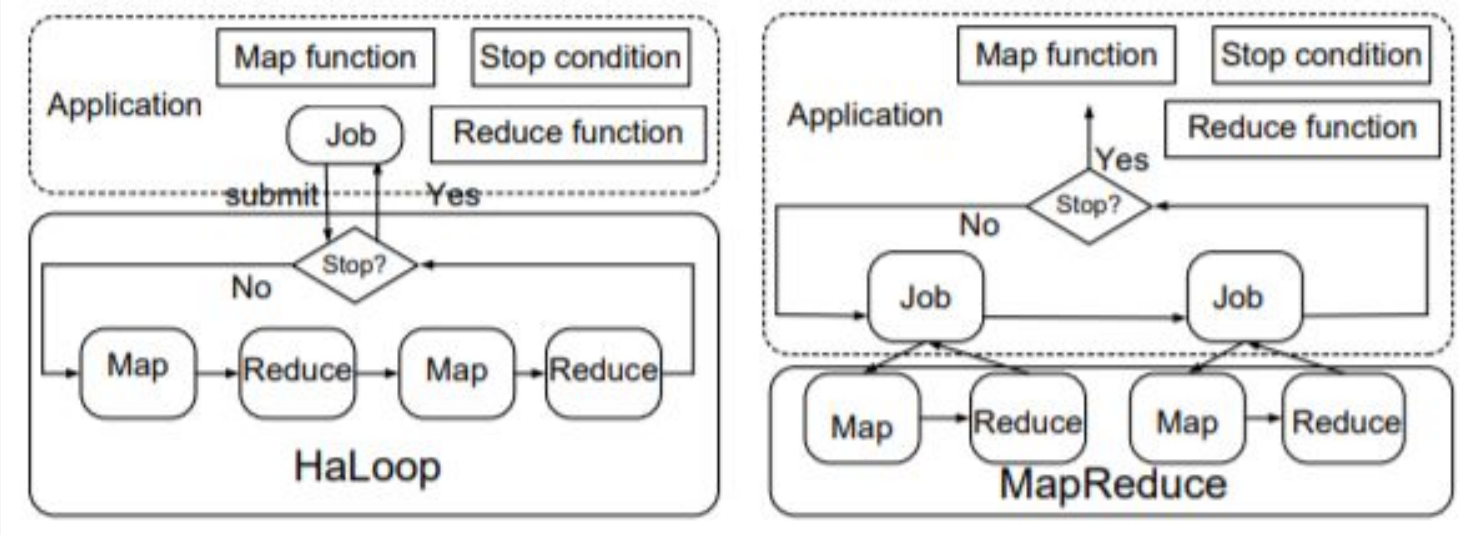


Figure 3: The HaLoop framework, a variant of Hadoop MapReduce framework.

<haloop architecture>



workflow: <haloop vs hadoop>

02 API and Programming

```
Map_Rank
Input: Key k, Value v, int iteration
 1: if v from L then
 2:     Output(v.url_src, v.url_dest, #1);
 3: else
 4:     Output(v.url, v.rank, #2);
 5: end if

Reduce_Rank
Input: Key key, Set values, Set invariantValues,
int iteration
 1: for url_dest in invariantValues do
 2:     Output(url_dest, values.get(0)/invariantValues.size());
 3: end for
```

→ Loopbody (MR1)

```
Map_Aggregate
Input: Key k, Value v, int iteration
 1: Output(v.url, v.rank);

Reduce_Aggregate
Input: Key key, Set values, int iteration
 1: Output(key, AggregateRank(values));
```

→ Loopbody (MR2)

```
ResultDistance
Input: Key out_key, Set v_{i-1}, Set v_i
 1: return |v_i.get(0) − v_{i-1}.get(0)|;
```

→ Difference between current and previous iteration's output

```
IterationInput
Input: int iteration
 1: if iteration==1 then
 2:     return L ∪ R_0;
 3: else
 4:     return R_{iteration-1}
 5: end if
```

→ Setting input for each iteration

```
Main
 1: Job job = new Job();
 2: job.AddMap(Map_Rank, 1);
 3: job.AddReduce(Reduce_Rank, 1);
 4: job.AddMap(Map_Aggregate, 2);
 5: job.AddReduce(Reduce_Aggregate, 2);
 6: job.SetDistanceMeasure(ResultDistance);
 7: job.AddInvariantTable(#1);
 8: job.SetInput(IterationInput);
 9: job.SetFixedPointThreshold(0.1);
10: job.SetMaxNumOfIterations(10);
11: job.SetReducerInputCache(true);
12: job.SetReducerOutputCache(true);
13: job.Submit();
```

→ Main function

Name	Functionality
AddMap & AddReduce	specify a step in loop
SetDistanceMeasure	specify a distance for results
SetInput	specify inputs to iterations
AddInvariantTable	specify loop-invariant data
SetFixedPointThreshold	a loop termination condition
SetMaxNumOfIterations	specify the max iterations
SetReducerInputCache	enable/disable reducer input caches
SetReducerOutputCache	enable/disable reducer output caches
SetMapperInputCache	enable/disable mapper input caches

Figure 16: HaLoop API

02

PageRank

1. MR1 - mapper

url_source	rank	table_id
www.a.com	1	#2
www.a.com	1	#2
www.c.com	1	#2
www.e.com	1	#2
www.d.com	1	#2

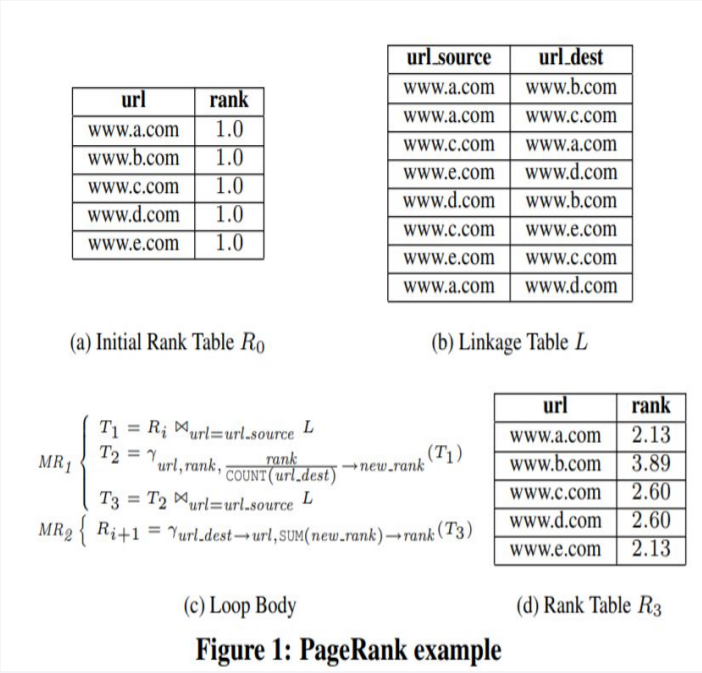
<from R0>

url_source	url_dest	table_id
www.a.com	www.b.com	#1
www.a.com	www.c.com	#1
www.c.com	www.a.com	#1
www.e.com	www.d.com	#1
www.d.com	www.b.com	#1
www.c.com	www.e.com	#1
www.e.com	www.c.com	#1
www.a.com	www.d.com	#1

<from L>

2. MR1 - Reducer

url_dest	weight
www.b.com	1/3
www.c.com	1/3
www.a.com	1/2
www.d.com	1/2
www.b.com	1
www.e.com	1/2
www.c.com	1/2
www.d.com	1/3



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

(a) Initial Rank Table R_0

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

(b) Linkage Table L

$$MR_1 \begin{cases} T_1 = R_i \bowtie_{url=url\_source} L \\ T_2 = \gamma_{url,rank,\frac{rank}{\text{COUNT}(url\_dest)} \rightarrow new\_rank}(T_1) \\ T_3 = T_2 \bowtie_{url=url\_source} L \end{cases}$$

$$MR_2 \left\{ R_{i+1} = \gamma_{url\_dest \rightarrow url,\text{SUM}(new\_rank) \rightarrow rank}(T_3) \right.$$

(c) Loop Body

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

(d) Rank Table R_3

Figure 1: PageRank example

02

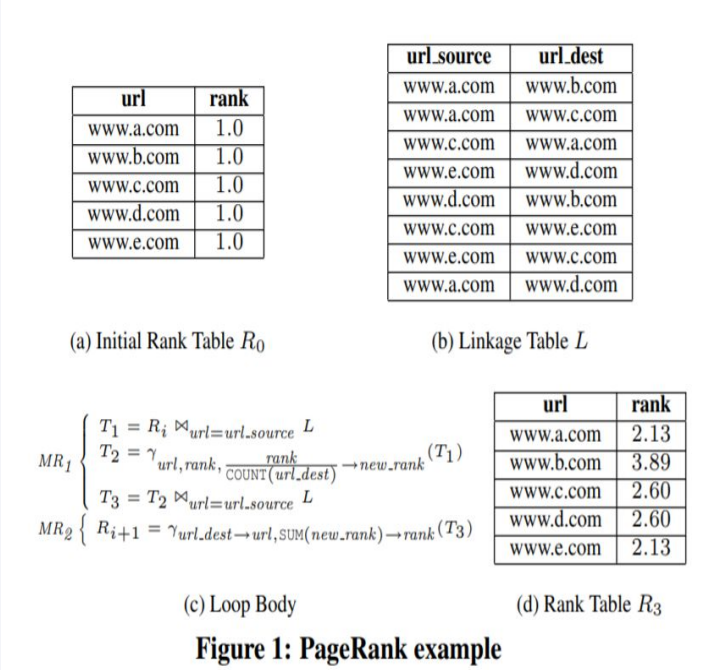
PageRank

3. MR2 - mapper

url(key)	rank(value)
www.b.com	1/3
www.c.com	1/3
www.a.com	1/2
www.d.com	1/2
www.b.com	1
www.e.com	1/2
www.c.com	1/2
www.d.com	1/3

4. MR2 - Reducer

url(key)	rank(value)
www.b.com	4/3
www.c.com	5/6
www.a.com	1/2
www.d.com	5/6
www.e.com	1/2



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

(a) Initial Rank Table R_0

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

(b) Linkage Table L

$$MR_1 \begin{cases} T_1 = R_i \bowtie_{url=url\_source} L \\ T_2 = \gamma_{url,rank,\frac{rank}{\mathrm{COUNT}(url\_dest)} \to new\_rank}(T_1) \\ T_3 = T_2 \bowtie_{url=url\_source} L \end{cases}$$

$$MR_2 \left\{ R_{i+1} = \gamma_{url\_dest \to url, \mathrm{SUM}(new\_rank) \to rank}(T_3) \right.$$

(c) Loop Body

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

(d) Rank Table R_3

Figure 1: PageRank example

02

API and Programming

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

$$R_{i+1} = R_i + \varepsilon$$

$$R_{i+1} - R_i = \varepsilon$$

General Formulation

1. Ri+1 is from computation using Ri, R0 and L
2. That means Ri+1 is concatnating Ri and added things from this iteration.
3. Then, we can get a difference between Ri+1 and Ri. If it is below threshold, the program terminates.

Scheduler

Haloop’s scheduler do loop awaring & task allocation

03

Scheduler's Goal

2019년 7월 - 포토샵 일러스트 경험자우대 취업, 일자리, 채용 ...
https://kr.indeed.com › 포토샵-일러스트-경험자우대직-취업
우대사항 해당직무 근무**경험** 컴퓨터활용능력 우수 엑셀 고급능력자 포토샵 능숙자 일러스트 능숙자 여자, 20세 ~ 27세 단체의류 인쇄시안 디자이너 / 일러스트, ...

위드시스템 정규직 기획 및 마케팅 (해당업무경험자) 우대 채용 ...
https://www.jobplanet.co.kr › companies › job_postings › 위드시스템
2019. 6. 12. - 위드시스템 정규직 기획 및 마케팅 (해당업무**경험자**) **우대** 채용(경력무관) 주요 업무, 자격 요건, 채용절차, 복리후생을 보시고 지금 바로 지원하세요.

"대졸신입 채용인데 관련분야 경험자 우대" 사실일까?
plus.hankyung.com › apps › newsinside
1일 전 - (공태윤 산업부 기자) '응용프로그램(API) 구축 **경험자**, 시스템 인터페이스 유경험자 **우대**'지난 15일 서울 강남구 코엑스에서 열린 '삼성 협력사 채용 ...

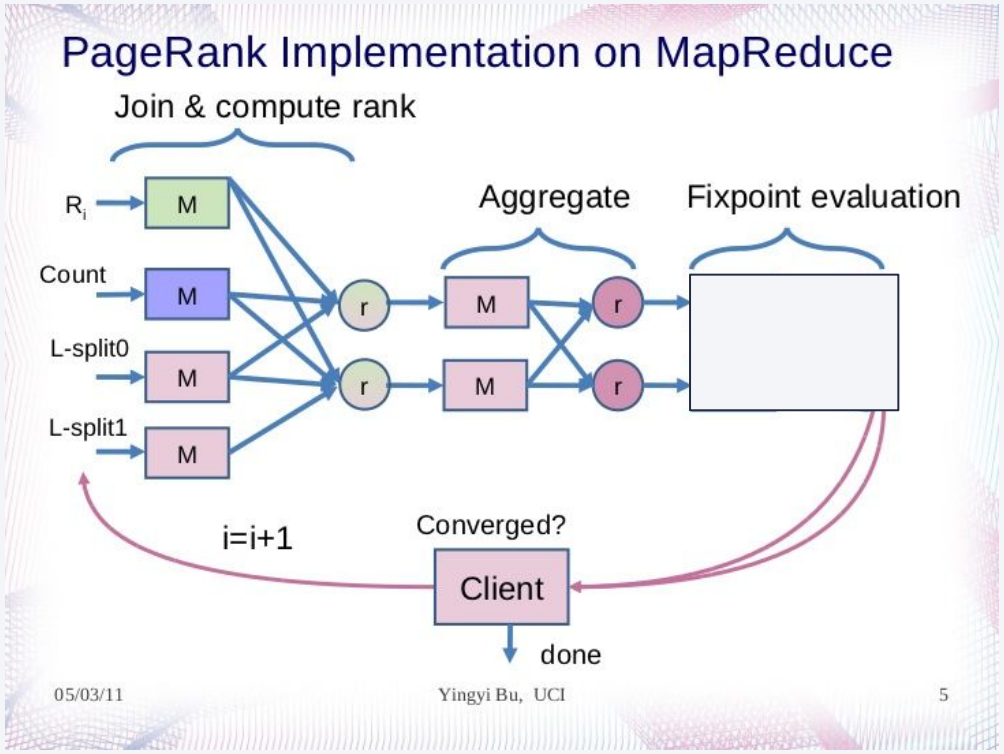
이제 막 졸업했는데 회사마다 '경험자 우대'...폭망 ㅠㅠ - 국민일보
m.kmib.co.kr › view
2019. 2. 16. - 이제 막 졸업했는데 회사마다 '**경험자 우대**'...폭망 ㅠㅠ. [취재대행소 왱] 아불대(아무불만대잔치), 20대가 직접 말하다 ①신입 채용에 왜 경력을 묻 ...

[그린피스이엔티] 여행사 항공발권(여행매니저 경험자 우대 ...
m.saramin.co.kr › job-search › view
2019. 9. 20. - 그린피스이엔티, 여행사 항공발권(여행매니저 **경험자 우대**) 사원 채용, 경력:경력 3년 이상, 학력:대학졸업(2,3년)이상, 연봉:2800~3000만원, ...

[(주)한국오리온] 용접사 경력사원 모집(경험자 우대) (D ... - 사람인
www.saramin.co.kr › zf_user › jobs › view
2019. 10. 1. - (주)한국오리온, 용접사 경력사원 모집(**경험자 우대**), 경력:경력 3년 이상, 학력:고등

03

Scheduler's Goal



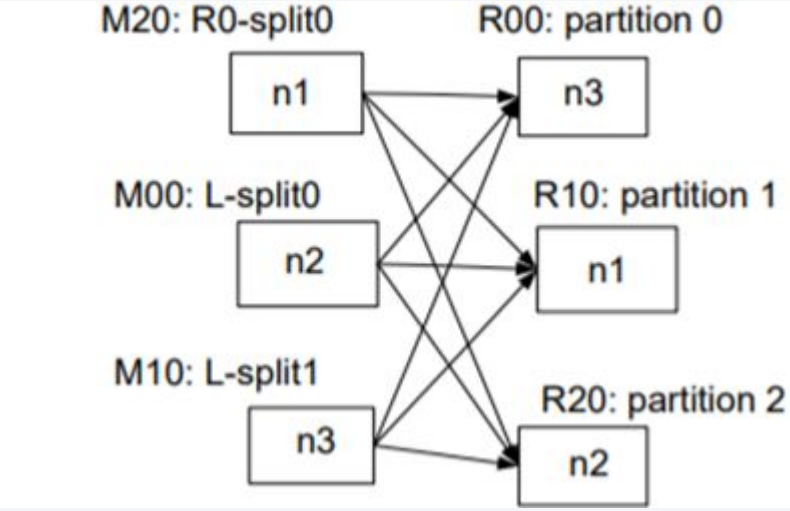
to place on the same physical machines those map and reduce tasks that occur in different iterations but access the same data.

So, data can more easily be cached and reused between iterations.

03

For Example

condition



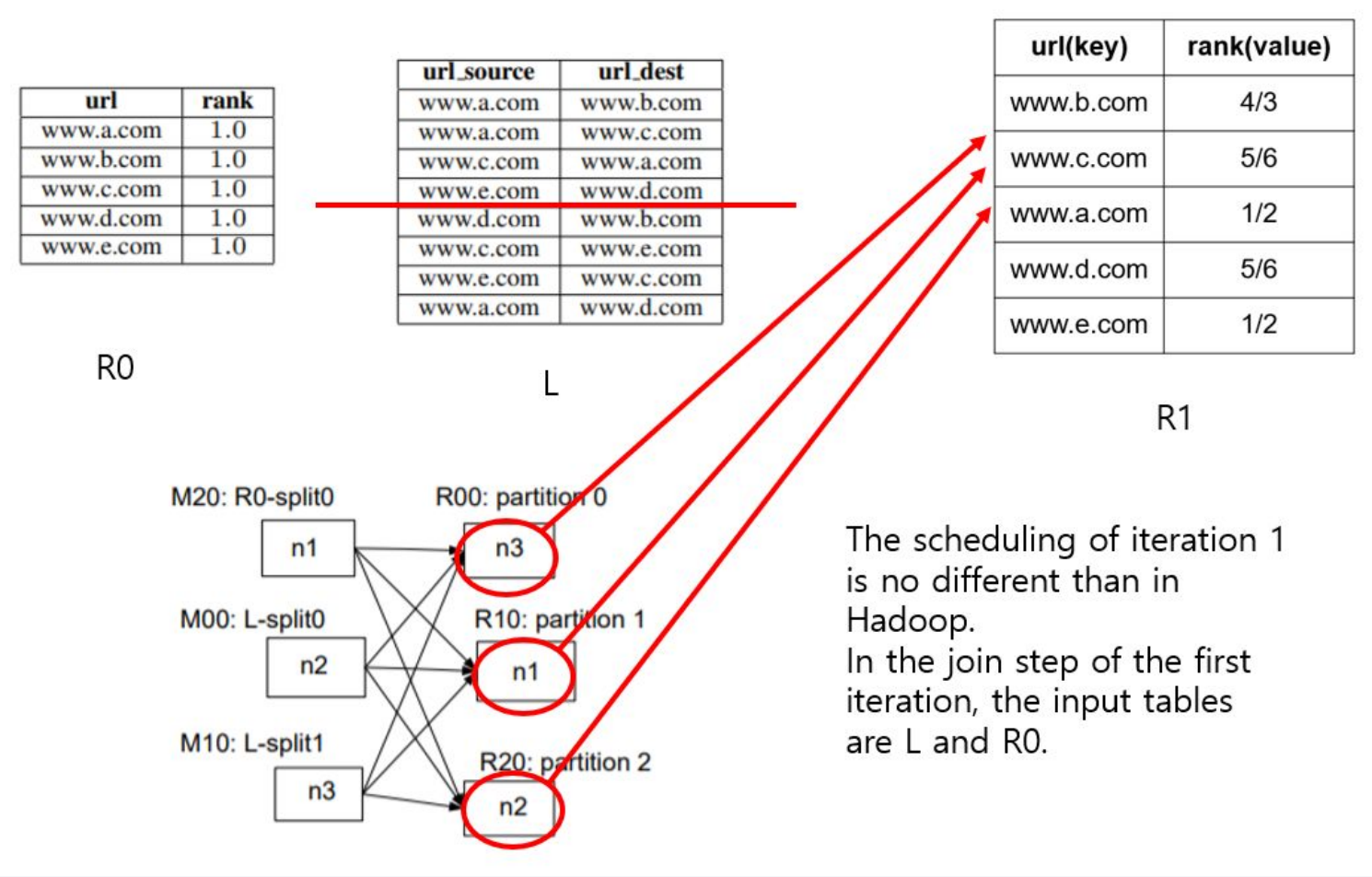
We want solve PageRank algorithm. (Very iterative algorithm)

This algorithm is divided join step & aggregation step.

We have 1 master node & 3 slave node.

For Example (2)

In First iteration.



03

For Example (3)

upper 2 iteration.

url(key)	rank(value)
www.b.com	4/3
www.c.com	5/6
www.a.com	1/2
www.d.com	5/6
www.e.com	1/2

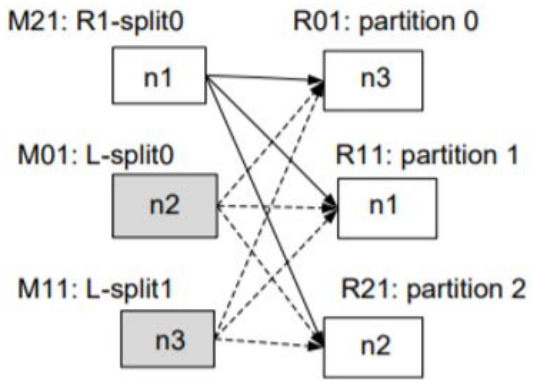
R1

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

L

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

R2

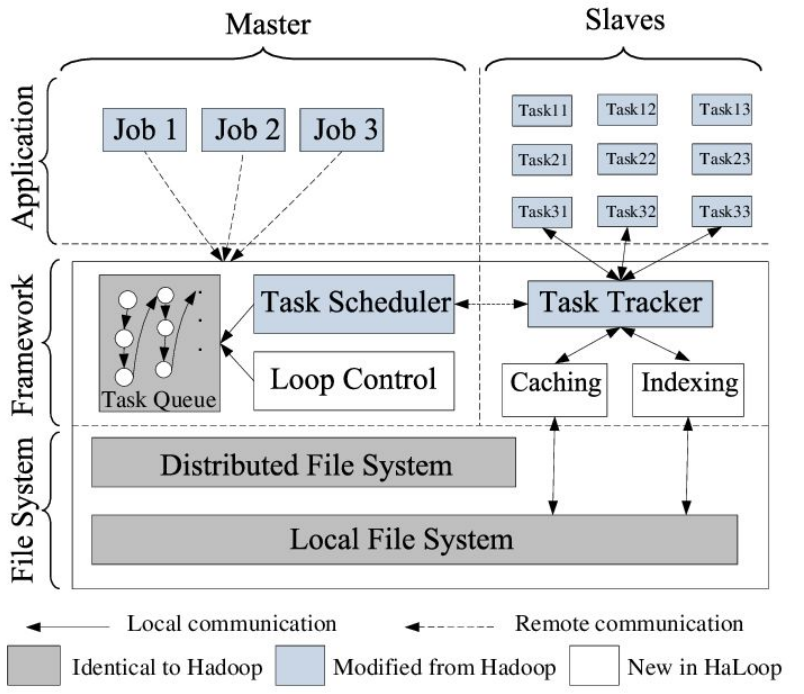


The scheduling of the join step of iteration 2 can take advantage of inter-iteration locality. Because they can reuse cache made from iteration 1.(L)

03

How to??

How to assign similar task to same node.



0)Task is done.

1)The master node receive a heartbeat from a slave node.

2)The master node maintains a mapping from each slave node.

3)In next Iteration, the master node tries to assign task to same node.

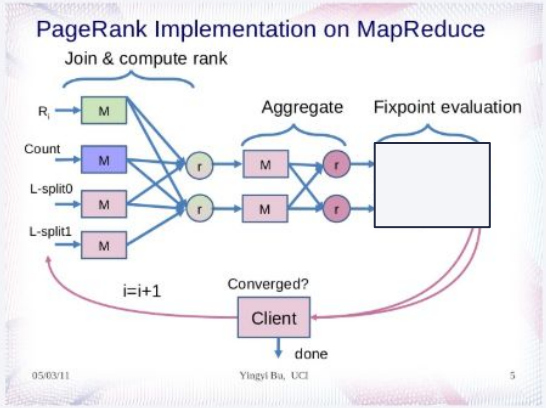
4)If the slave node already has a full load, the master re-assigns its tasks to a nearby slave node.

Cache

Haloop has 3 caches , also offer reloading

04 Cache

01 Scheduler's Goal



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

So, data can more easily be cached and reused between iterations.

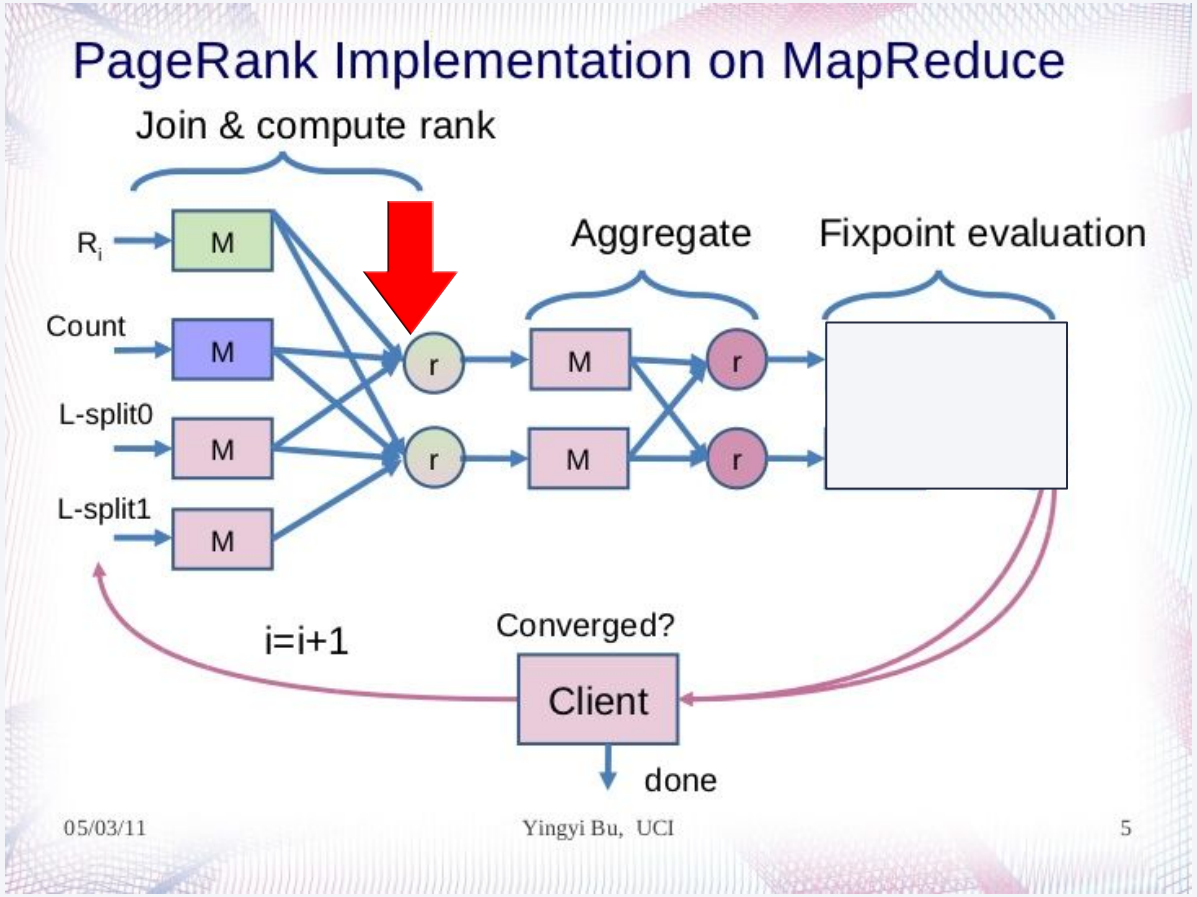
6

This cache saved in each node's local system.

Haloop has 3 caches, also offer reloading.

04

Reducer Input Cache



Goal : caching reducer input to reduce map cost.

can remove map step + sorting + grouping

Restriction : (partition function = f)
1.f must be deterministic
2.f must remain the same across iterations
3.f must not take any inputs other than the tuple t.

04

Reducer Input Cache

Example

name1	name2
Tom	Bob
Tom	Alice
Elisa	Tom
Elisa	Harry

(a) F-split0

name1	name2
Sherry	Todd
Eric	Elisa
Todd	John
Robin	Edward

(b) F-split1

name1	name2
Eric	Eric

(c) ΔS_0-split0

Figure 7: Mapper Input Splits in Example 2

name1	name2	table ID
Elisa	Tom	#1
Elisa	Harry	#1
Robin	Edward	#1
Tom	Bob	#1
Tom	Alice	#1

(a) partition 0

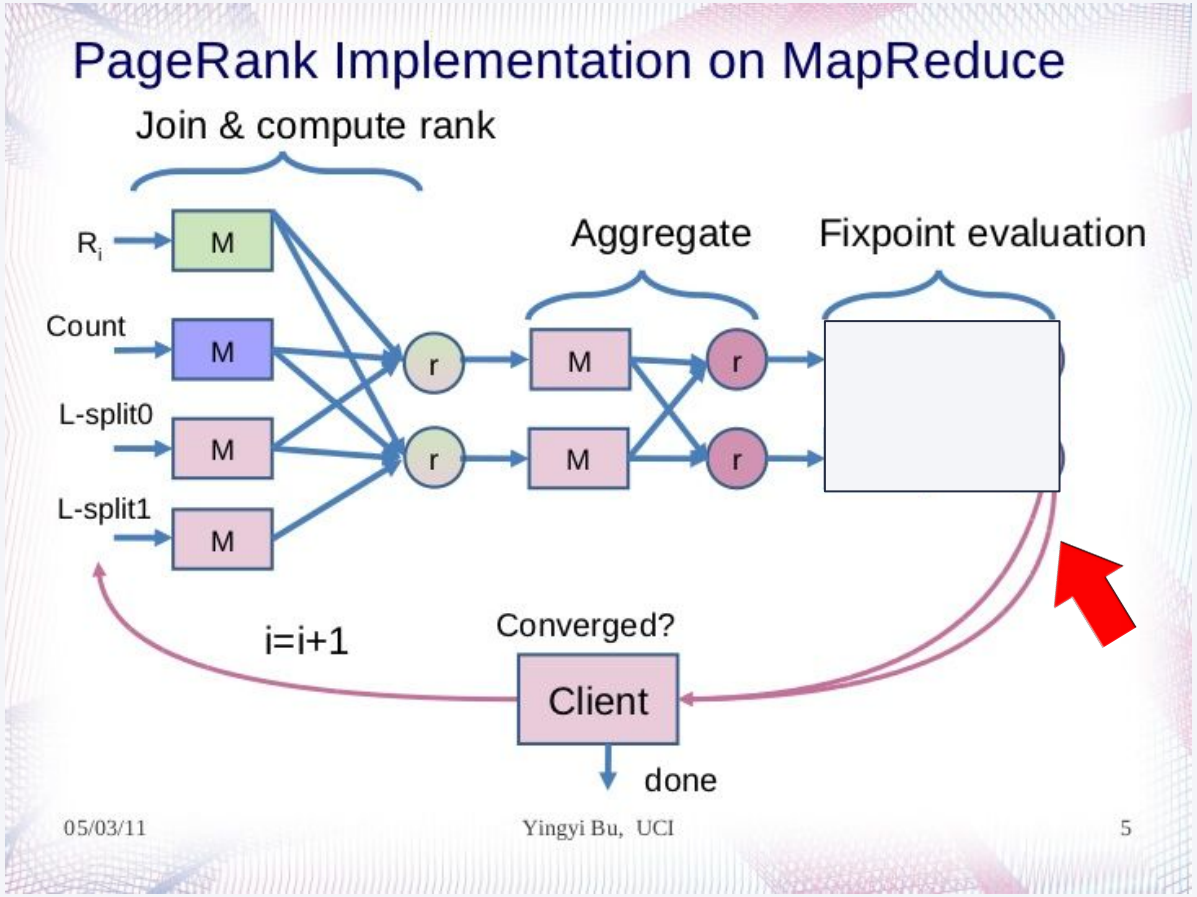
name1	name2	table ID
Eric	Elisa	#1
Eric	Eric	#2
Sherry	Todd	#1
Todd	John	#1

(b) partition 1

Figure 8: Reducer Input Partitions in Example 2

04

Reducer Ouput Cache



Goal : saving previous result to evaluate fixpoint termination conditions.

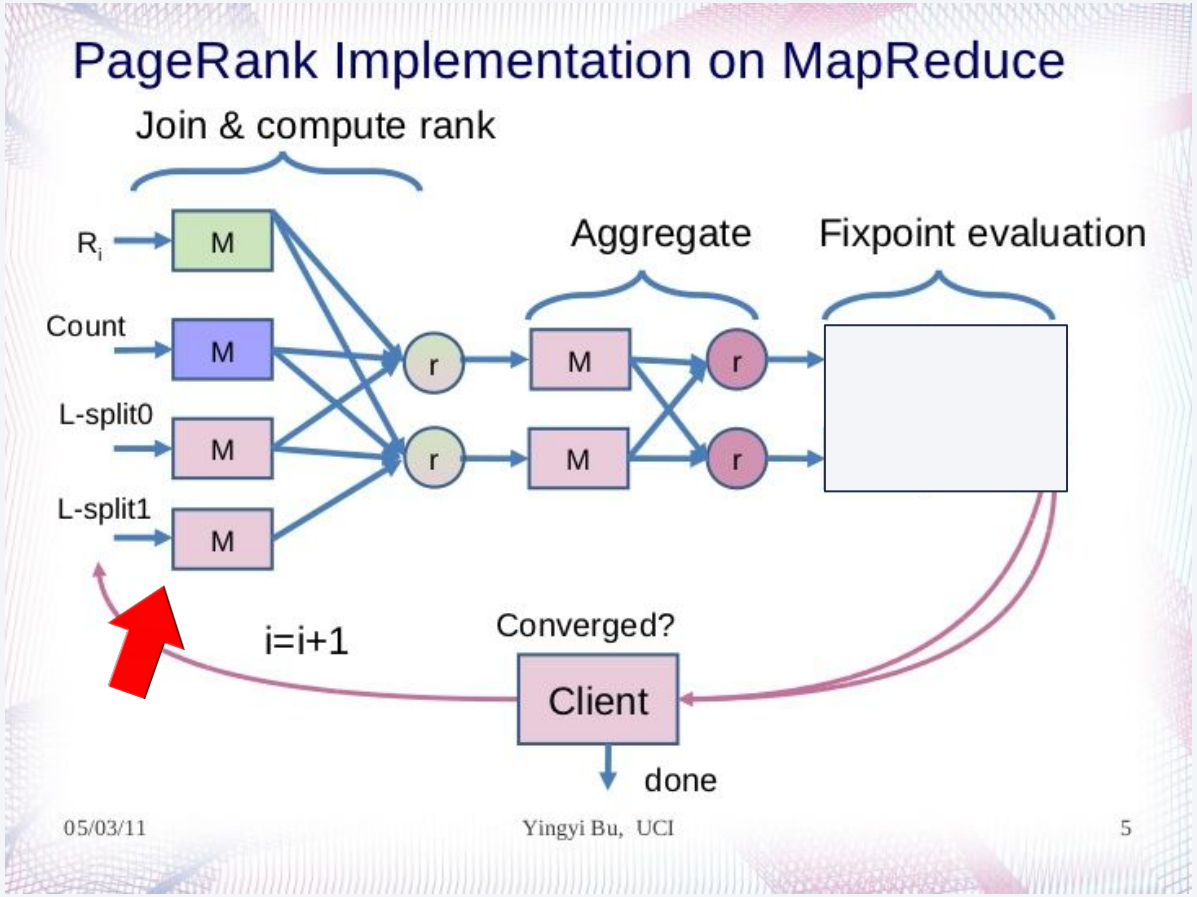
After all Reduce function invocations are done, each reducer evaluation results to the master node, which computes the final answer.

The reducer output cache enables the framework to perform the comparison in a distributed fashion.

Restriction :
if Reducer output's key is same, then input is from same partition.

04

Mapper Input Cache



Goal : caching mapper input data to enhance more locality.

Hadoop ensure locality 70~95%.
If you use this cache, you can increase this point.

If this node load from other node's data.
This node cache this data in this local.

Experiment

Checking Haloop’s output.

05

Setting

Use independently cache.

Hardware setting

- EC2 node 50 or 90

Data set

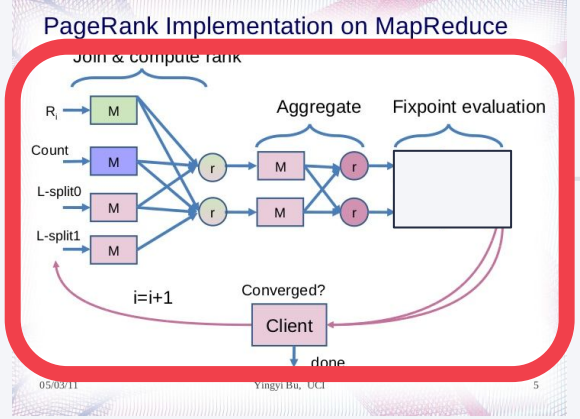
- Livejournal (18GB, social network)
- Freebase (12GB, concept linkage graph)
- Triples (120GB, web)

Test Program

- PageRank
- Descendant query

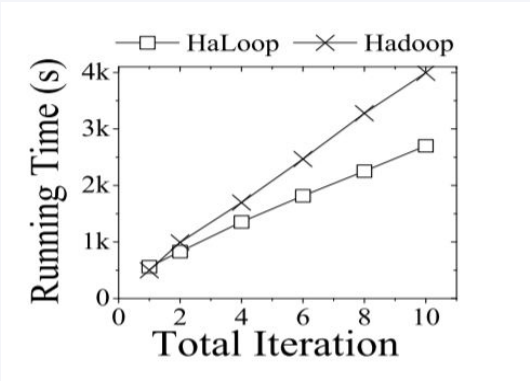
05

Reducer Input Cache

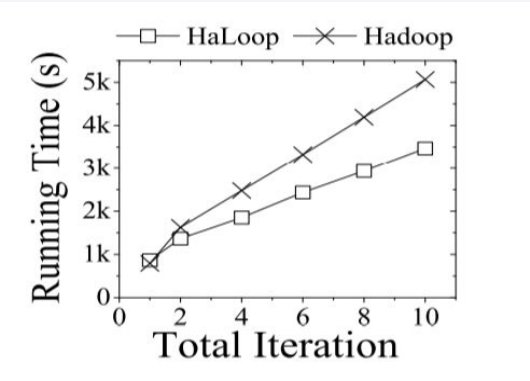


Overall

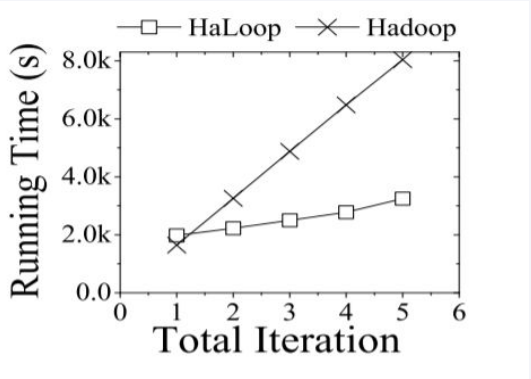
Haloop always performs better than Hadoop.



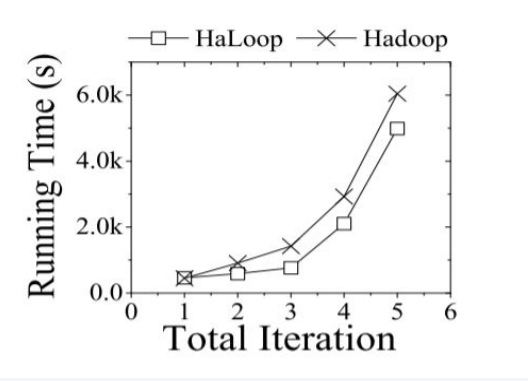
PageRank
(Livejournal , 50 nodes)



PageRank
(Freebase , 90 nodes)



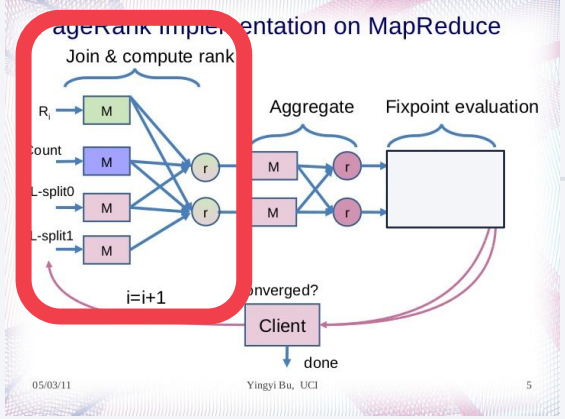
Descendant Query
(Triples , 90 nodes)



Descendant Query
(Livejoural , 90 nodes)

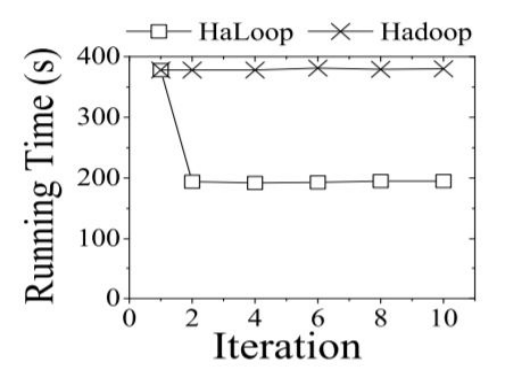
05

Reducer Input Cache

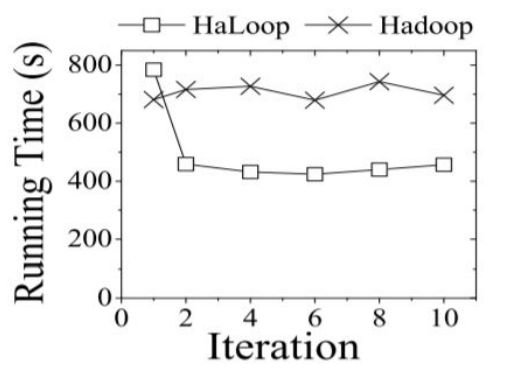


Join Step

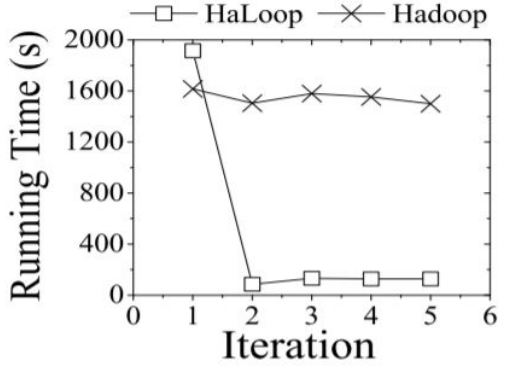
In the first iteration, Haloop is slower than Hadoop.
Because of caching.



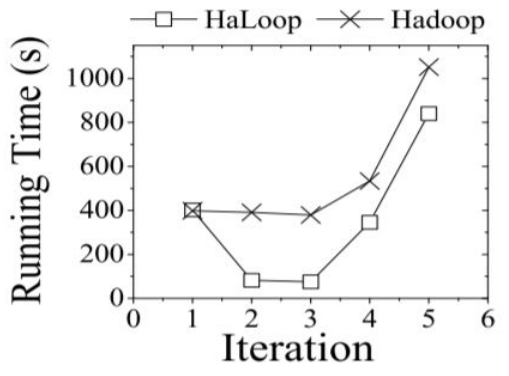
PageRank
(Livejournal , 50 nodes)



PageRank
(Freebase , 90 nodes)



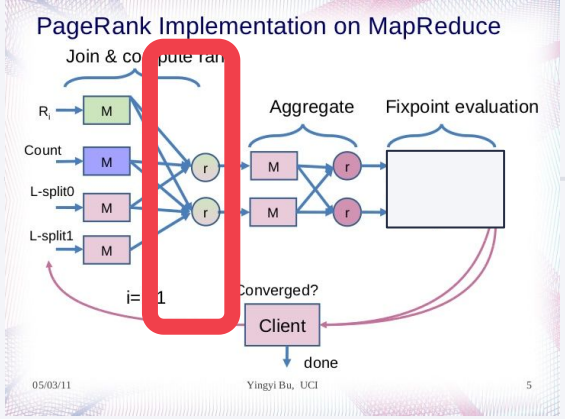
Descendant Query
(Triples , 90 nodes)



Descendant Query
(Livejoural , 90 nodes)

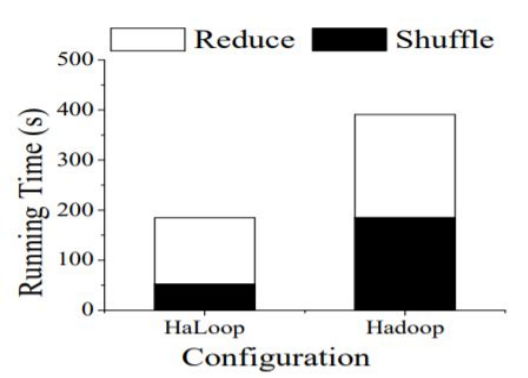
05

Reducer Input Cache

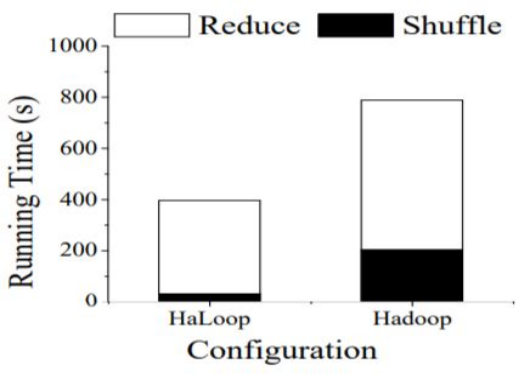


Cost Distribution

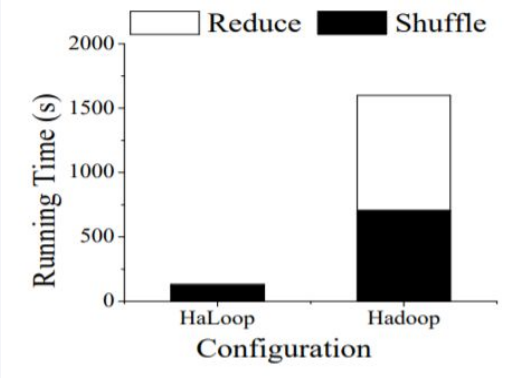
Reduce time : sorting time + grouping time + reducer's running time
Shuffle time : shuffle time(just before sorting) + mapper's running time



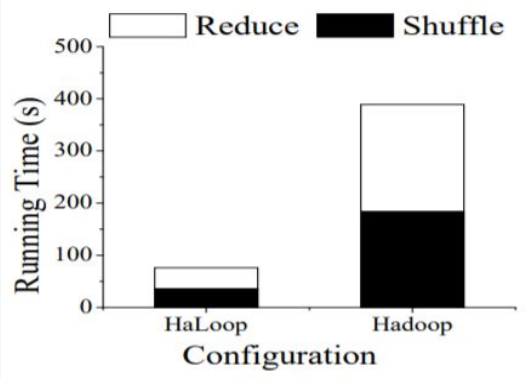
PageRank
(Livejournal , 50 nodes)



PageRank
(Freebase , 90 nodes)



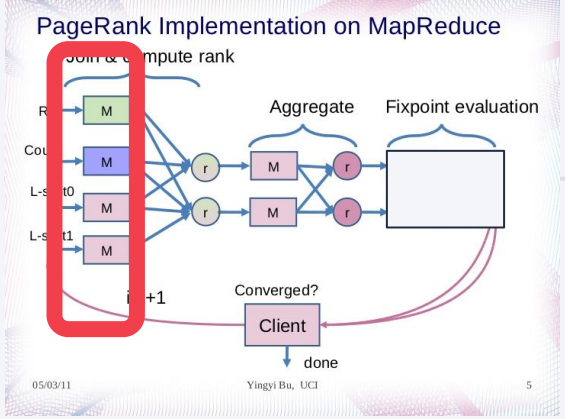
Descendant Query
(Triples , 90 nodes)



Descendant Query
(Livejoural , 90 nodes)

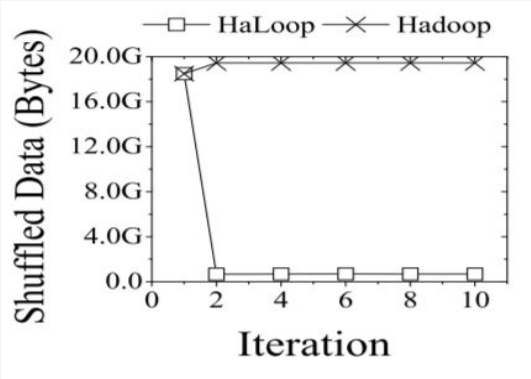
05

Reducer Input Cache

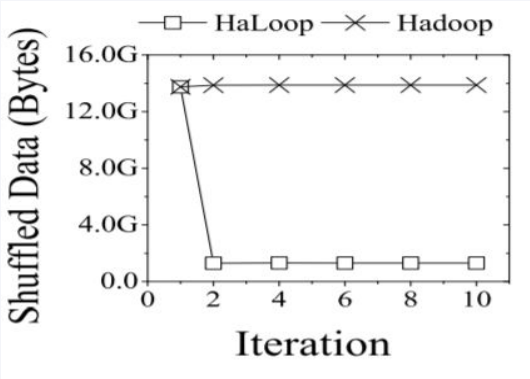


Shuffled Bytes

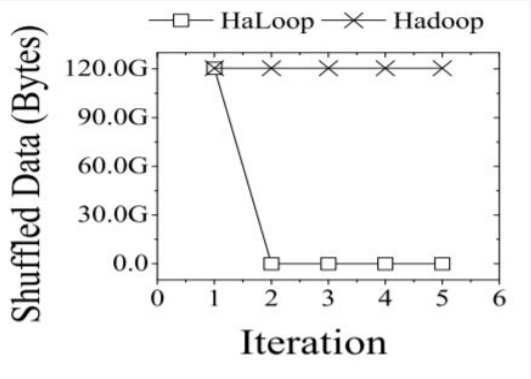
Haloop's join step shuffles 4% as much data as Hadoop's does.



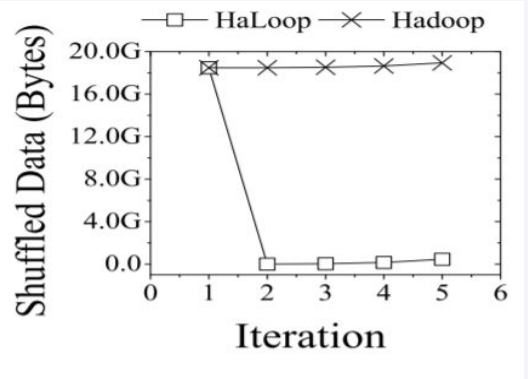
PageRank
(Livejournal , 50 nodes)



PageRank
(Freebase , 90 nodes)



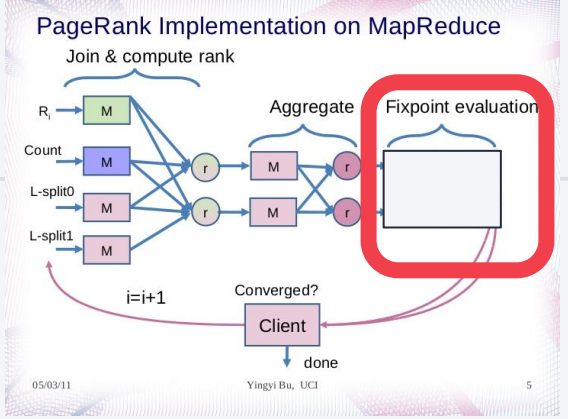
Descendant Query
(Triples , 90 nodes)



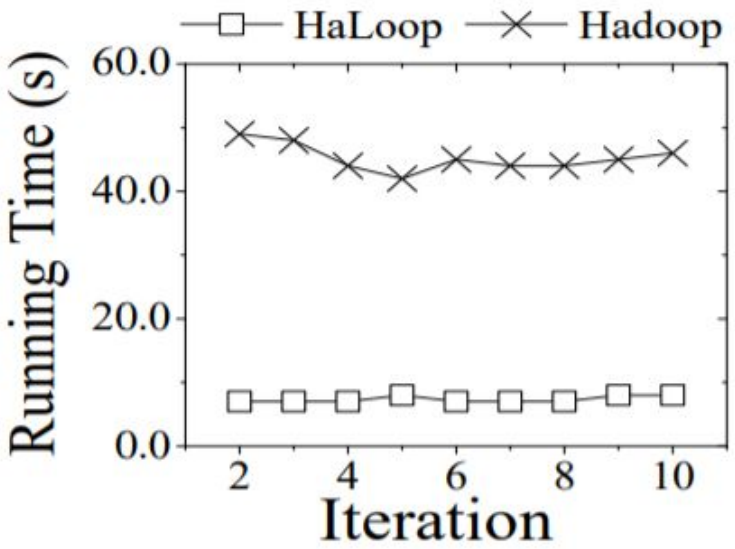
Descendant Query
(Livejoural , 90 nodes)

05

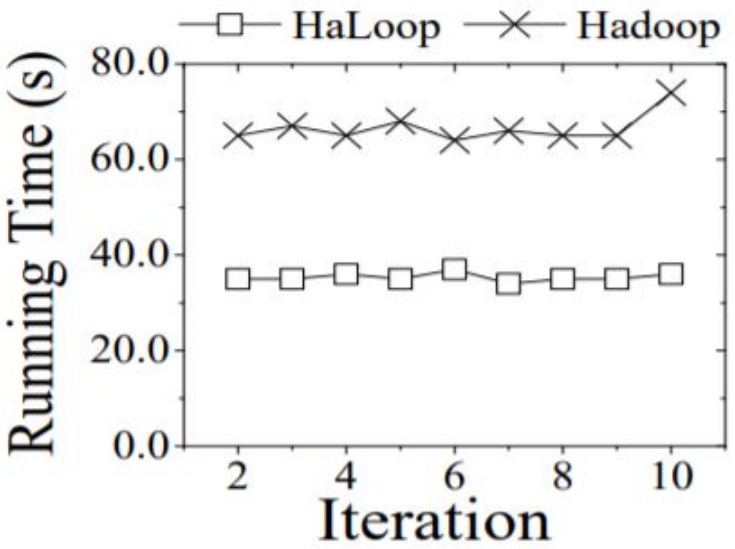
Reducer Output Cache



Hadoop : additional fixpoint checking task.
Haloop : check fixpoint by using reducer output cache.



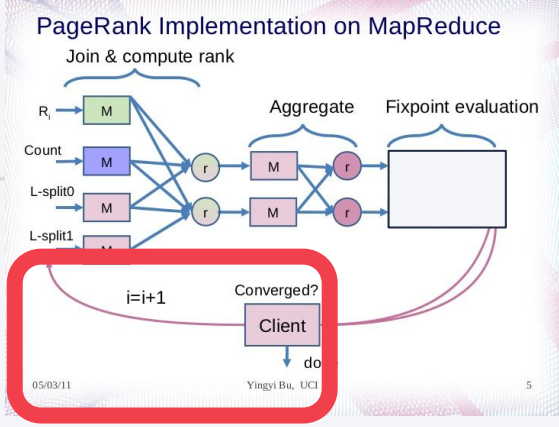
(a) Livejournal, 50 nodes



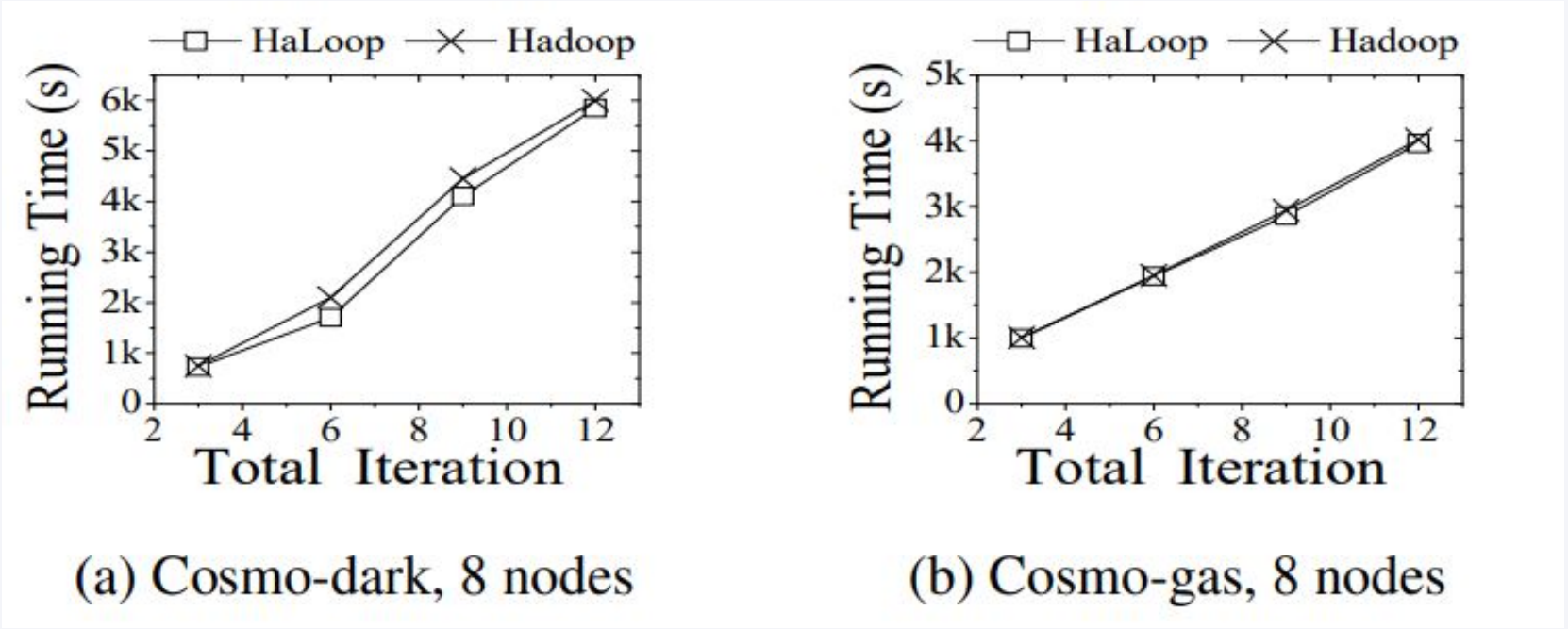
(b) Freebase, 90 nodes

05

Mapper Input Cache



By avoiding non-local data loading, Haloop performs better than Hadoop.



Overall

haloop is very efficient in iterative process.

06

Overall

1. Haloop is built on top of Haloop and extends it with a new programming model and several important optimizations that include
   (1) a loop-aware task scheduler, (2) loop-invariant data caching, (3) caching for efficient fixpoint verification.

2. Haloop is single pipeline program and synchronized iteration and pursue Disk cache.

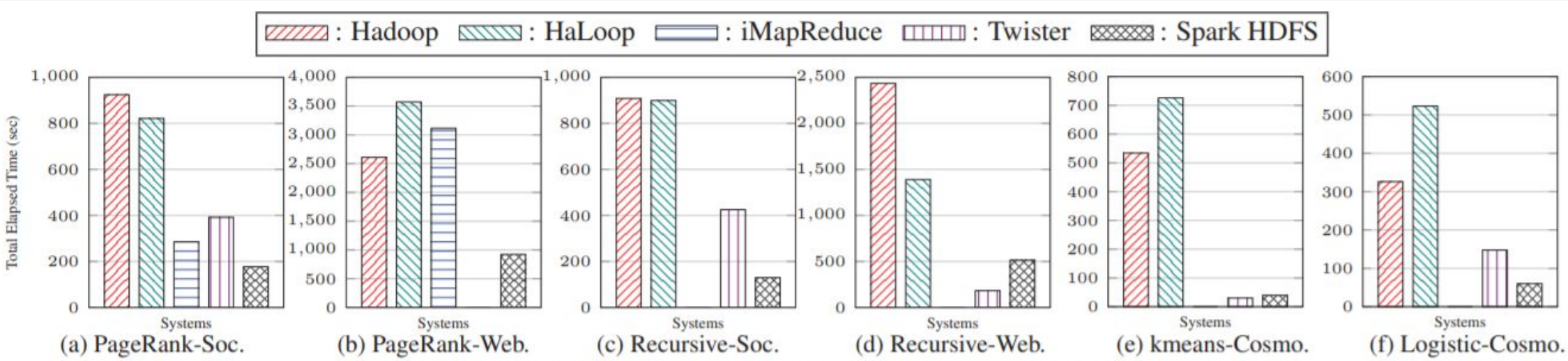


Figure 1: Total elapsed time of the five systems (50 machines).

출처 : An Experimental Comparison of Iterative MapReduce Frameworks,2016
https://dm.kaist.ac.kr/lab/papers/cikm16_expr.pdf

06

Thank you

Q & A