HaLoop: Efficient Iterative Data Processing On Large Scale Clusters

Yingyi Bu, UC Irvine

Bill Howe, UW

Magda Balazinska, UW

Michael Ernst, UW

Horizon

http://clue.cs.washington.edu/

Award IIS 0844572

Cluster Exploratory (CluE)

QuickTime™ and a decompressor are needed to see this picture.

http://escience.washington.edu/

VLDB 2010, Singapore

Thesis in one slide

- <u>Observation:</u> MapReduce has proven successful as a *common runtime* for non-recursive declarative languages
  - HIVE (SQL)
  - Pig (RA with nested types)

- <u>Observation:</u> Many people roll their own loops
  - Graphs, clustering, mining, recursive queries
  - iteration managed by external script
- <u>Thesis:</u> With minimal extensions, we can provide an efficient common runtime for ***recursive languages***
  - ***Map, Reduce, Fixpoint***

Related Work: Twister [Ekanayake HPDC 2010]

- Redesigned evaluation engine using pub/sub
- Termination condition evaluated by main()

```
while(!complete){
monitor = driver.runMapReduceBCast(cData);
monitor.monitorTillCompletion();

DoubleVectorData newCData = ((KMeansCombiner) driver
                .getCurrentCombiner()).getResults();
totalError = getError(cData, newCData);
cData = newCData;
  if (totalError < THRESHOLD) {
      complete = true;
      break;
  }
}
```

O(k)

In Detail: PageRank (Twister)

run MR

```
while (!complete) {
  // start the pagerank map reduce process
  monitor = driver.runMapReduceBCast(new
        BytesValue(tmpCompressedDvd.getBytes()));
  monitor.monitorTillCompletion();
  // get the result of process
  newCompressedDvd = ((PageRankCombiner)
        driver.getCurrentCombiner()).getResults();
  // decompress the compressed pagerank values
  newDvd = decompress(newCompressedDvd);
  tmpDvd = decompress(tmpCompressedDvd);
  totalError = getError(tmpDvd, newDvd);
  // get the difference between new and old pagerank values
  if (totalError < tolerance) {
    complete = true;
  }
  tmpCompressedDvd = newCompressedDvd;
}
```

term. cond.

O(N) in the size of the graph

Related Work: Spark [Zaharia HotCloud 2010]

- Reduction output collected at driver program
  - "…does not currently support a grouped reduce operation as in MapReduce"

```
val spark = new SparkContext(<Mesos master>)
var count = spark.accumulator(0)
for (i <- spark.parallelize(1 to 10000, 10)) {
  val x = Math.random * 2 - 1
  val y = Math.random * 2 - 1
  if (x*x + y*y < 1) count += 1
}
println("Pi is roughly " + 4 * count.value / 10000.0)
```

all output sent to driver.

Related Work: Pregel [Malewicz PODC 2009]

- Graphs only
  - clustering: k-means, canopy, DBScan
- Assumes each vertex has access to outgoing edges
- So an edge representation …

Edge(from, to)

- …requires offline preprocessing
  - perhaps using MapReduce

Related Work: Piccolo [Power OSDI 2010]

- Partitioned table data model, with user-defined partitioning
- Programming model:
  - message-passing with global synchronization barriers
- User can give locality hints

GroupTables(curr, next, graph)

- Worth exploring a direct comparison

Related Work: BOOM [c.f. Alvaro EuroSys 10]

- Distributed computing based on Overlog (Datalog + temporal logic + more)
- Recursion supported naturally
  - app: API-compliant implementation of MR
- Worth exploring a direct comparison

Details

- ~~Architecture~~
- ~~Programming Model~~
- Caching (and Indexing)
- ~~Scheduling~~

Example 1: PageRank

Rank Table R_0

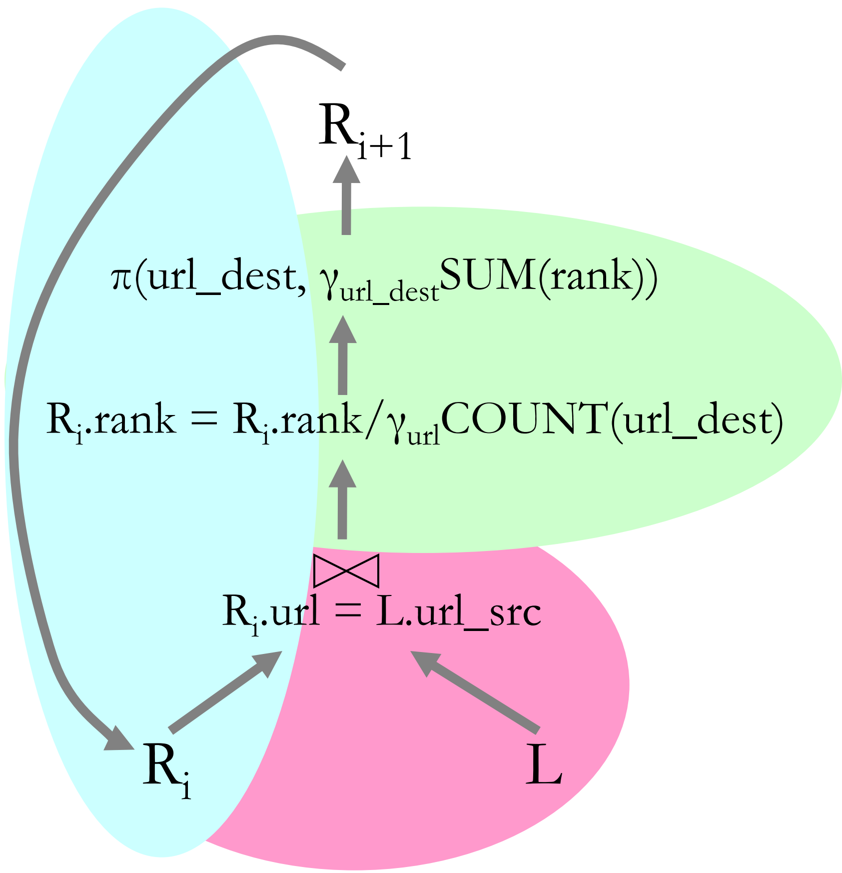
url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

Linkage Table *L*

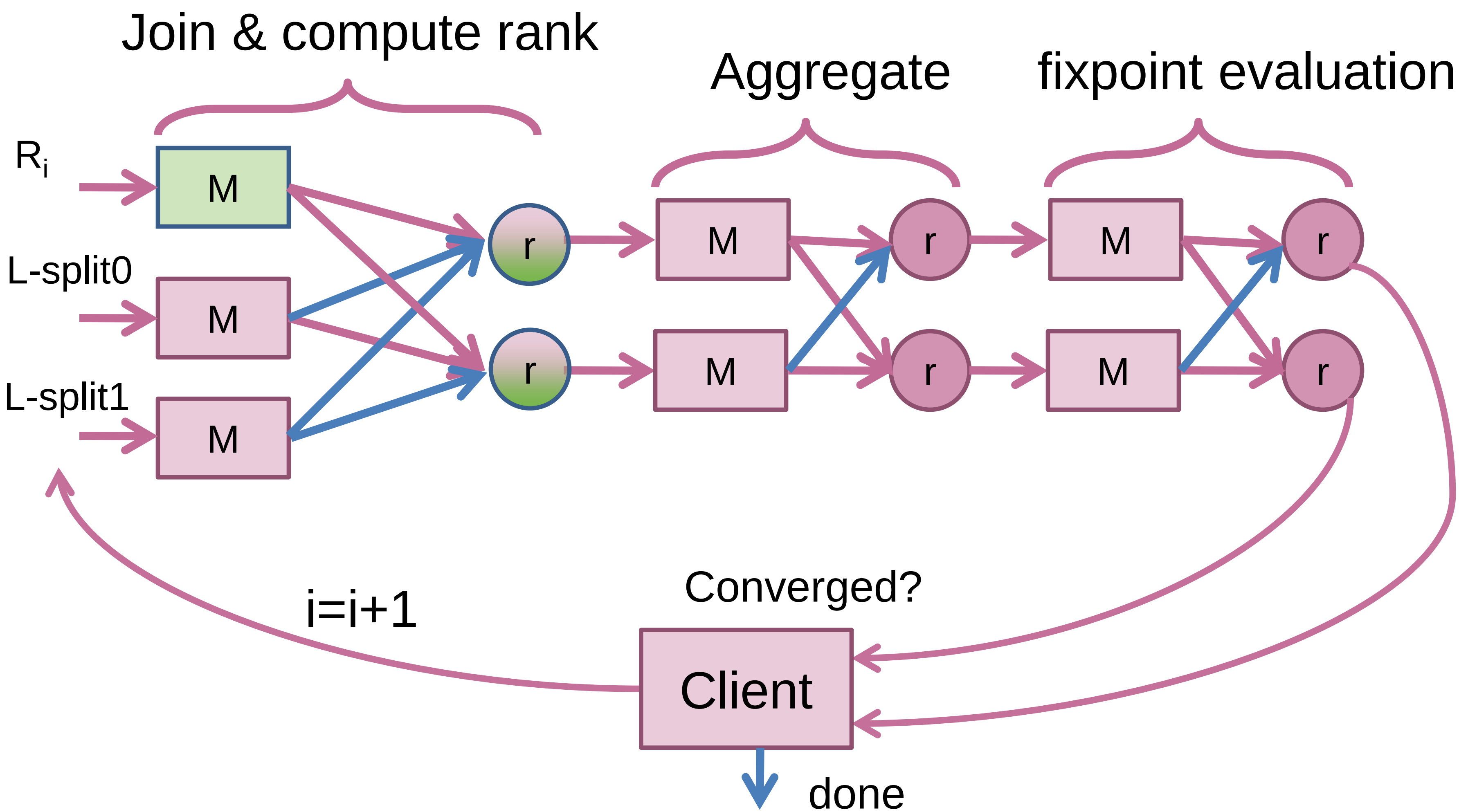
url_src	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.c.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.om
www.a.com	www.d.com

Rank Table R_3

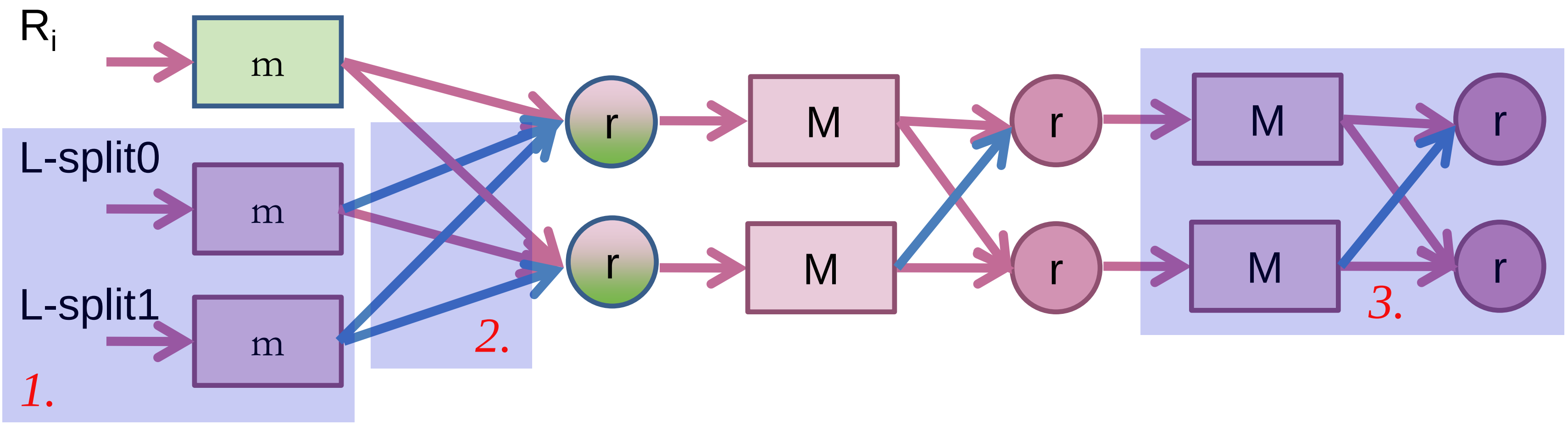
url	rank
www.a.com	2.13
www.b.com	3.89
www.c.com	2.60
www.d.com	2.60
www.e.com	2.13



A MapReduce Implementation



What's the problem?



L is loop invariant, but

1. *L is loaded on each iteration*
2. *L is shuffled on each iteration*

plus

3. *Fixpoint evaluated as a separate MapReduce job per iteration*

Example 2: Transitive Closure

Friend

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

Find all transitive friends of Eric

R_0 {Eric, Eric}

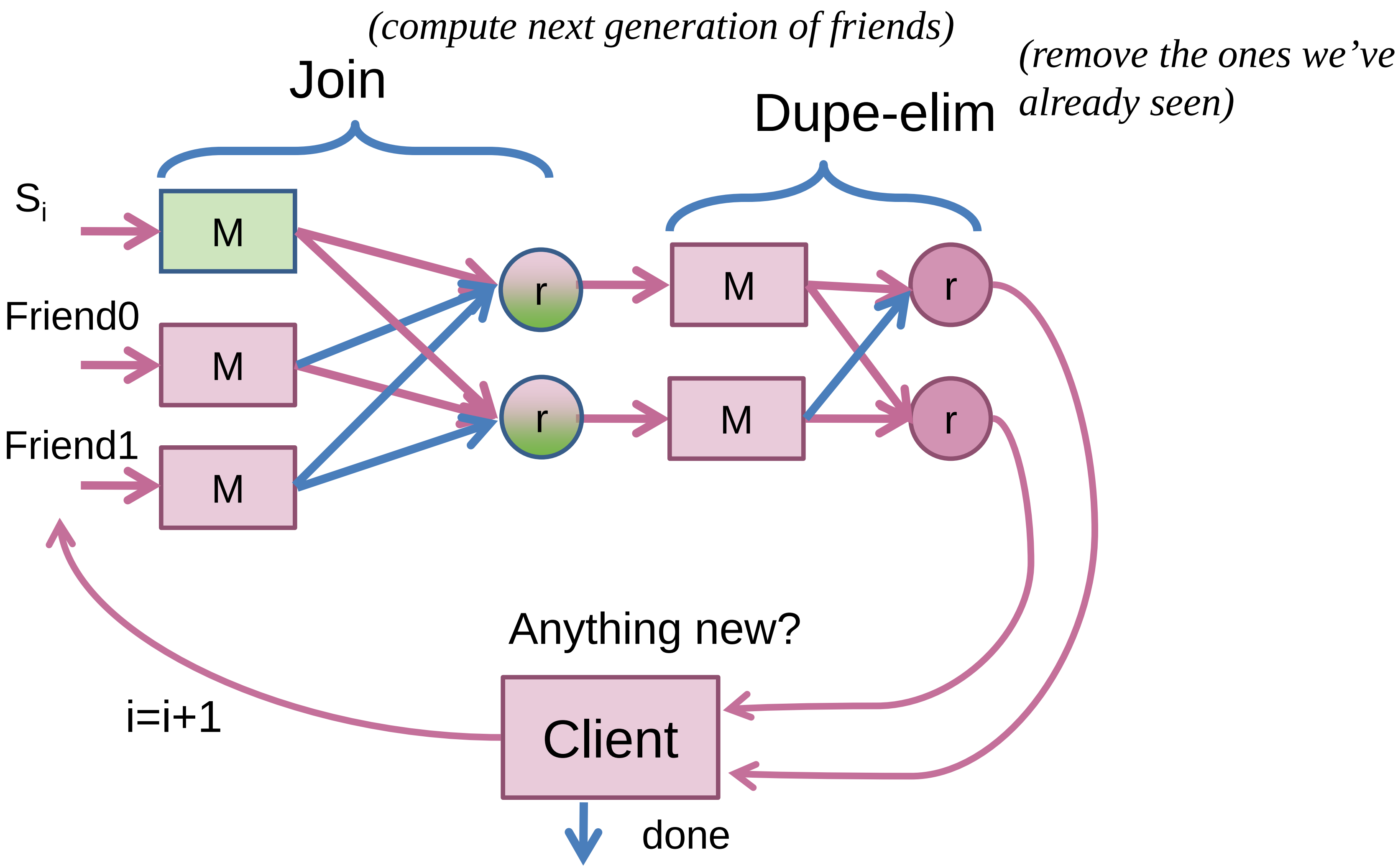
R_1 {Eric, Elisa}

R_2 {Eric, Tom
Eric, Harry}

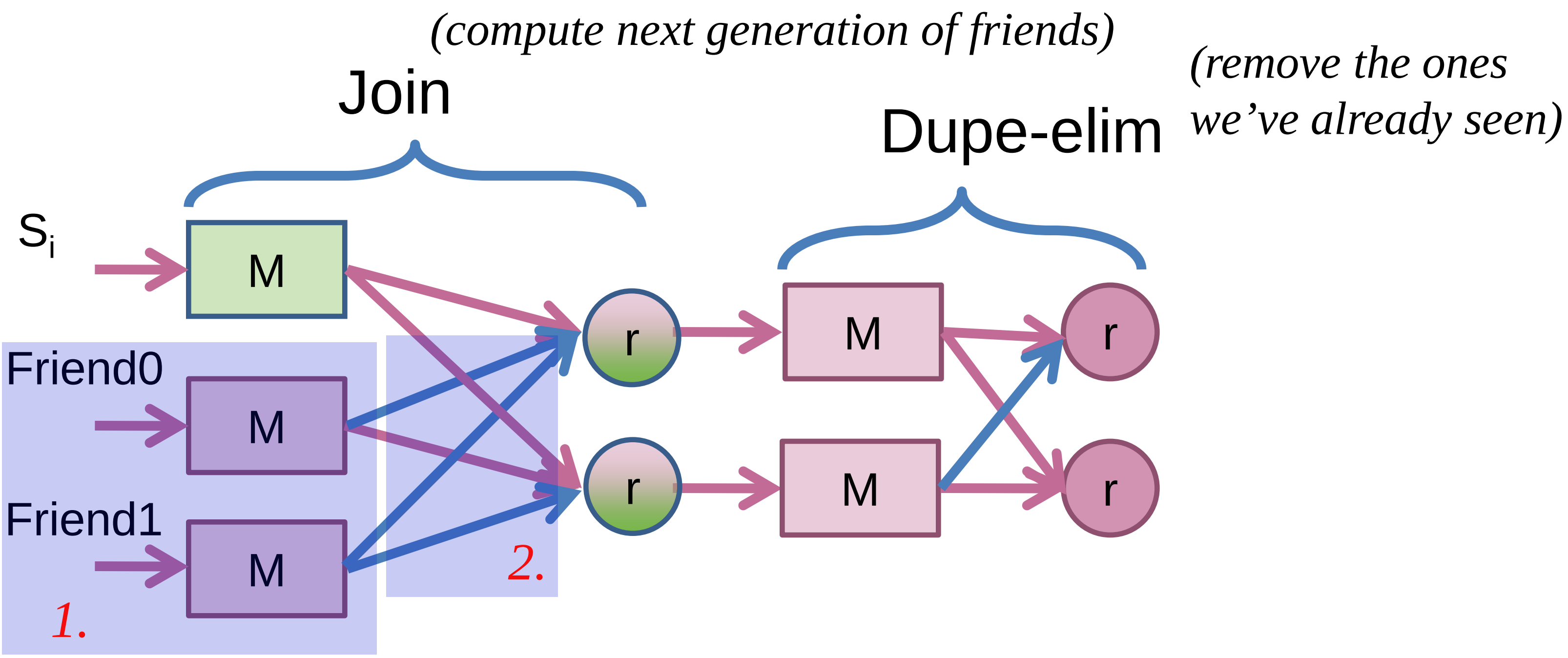
R_3 {}

(semi-naïve evaluation)

Example 2 in MapReduce



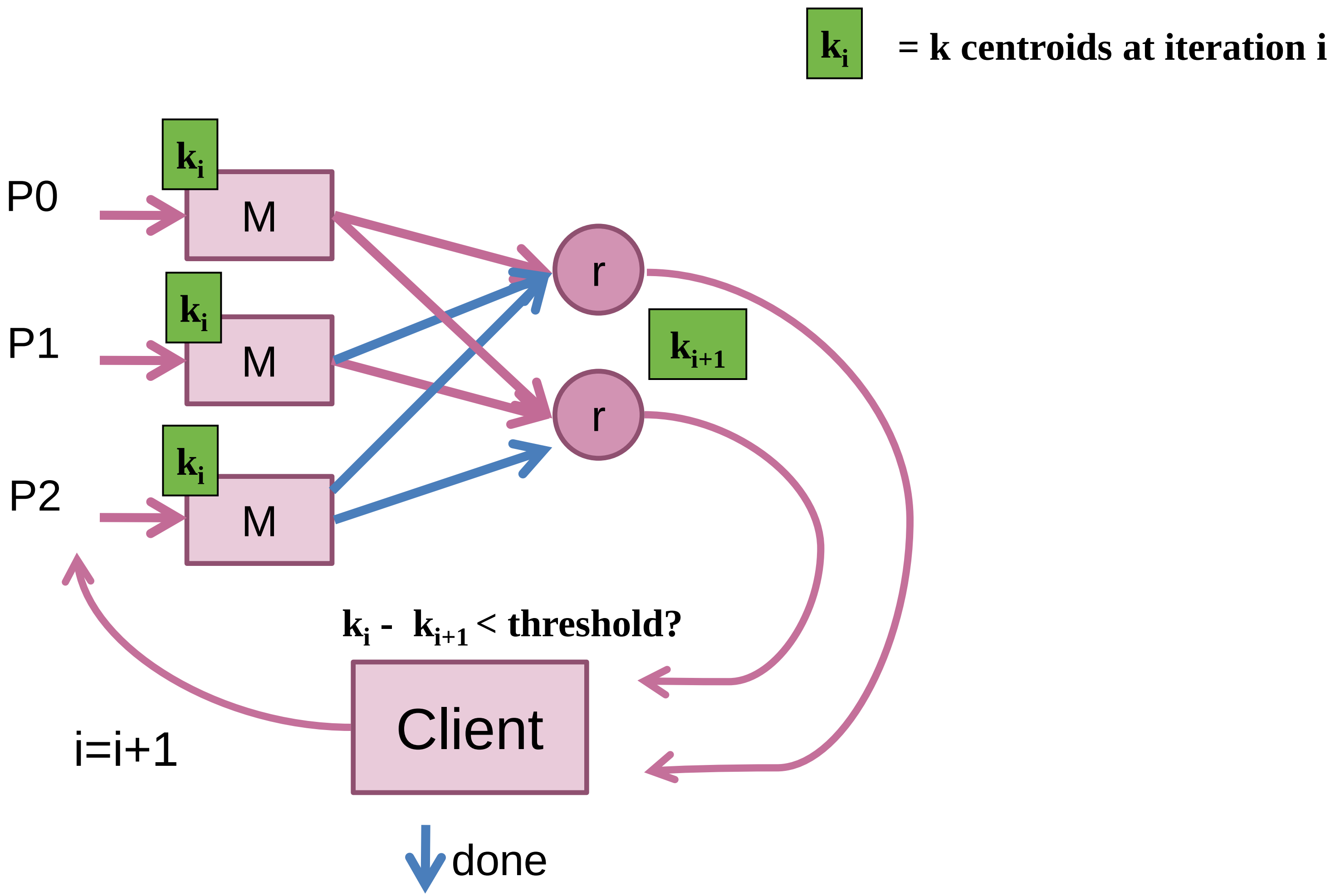
What's the problem?



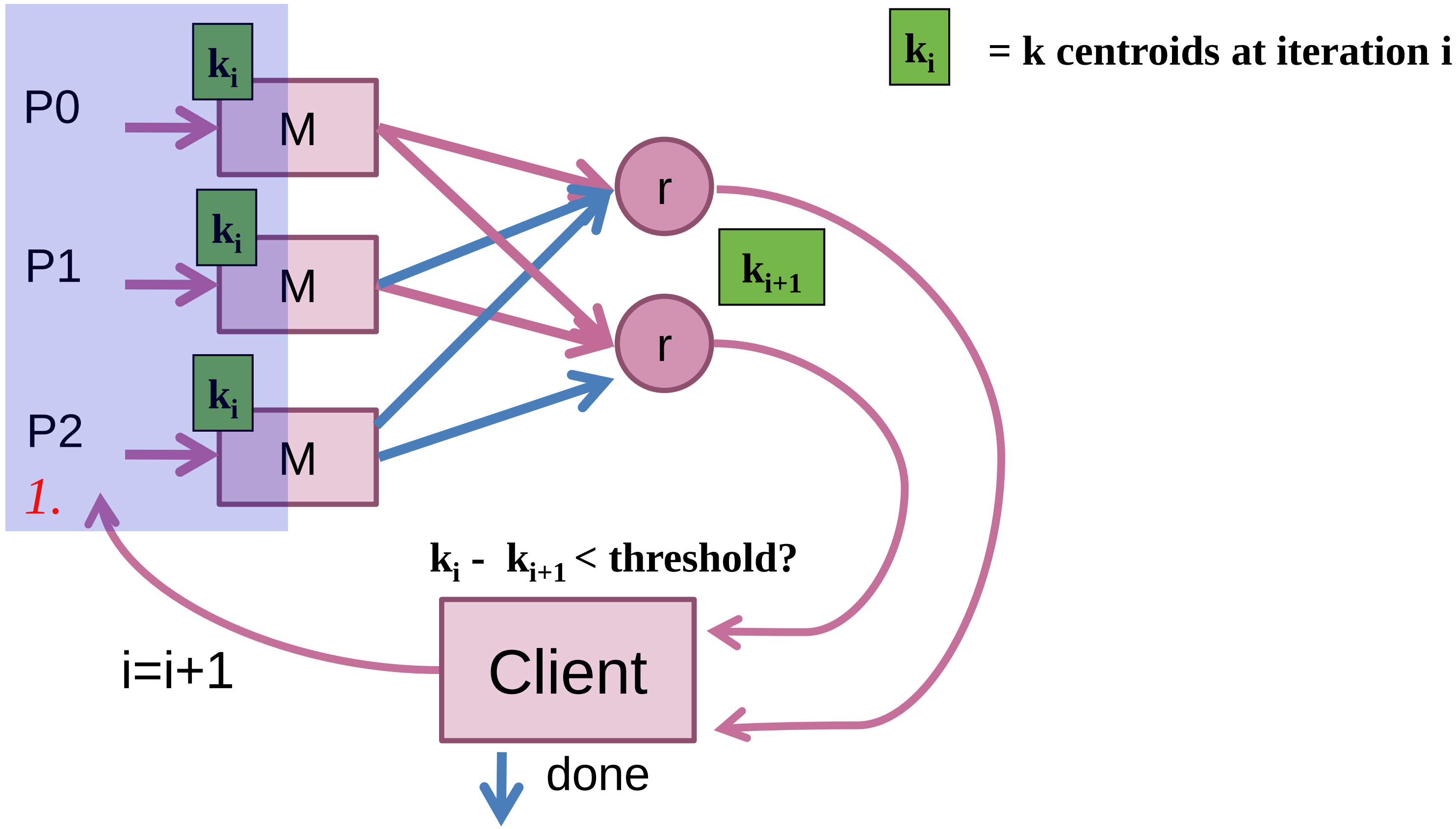
Friend is loop invariant, but

1. *Friend is loaded on each iteration*
2. *Friend is shuffled on each iteration*

Example 3: k-means



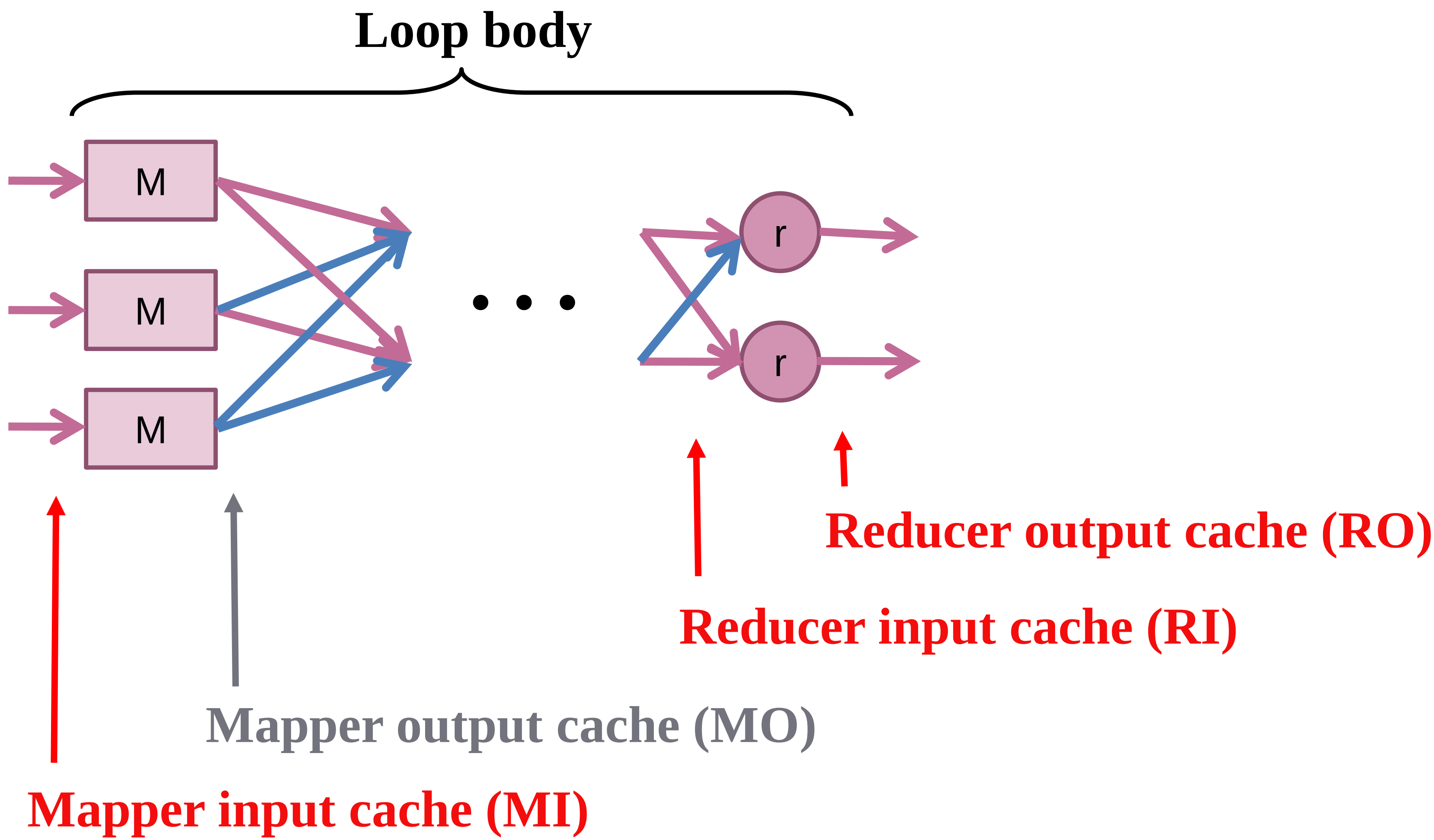
What’s the problem?



P is loop invariant, but

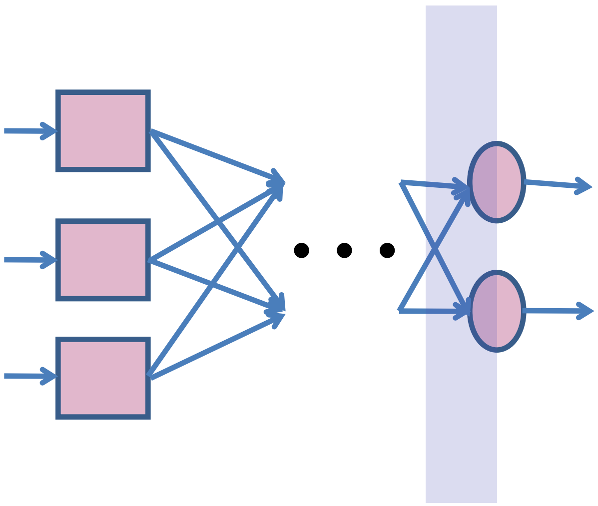
1. P is loaded on each iteration

Approach: Inter-iteration caching



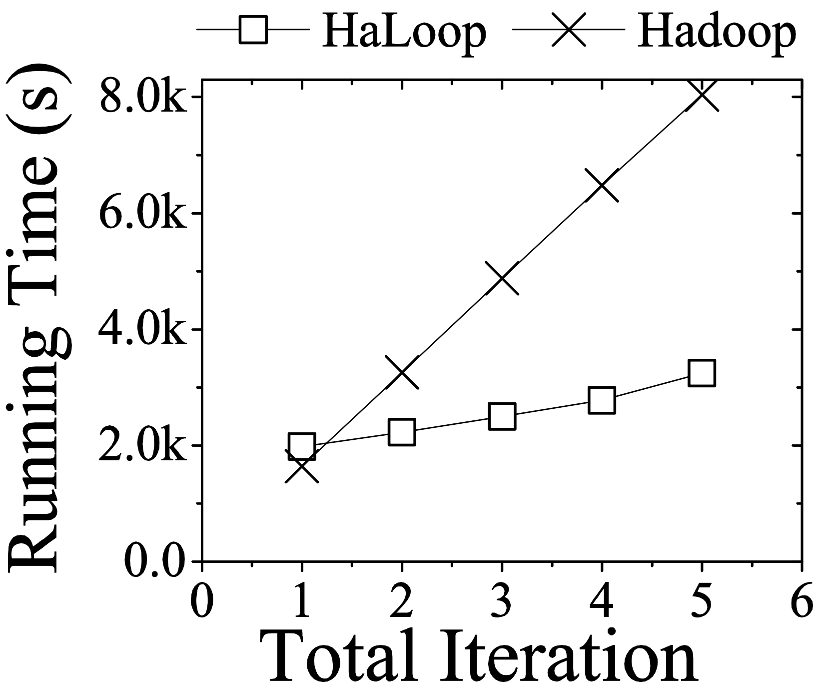
RI: Reducer Input Cache

- 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
- Transitive Closure
  - Avoid shuffling the graph at every step
- K-means
  - No help

Reducer Input Cache Benefit



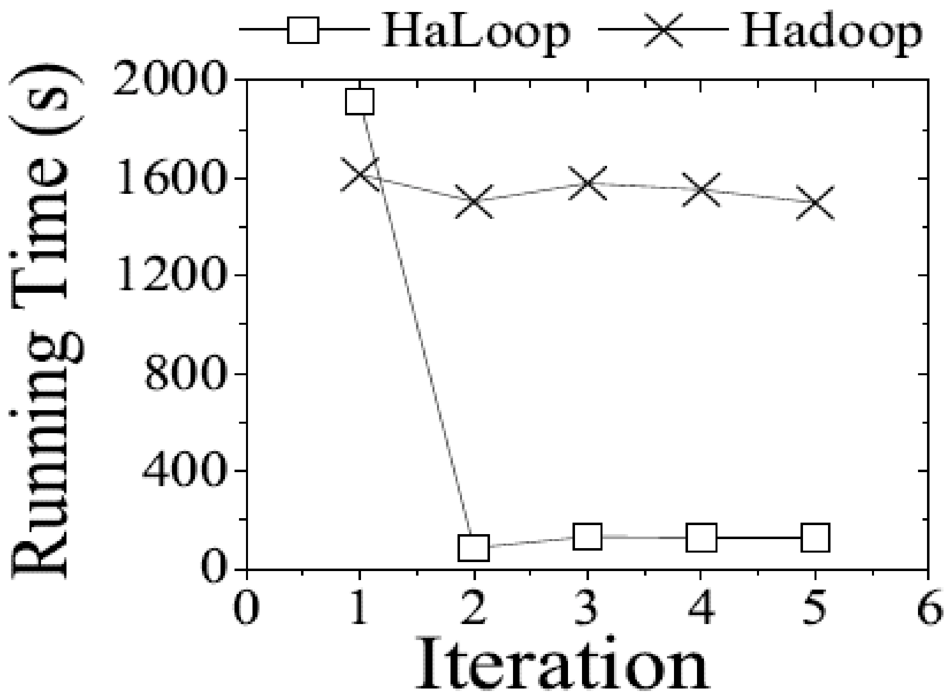
Transitive Closure

Billion Triples Dataset (120GB)

90 small instances on EC2

Overall run time

Reducer Input Cache Benefit



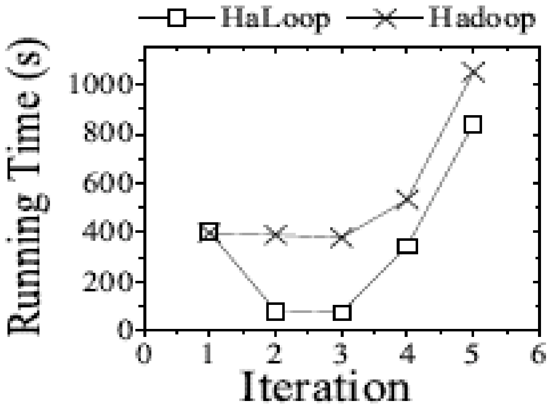
Transitive Closure

Billion Triples Dataset (120GB)

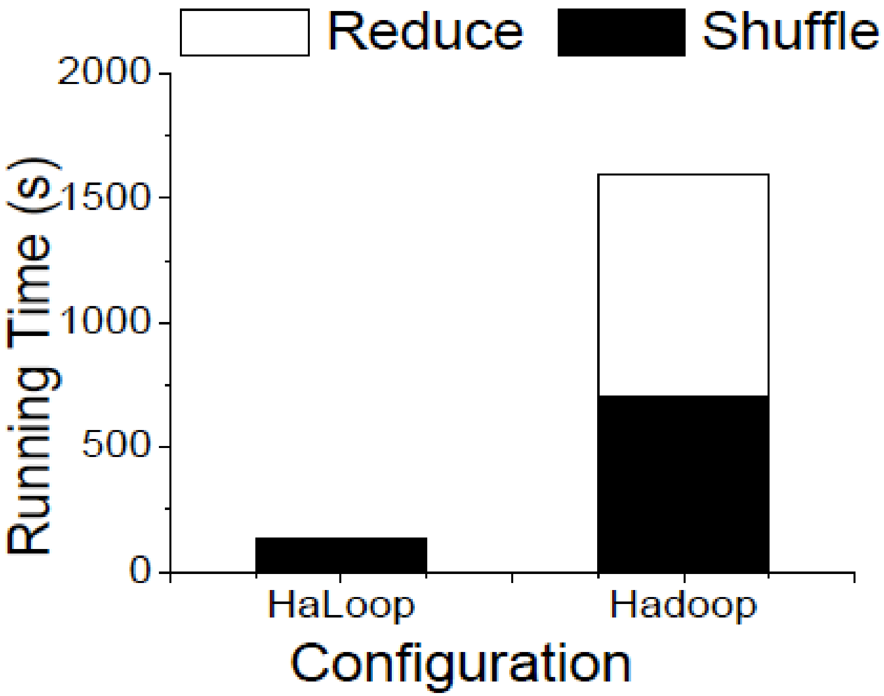
90 small instances on EC2

Join step only

Livejournal, 12GB



Reducer Input Cache Benefit



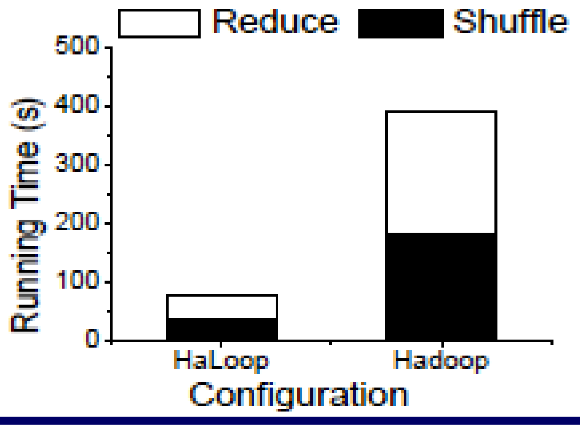
Transitive Closure

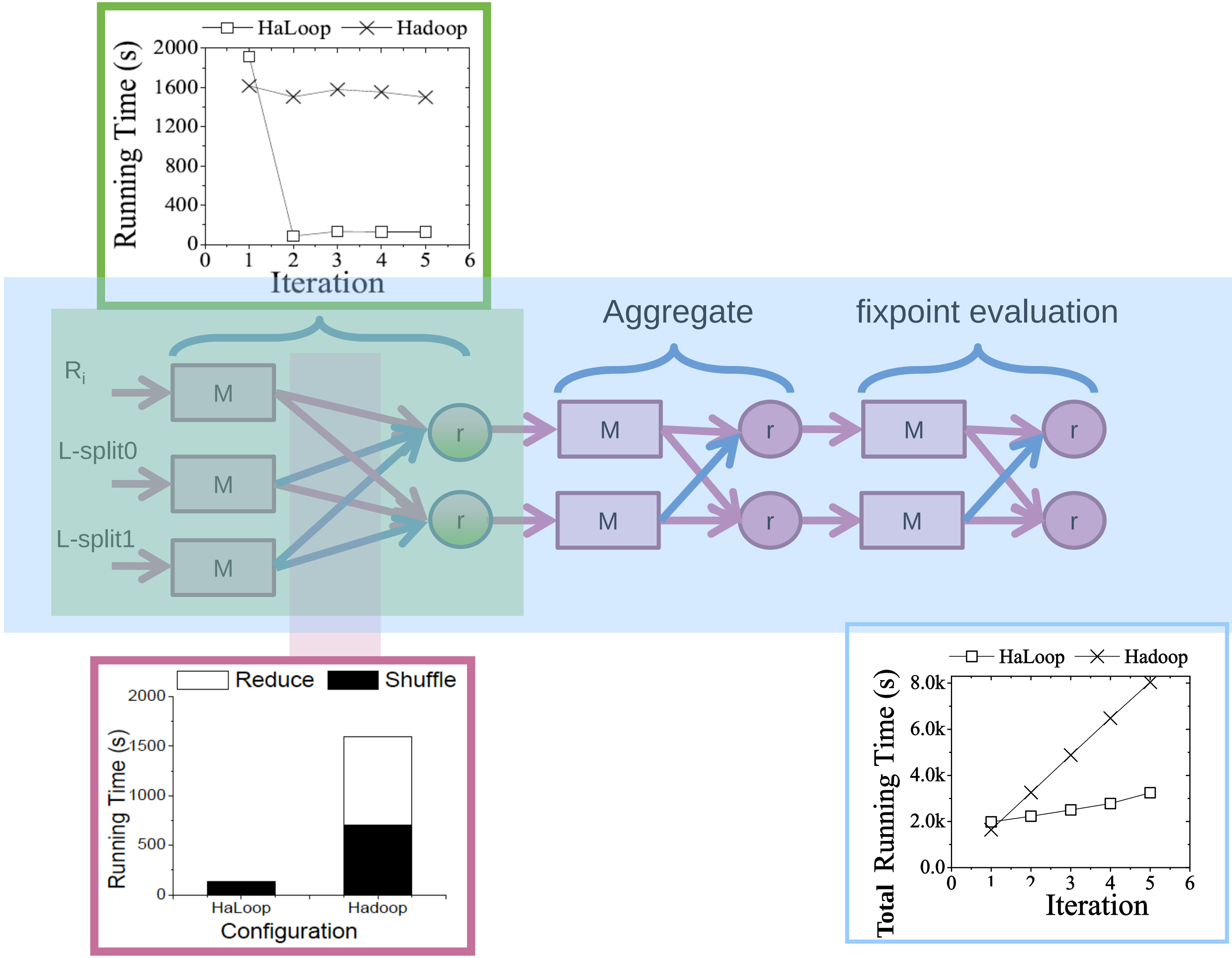
Billion Triples Dataset (120GB)

90 small instances on EC2

Reduce and Shuffle of Join Step

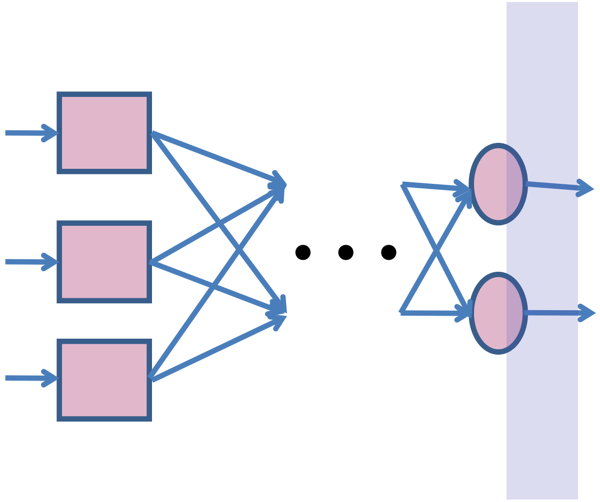
Livejournal, 12GB




HaLoop Hadoop
Running Time (s)
2000
1600
1200
800
400
0
0 1 2 3 4 5 6
Iteration
Ri
L-split0
L-split1
M
M
M
r
r
Aggregate
fixpoint evaluation
M
M
r
r
M
M
r
r
Reduce Shuffle
Running Time (s)
2000
1500
1000
500
0
HaLoop Hadoop
Configuration
HaLoop Hadoop
Total Running Time (s)
8.0k
6.0k
4.0k
2.0k
0.0
0 1 2 3 4 5 6
Iteration

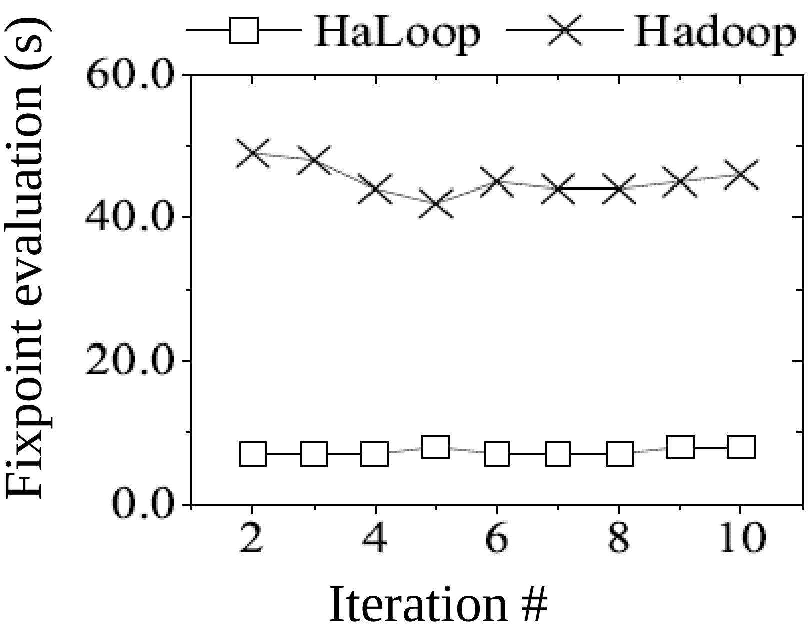
RO: Reducer Output Cache

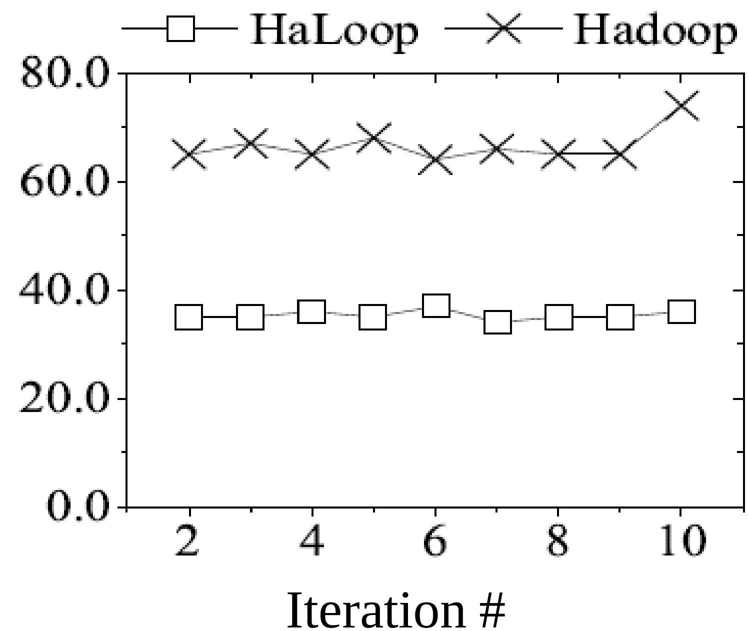
- Provides:
  - Distributed access to output of previous iterations
- Used By:
  - Fixpoint evaluation
- Assumes:
  1. Partitioning constant across iterations
  2. Reducer output key functionally determines Reducer input key



- PageRank
  - Allows distributed fixpoint evaluation
  - Obviates extra MapReduce job
- Transitive Closure
  - No help
- K-means
  - No help

Reducer Output Cache Benefit





Livejournal dataset

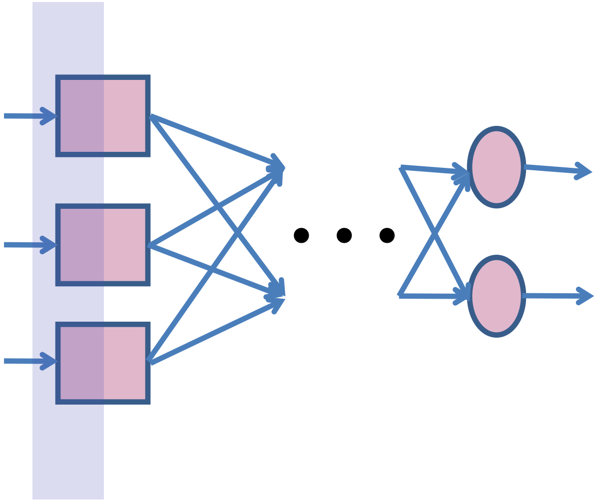
50 EC2 small instances

Freebase dataset

90 EC2 small instances

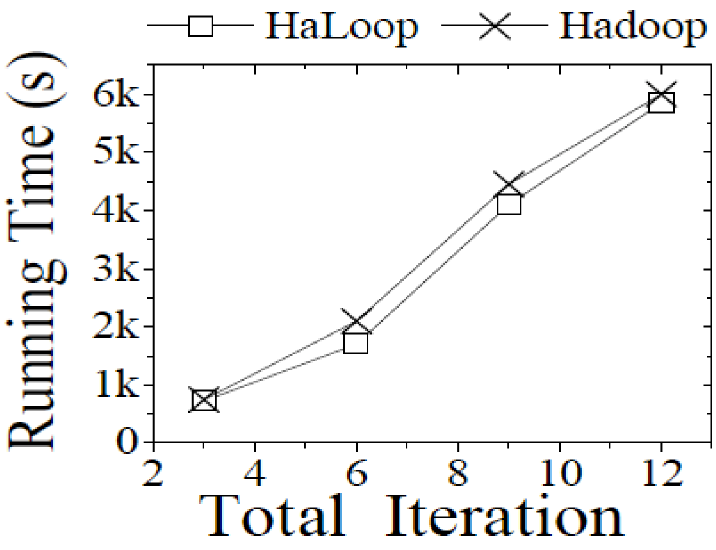
MI: Mapper Input Cache

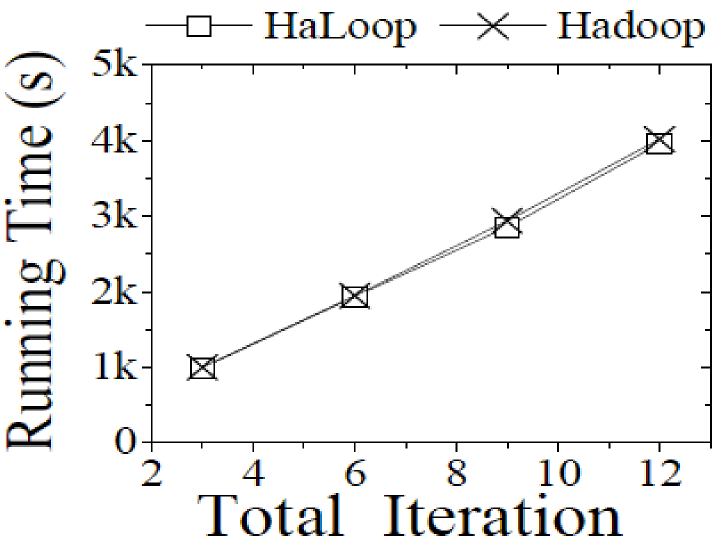
- Provides:
  - Access to non-local mapper input on later iterations
- Used:
  - During scheduling of map tasks
- Assumes:
  1. Mapper input does not change



- PageRank
  - Subsumed by use of Reducer Input Cache
- Transitive Closure
  - Subsumed by use of Reducer Input Cache
- K-means
  - Avoids non-local data reads on iterations > 0

Mapper Input Cache Benefit





5% non-local data reads;
~5% improvement

Conclusions (last slide)

- *Relatively simple changes to MapReduce/Hadoop can support arbitrary recursive programs*
  - TaskTracker (Cache management)
  - Scheduler (Cache awareness)
  - Programming model (multi-step loop bodies, cache control)
- Optimizations
  - Caching loop invariant data realizes largest gain
  - Good to eliminate extra MapReduce step for termination checks
  - Mapper input cache benefit inconclusive; need a busier cluster
- *Future Work*
  - *Analyze expressiveness of **Map Reduce Fixpoint***
  - Consider a model of ***Map (Reduce^{+}) Fixpoint***

Data-Intensive Scalable Science

http://escience.washington.edu

Award IIS 0844572
Cluster Exploratory (CluE)
http://clue.cs.washington.edu

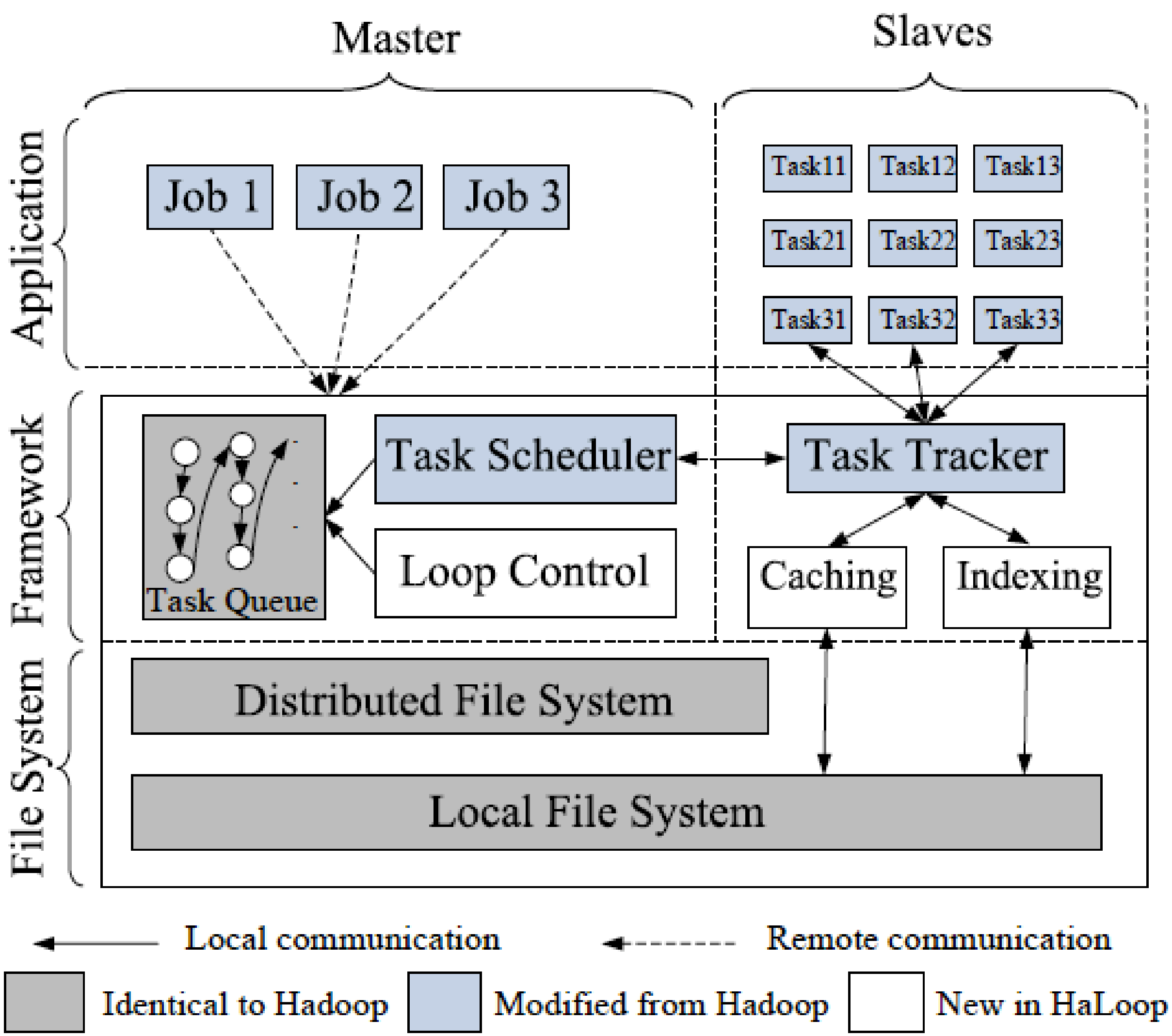
Motivation in One Slide

- MapReduce can't express recursion/iteration
- Lots of interesting programs need loops
  - graph algorithms
  - clustering
  - machine learning
  - recursive queries (CTEs, datalog, WITH clause)
- Dominant solution: Use a driver program outside of mapreduce
- Hypothesis: making MapReduce loop-aware affords optimization
  - …and lays a foundation for scalable implementations of recursive languages

Experiments

- Amazon EC2
  - 20, 50, 90 default small instances
- Datasets
  - Billions of Triples (120GB) [1.5B nodes 1.6B edges]
  - Freebase (12GB) [7M ndoes 154M edges]
  - Livejournal social network (18GB) [4.8M nodes, 67M edges]
- Queries
  - Transitive Closure
  - PageRank
  - k-means

HaLoop Architecture



Scheduling Algorithm

```
Input:  Node node
Global variable: HashMap<Node, List<Parition>> last, HashMaph<Node, List<Partition>> current
if (iteration ==0) {
        Partition part = StandardMapReduceSchedule(node);
        current.add(node, part);
}else{
        if (node.hasFullLoad()) {
                Node substitution = findNearbyNode(node);
                last.get(substitution).addAll(last.remove(node));
                return;
        }
        if (last.get(node).size()>0) {
                Partition part = last.get(node).get(0);
                schedule(part, node);
                current.get(node).add(part);
                list.remove(part);
        }
}
```

The same as MapReduce

Find a substitution

Iteration-local Schedule

Programming Interface

```
Job job = new Job();

job.AddMap(Map Rank, 1);
job.AddReduce(Reduce Rank, 1);
job.AddMap(Map Aggregate, 2);
job.AddReduce(Reduce Aggregate, 2);
```

define loop body

```
job.AddInvariantTable(#1);
```

← **Declare an input as invariant**

```
job.SetInput(IterationInput);
```

← **Specify loop body input, parameterized by iteration #**

```
job.SetFixedPointThreshold(0.1);
job.SetDistanceMeasure(ResultDistance);
job.SetMaxNumOfIterations(10);
```

Termination condition

```
job.SetReducerInputCache(true);
job.SetReducerOutputCache(true);
```

Turn on caches

```
job.Submit();
```

Cache Infrastructure Details

- Programmer control
- Architecture for cache management
- Scheduling for *inter-iteration locality*
- Indexing the values in the cache

Other Extensions and Experiments

- Distributed databases and Pig/Hadoop for Astronomy [IASDS 09]
- Efficient "Friends of Friends" in Dryad [SSDBM 2010]
- SkewReduce: Automated skew handling [SOCC 2010]
- Image Stacking and Mosaicing with Hadoop [Hadoop Summit 2010]
- HaLoop: Efficient iterative processing with Hadoop [VLDB2010]

MapReduce Broadly Applicable

- Biology
  - [Schatz 08, 09]
- Astronomy
  - [IASDS 09, SSDBM 10, SOCC 10, PASP 10]
- Oceanography
  - [UltraVis 09]
- Visualization
  - [UltraVis 09, EuroVis 10]

Key idea

- When the loop output is large…
  - transitive closure
  - connected components
  - PageRank (with a convergence test as the termination condition)
- …need a distributed fixpoint operator
  - typically implemented as yet another MapReduce job -- on every iteration

Background

- Why is MapReduce popular?
  - Because it's fast?
  - Because it scales to 1000s of commodity nodes?
  - Because it's fault tolerant?
- Witness
  - MapReduce on GPUs
  - MapReduce on MPI
  - MapReduce in main memory
  - MapReduce on <10 nodes

So why is MapReduce popular?

- The programming model
  - Two serial functions, parallelism for free
  - Easy and expressive
- Compare this with MPI
  - 70+ operations
- But it can't express recursion
  - graph algorithms
  - clustering
  - machine learning
  - recursive queries (CTEs, datalog, WITH clause)

Fixpoint

- A fixpoint of a function *f* is a value *x* such that $f(x) = x$
- The fixpoint queries FIX can be expressed with the relational algebra plus a *fixpoint operator*
- Map - Reduce - Fixpoint
  - hypothesis: sufficient model for all recursive queries