HaLoop: Efficient Iterative Data Processing On Large Clusters

Yingyi Bu, UC Irvine

Bill Howe, UW

Magda Balazinska, UW

Michael D. Ernst, UW

UCIRVINE

http://isg.ics.uci.edu

Horizon

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

QuickTime and a decompressor are needed to see this picture.

eScience Institute

http://escience.washington.edu/

May-2011, UC Berkeley Cloud Computing Seminar

Outline

- Motivation
- Caching & scheduling
- Fault-tolerance
- Programming model
- Related work
- Conclusion
- Cloud Computing Projects in UCI

Motivation

- 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
  - lays a foundation for scalable implementations of recursive languages

Example 1: PageRank

Rank Table R_0

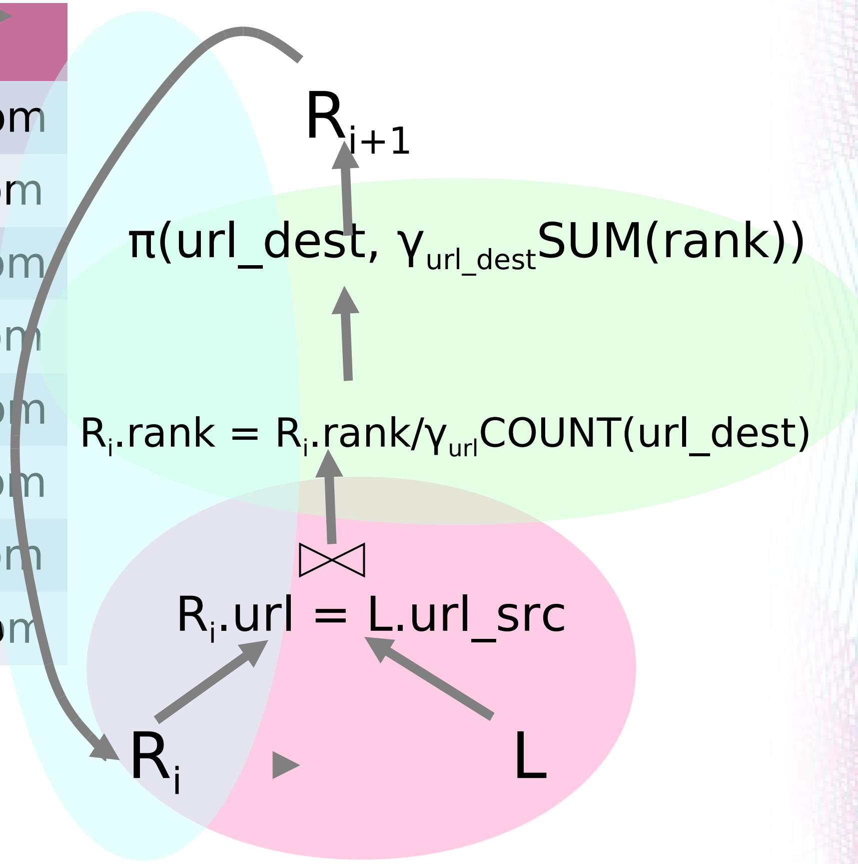
url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

Rank Table R_3

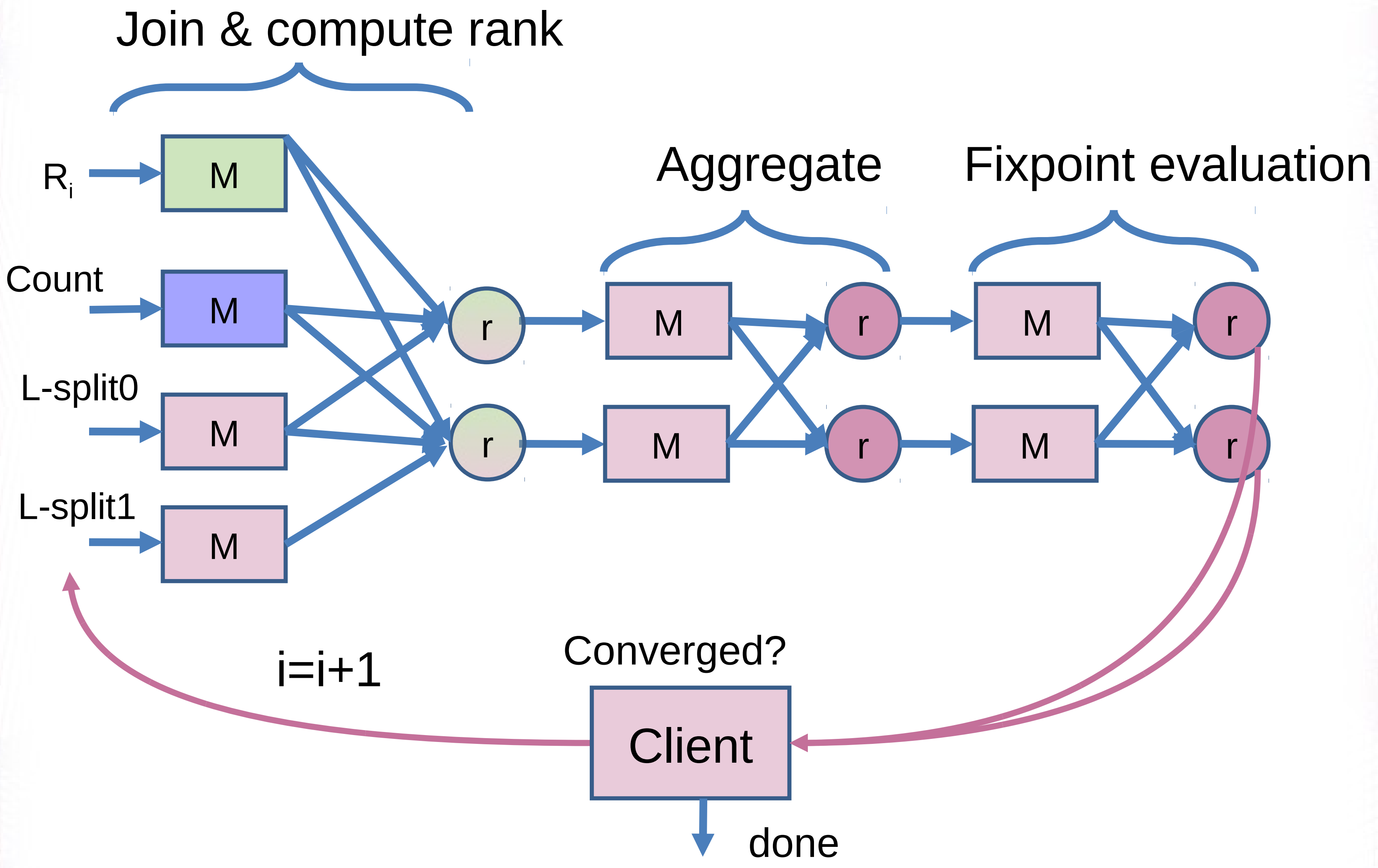
url	rank
www.a.com	2.13
www.b.com	3.89
www.c.com	2.60
www.d.com	2.60
www.e.com	2.13

Linkage Table *L*

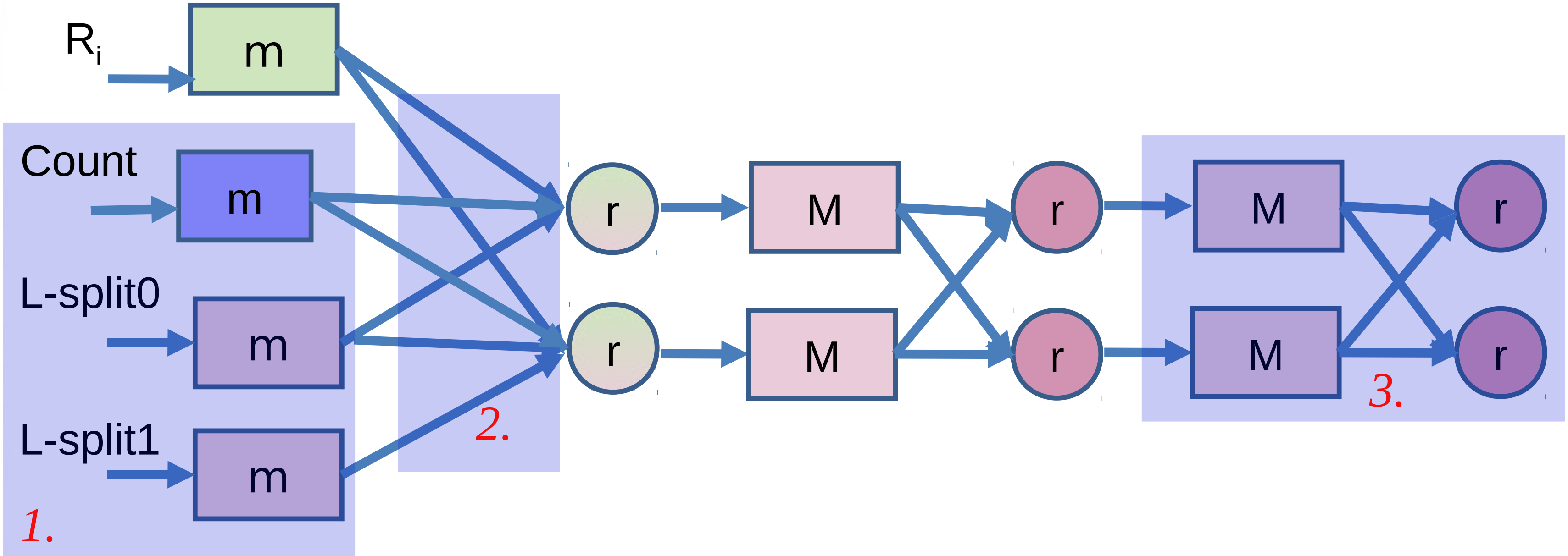
url_src	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.c.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.com
www.a.com	www.d.com



PageRank Implementation on MapReduce



What's the problem?



L and Count are loop invariants, but

1. *They are loaded on each iteration*
2. *They are shuffled on each iteration*
3. *Also, 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

(semi-naïve evaluation)

Find all transitive friends of Eric

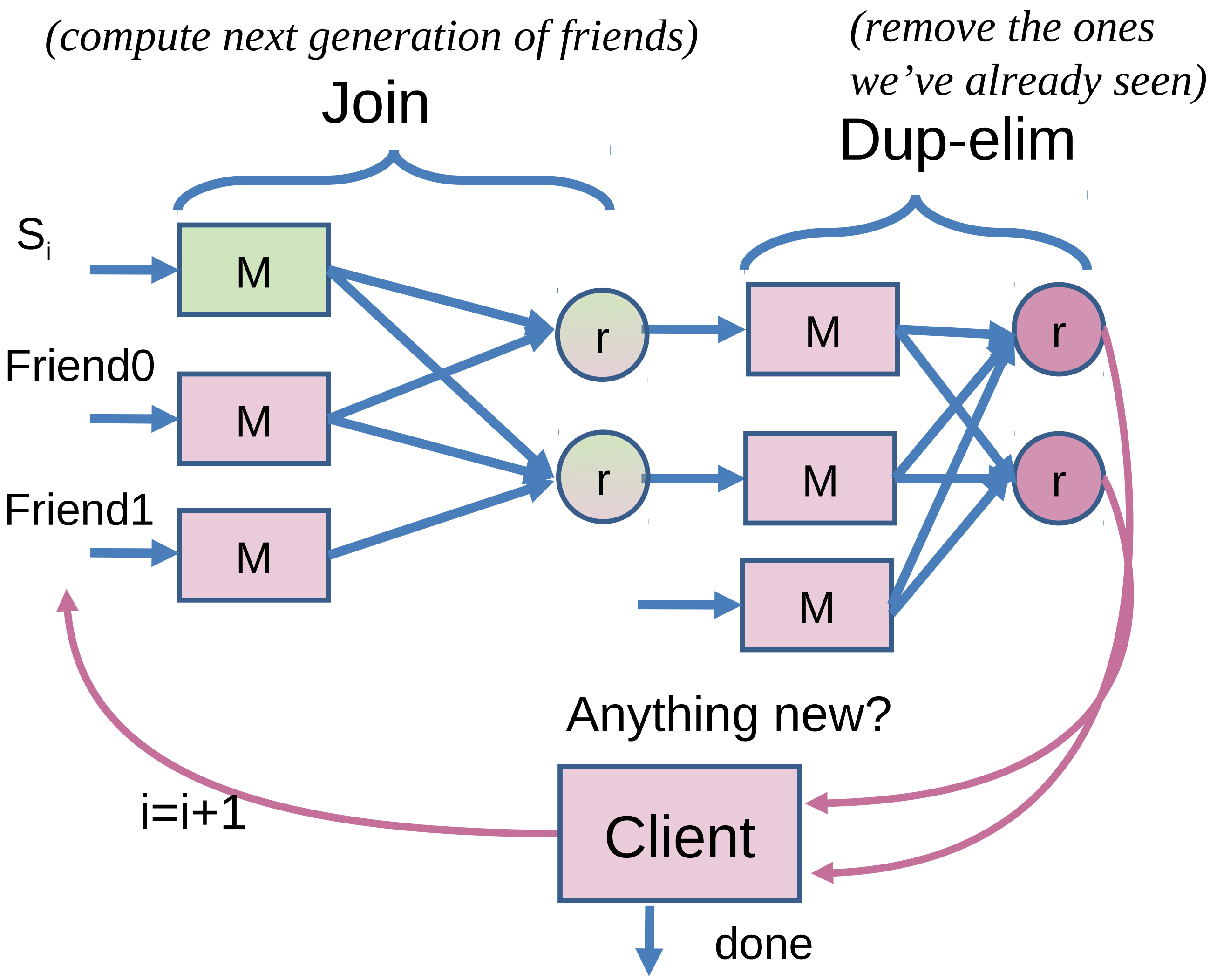
R_0 {Eric, Eric}

R_1 {Eric, Elisa}

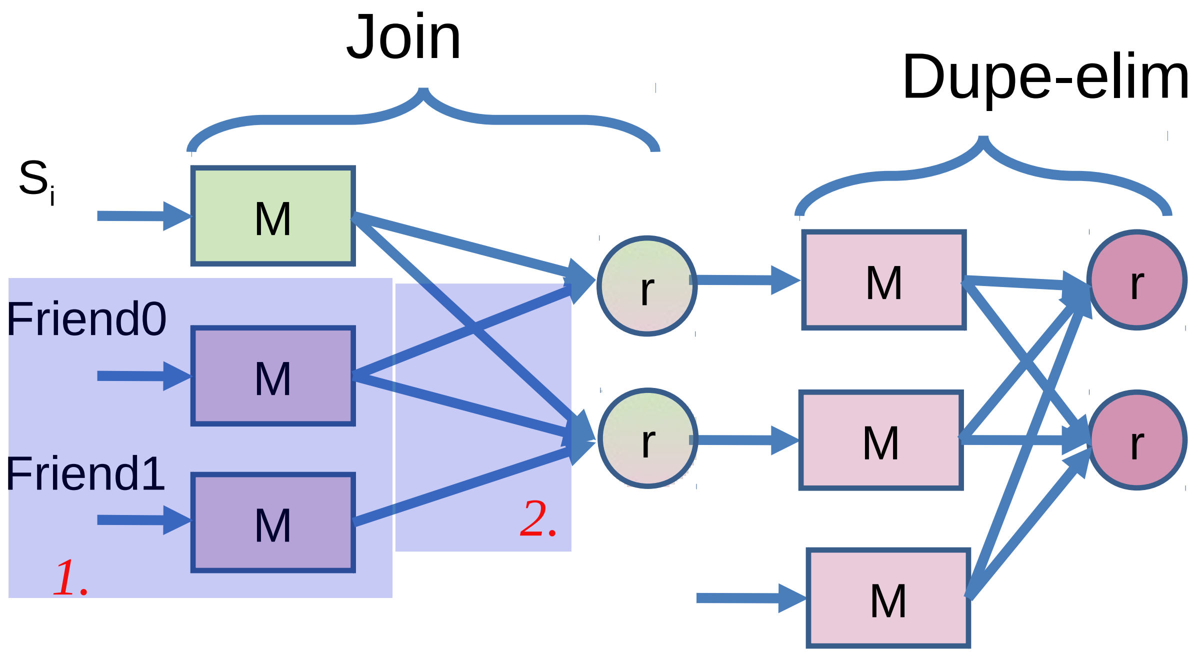
R_2 {Eric, Tom
Eric, Harry}

R_3 {}

Transitive Closure on MapReduce



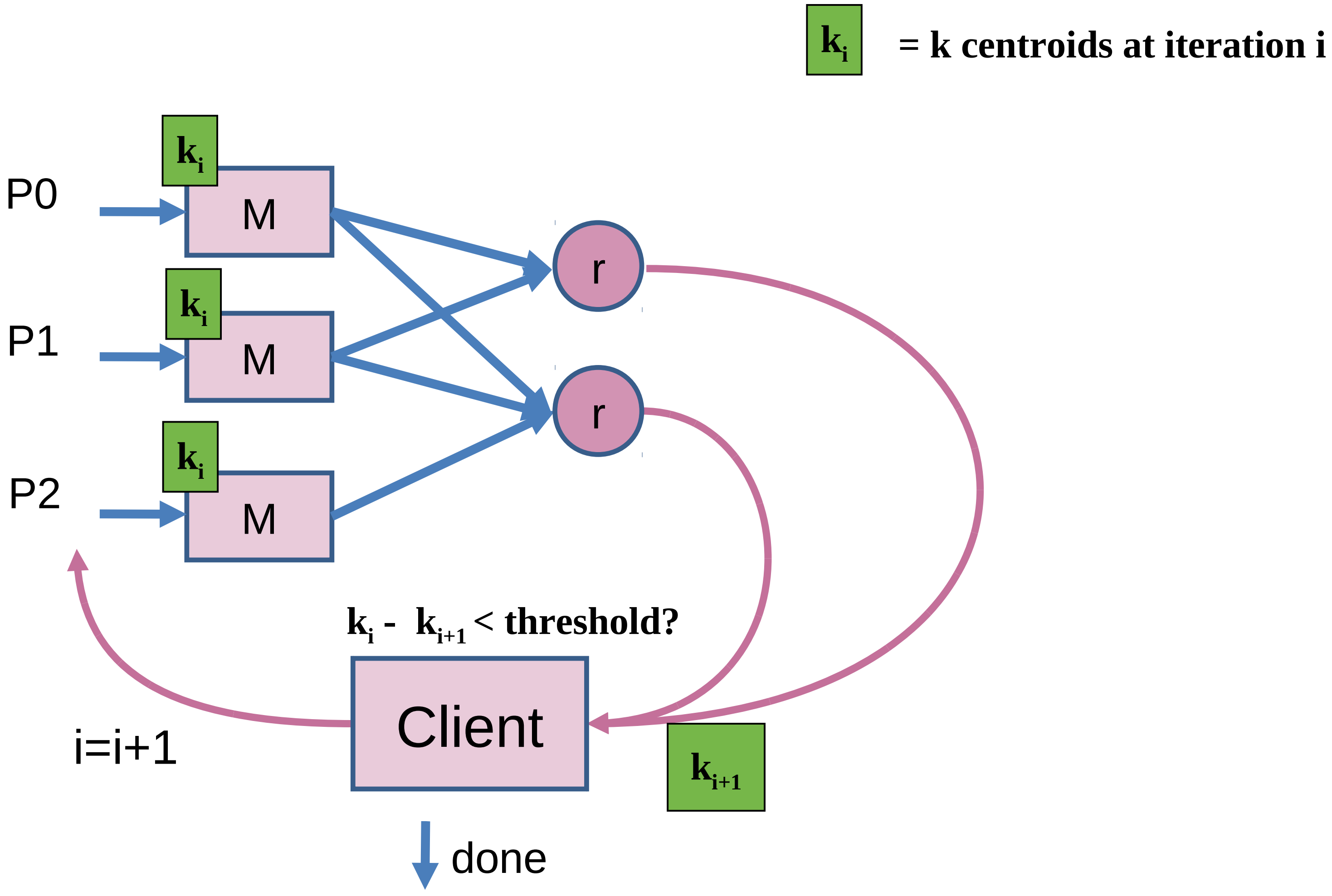
What's the problem?



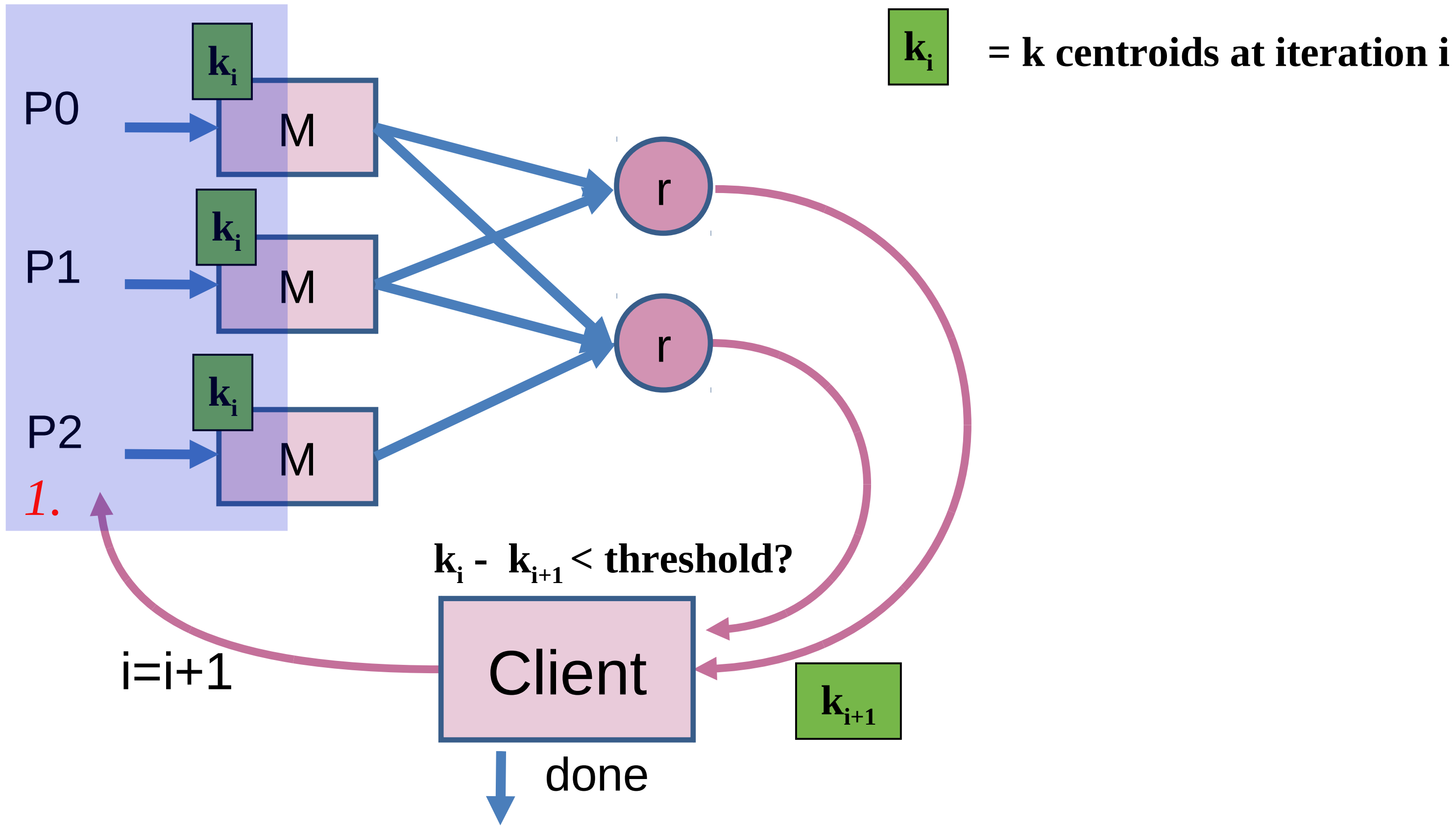
Friend is loop invariant, but

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

Example 3: k-means



What's the problem?



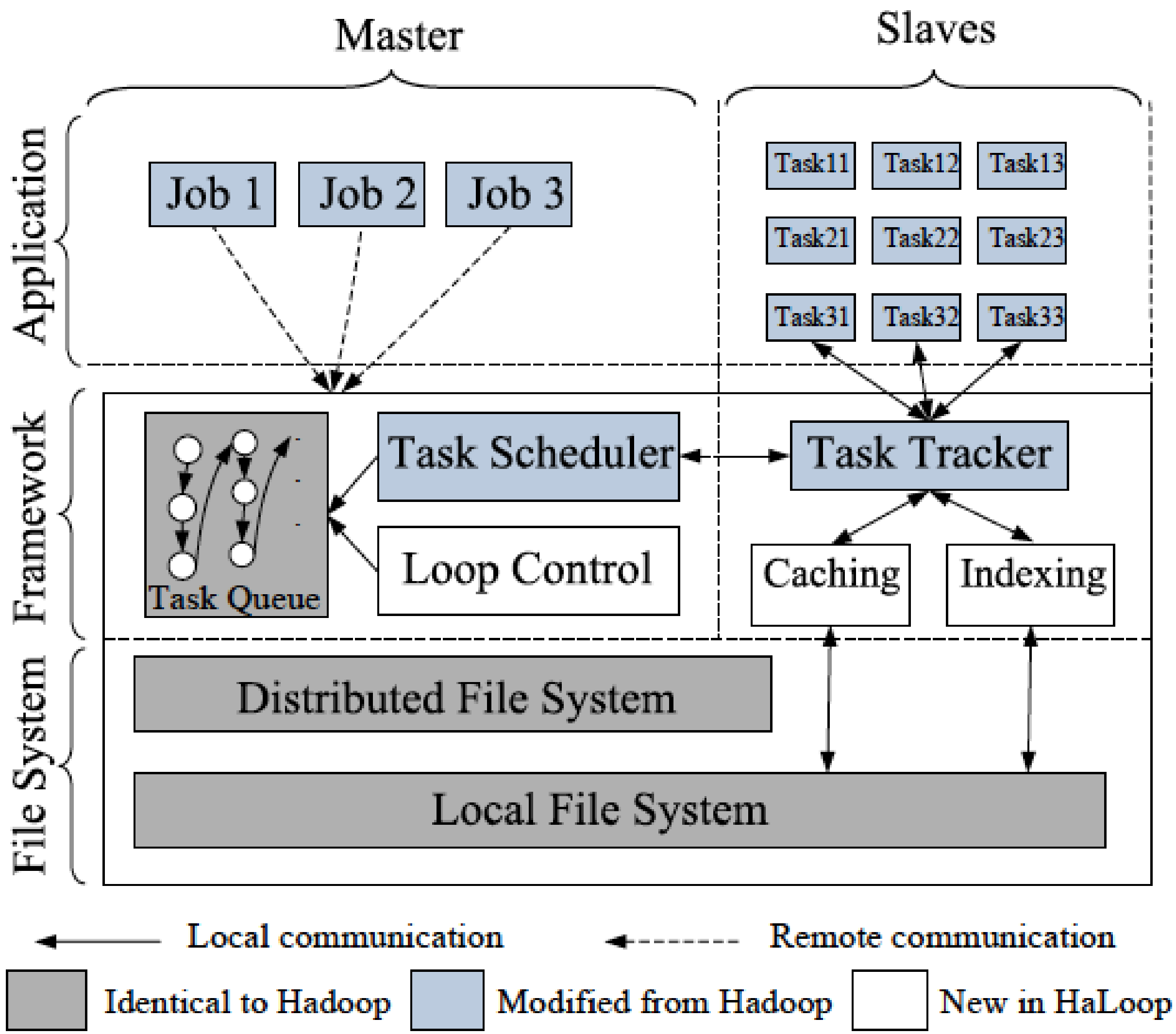
P is loop invariant, but

1. *P is loaded on each iteration*

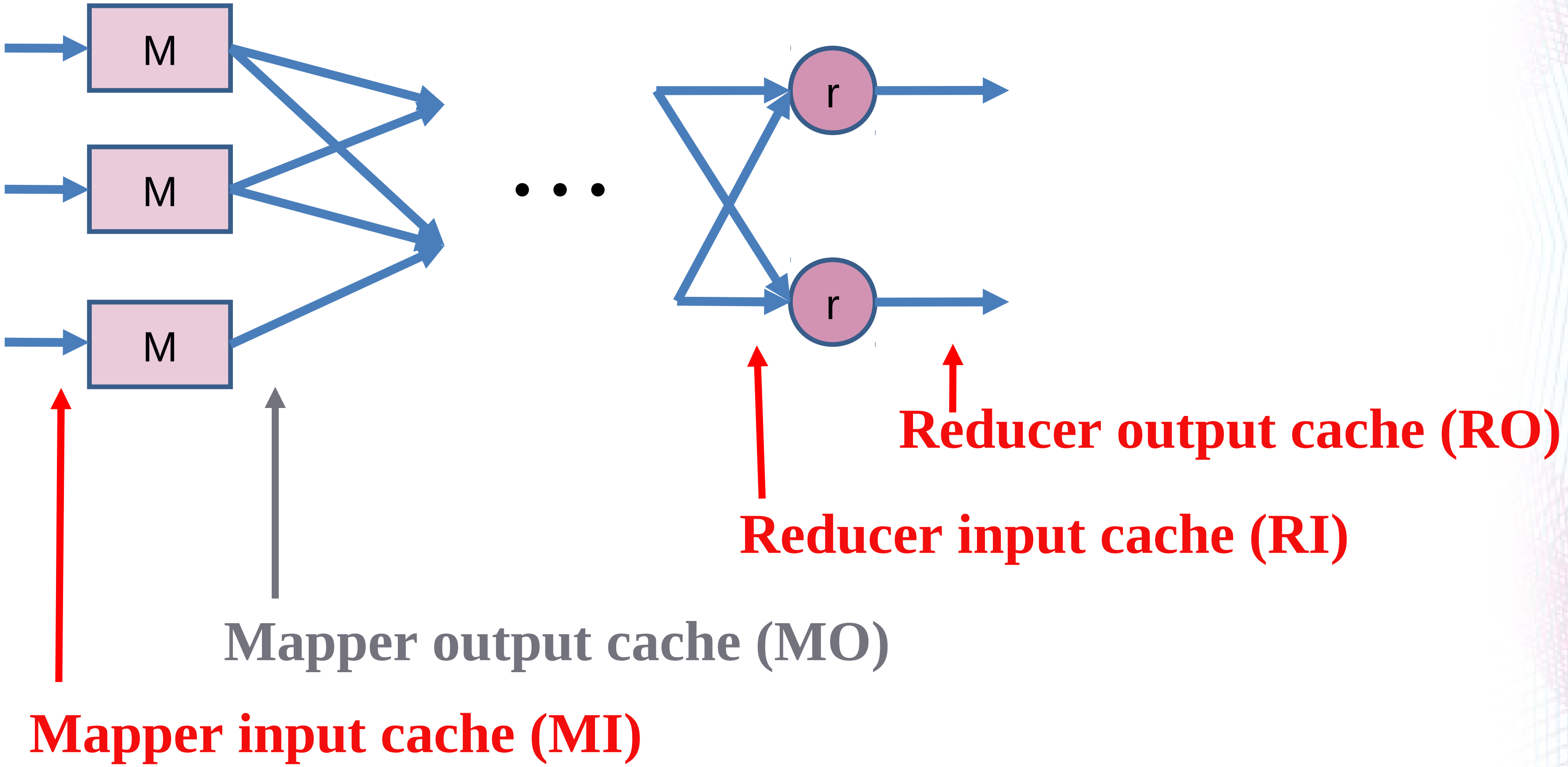
Push loops into MapReduce!

- Architecture
- Cache loop-invariant data
- Scheduling
- Fault-tolerance
- Programming Model

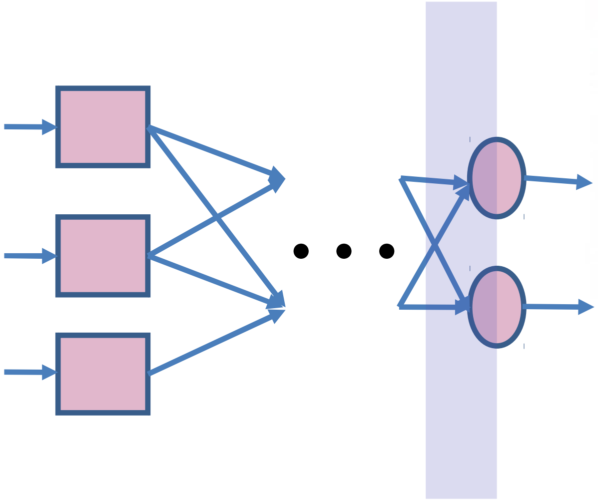
HaLoop Architecture



Inter-iteration caching



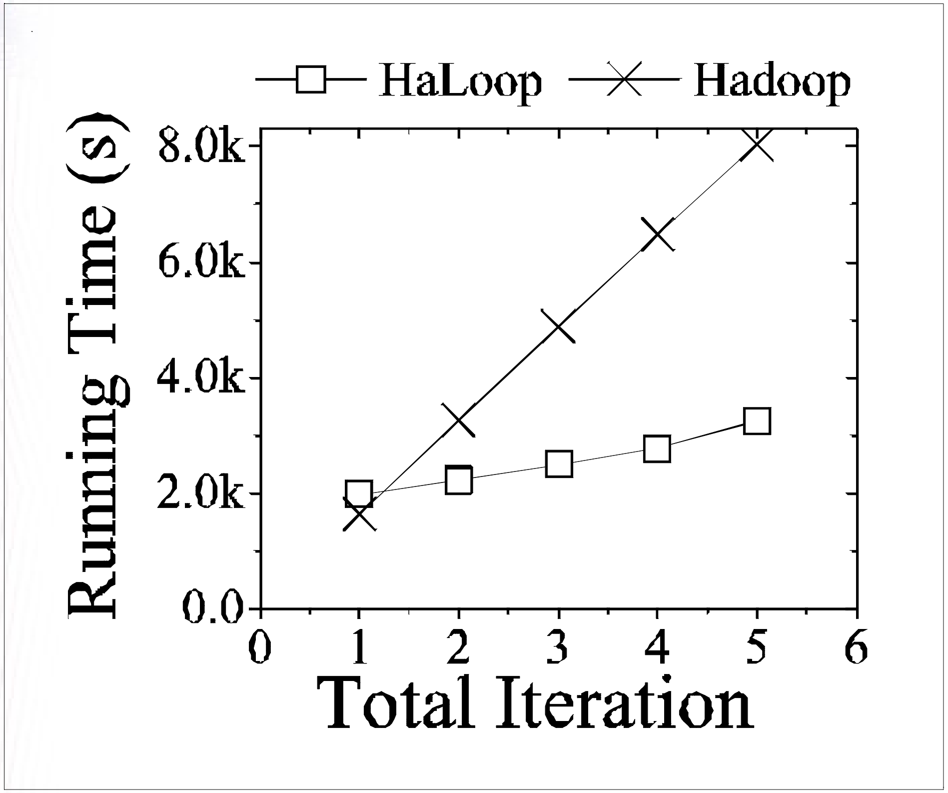
RI: Reducer Input Cache



- Provides:
  - Access to loop invariant data without map/shuffle
- Data:
  - Reducer function
- Assumes:
  1. Static partitioning (implies: no new nodes)
  2. Deterministic mapper implementation

- PageRank
  - Avoid loading and shuffling the web graph at every iteration
- Transitive Closure
  - Avoid loading and shuffling the friends graph at every iteration
- K-means
  - No help

Reducer Input Cache Benefit



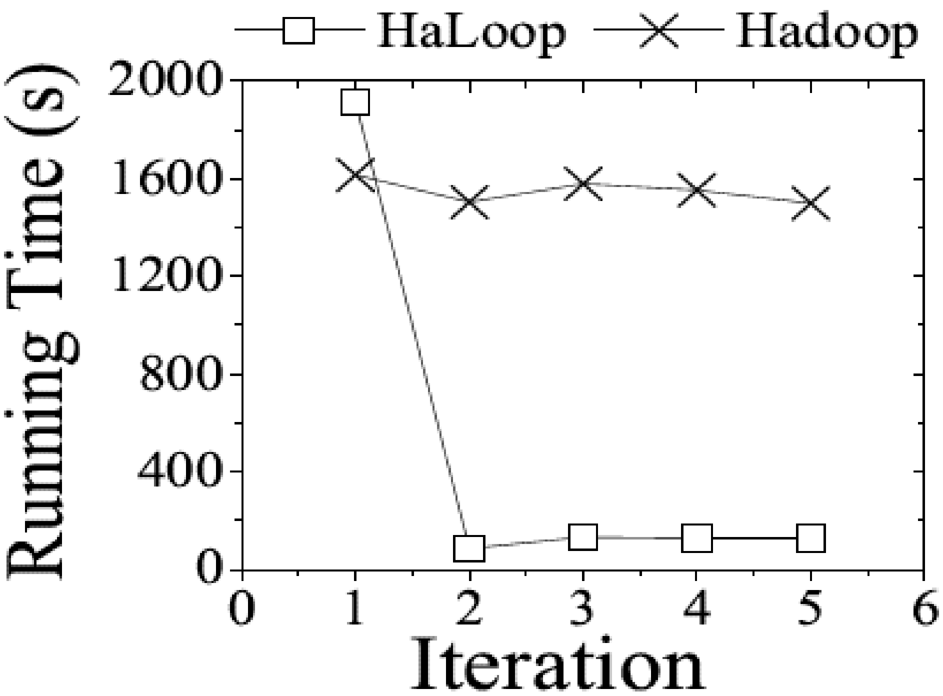
Overall run time

Friends-of-friends query

Billion Triples Dataset (120GB)

90 small instances on EC2

Reducer Input Cache Benefit

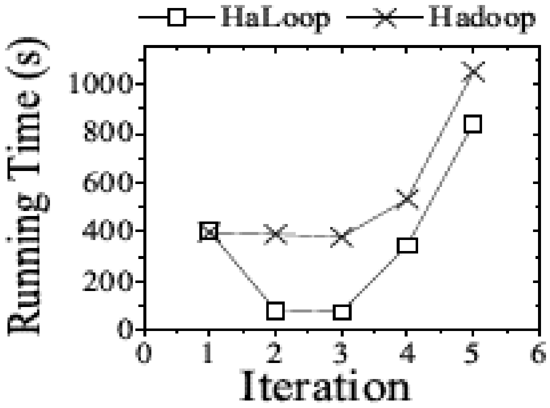


Friends-of-Friends query

Billion Triples Dataset (120GB)

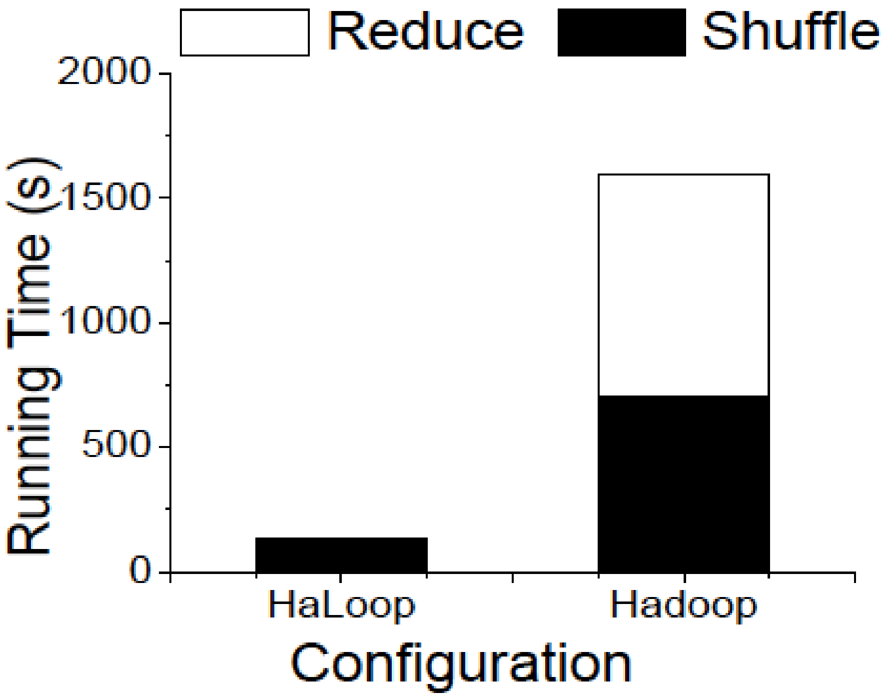
90 small instances on EC2

Livejournal, 12GB



Join step only

Reducer Input Cache Benefit

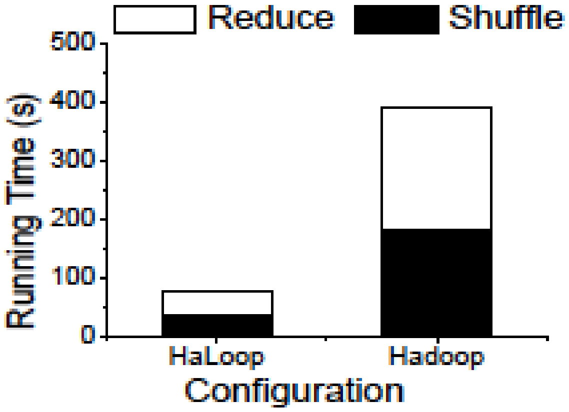


Friends-of-friends query

Billion Triples Dataset (120GB)

90 small instances on EC2

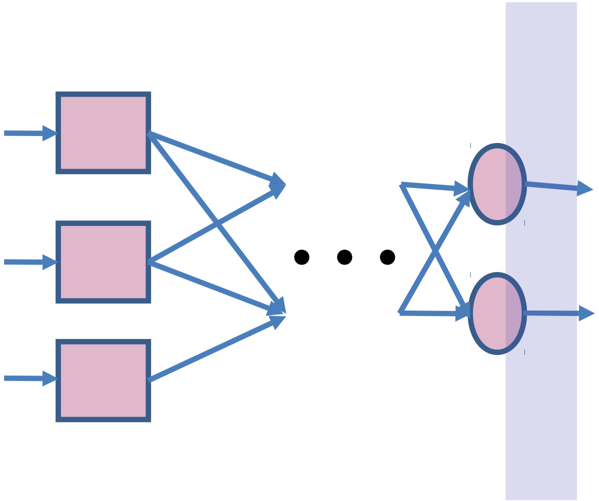
Livejournal, 12GB



Reduce and Shuffle of Join Step

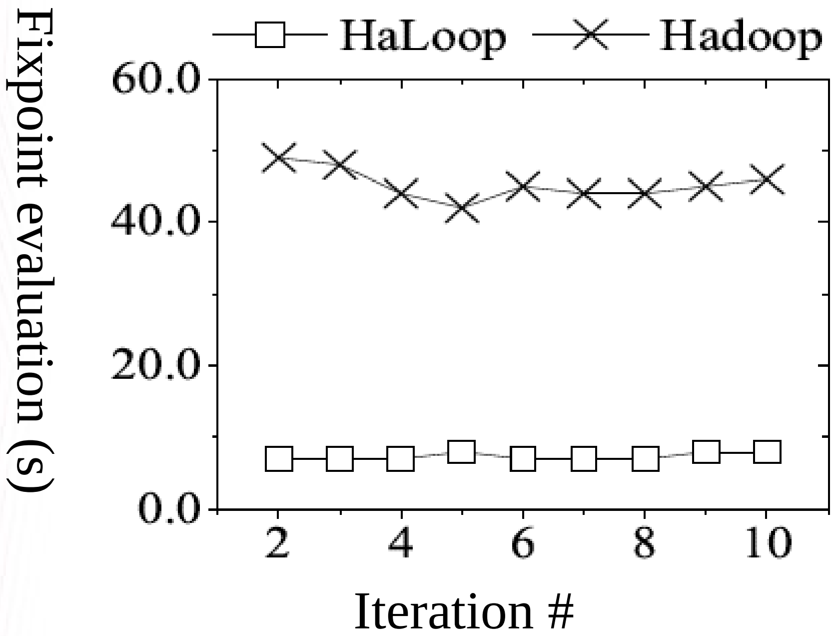
RO: Reducer Output Cache

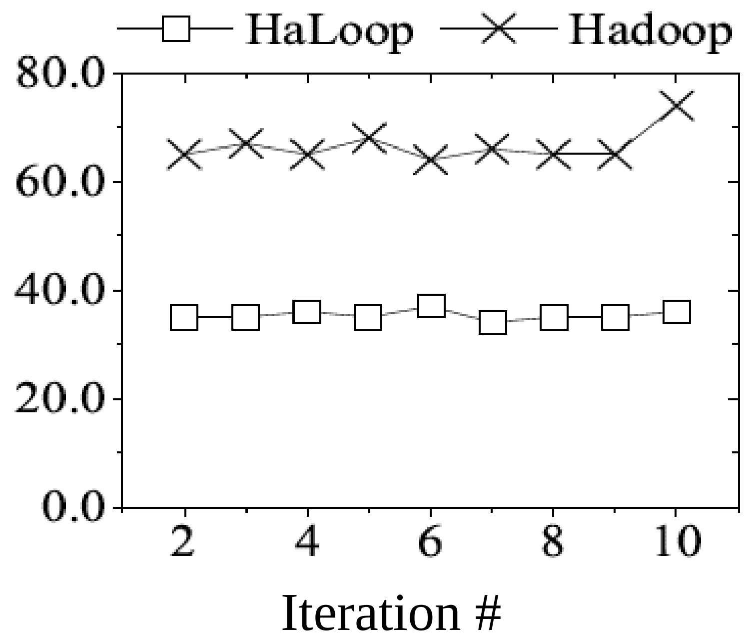
- Provides:
  - Distributed access to output of previous iterations
- Used By:
  - Fixpoint evaluation
- Assumes:
  1. Partitioning constant across iterations
  2. Reducer output key functionally determines Reducer input key



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

Reducer Output Cache Benefit





Livejournal dataset

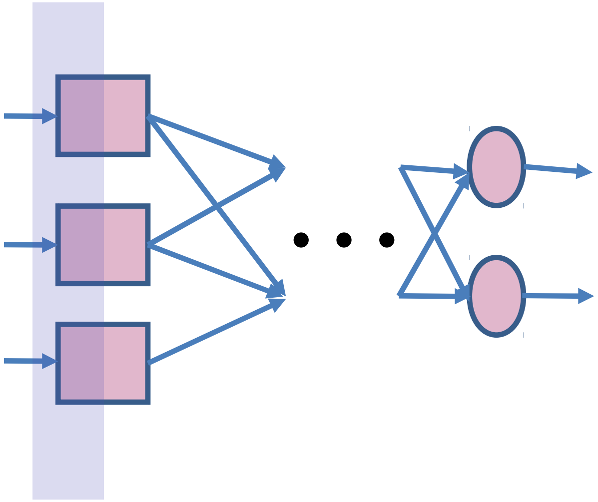
50 EC2 small instances

Freebase dataset

90 EC2 small instances

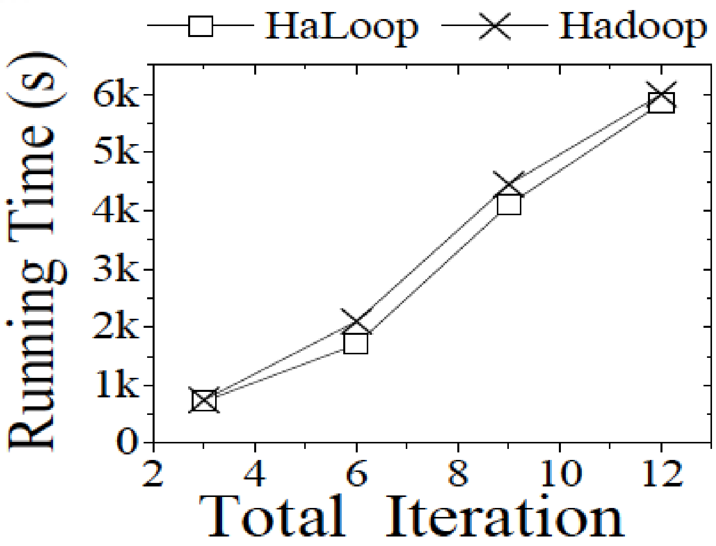
MI: Mapper Input Cache

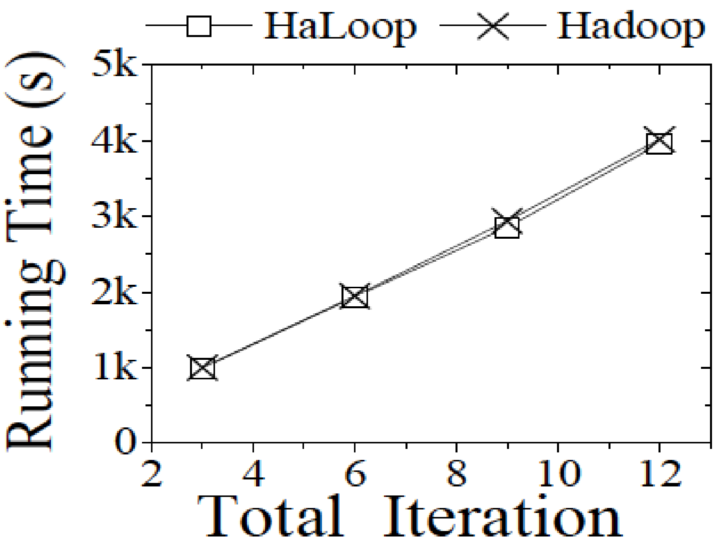
- Provides:
  - Access to non-local mapper input on later iterations
- Data for:
  - Map function
- Assumes:
  1. Mapper input does not change



- PageRank
  - No help
- Transitive Closure
  - No help
- K-means
  - Avoids non-local data reads on iterations > 0

Mapper Input Cache Benefit





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

However, Facebook has 70% non-local data reads!!

Loop-aware Task Scheduling

Input: Node node, int iteration

Global variable: HashMap<Node, List<Parition>> last, HashMaph<Node, List<Partition>> current

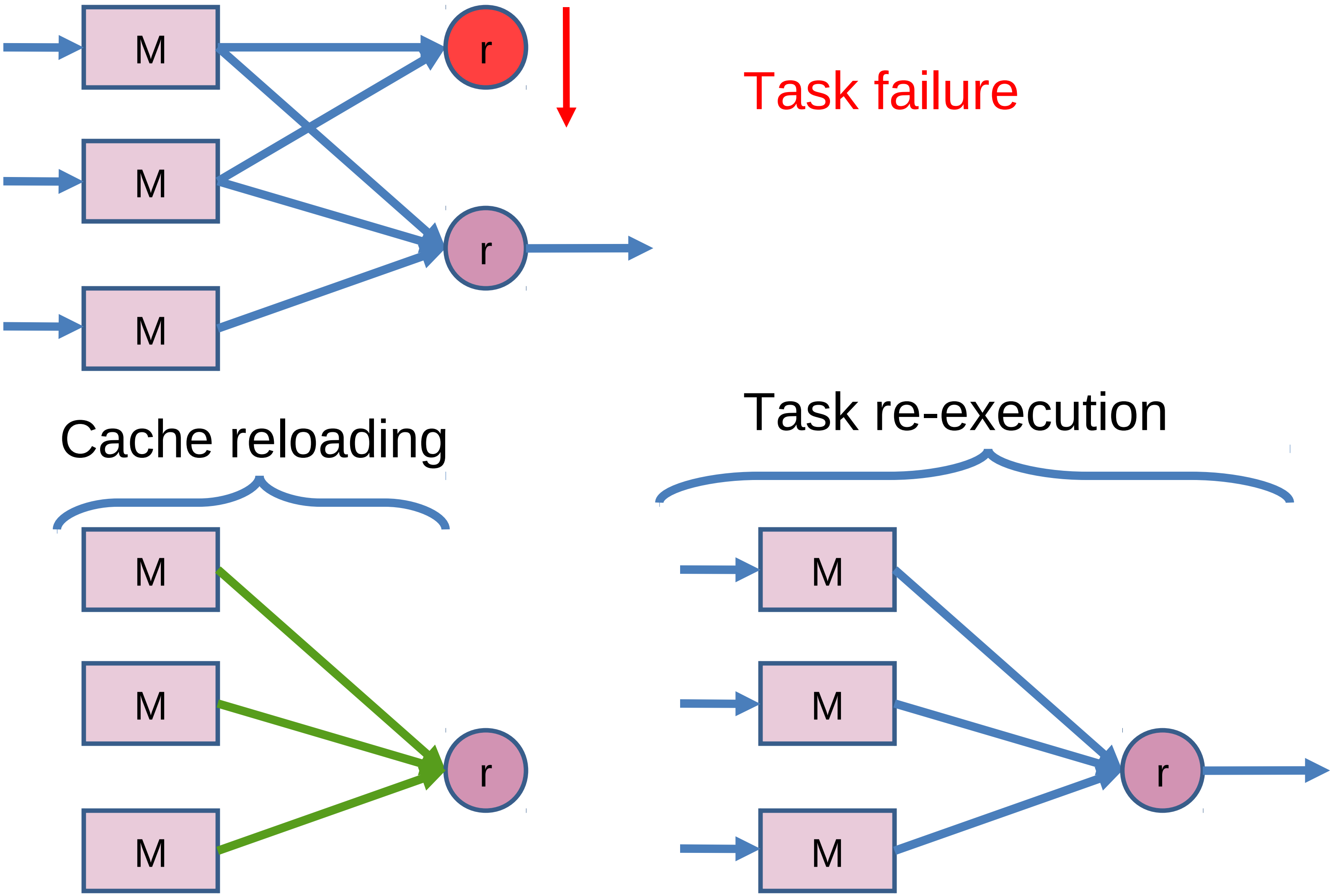
```
if (iteration ==0) {
        Partition part = StandardMapReduceSchedule(node);
        current.add(node, part);
}else{
        if (node.hasFullLoad()) {
                Node substitution = findNearbyNode(node);
                last.get(substitution).addAll(last.remove(node));
                return;
        }
        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

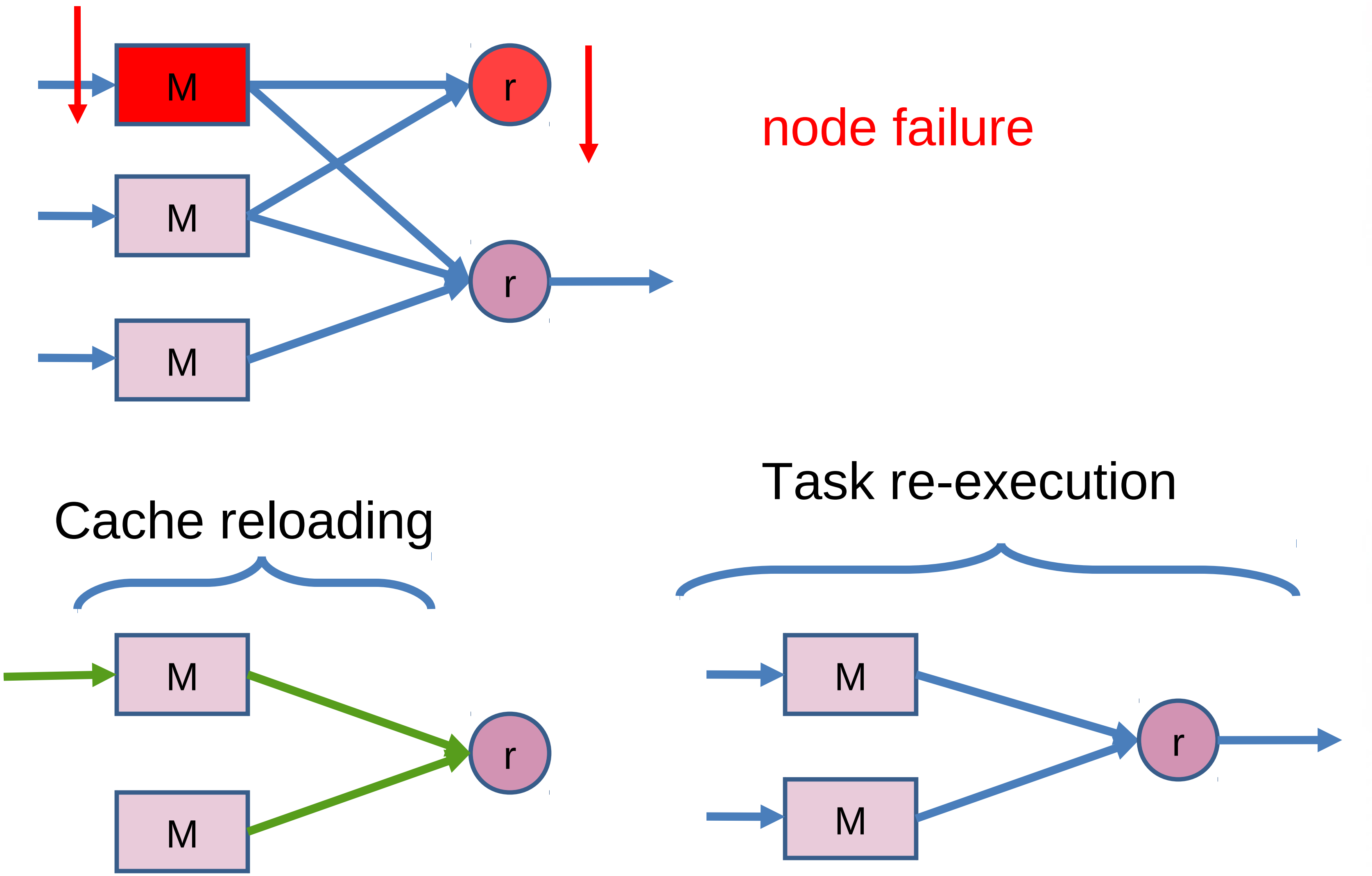
Find a substitution

Iteration-local Schedule

Fault-tolerance (task failures)



Fault-tolerance (node failures)



Programming Model

- Mapper/reducer stay the same!
- Touch points
  - Input/Output: for each <iteration, step>
  - Cache filter: which tuple to cache?
  - Distance function: optional
- Nested job containing child jobs as loop body
- Minimize extra programming efforts

Related Work: Twister [Ekanayake HPDC 2010]

- Pipelining mapper/reducer
- 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 SIGMOD 2010]

- 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: BOOM [Alvaro EuroSys 10]

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

Conclusions

- *Relatively simple changes to MapReduce/Hadoop can support iterative/recursive programs*
  - TaskTracker (Cache management)
  - Scheduler (Cache awareness)
  - Programming model (multi-step loop bodies, cache control)
- Optimizations
  - Caching reducer input realizes the largest gain
  - Good to eliminate extra MapReduce step for termination checks
  - Mapper input cache benefit inconclusive; need a busier cluster
- *Future Work*
  - *Iteration & Recursion on top of Hyracks core!*

Hyracks [Borkar et al., ICDE'11]

- Partitioned-Parallel Platform for data-intensive computing
  - Flexible (DAGs, location constraints)
  - Extensible (“micro-kernel” core, online-aggregation plugin (VLDB'11), B-tree plugin, R-tree plugin, Dataflow plugin...), **an iteration/recursion plugin?**
- Jobs
  - Dataflow DAG of operators and connectors
  - Can set location constraints
  - Can use a library of operators: joins, group-by, sort and so on
- V.s. “competitors”
  - V.s. Hadoop: more flexible model and less pressimistic
  - V.s. Dryad: support data as first class citizens

Questions?