MapReduce Optimizations and Algorithms

2015

Professor Sasu Tarkoma

Optimizations

Reduce tasks cannot start before the whole map phase is complete

Thus single slow machine can slow down the whole process

Master can execute many redundant map tasks and then use the results of the first task to complete

Optimizations: Combining Phase

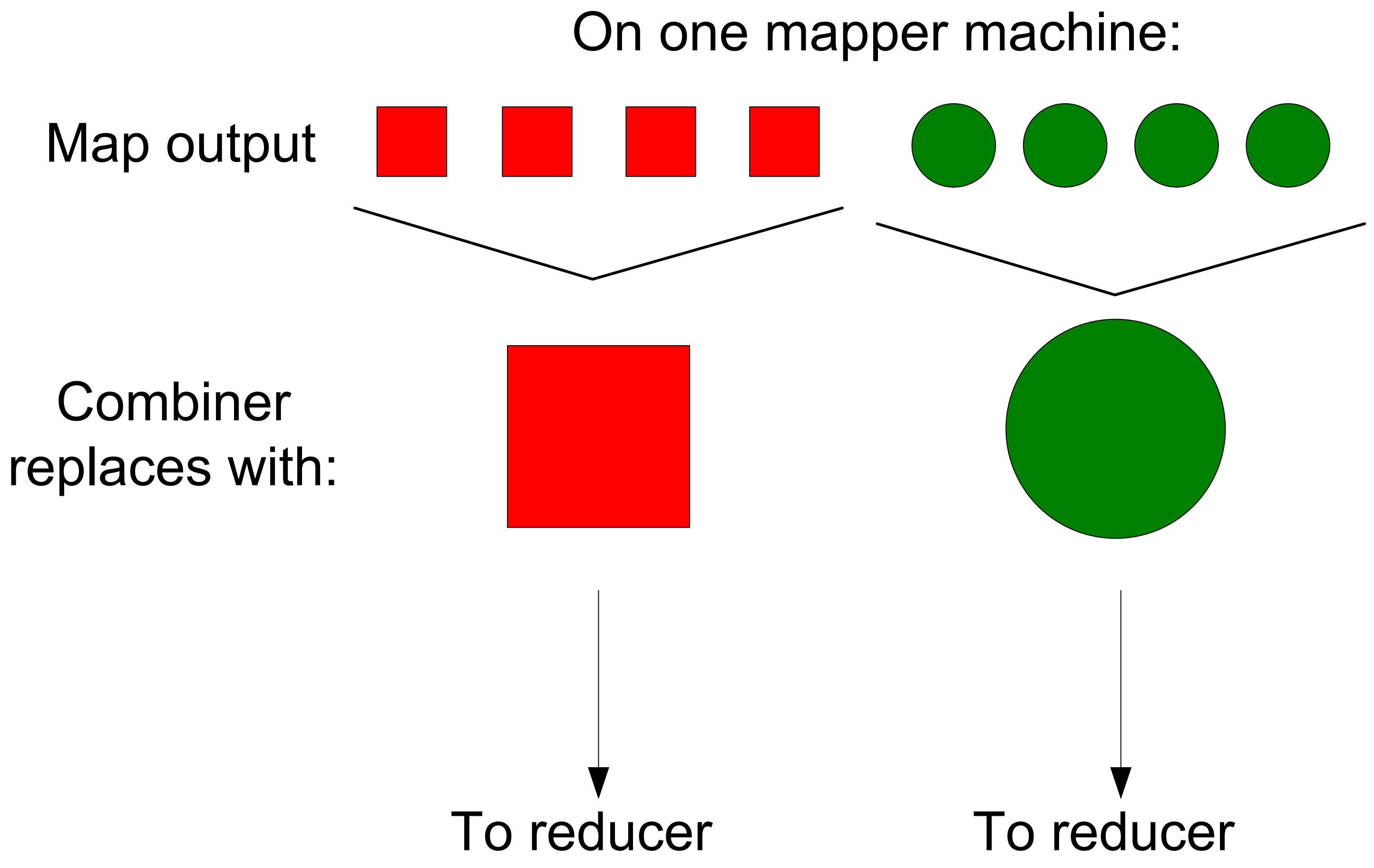
Performance can be increased by running a mini reduce phase on local map output

Executed on mapper nodes after map phase

Saves bandwidth before sending data to a full reducer

Reducer can be a combiner if it is commutative and associative

Combiner



Source: https://courses.cs.washington.edu/courses/cse490h/08au/lectures/mapred.pdf

Partitioner

Partitioner divides the **intermediate key space**

Assigns intermediate key-value pairs to reducers

Thus **n partitions results in n reducers**

Between map and reduce phases:

- data is shuffled: parallel-sorted and exchanged
- data is moved to the correct shard for reducing
- partition function accepts the key and the number of reducers and then returns the index of the reducer

Supports load balancing

MapReduce Summary

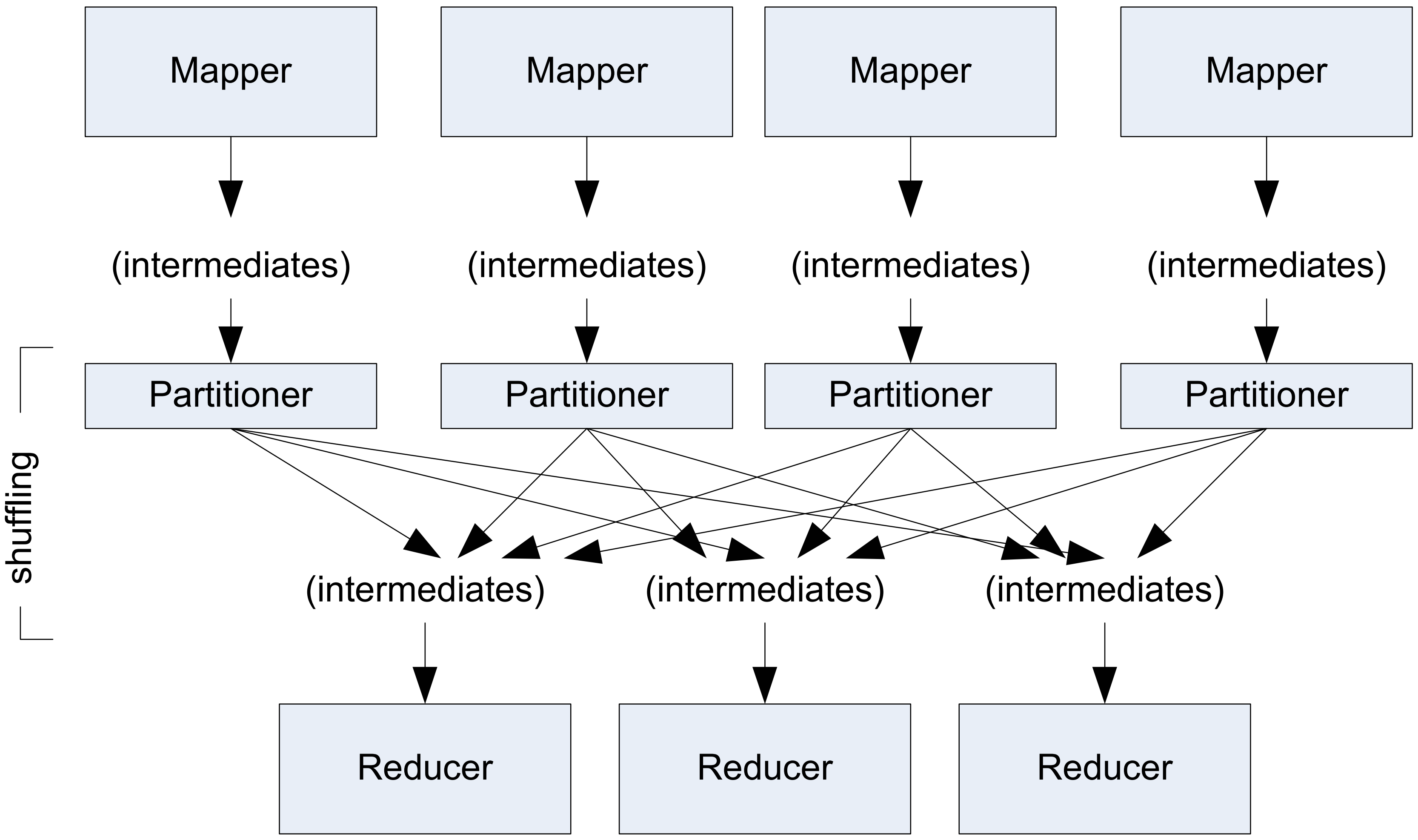
Two key functions that need to be implemented:

- **map** (in_key, in_value) → (out_key, intermediate_value) list
- **reduce** (out_key, intermediate_value list) → out_value list

With two optimizations:

- **combine** (key, intermediate_value list) → intermediate_out_value list
- **partition** (key, number of partitions) → partition for key

Partition and Shuffle



Synchronization

Intermediate key-value pairs must be grouped by key through a distributed sort

Shuffle and sort

A job with m mappers and r reduces involves up to m * r different copy operations

Each mapper may have intermediate output destined to every reducer

Data management as part of MapReduce

Purlieus and CAM

Data management as part of MapReduce

Need to know task types (map/reduce intensive)

Map intensive jobs benefit from locality awareness

Up to 80% reduction in execution time

Key algorithms for MapReduce

Inverted Index

Statistics

Sorting

Searching

K-Means

Transitive closure

PageRank

Advanced algorithms

Filtering Algorithms

Finding files or items with specific characteristics

Searching for patterns in web logs or files

Filtering is mostly done in the map phase

Reduce can be simply the identity

Search

Mapper key is the file name + line number

Mapper value is the contents of the file

Search pattern is a special parameter

Mapper:

- Input: (filename, *text*) and *pattern*
- If *text* matches *pattern* output (filename, _)

Reducer:

- Identity function

Optimization:

- Mark file only once
- Use a **combiner function** to collapse (filename, _) pairs into one
  - Alleviates I/O issues

Aggregation Algorithms

Computing the minimum, maximum, sum, average, ... of the given values

Count the number of Tweets per day

Map is simple or the identity, reduce is doing most of the work

Sorting

- The Mapper output is automatically sorted by keys
- Reducer plays a crucial role in sorting
- Terasort Benchmark
  - http://sortbenchmark.org/YahooHadoop.pdf
  - (key, value) pairs are handled in order by key and sent to a specific reducer based on hash(key)
  - Hash function must be chosen so that
  - **k1 < k2 => hash(k1) < hash(k2)**
  - If we have a single reducer, the output is sorted
  - If we have multiple reducers, we get partially sorted results: last-stage merge of the interim results

http://hadooptutorial.wikispaces.com/Sorting+feature+of+MapReduce

TeraSort

TeraSort is a standard map/reduce sort

A custom partitioner that uses a sorted list of N − 1 sampled keys that define the key range for each reduce.

In particular, all keys such **that sample[i − 1] <= key < sample[i]** are sent to reduce **i**.

This guarantees that the output of reduce **i** are all less than the output of reduce **i+1**.

To speed up the partitioning, the partitioner builds a two level trie that quickly indexes into the list of sample keys based on the first two bytes of the key.

TeraSort generates the sample keys by sampling the input **before** the job is submitted and writing the list of keys into HDFS.

Iterative MapReduce Algorithms

K-Means Clustering Algorithm

Iterative algorithm that is run until it converges

1. K initial points (centers) are chosen at random.
2. K clusters are formed by associating every data point (observation) with the nearest center.
3. For each cluster, recompute the centers (determine centroid)
4. Repeat from 2 until convergence.

Source: Riccardo Torlone. Analytics on Big Data. Universita Roma Tre.

K-Means for MapReduce

Map phase

- Each map reads the K centroids and a block from the input dataset
- Each point is assigned to the closest centroid
- **Output: <centroid, point>**

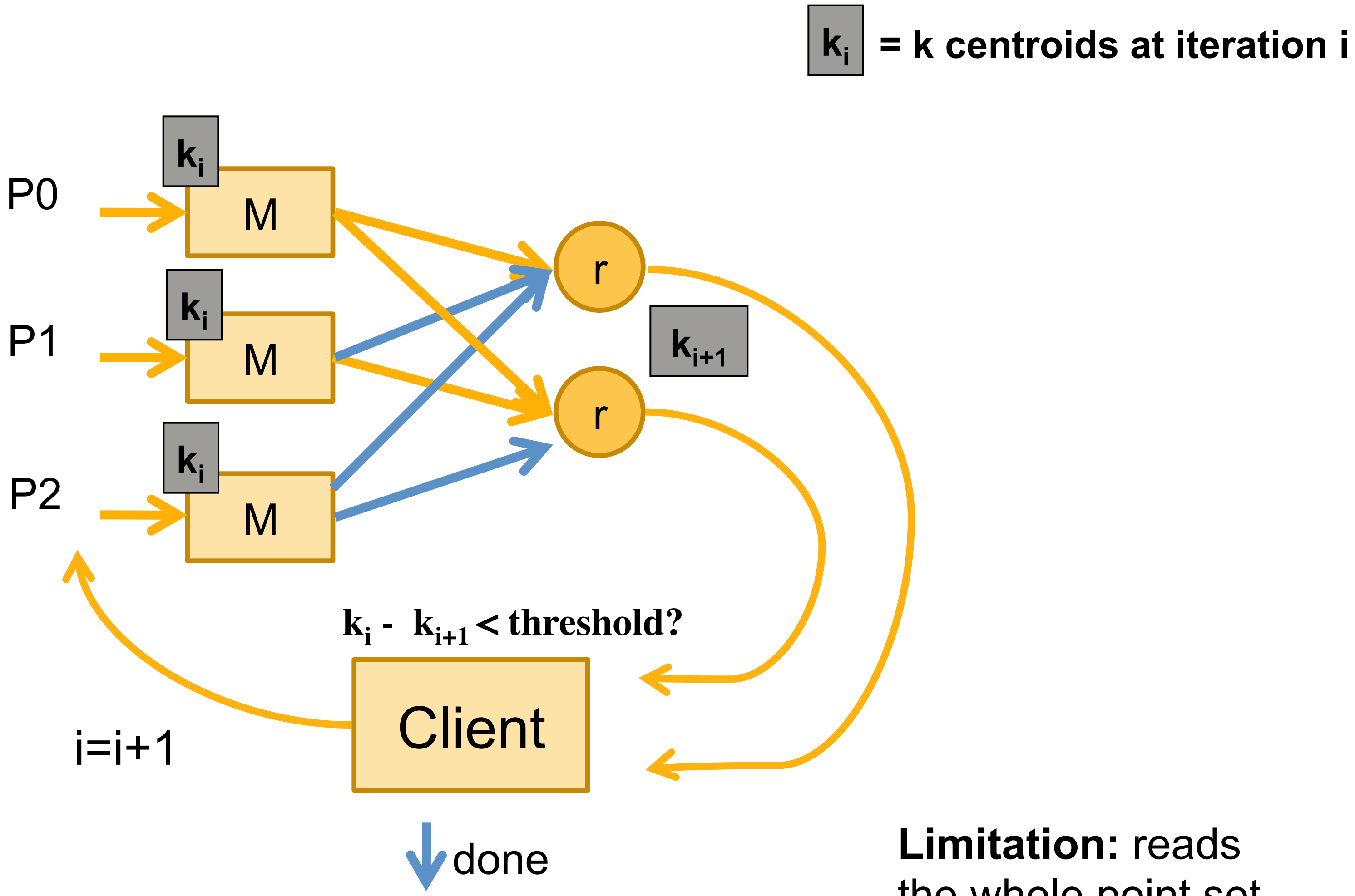
Reduce phase

- Obtain all points for a given centroid
- Recompute the new centroid
- **Output: <new centroid>**

Iteration:

- Compare the old and new set of K centroids
- If they are similar then Stop
- Else Start another iteration unless maximum of iterations has been reached.

MapReduce K-Means



k_i = k centroids at iteration i
P0
P1
P2
k_i
M
r
k_{i+1}
$k_i - k_{i+1} <$ threshold?
Client
i=i+1
done

Limitation: reads the whole point set P at each iteration

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

Optimizing K-Means for MapReduce

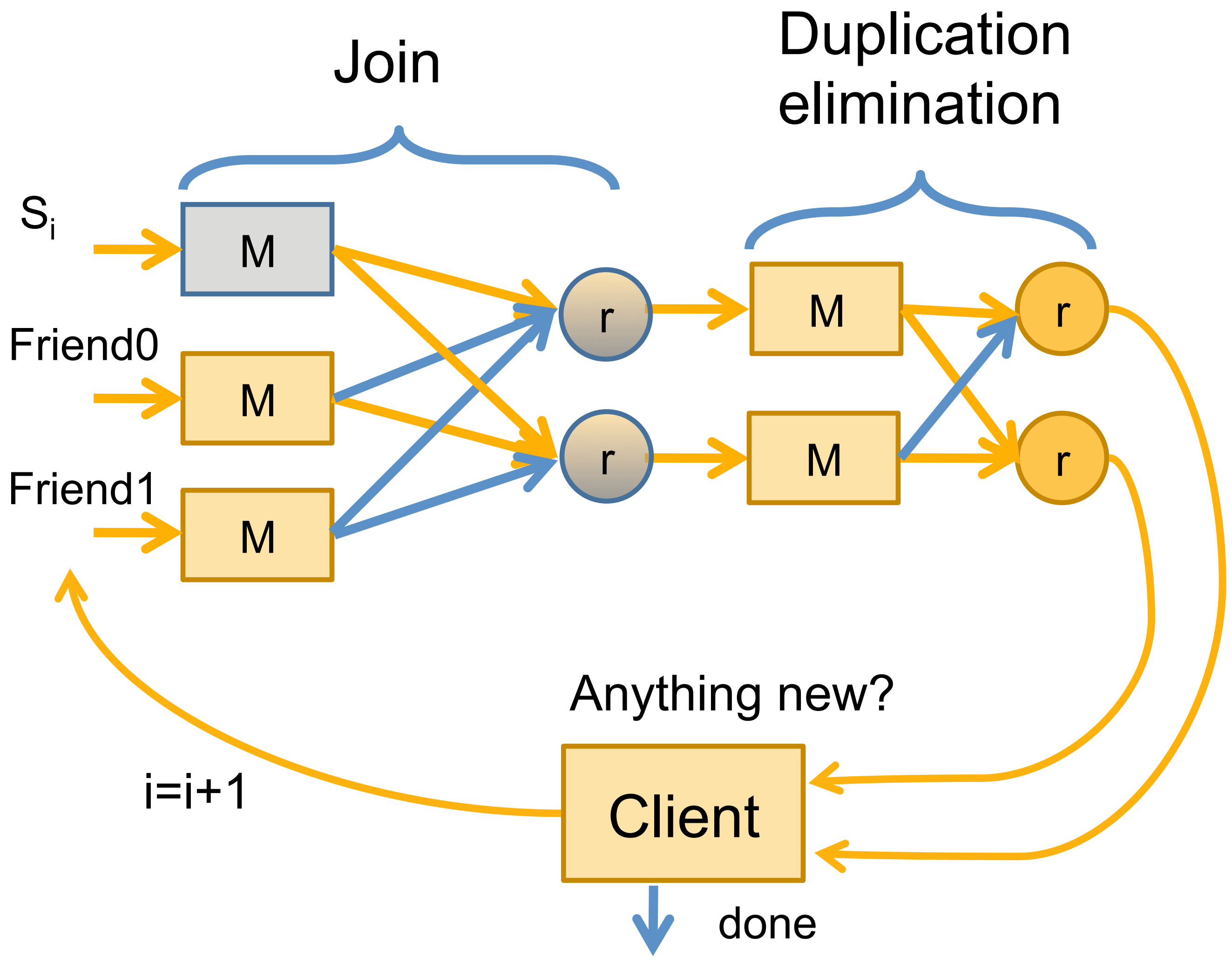
Combiners can be used to optimize the distributed algorithm

- Compute for each centroid the local sums of the points
- Send to the reducer: <centroid, partial sums>

Use of a single **reducer**

- Data to reducers is very small
- Single reducer can tell immediately if the computation has converged
- Creation of a single output file

Transitive Closure in MapReduce



Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

PageRank Algorithm

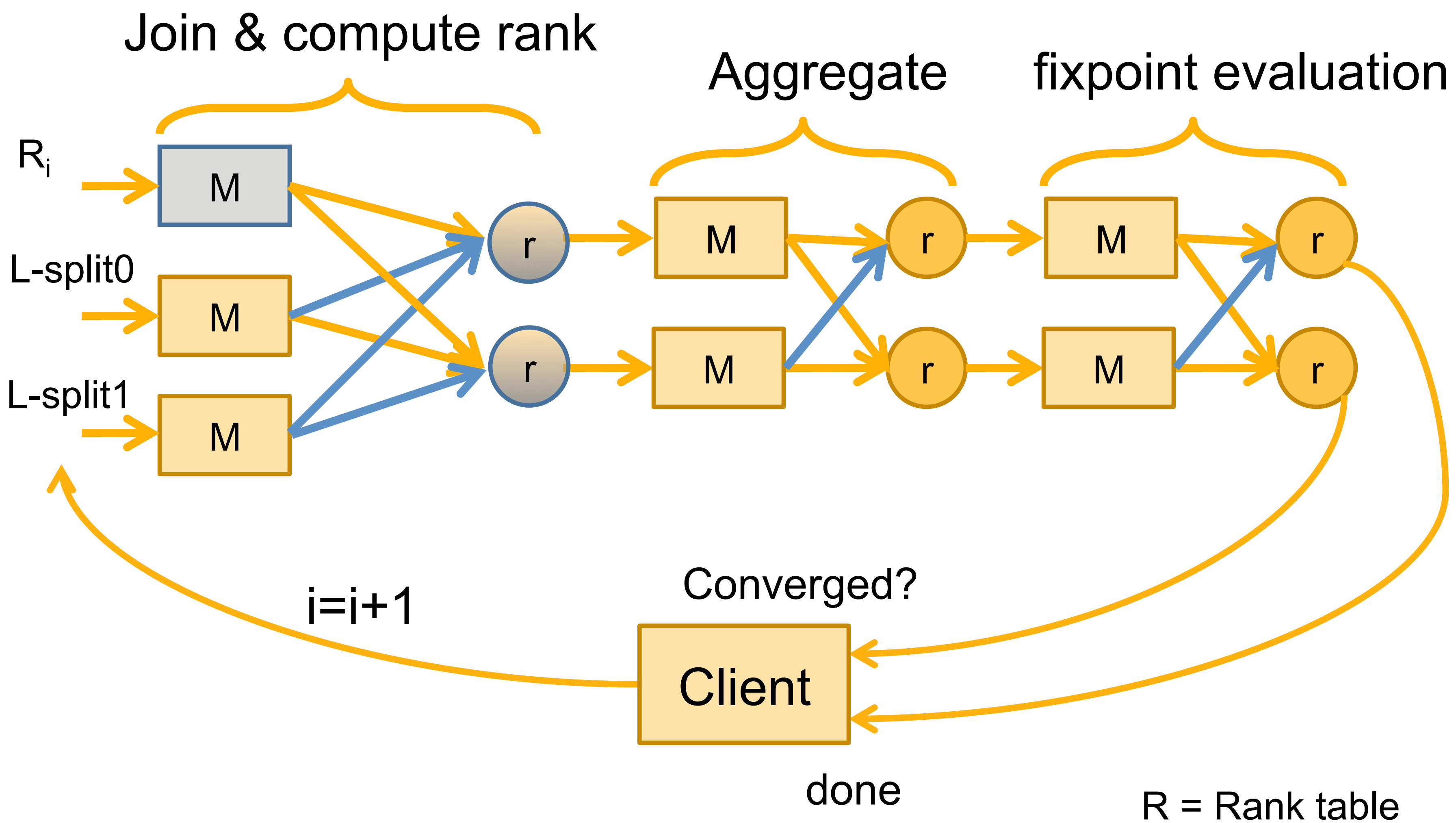
Link analysis algorithm that assigns weights to each vertex in a graph by iteratively computing the weight of each vertex based on the weight of its inbound neighbours.

In relational algebra, PageRank can be expressed as: a join followed by an update with two aggregations that are repeated until stopping condition

The **first MapReduce job** joins the rank and linkage tables; Mappers emit the join column as the key and Reducers compute the join for each unqiue source URL and ran contribution of each outbound edge.

The **second MapReduce job** computes the aggregate rank of each unique destination URL. The Map is the identity and the reducers sum the rank contributions of each incoming edge.

PageRank Algorithm



R = Rank table
L = Linkage table

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

Limitation: iterative algorithms

MapReduce tasks must be written as acyclic dataflow programs

- Stateless mappers and reducers
- Batch model

Difficult to implement iterative processing of datasets

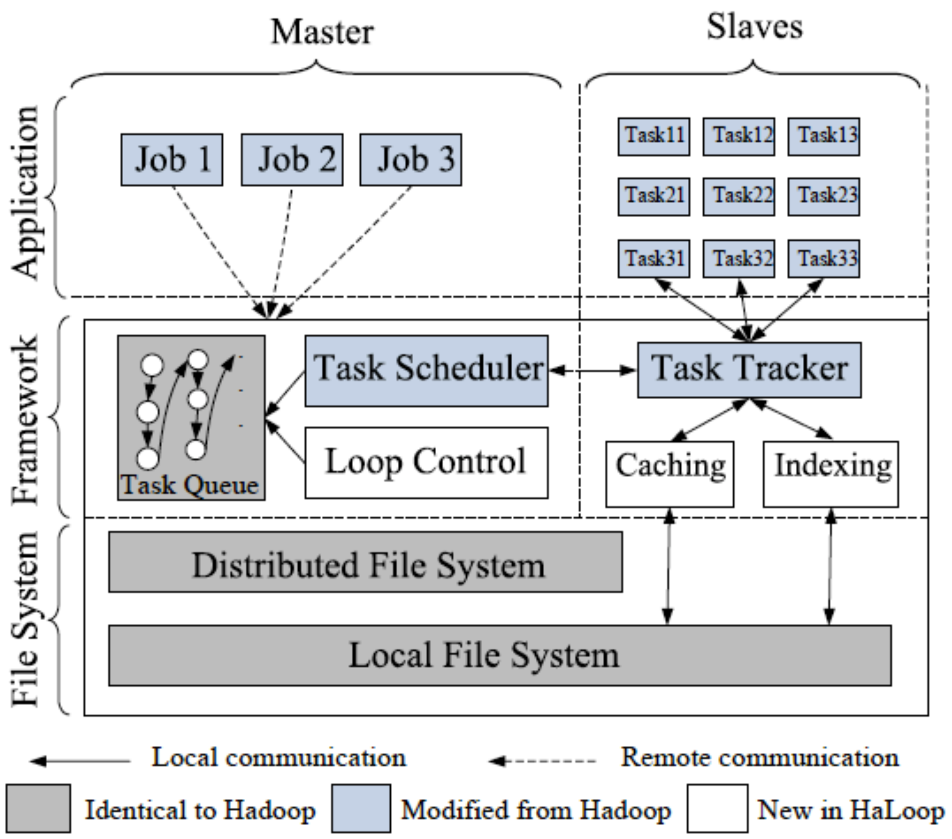
- Machine learning typically requires iterative operation: the dataset is visited multiple times by the algorithm

HaLoop for iterative MapReduce

MapReduce cannot express iteration or recursion

HaLooP modifies Hadoop for supporting fixpoint operations, loop-aware task scheduling, and cache management

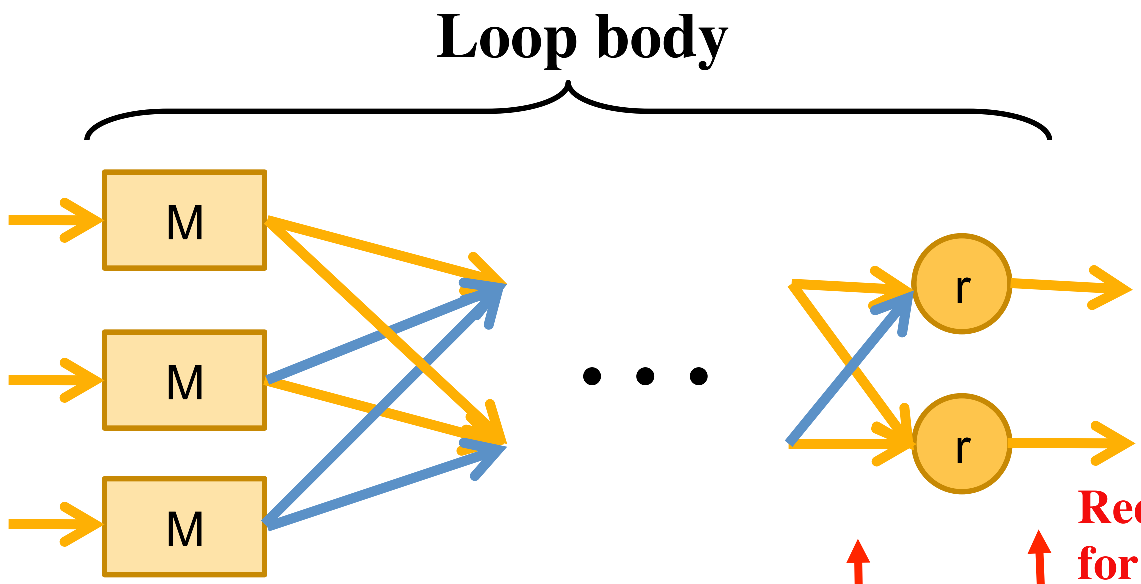
Map – Reduce – Fixpoint model for recursive languages



For example: the vector of PageRank values of web pages is the fixed point of a linear transformation derived from the link structure

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

HaLoop: Inter-iteration caching



Reducer output cache (RO) for access to output of previous iterations, for fixpoint evaluation

Reducer input cache (RI) for loop invariant data without map/shuffle

Mapper output cache (MO)

Mapper input cache (MI) for access to non-local mapper input on later iterations

Largest gain by caching loop invariant data

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010