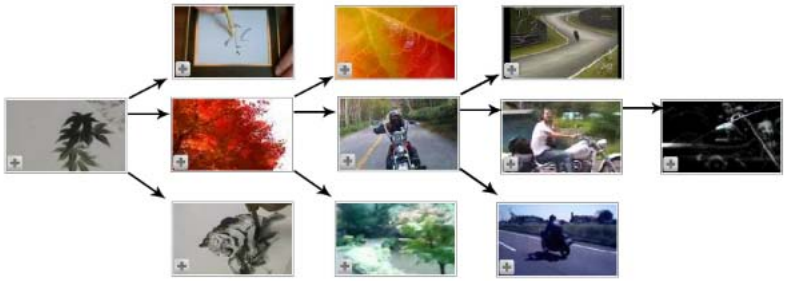
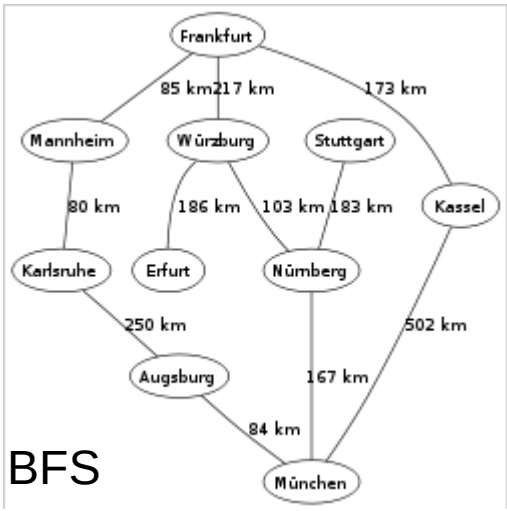
iMapReduce:
A Distributed Computing Framework for Iterative Computation

Yanfeng Zhang, Northeastern University, China
Qixin Gao, Northeastern University, China
Lixin Gao, UMass Amherst
Cuirong Wang, Northeastern University, China

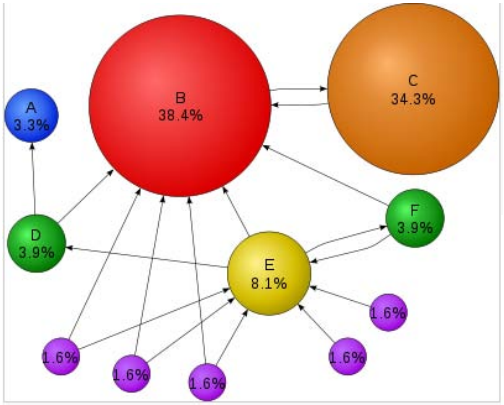
Iterative Computation Everywhere



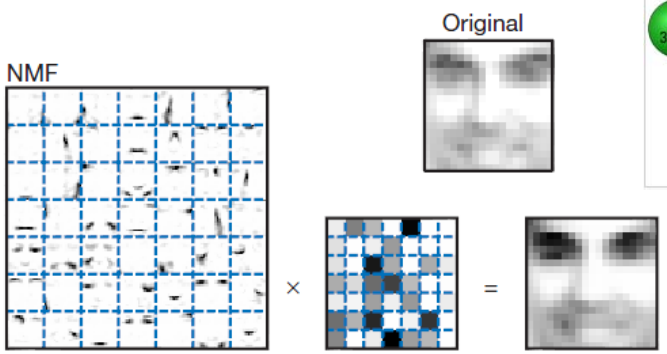
Video suggestion



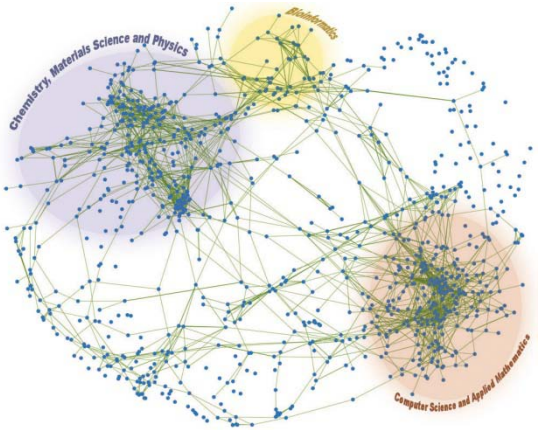
BFS



PageRank

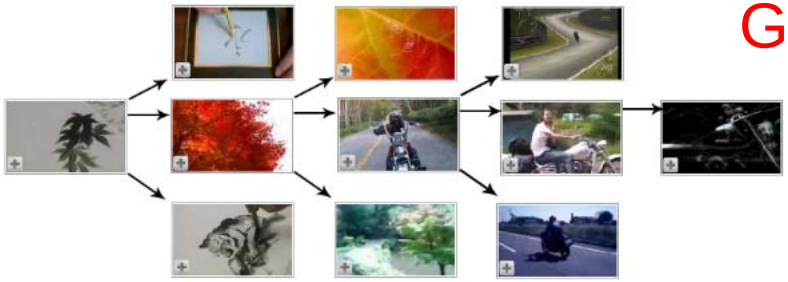


Pattern Recognition



Clustering

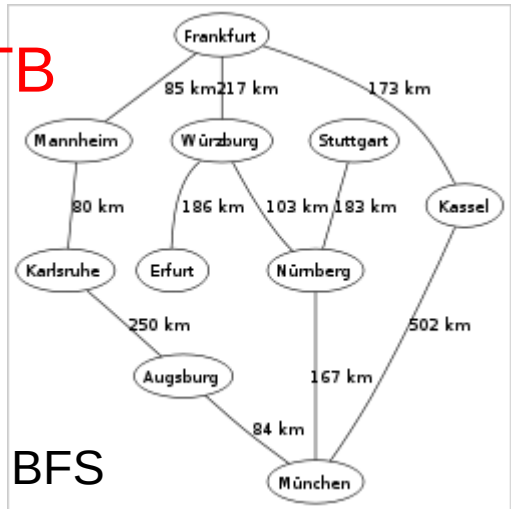
Iterative Computation on Massive Data



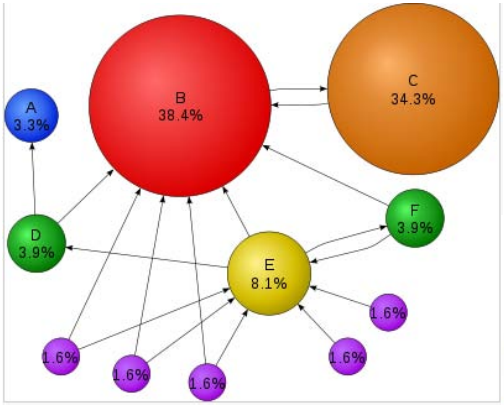
Video suggestion

100 million videos

Google maps 70.5TB

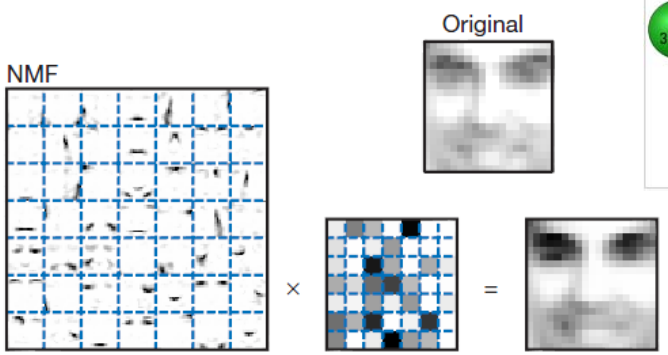


BFS



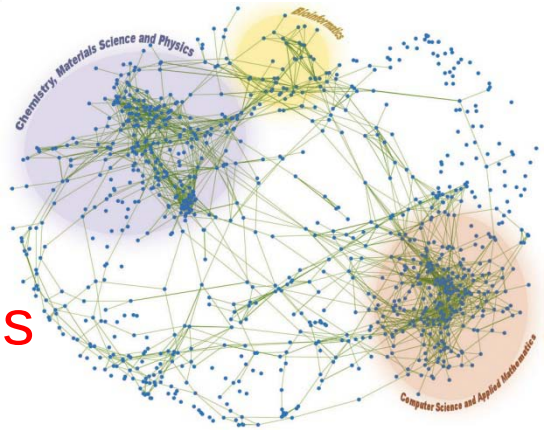
PageRank

29.7 billion pages



Pattern Recognition

500TB/yr hospital



700PB human genomics Clustering

Large-Scale Solution MapReduce

- Programming model
  - Map: distribute computation workload
  - Reduce: aggregate the partial results
- But, no support of iterative processing
  - Batched processing
  - Must design a series of jobs, do the same thing many times
    - Task reassignment, data loading, data dumping

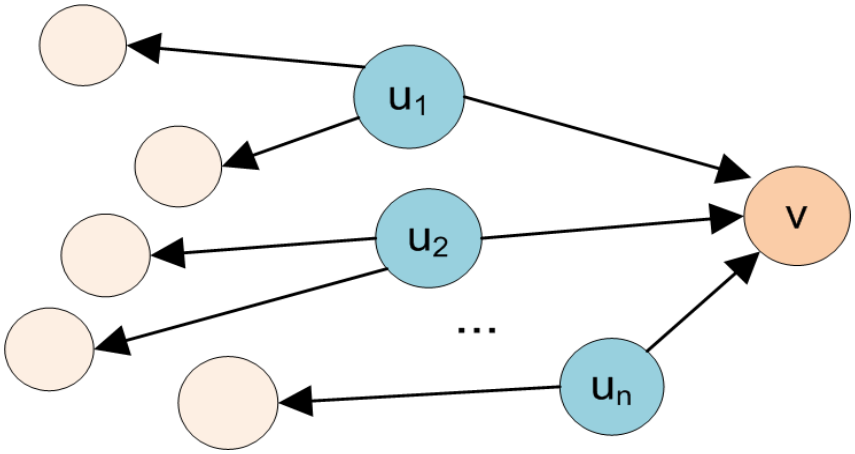
Outline

- Introduction & Motivation
- Example: PageRank in MapReduce
- System Design: iMapReduce
- Evaluation
- Conclusions

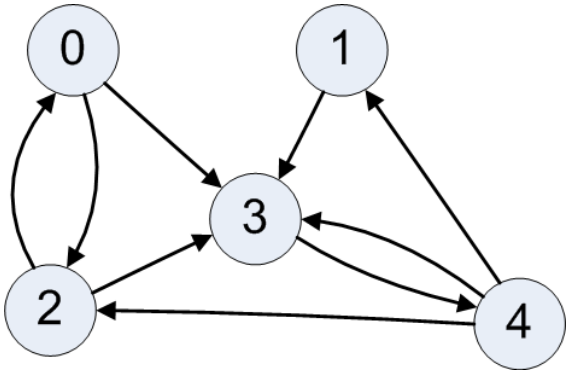
PageRank

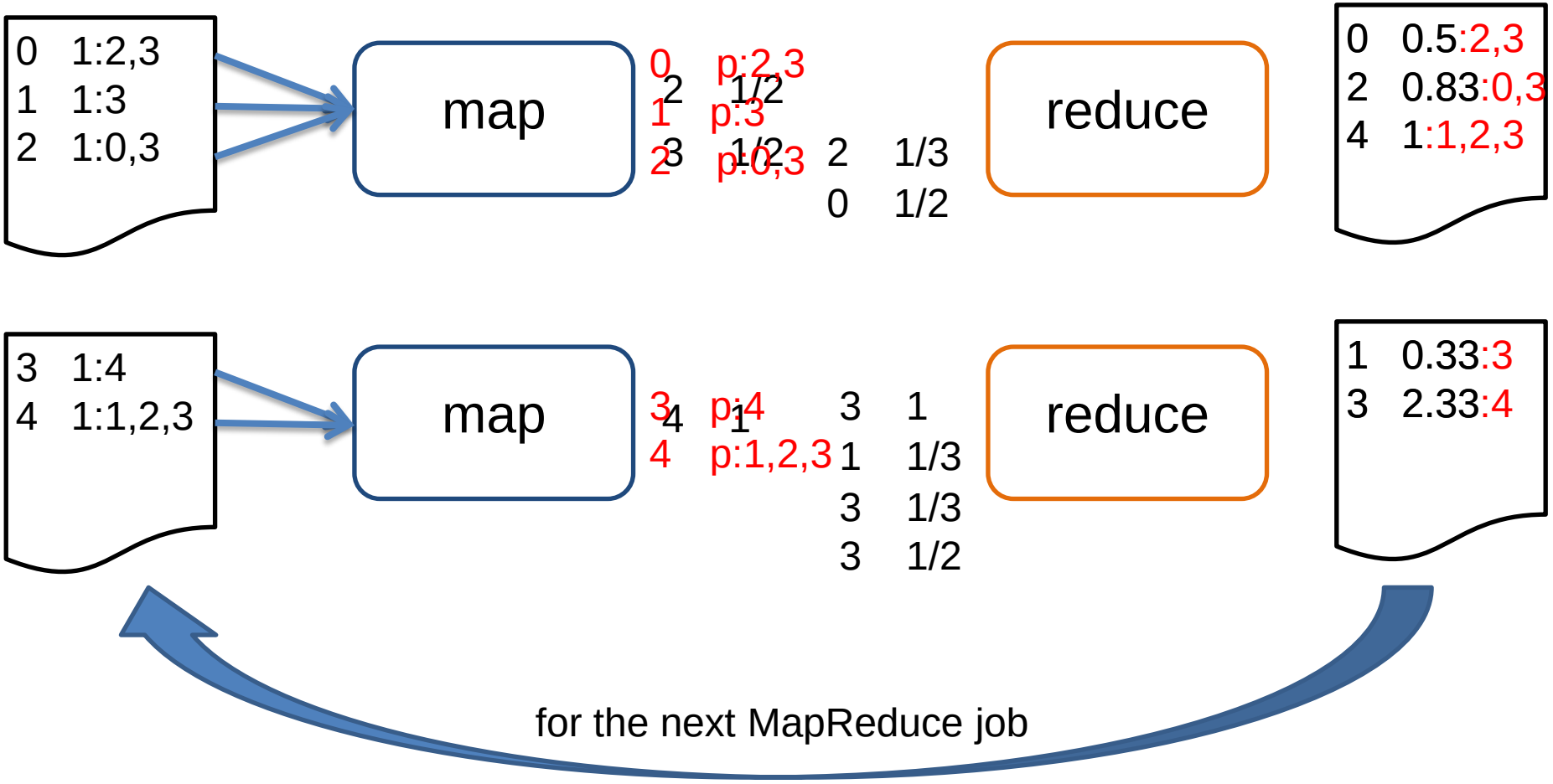
- Each node associated with a ranking score R
  - Evenly to its neighbors with portion d
  - Retain portion (1-d)

$$R^{k+1}(v) = d \cdot \left(\frac{R^k(u_1)}{N(u_1)} + \frac{R^k(u_2)}{N(u_2)} + \ldots + \frac{R^k(u_n)}{N(u_n)} \right) + (1-d) \cdot \frac{1}{|V|}$$



PageRank in MapReduce





Problems of Iterative Implementation

- 1. Multiple times job/task initializations
- 2. Re-shuffling the static data
- 3. Synchronization barrier between jobs

Outline

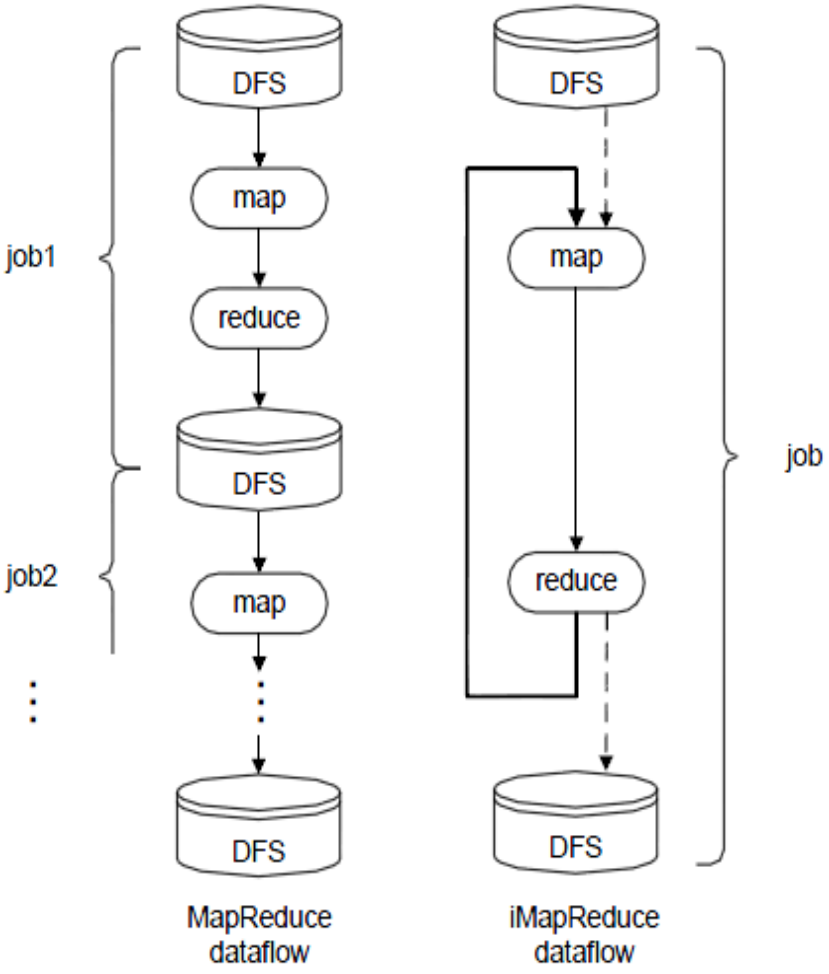
- Introduction
- Example: PageRank
- System Design: iMapReduce
- Evaluation
- Conclusions

iMapReduce Solutions

- 1. Multiple job/task initializations
  - Iterative processing
- 2. Re-shuffling the static data
  - Maintain static data locally
- 3. Synchronization barrier between jobs
  - Asynchronous map execution

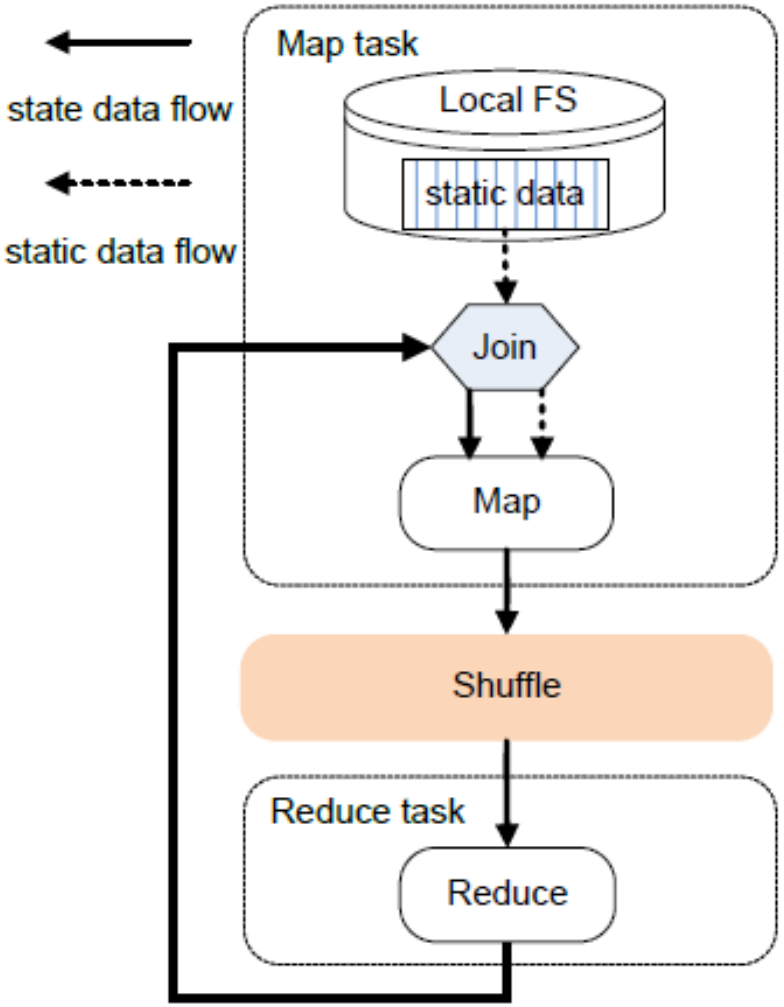
Iterative Processing

- Reduce → Map
  - Load/dump once
  - Persistent map/reduce task
  - Each reduce task has a correspondent map task
  - Locally connected

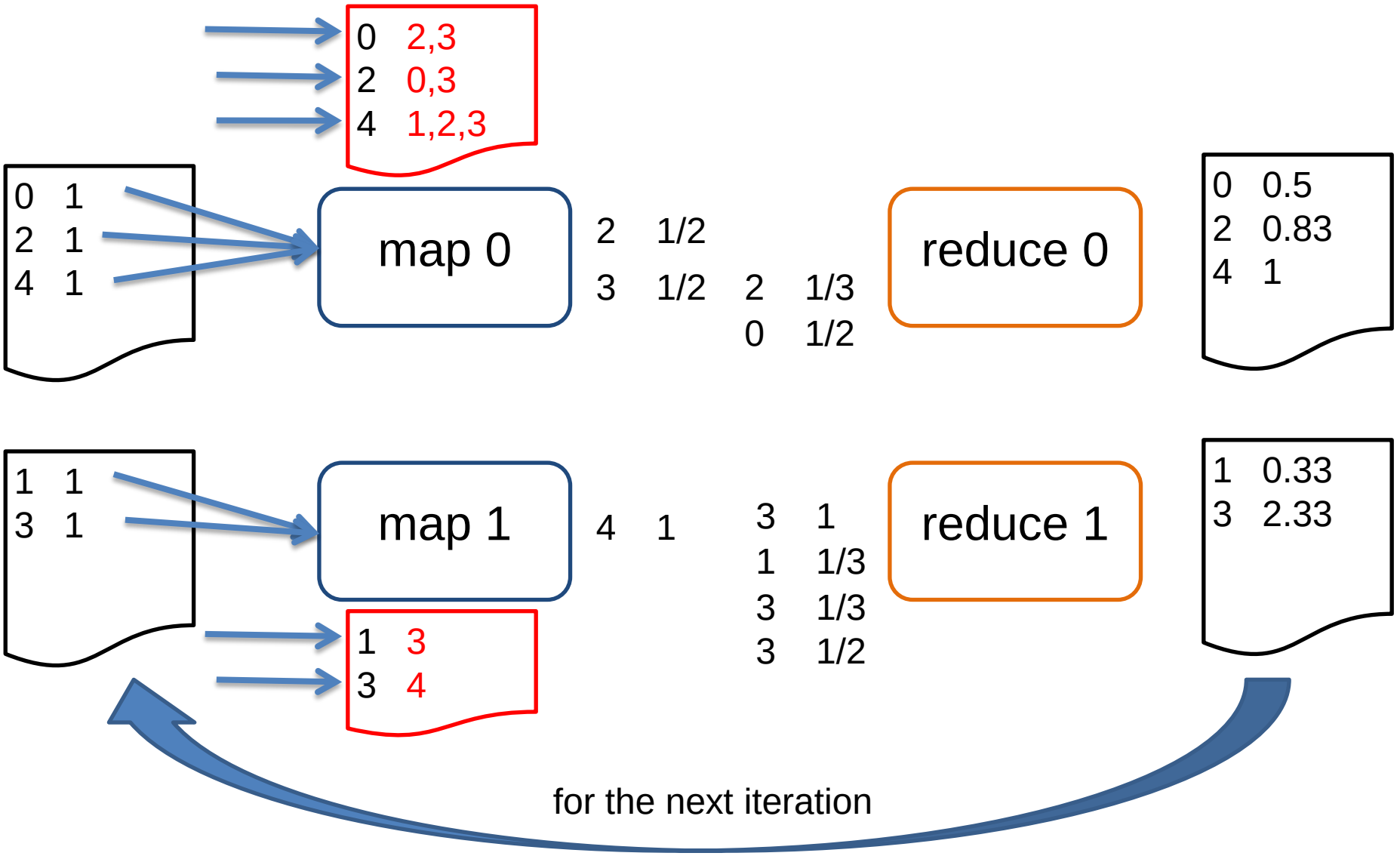


Maintain Static Data Locally

- Iterate state data
  - Update iteratively
- Maintain static data
  - Join with state data at map phase

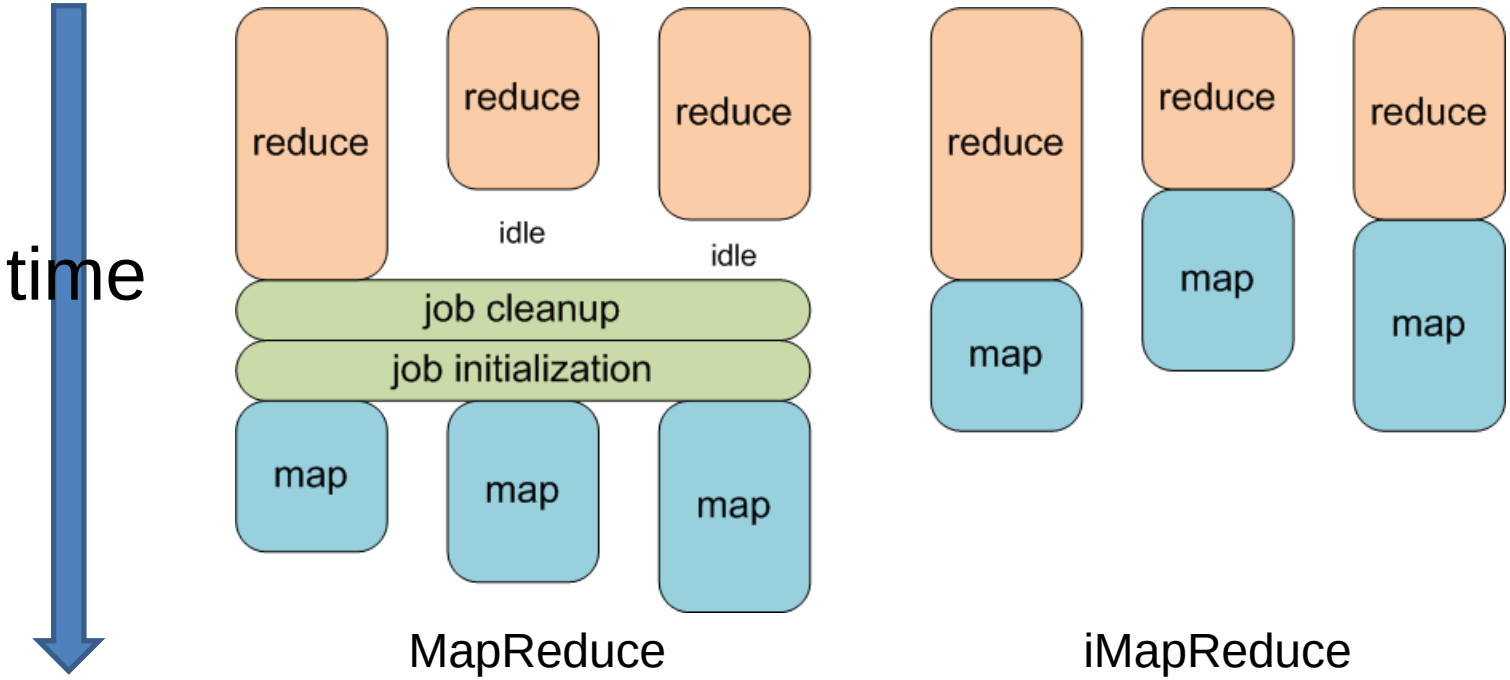


State Data ⋈ Static Data



Asynchronous Map Execution

- Persistent socket reduce -> map
- Reduce completion directly trigger map
- Map tasks no need to wait



Outline

- Introduction
- Example: PageRank
- System Design: iMapReduce
- Evaluation
- Conclusions

Experiment Setup

- Cluster
  - Local cluster: 4 nodes
  - Amazon EC2 cluster: 80 small instances
- Implemented algorithms
  - Shortest path, PageRank, K-Means
- Data sets

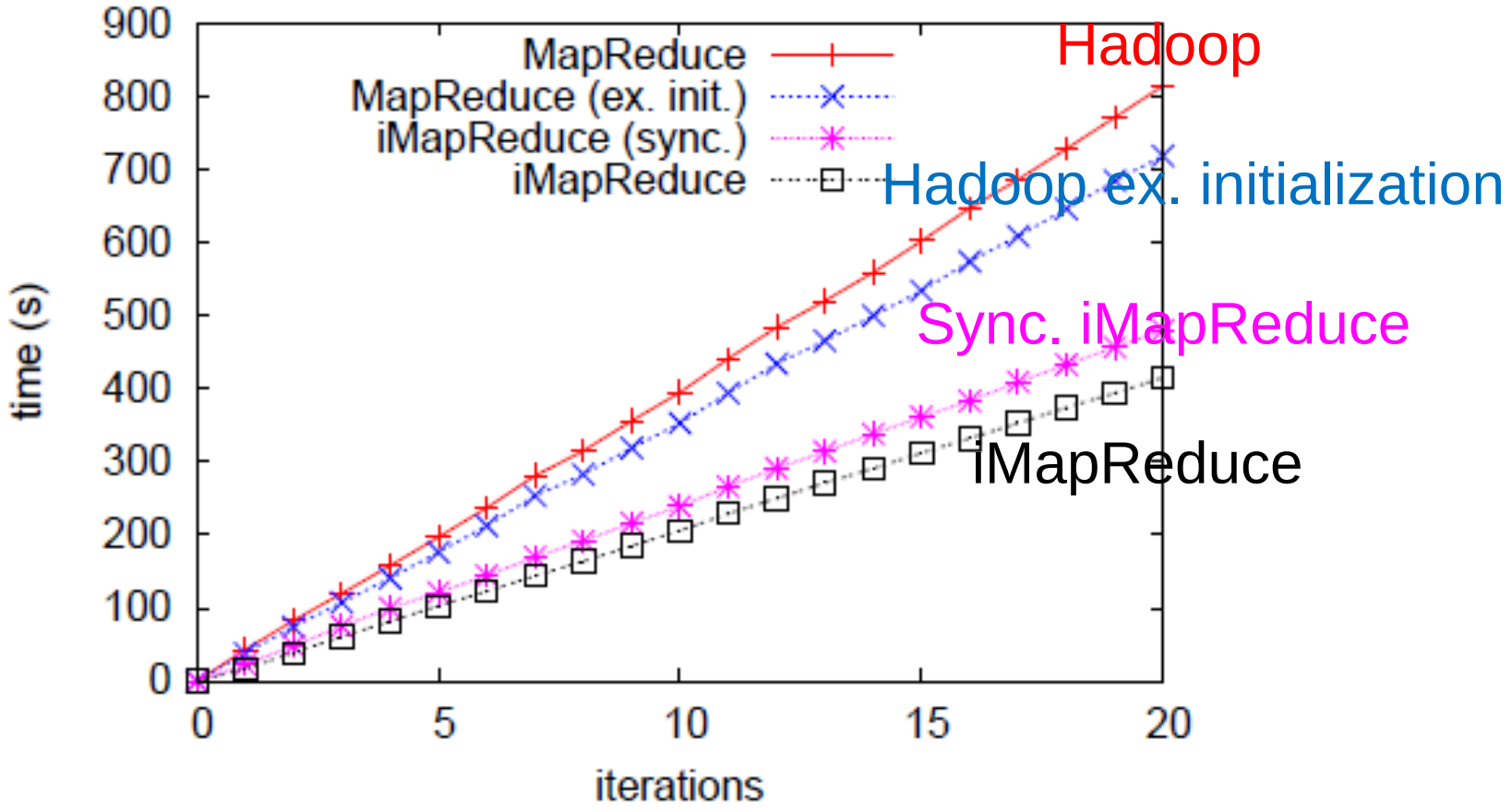
graph	# of nodes	# of edges	file size
DBLP	310,556	1,518,617	16MB
Facebook	1,204,004	5,430,303	58MB
SSPP-s	1M	7,868,140	87MB
SSPP-m	10M	78,873,968	958MB
SSPP-l	50M	369,455,293	5.19GB

SSSP

graph	# of nodes	# of edges	file size
Google	916,417	6,078,254	49MB
Berk-Stan	685,230	7,600,595	57MB
PageRank-s	1M	7,425,360	61MB
PageRank-m	10M	75,061,501	690MB
PageRank-l	30M	224,493,620	2.26GB

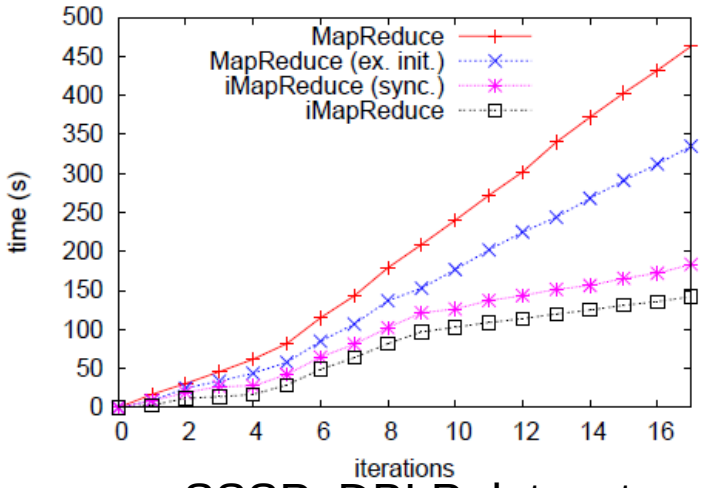
PageRank

Results on Local Cluster

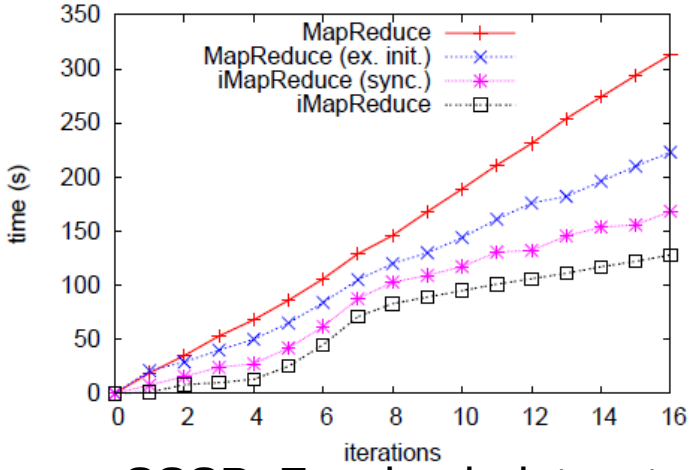


PageRank: google webgraph

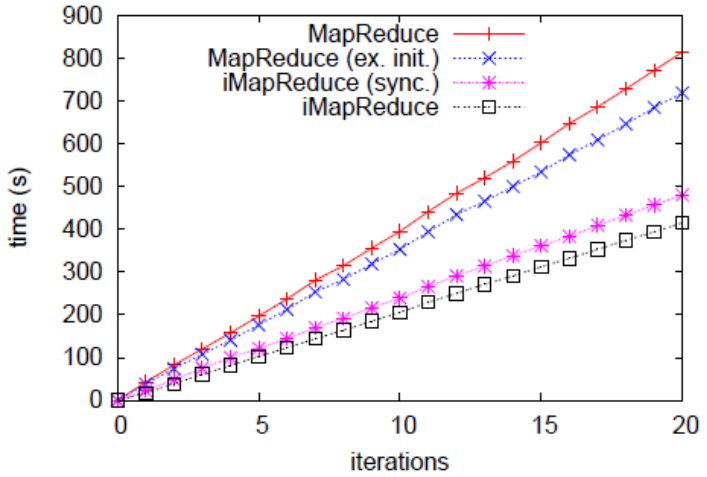
Results cont.



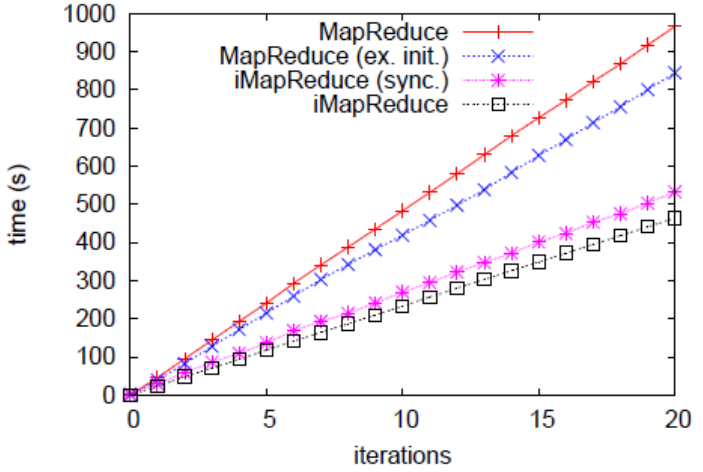
SSSP: DBLP dataset



SSSP: Facebook dataset



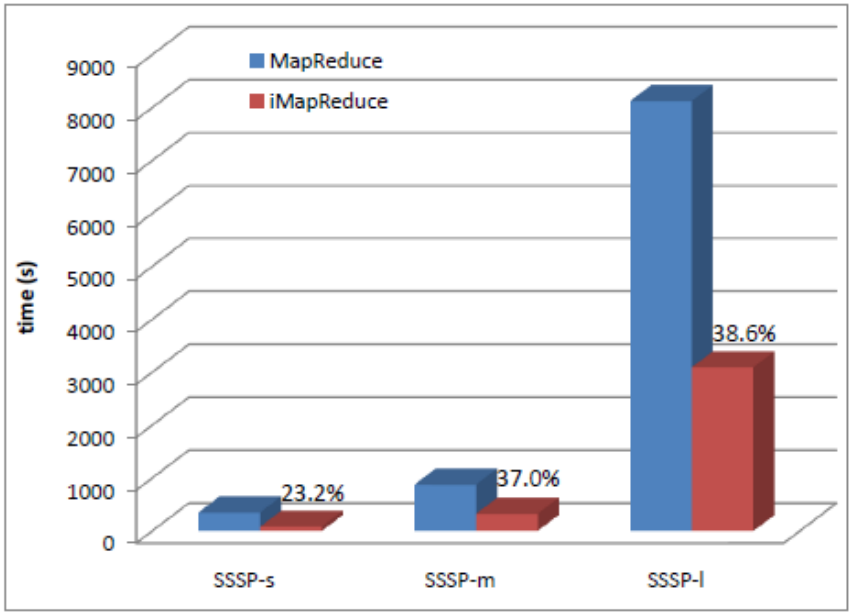
PageRank: google webgraph



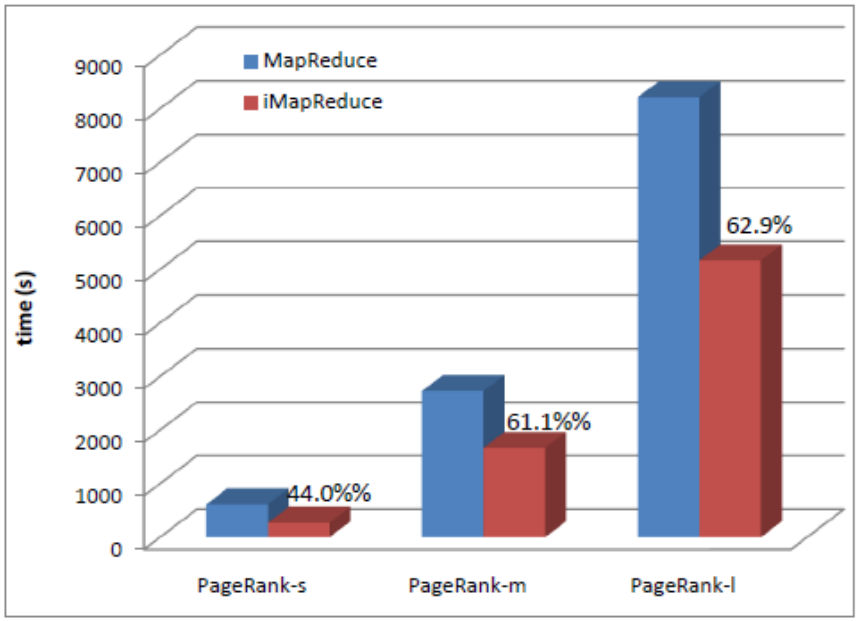
PageRank: Berk-Stan webgraph

Scale to Large Data Sets

- On Amazon EC2 cluster
- Various-size graphs

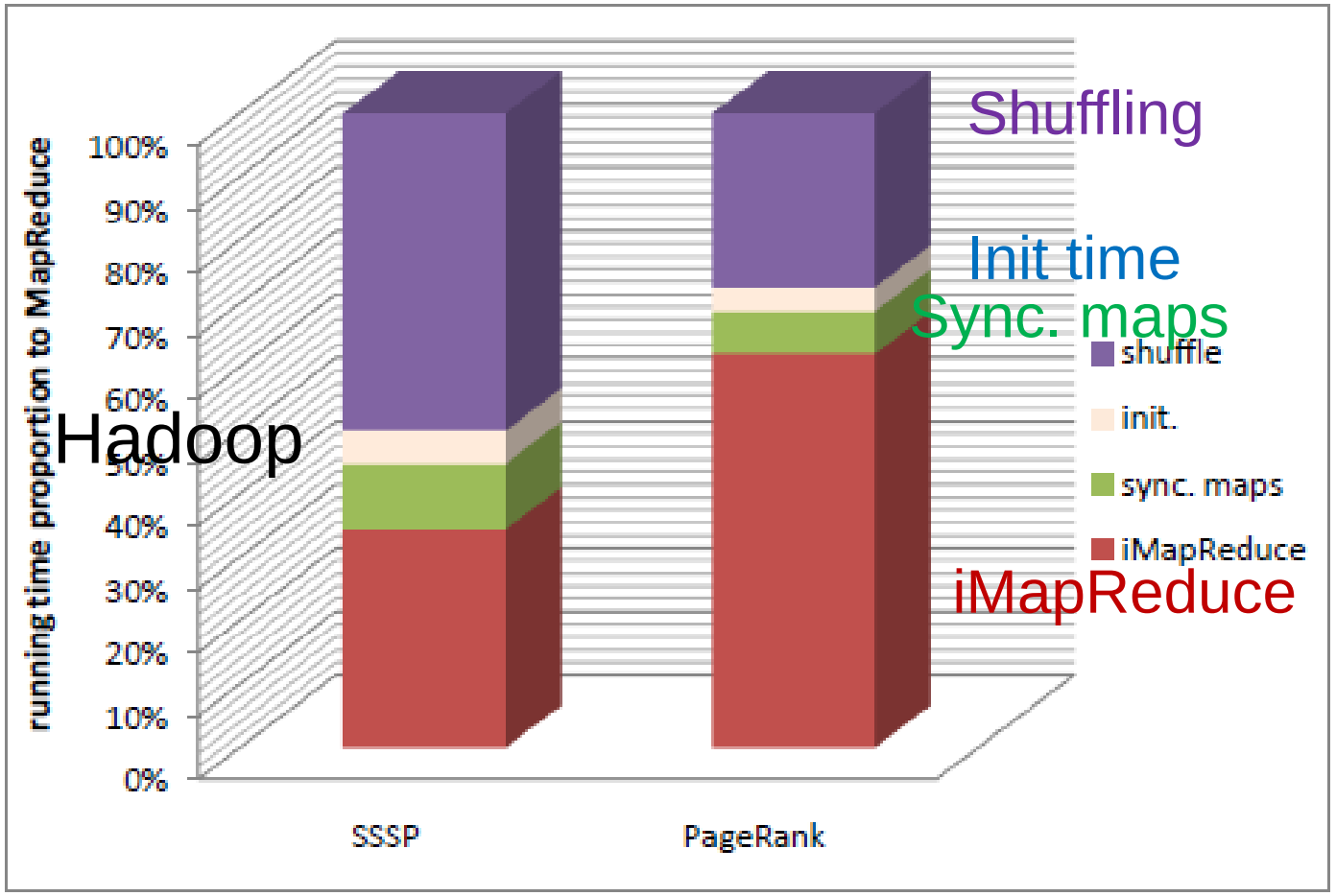


SSSP



PageRank

Different Factors on Running Time Reduction



Conclusion

- iMapReduce reduces running time by
  - Reducing the overhead of job/task initialization
  - Eliminating the shuffling of static data
  - Allowing asynchronous map execution
- Achieve a factor of 1.2x to 5x speedup
- Suitable for most iterative computations

Q & A

- Thank you!