Large-Scale Matrix Operations Using a Data Flow Engine

Reza Zadeh

MapReduce for Matrix Operations

Matrix-vector multiply

Power iteration (e.g. PageRank)

Gradient descent methods

Stochastic SVD

Tall skinny QR

Many others!

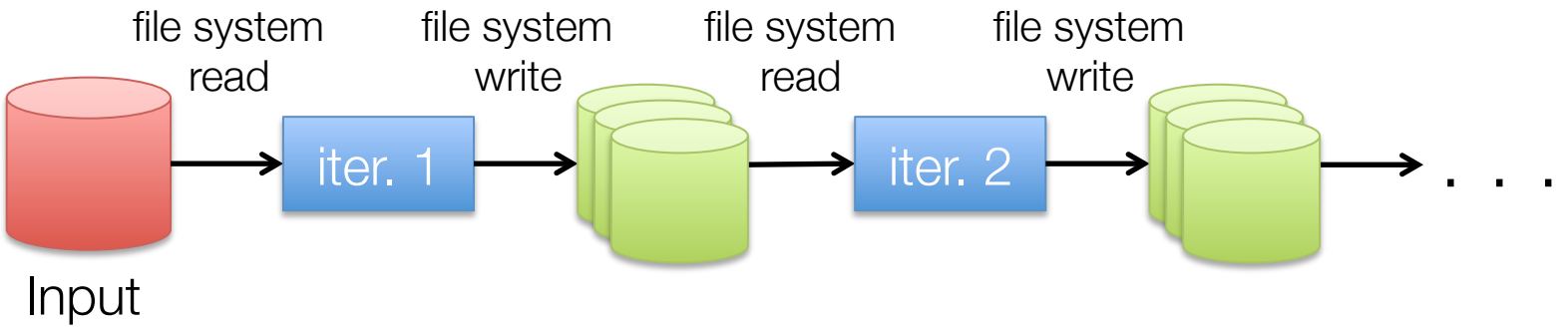
Limitations of MapReduce

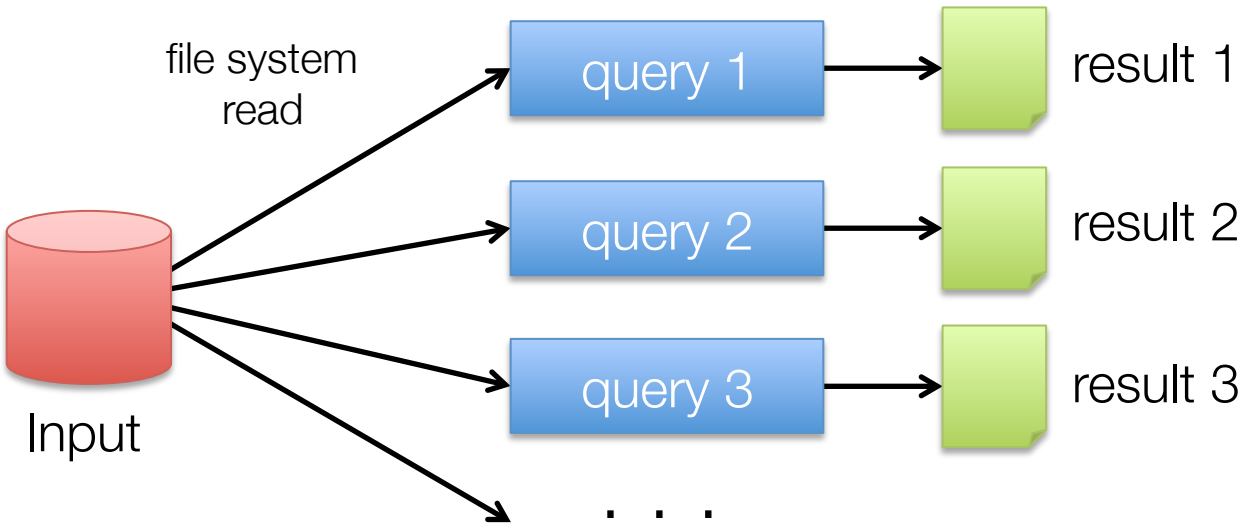
MapReduce is great at one-pass computation, but inefficient for *multi-pass* algorithms

No efficient primitives for data sharing

- » State between steps goes to distributed file system
- » Slow due to replication & disk storage
- » No control of data partitioning across steps

Example: Iterative Apps



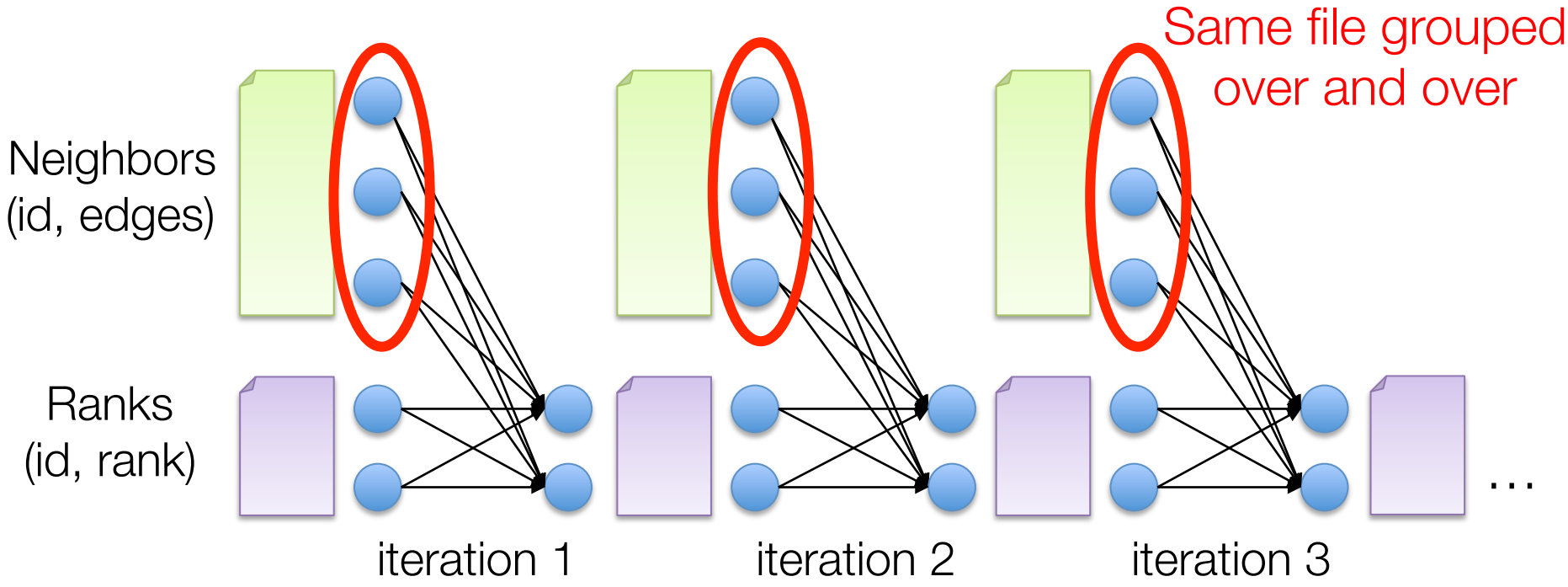


Commonly spend 90% of time doing I/O

Example: PageRank

Repeatedly multiply sparse matrix and vector

Requires repeatedly hashing together page adjacency lists and rank vector



Spark Programming Model

Extends MapReduce with primitives for efficient data sharing

» “Resilient distributed datasets”

APIs in Java, Scala & Python

Resilient Distributed Datasets (RDDs)

Collections of objects stored across a cluster

User-controlled partitioning & storage (memory, disk, …)

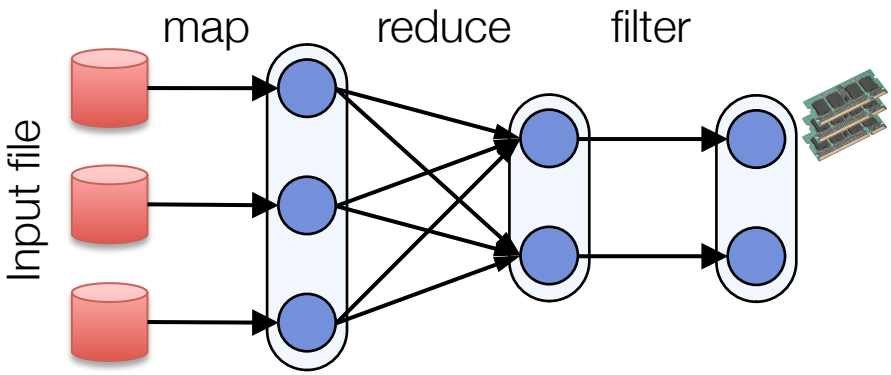
Automatically rebuilt on failure

```
urls = spark.textFile(“hdfs://...”)
records = urls.map(lambda s: (s, 1))
counts = records.reduceByKey(lambda a, b: a + b)
bigCounts = counts.filter(lambda (url, cnt): cnt > 10)
```

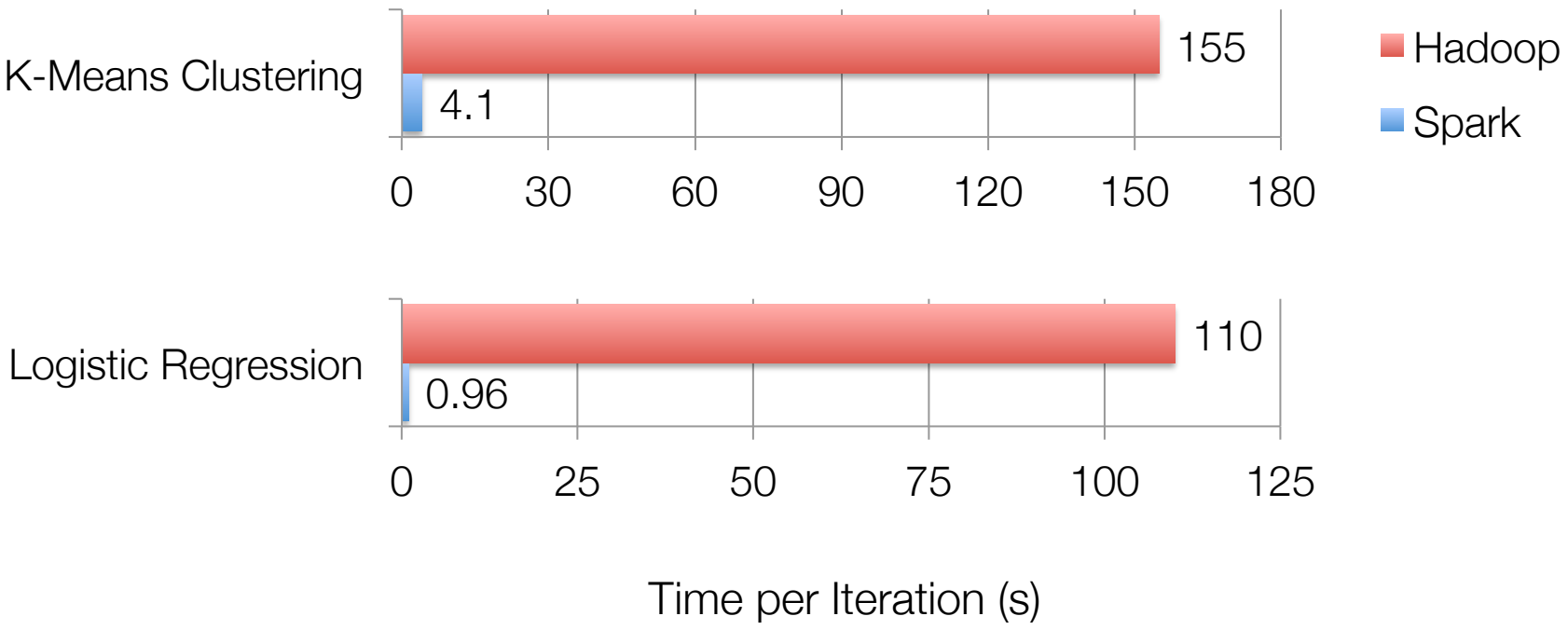
```
bigCounts.cache()

bigCounts.filter(
  lambda (k,v): “news” in k).count()

bigCounts.join(otherPartitionedRDD)
```



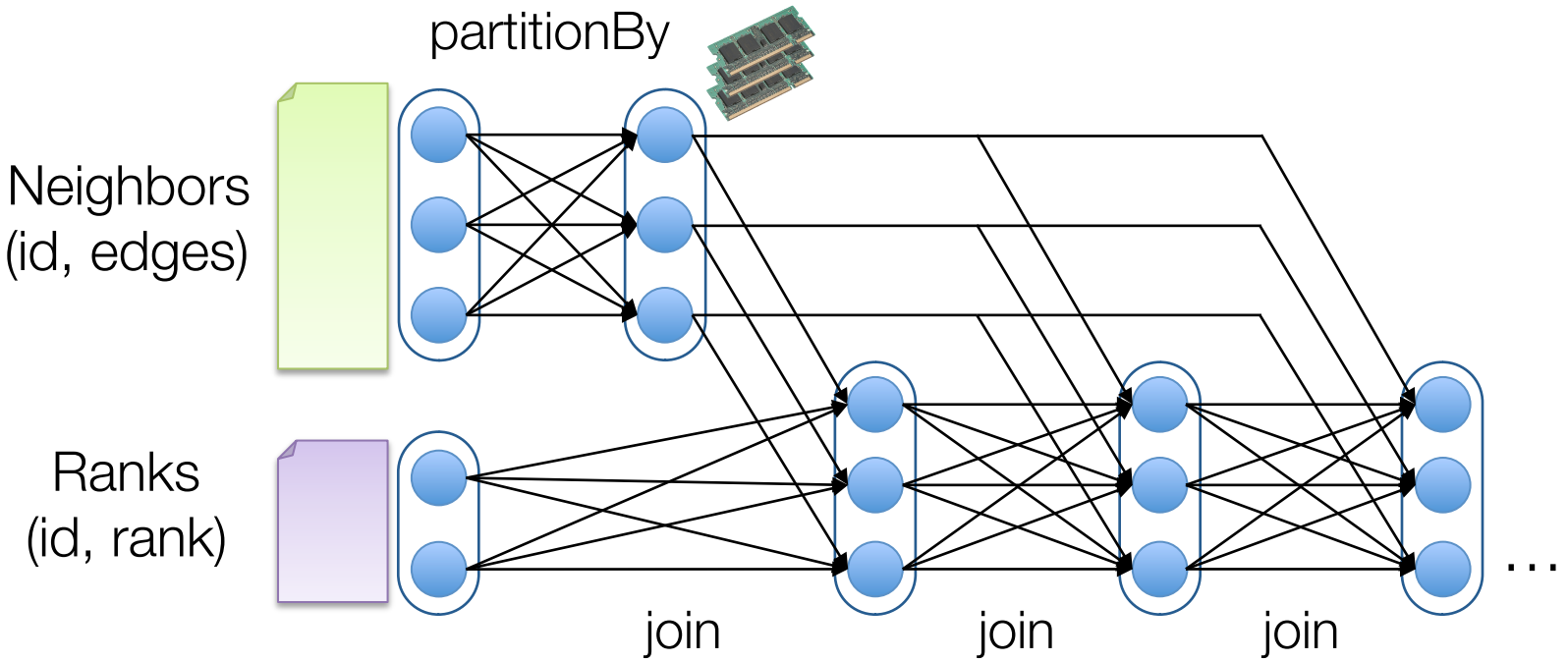
Performance



PageRank

Using cache(), keep neighbors in RAM

Using partitioning, avoid repeated hashing



PageRank Results