Fault Tolerant Distributed Main Memory Systems

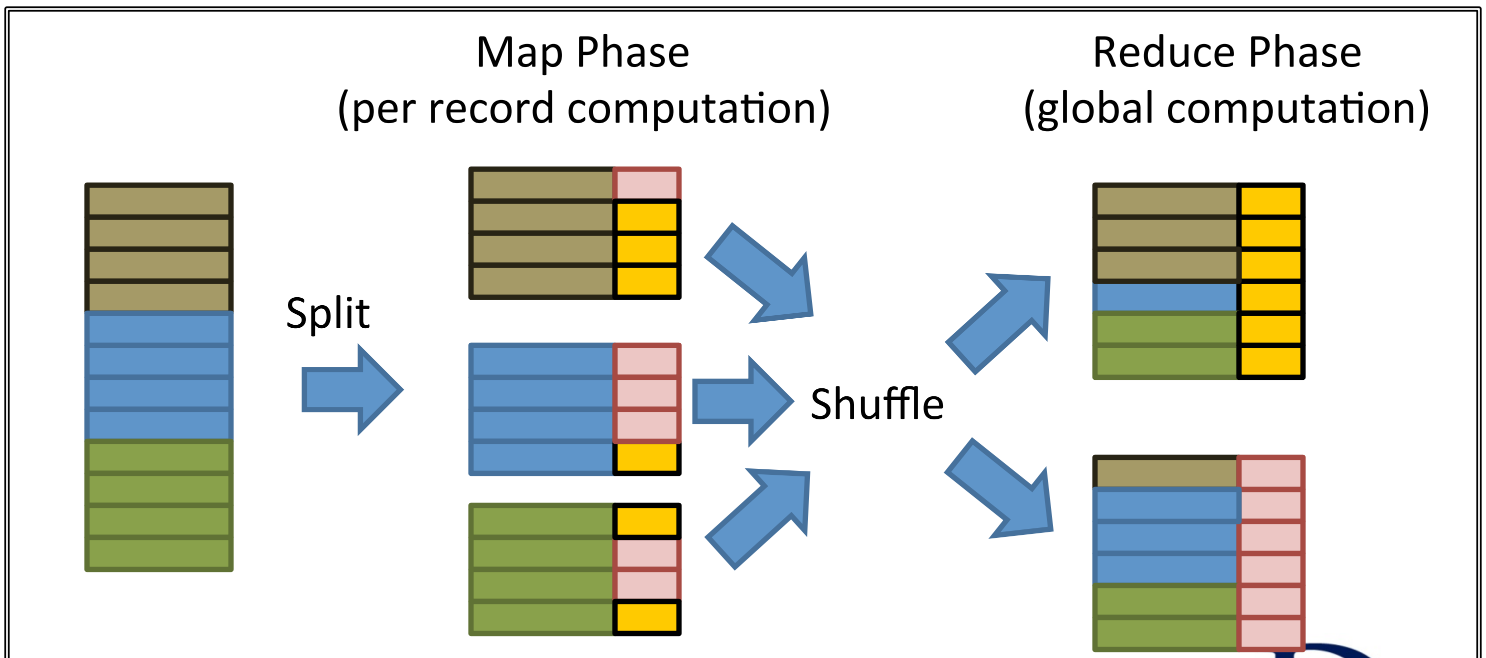
CompSci 590.04
Instructor: Ashwin Machanavajjhala

Duke
UNIVERSITY

Recap: Map Reduce

$$\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$

$$\text{reduce}(k_2, \text{list}(v_1)) \rightarrow \text{list}(k_3, v_3)$$



Duke UNIVERSITY

Recap: Map Reduce

Programming Model **+** **Distributed System**

Programming Model

- Simple model
- Programmer only describes the logic

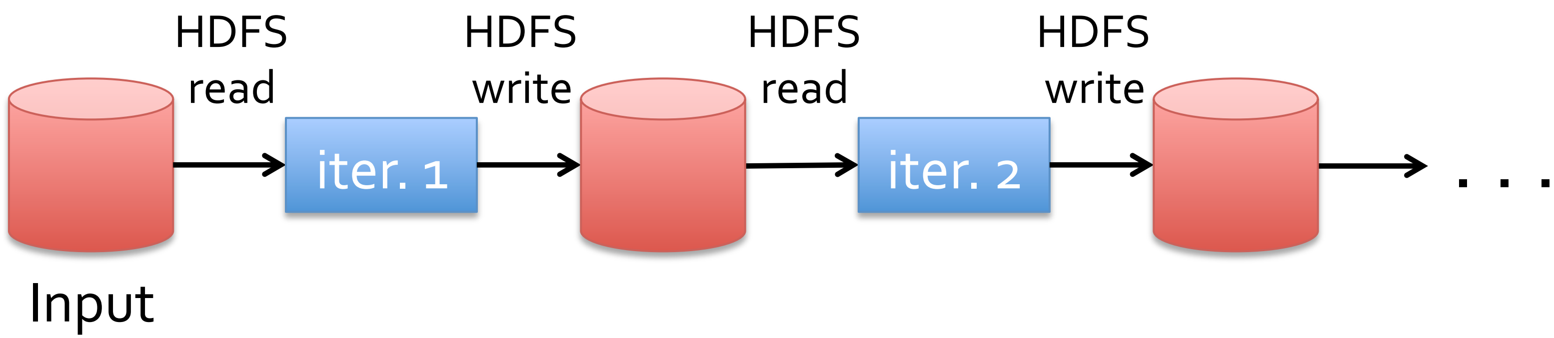
Distributed System

- Works on commodity hardware
- Scales to thousands of machines
- Ship code to the data, rather than ship data to code
- Hides all the hard systems problems from the programmer
  - Machine failures
  - Data placement
  - …

Recap: Map Reduce

But as soon as it got popular, users wanted more:

- More complex, multi-stage applications (e.g. iterative machine learning & graph processing)
- More interactive ad-hoc queries



Recap: Map Reduce

But as soon as it got popular, users wanted more:

- More complex, multi-stage applications
  (e.g. iterative machine learning & graph processing)
- More interactive ad-hoc queries

Thus arose many *specialized* frameworks for parallel processing

UNIVERSITY

Recap: Pregel

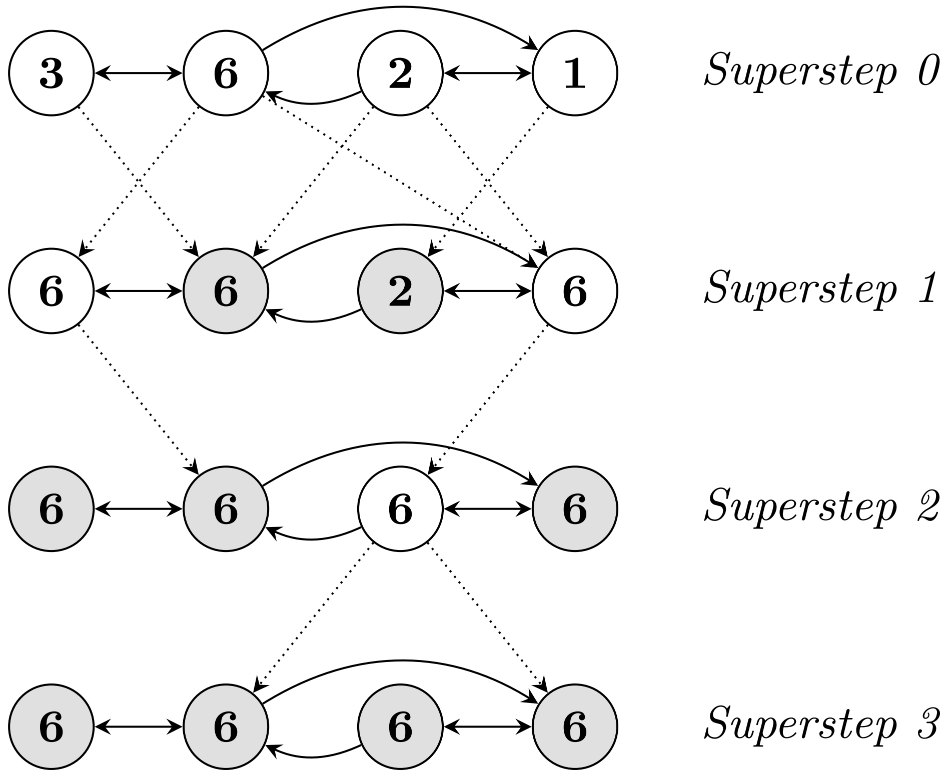
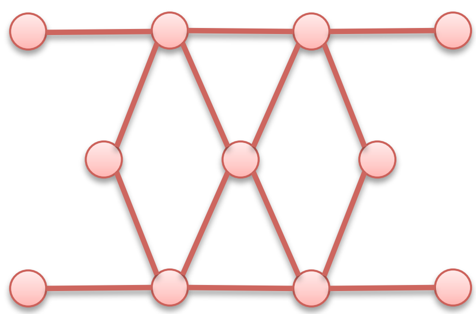


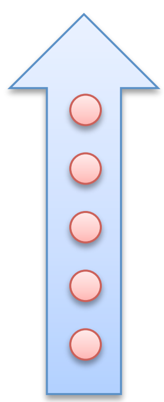
Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

GraphLab

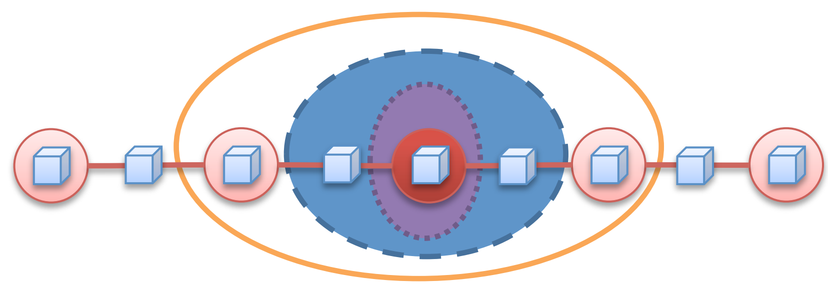
Data Graph



Shared Data Table



Scheduling



Update Functions and Scopes

Problem with specialized frameworks

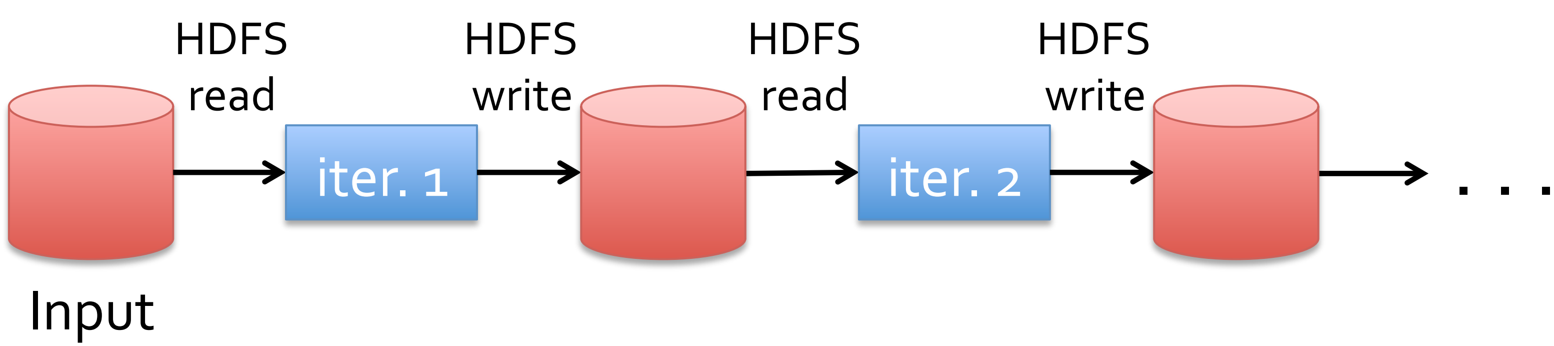
- Running multi-stage workflows is hard
  - Extract a mentions of celebrities from news articles
  - Construct a co-reference graph of celebrities (based on cooccurence in the same article)
  - Analyze this graph (say connected components / page rank)

- Graph processing on Map Reduce is slow.
- The input does not have a graph abstraction. Map Reduce is a good candidate to construct the graph in the first place.

Duke UNIVERSITY

Root Cause Analysis

- Why do graph processing algorithms and iterative computation do poorly on Map Reduce?



- There is usually some (large) input that does not change across iterations.
  Map reduce unnecessarily keeps writing to and reading from disk.

Duke UNIVERSITY

Examples

- Page Rank
  Links in the graph do not change, only the rank of each node changes.

- Logistic Regression
  The original set of points do not change, only the model needs to be updated

- Connected components / K-means clustering
  The graph/dataset does not change, only the labels on the nodes/points changes.

Examples

- Page Rank
  Links in the graph do not change, only the rank of each node changes.

LARGE

- Logistic Regression
  The original set of points do not change, only the model needs to be updated

- Connected components / K-means clustering
  The graph/dataset does not change, only the labels on the nodes/points changes.

Examples

- Page Rank
  Links in the graph do not change, only the rank of each node changes.

small

- Logistic Regression
  The original set of points do not change, only the model needs to be updated

- Connected components / K-means clustering
  The graph/dataset does not change, only the labels on the nodes/ points changes.

Duke
UNIVERSITY

Idea: Load the “immutable” part into memory

- Twitter follows graph: 26GB uncompressed

- Can be stored in memory using 7 off the shelf machines each having 4 GB memory each.

Idea: Load the "immutable" part into memory

- Twitter follows graph: 26GB uncompressed
- Can be stored in memory using 7 off the shelf machines each having 4 GB memory each.
- **Problem: Fault Tolerance!**

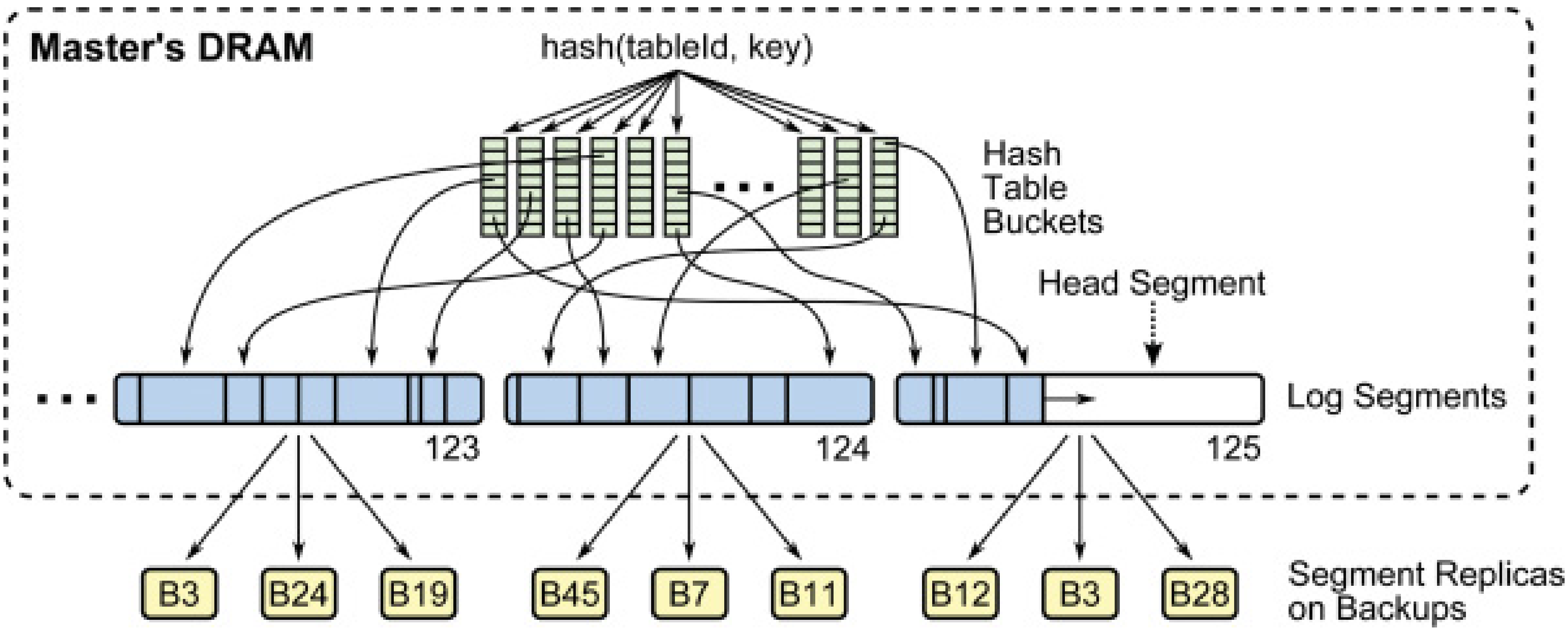
Duke UNIVERSITY

Fault Tolerant Distributed Memory

- Solution 1: Global Checkpointing
- E.g., Piccolo (http://piccolo.news.cs.nyu.edu/)
- Problem: need to redo a lot of computation.
  (In Map Reduce: need to only to redo a Mapper or Reducer)

Fault Tolerant Distributed Memory

- Solution 2: Replication (e.g., RAMCloud)



RAMCloud

- Log Structured Storage
- Each master maintains in memory
  - An append only log
  - Hash Table (object id, location on the log)

- Every write becomes an append on the log
  - Plus a write to the hash table

- Log is divided into log segments

Durable Writes

- Write to the head of log (in master's memory)
- Write to hash table (in master's memory)
- Replication to 3 other backups
  - They each write to the backup log in memory and return
- Master returns as soon as ACK is received from replicas.

- Backups write to disk when the log segment becomes full.

Fault Tolerant Distributed Memory

- Solution 2: Replication
- Log Structured Storage (e.g., RAMCloud) + Replication
- Problem:
  - Every write triggers replication across nodes, which can become expensive.
  - Log needs constant maintenance and garbage cleaning.

Fault Tolerant Distributed Memory

- Moreover, existing solutions (Piccolo, RAMCloud, memcacheD) assume that objects in memory can be read as well as written
- But, in most applications we only need objects in memory that are read (and hence immutable).

Fault Tolerant Distributed Memory

- Solution 3: Resilient Distributed Datasets

Restricted form of distributed shared memory

- Data in memory is immutable
- Partitioned collection of records
- Can only be built through coarse grained deterministic transformations (map, filter, join, etc)

Fault Tolerance through lineage

- Maintain a small log of operations
- Recompute lost partitions when failures occur

Example: Log Mining

Original File

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
```

This is the RDD that is stored

```
messages.persist()
```

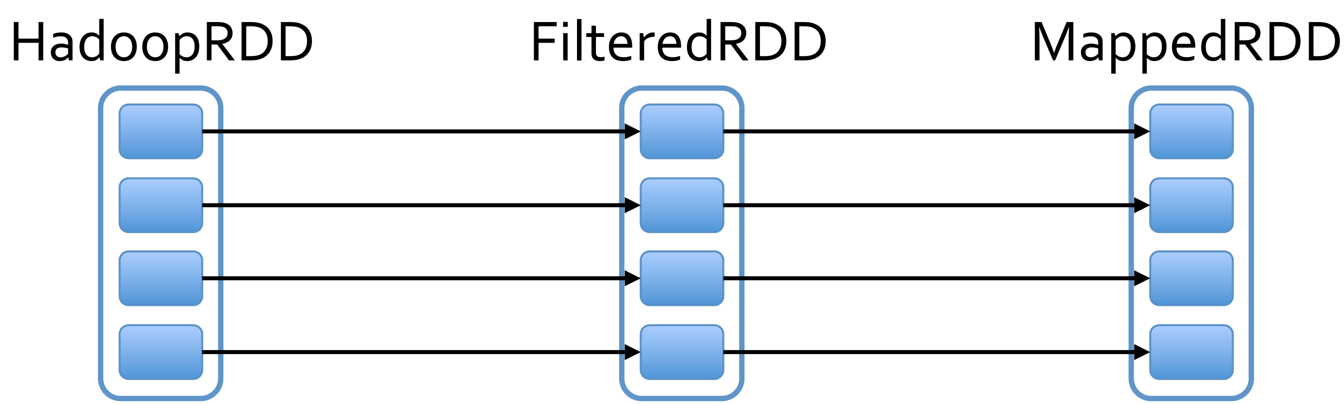
First action triggers RDD computation and load into memory

```
messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
```

RDD Fault Tolerance

- RDDs track the graph of operations used to construct them, called *lineage*.
- Lineage is used to rebuild data lost due to failures

`lines = spark.textFile("hdfs://...")`	HadoopRDD
`errors = lines.filter(_.startsWith("ERROR"))`	FilteredRDD
`messages = errors.map(_.split('\t')(2))`	MappedRDD



RDD Fault Tolerance

- The larger the lineage, more computation is needed, and thus recovery from failure will be longer.

- Therefore, RDDs only allow operations that touch a large number of records at the same time.

Transformations (define a new RDD)	map filter sample groupByKey reduceByKey sortByKey	flatMap union join cogroup cross mapValues
Actions (return a result to driver program)	collect reduce count save lookupKey	

RDD Fault Tolerance

- The larger the lineage, more computation is needed, and thus recovery from failure will be longer.

- Therefore, RDDs only allow operations that touch a large number of records at the same time.
  - Great for batch operations
  - Not so good for random access or asynchronous algorithms.

Iterative Computation

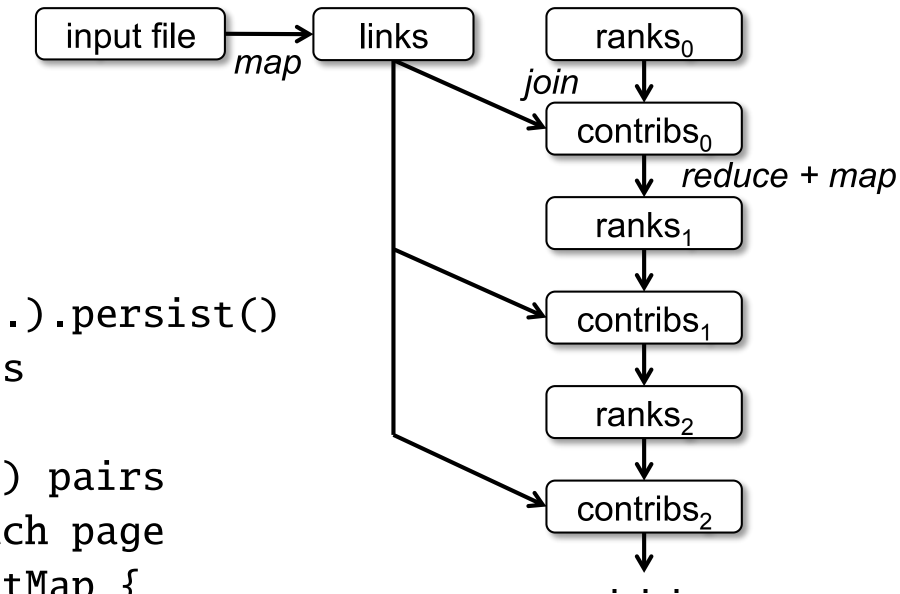
- Logistic Regression

```
val points = spark.textFile(...)
                  .map(parsePoint).persist()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

Page Rank

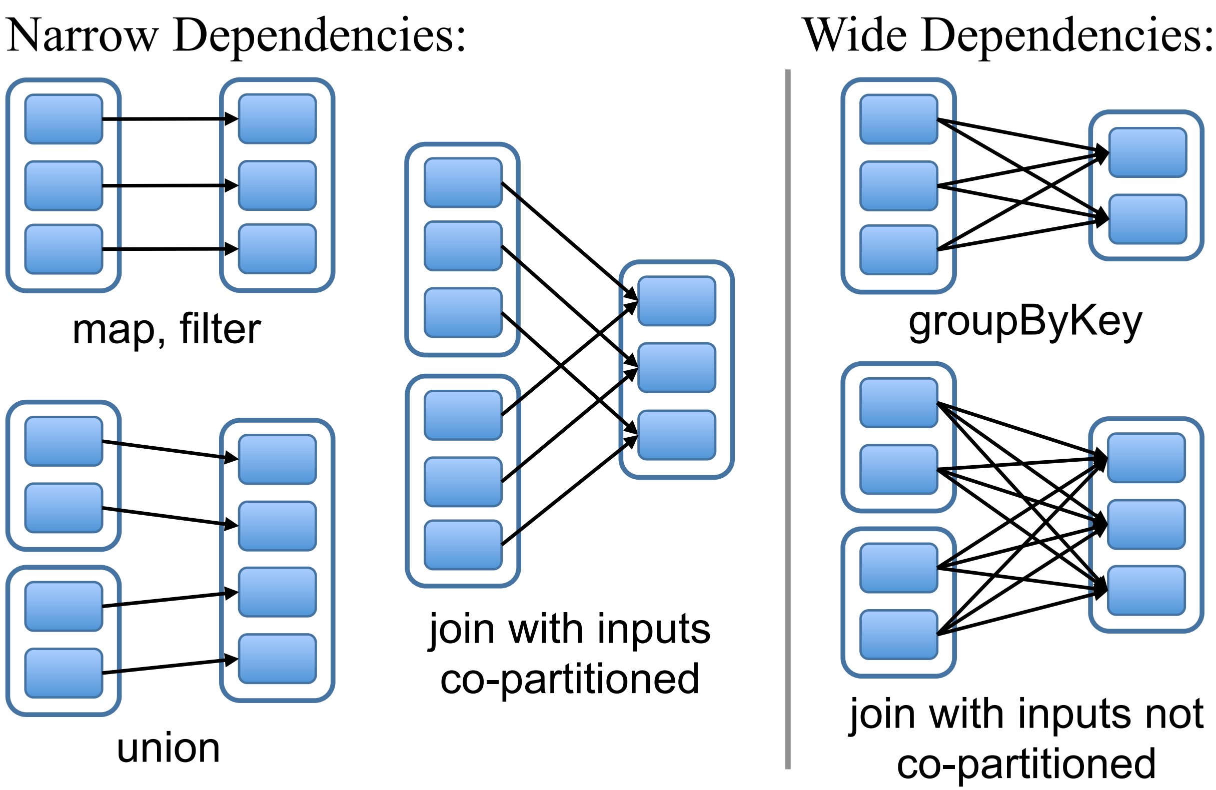
Lineage graphs can be long. Uses checkpointing in such cases.

```
val links = spark.textFile(...).map(...).persist()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS) {
  // Build an RDD of (targetURL, float) pairs
  // with the contributions sent by each page
  val contribs = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  // Sum contributions by URL and get new ranks
  ranks = contribs.reduceByKey((x,y) => x+y)
                  .mapValues(sum => a/N + (1-a)*sum)
}
```



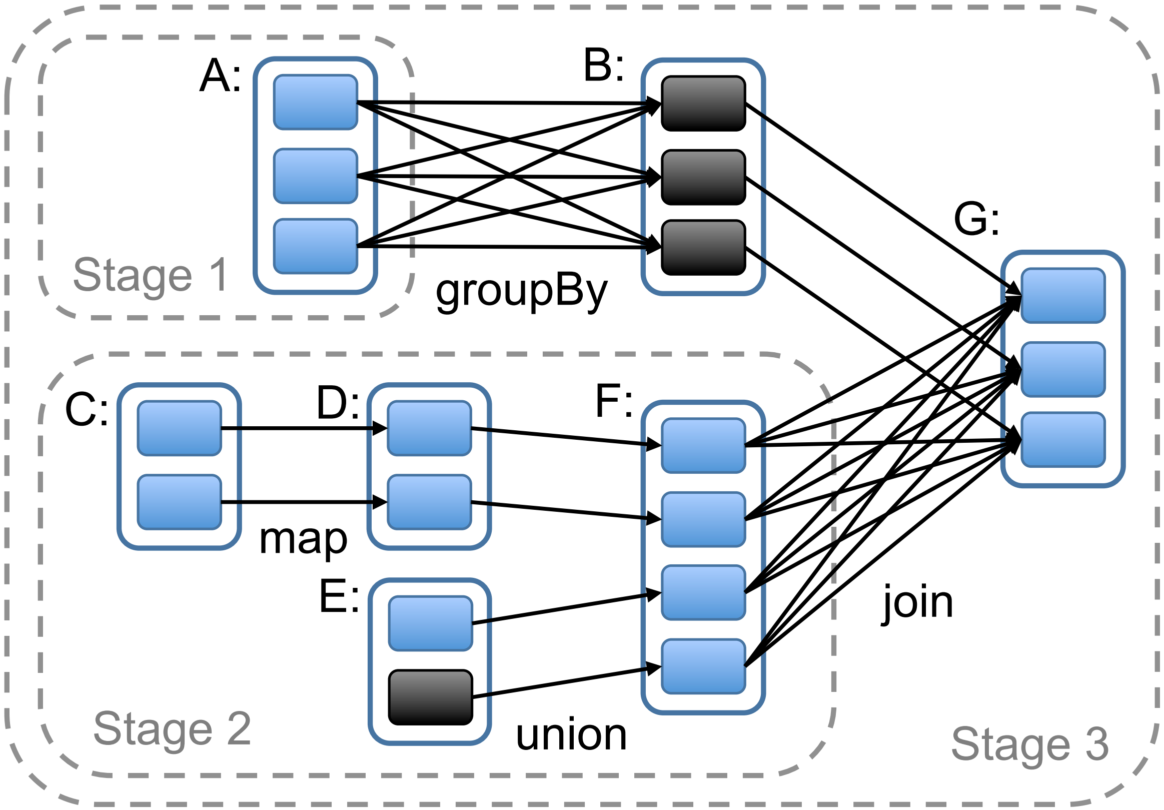
Duke UNIVERSITY

Transformations and Lineage Graphs



User can specify how data is partitioned to ensure narrow dependencies

Scheduling



Can pipeline execution as long as dependencies are narrow

Summary

- Map Reduce requires writing to disk for fault tolerance
- Not good for iterative computation.

RDD: Restricted form of distributed shared memory

- Data in memory is immutable
- Partitioned collection of records
- Can only be built through coarse grained deterministic transformations (map, filter, join, etc)

Fault Tolerance through lineage

- Maintain a small log of operations
- Recompute lost partitions when failures occur

Duke UNIVERSITY