Spark for Scientific Computing

A. Zonca, M. Tatineni - SDSC

What is Spark?

- A distributed computing framework

Problem 1: Storage

- Big data
- Commodity hardware (Cloud)

Solution: Distributed File System

- redundant
- fault tolerant

Problem 2: Computation

- Slow to move data across network
- Computations fail

Solution: Hadoop Mapreduce / Spark

- Execute computation where data are located
- Rerun failed jobs

Problem 3: Communication

- Most of the times, need to summarize data to get a result
- Reduction phase in MapReduce
- Need data transfer across network

Solution: highly optimized Shuffle (All-to-All)

Spark and Hadoop

- Works within the Hadoop ecosystem
- Extends MapReduce
- Initially developed at UC Berkeley
- Now within the Apache Foundation
- ~400 and more developers

Key features of Spark

- **Resiliency**: tolerant to node failures
- **Speed**: supports in-memory caching
- **Ease of use**:
  - Python/Scala interfaces
  - interactive shells
  - many distributed primitives

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Spark 100TB benchmark

HPC: Distributed TBs of data

- Fault-tolerant batch processing
- Data exploration with an interactive console
- SQL operations with Spark-SQL
- Iterative Machine Learning algorithms with Spark-MLlib

Comparison with MPI

- MPI: describe computation and communication explicitly
- Spark: use a graph of high-level operators, the framework decides how and where to run tasks

Million Songs Dataset

- Metadata about 1 million songs
- Fields like title, artist, year, loudness, hotness
- Created by Columbia University
- Dedicated to Machine Learning studies
- 280 GB compressed

Example Input

input file: “songs.tsv”

0	.4
1	.9
2	.2
3	.12
4	.55
5	.98

Example (Serial)

```
lines = open("songs.tsv").readlines()

def extract_hotness(line):
	return float(line.split()[1])

songs_hotness =map(extract_hotness, lines)
max_hotness = max(songs_hotness)
```

Example (Lambda)

```
def extract_hotness(line):
	return float(line.split()[1])

songs_hotness =map(extract_hotness, lines)

songs_hotness =map(lambda x:float(x.split()
[1]), lines)
```

Interactive spark on Comet

Add to .bashrc:
module load python scipy

Navigate to the comet-interactive/ folder,
submit a spark job with:
> qsub spark.cmd

Connect to Spark

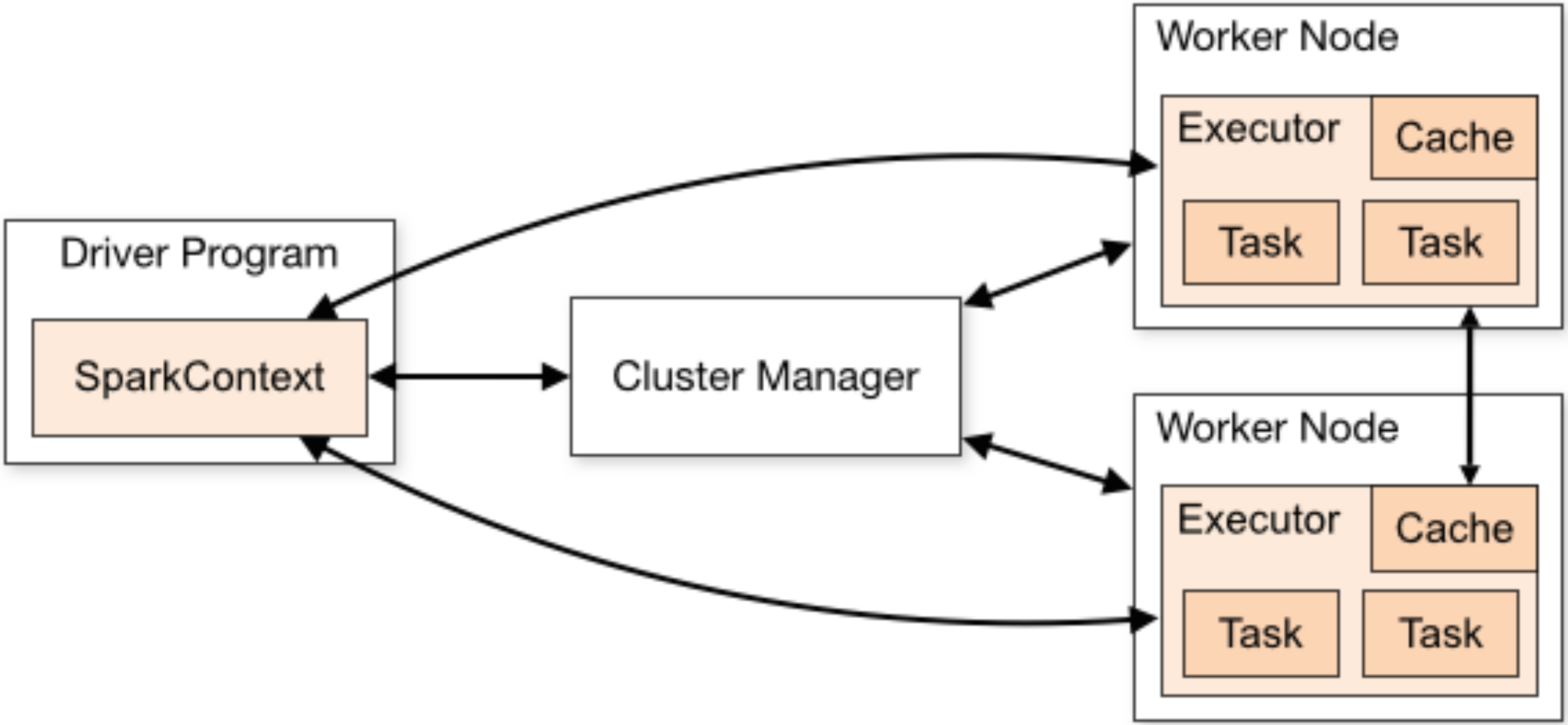
```
source slurm-env.sh
ssh $SLURMD_NODENAME

source slurm-env.sh
pyspark
```

First example

```
local_data = range(100)
data = sc.parallelize(local_data)
def myfilter(d):
    return d < 10
data.glom().collect()
data.filter(myfilter).collect()
```

Example (PySpark)

```
data = sc.textFile('songs.tsv')


def extract_hotness(line):
        return float(line.split()[1])


songs_hotness = data.map(extract_hotness)
max_hotness = songs_hotness.max()
```



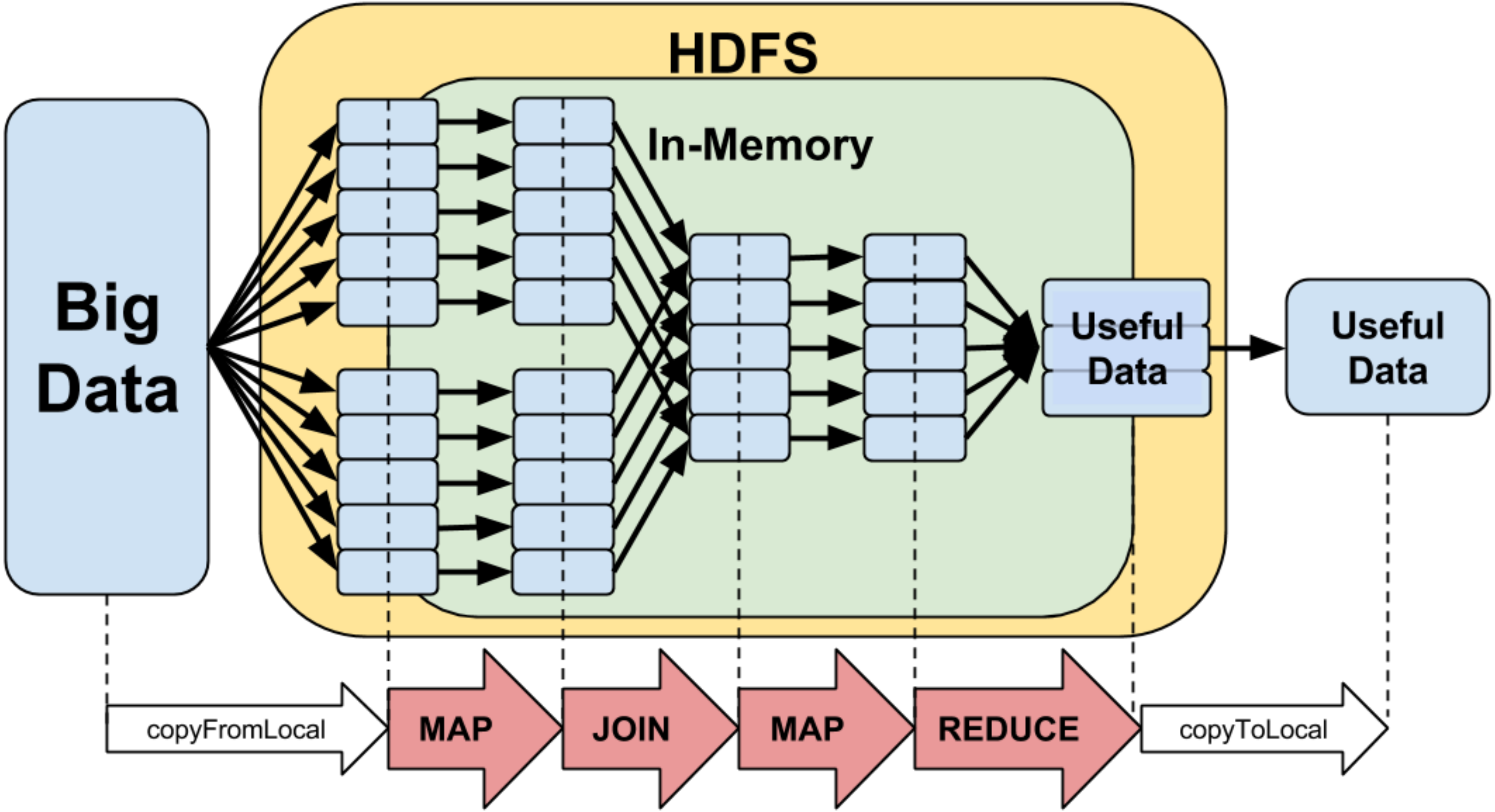
Spark architecture

Resilient Distributed Dataset

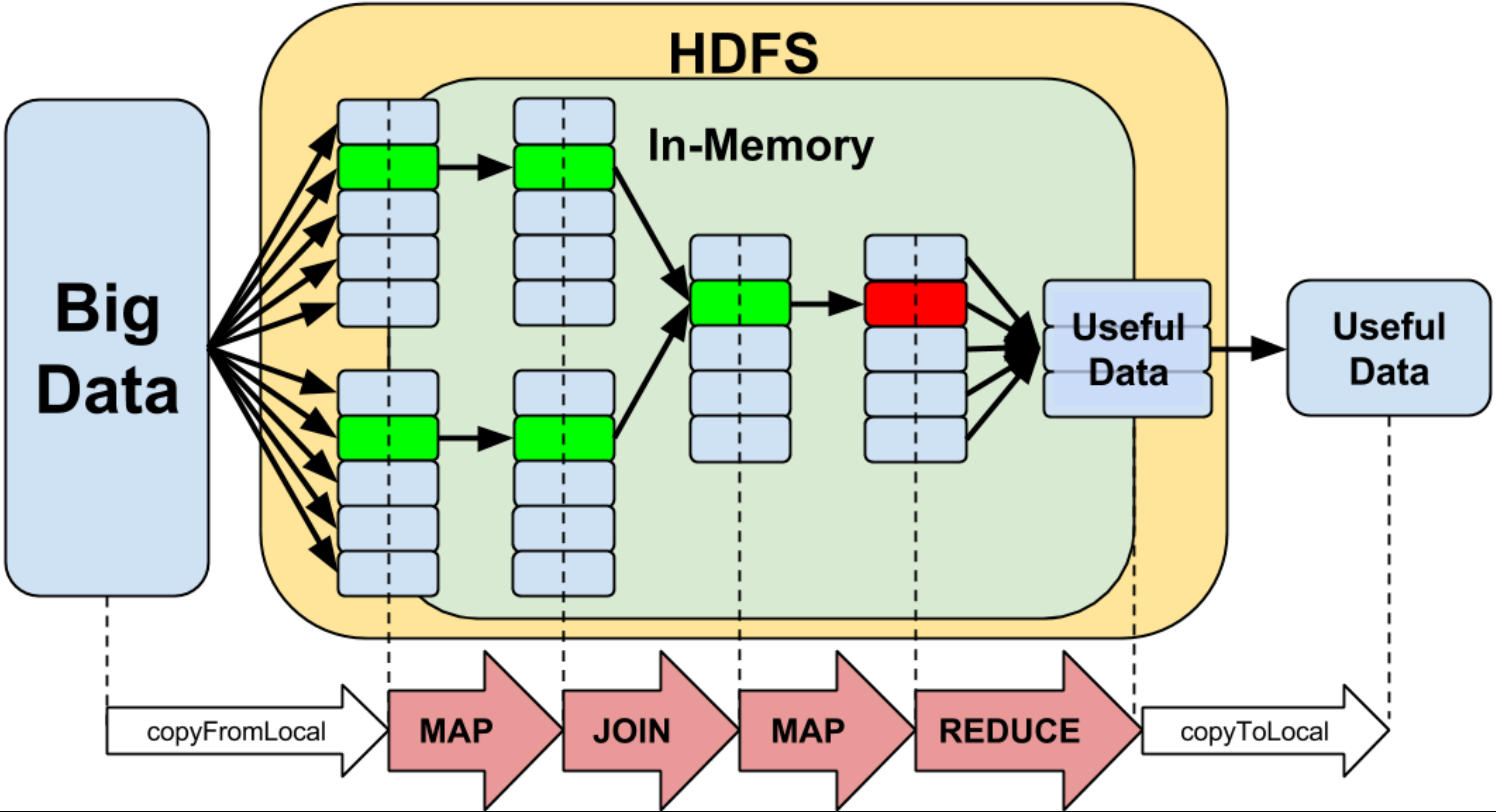
- Resilient: fault tolerant, lineage is saved, lost partitions can be recovered
- Distributed: partitions are automatically distributed across nodes
- Created from: HDFS, S3, HBase, Local file, Local hierarchy of folders

Read from HDFS

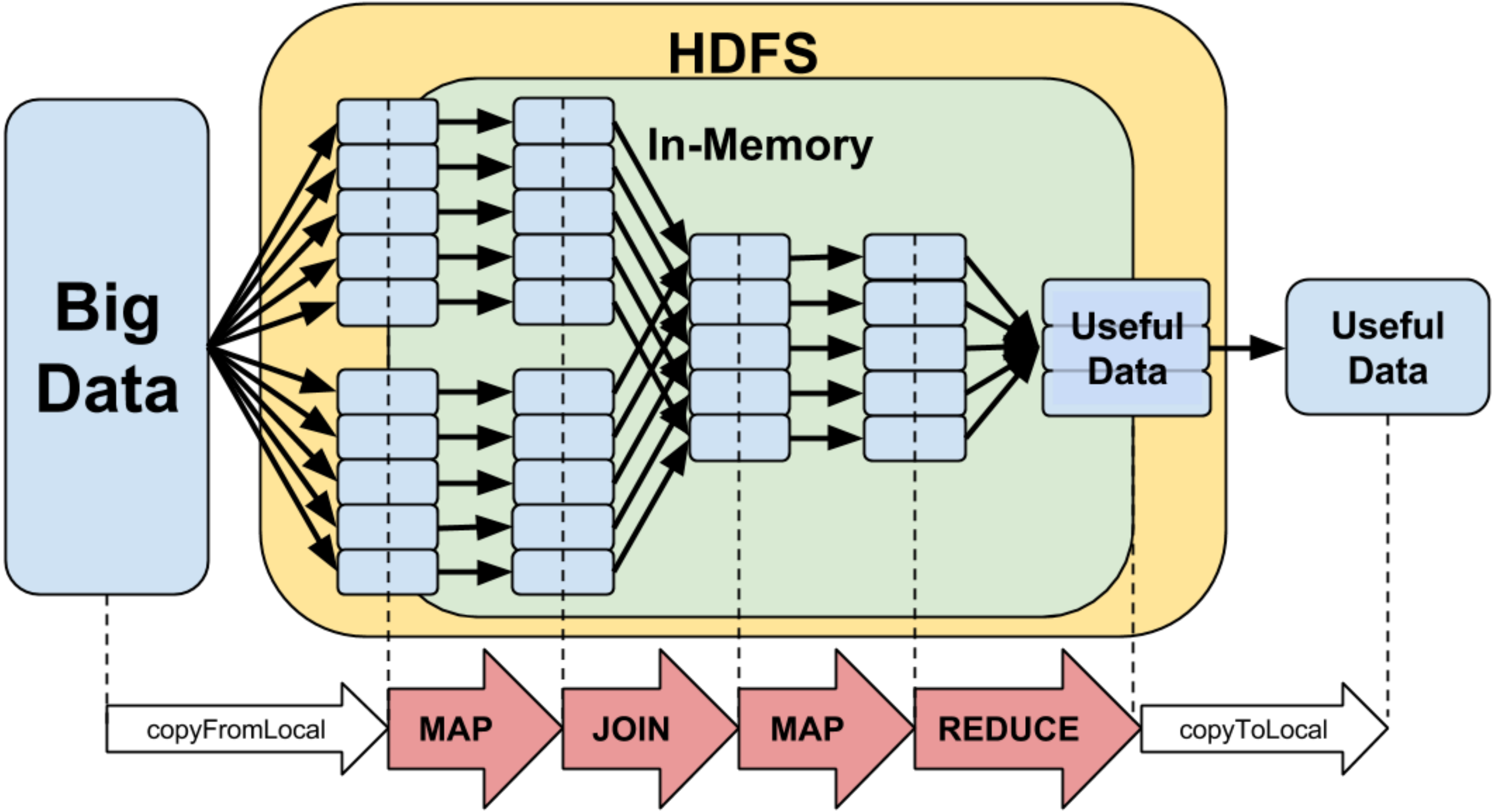
```
$ hdfs dfs -mkdir -p /user/$USER
$ hdfs dfs -put songs.tsv /user/$USER/

sc.textFile('hdfs:///user/%s/songs.tsv' % os.
environ["USER"])
```


HDFS
In-Memory
Big Data
Useful Data
Useful Data
copyFromLocal
MAP
JOIN
MAP
REDUCE
copyToLocal

Spark flow

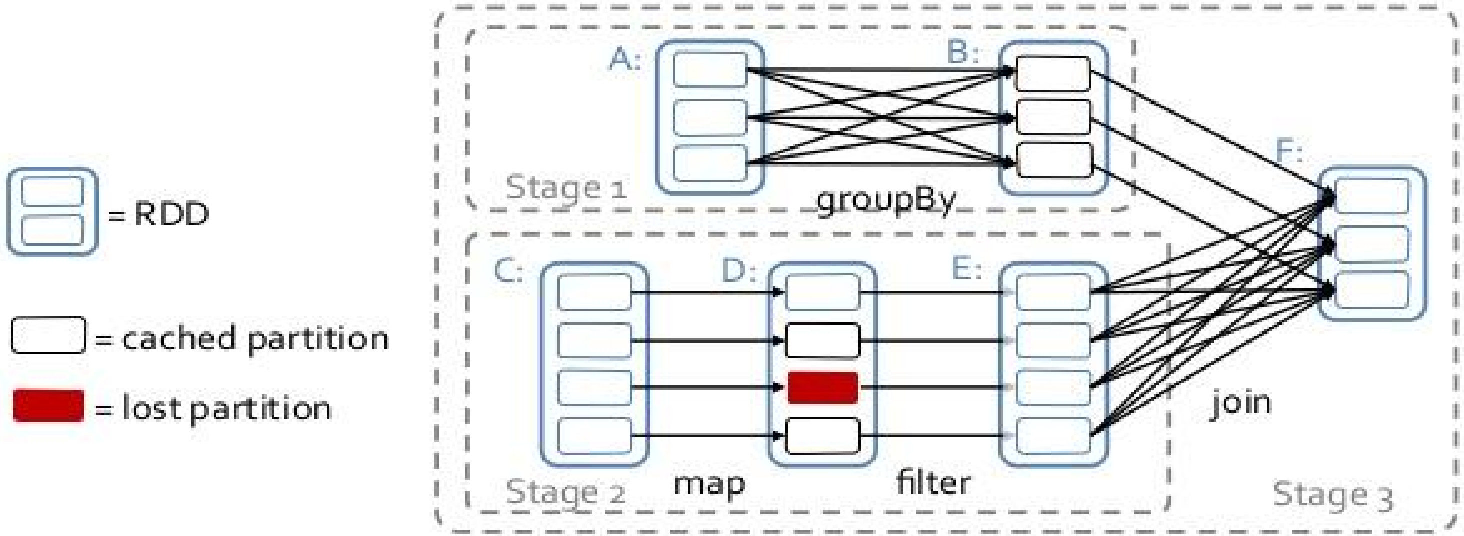

HDFS
In-Memory
Big Data
Useful Data
Useful Data
copyFromLocal
MAP
JOIN
MAP
REDUCE
copyToLocal

Resiliency



Transformations and actions

Example Stages



Transformations

- map(func) - apply func to all elements
- flatMap(func) - map then flatten output
- filter(func) - keep only elements where func is true
- sample(withReplacement, fraction, seed) - get a random data fraction

Hands-on

Print all the values of hotness larger than .5

Example (PySpark)

```
songs_hotness.filter(lambda x:x>.5).collect()
[0.9, 0.55, 0.98]
```

Transformations (2)

- union(otherDataset) - merge datasets
- coalesce(numPartitions) - decrease number of partitions

Extract data from RDD

- collect() - copy all elements to the driver
- take(n) - copy first n elements
- saveAsTextFile(filename) - save to file
- reduce(func) - aggregate elements with func (takes 2 elements, returns 1)

Cache data in memory

```
max_hotness = songs_hotness.max()
min_hotness = songs_hotness.min()
```

Each operation reads the data back again, Spark does not keep intermediate results in memory. Can trigger cache with:

```
songs_hotness.cache()
```

Cached RDD

- Generally recommended after data cleaning
- Reusing cached data: 10x speedup
- Great for iterative algorithms
- If RDD too large, will only be partially cached in memory

```python
data = sc.textFile('songs.tsv')


def extract_hotness(line):
        return float(line.split()[1])


songs_hotness = data.map(extract_hotness)
songs_hotness.cache()
max_hotness = songs_hotness.max()
```

Spark is lazy

Broadcast variables

- Large variable used in all nodes, possibly in many functions
- Transfer just once per Executor
- For example large configuration dictionary or lookup table

```
config = sc.broadcast({"order":3, "filter":True})
config.value
```

Accumulators

- Common pattern of accumulating to a variable across the cluster

```
accum = sc.accumulator(0)
sc.parallelize([1, 2, 3, 4]).foreach(lambda x:
accum.add(x))
accum.value
10
```

SparkUI

from your local machine:

$ ssh username@comet.sdsc.edu -L 4040:IP:
4040

Open browser at localhost:4040

Spark SQL

Tabular data processing in Spark

What is Spark SQL?

- High-level interface for structured data (i.e. tables)
- Provides Dataframes (data organized in columns), ~pandas, R
- Runs distributed SQL queries

Advantages of Spark SQL

- Same interface in Java/Scala/Python/R
- Native speed even in Python/R
- More expressive code -> easier to maintain

Spark SQL demo

Open pyspark

Hands-on

1. find maximum hotness for each decade
2. find how many songs for each decade

Spark MLlib

Machine Learning with Spark

Spark MLlib introduction

- Machine learning library
- Built on top of Spark
- Distributed linear algebra primitives:
  - Labeled points [y, X]
  - Dense vectors and matrices
  - Sparse vectors, matrices, block matrices

Spark MLlib features

- linear SVM and logistic regression
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power iteration clustering
- singular value decomposition

Spark MLlib demo

- open pyspark

Hands-on

- print the KMeans cluster centers (k=4) adding also the year as a features column

Thanks

Questions?

Andrea Zonca
zonca@sdsc.edu

View partitioning

- glom gathers all data in a partition as a list:

songs_hotness.repartition(2).glom().collect()

[[0.4, 0.12, 0.55], [0.98, 0.9, 0.2]]

songs_hotness.repartition(3).glom().collect()

[[0.12], [0.55, 0.4], [0.9, 0.2, 0.98]]