Introduction to Data Management
CSE 344

Lecture 26: More Spark

Announcements

- HW8 due Wednesday
  - Make sure you monitor your AWS usage!
- Final next Monday
  - 2:30 – 4:20pm, JHN 102
  - 2 sheets of notes
  - Review session this Saturday afternoon
  - Previous exams are posted on course website

Announcements

- No sections this week
- Wednesday will be our last lecture ☹
- Today: Spark

Issues with MapReduce

- Difficult to write more complex queries
- Need multiple MapReduce jobs: dramatically slows down because it writes all results to disk

Implementing Relational Operators in MapReduce

Given relations R(A,B) and S(B, C) compute:

- Selection: $\sigma_{A=123}(R)$
- Group-by: $\gamma_{A,sum(B)}(R)$
- Join: $R \bowtie S$

Selection $\sigma_{A=123}(R)$

```
map(String key, Tuple t):
    if  t.A = 123:
        EmitIntermediate(key, t);
```

```
reduce(String k, Iterator values):
    for each v in values:
        Emit(v);
```

Selection $\sigma_{A=123}(R)$

```
map(String key, Tuple t):
    if  t.A = 123:
        EmitIntermediate(key, t);
```

```
reduce(String k, Iterator values):
    for each v in values:
        Emit(v);
```

No need for reduce.
But need system hacking
to remove reduce from MapReduce

Spark Interface

- Spark supports a Scala interface
- Scala = extension of Java with closures

- We will illustrate Scala/Spark in the lectures

- Spark also supports a SQL interface, and compiles SQL to its Scala interface

- For HW8: you only need the SQL interface!

RDD

- RDD = Resilient Distributed Datasets
  - A distributed collection of data items, together with its *lineage*
  - Collection = list of ints, list of KV pairs, etc
  - Lineage = expression that says how that relation was computed = a relational algebra plan

- Spark stores intermediate results as RDD
- Spark operators take in (and generate) RDDs

RDD

- If a server crashes, its RDD in memory is lost
- But the master node knows the lineage, and will simply recompute the RDDs
  - Improve over MapReduce: we can recompute even within a map / reduce task
- How is this done?
  - Store intermediate RDDs to disk
  - Separate operators into lazy and eager
  - Construct a graph of operators

Programming in Spark

- A Spark/Scala program consists of:
  - Transformations (`map, reduceByKey, join…`). Lazy
  - Actions (`count, reduce, save…`). Eager
- RDD[T] = an RDD collection of type T
  - Partitioned, recoverable (through lineage), not nested
- Seq[T] = a Scala sequence
  - Local to a server, may be nested

Scala Primer

- Functions with one argument:

```
_.contains(“sqlite”)
_ > 6
```

- Functions with more arguments

```
(x => x.contains(“sqlite”))
(x => x > 6)
((x,y) => x+3*y)
```

- Closures (functions with free variables):

```
var x = 5; rdd.filter(_ > x)
var s = “sqlite”; rdd.filter(x => x.contains(s))
```

Example

Given a large log file hdfs://logfile.log retrieve all lines that:

- Start with ERROR
- Contain the string “sqlite”

```
lines = spark.textFile(“hdfs://logfile.log”);

errors = lines.filter(_.startsWith(“ERROR”));

sqlerrors = errors.filter(_.contains(“sqlite”));

sqlerrors.collect()
```

Example

Given a large log file hdfs://logfile.log retrieve all lines that:

- Start with ERROR
- Contain the string “sqlite”

```
lines = spark.textFile(“hdfs://logfile.log”);

errors = lines.filter(_.startsWith(“ERROR”));

sqlerrors = errors.filter(_.contains(“sqlite”));

sqlerrors.collect()
```

Transformations: Not executed yet…

Action: triggers execution of entire program

Example

```
lines = spark.textFile("hdfs://logfile.log");

errors = lines.filter(_.startsWith("ERROR"));

sqlerrors = errors.filter(_.contains("sqlite"));

sqlerrors.collect();
```

```
lines = spark.textFile("hdfs://logfile.log")
             .filter(_.startsWith("ERROR"))
             .filter(_.contains("sqlite"))
             .collect();
```

MapReduce Again…

Steps in Spark resemble MapReduce:

- `col.filter(p)` applies in parallel the predicate p to all elements x of the partitioned collection, and returns collection with those x where `p(x) = true`
- `col.map(f)` applies in parallel the function f to all elements x of the partitioned collection, and returns a new partitioned collection

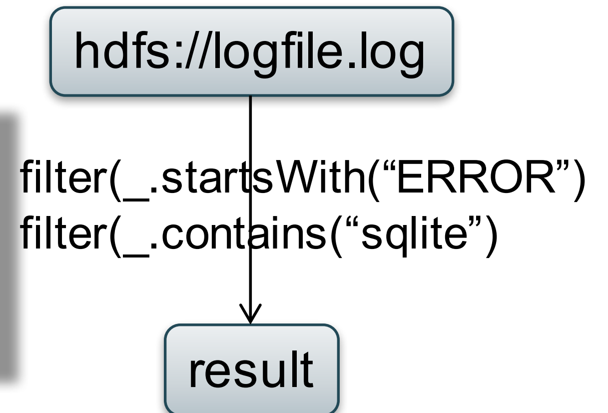
Persistence

```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```

If any server fails before the end, then Spark must restart

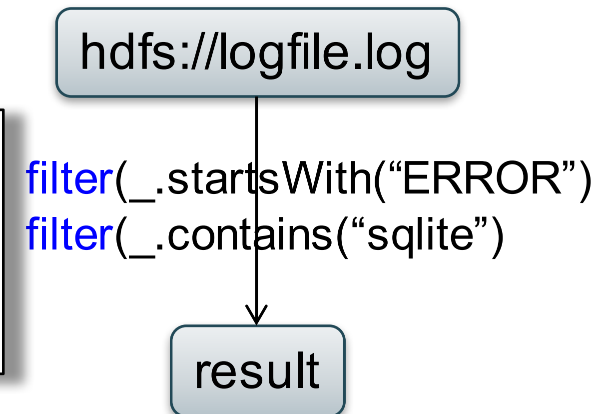
Persistence

```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```



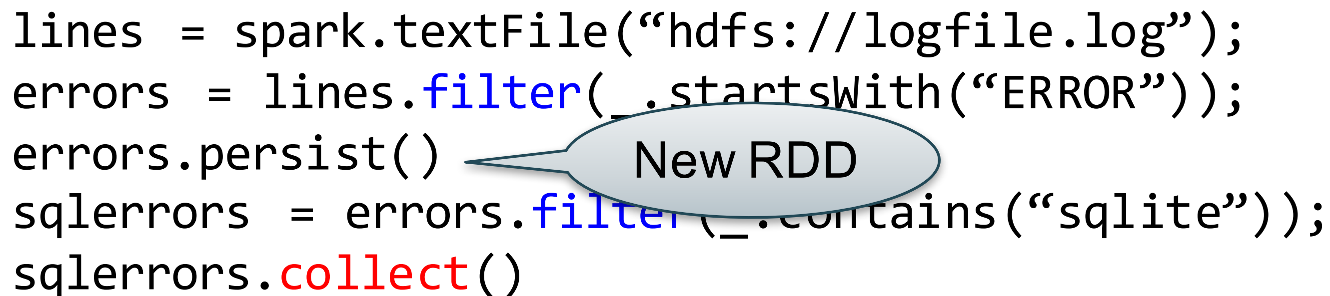
If any server fails before the end, then Spark must restart

Persistence




```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```

If any server fails before the end, then Spark must restart

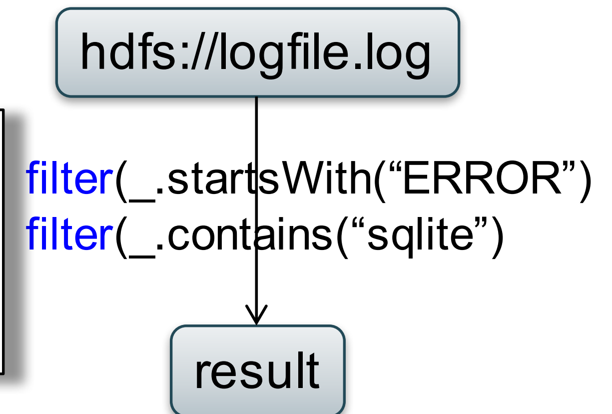



```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
errors.persist()
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```

Spark can recompute the result from errors

Persistence

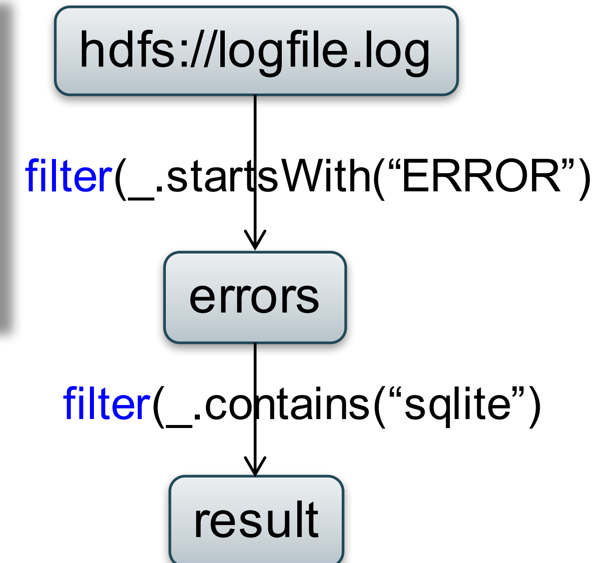
```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```



If any server fails before the end, then Spark must restart

```
lines = spark.textFile("hdfs://logfile.log");
errors = lines.filter(_.startsWith("ERROR"));
errors.persist()
sqlerrors = errors.filter(_.contains("sqlite"));
sqlerrors.collect()
```

New RDD



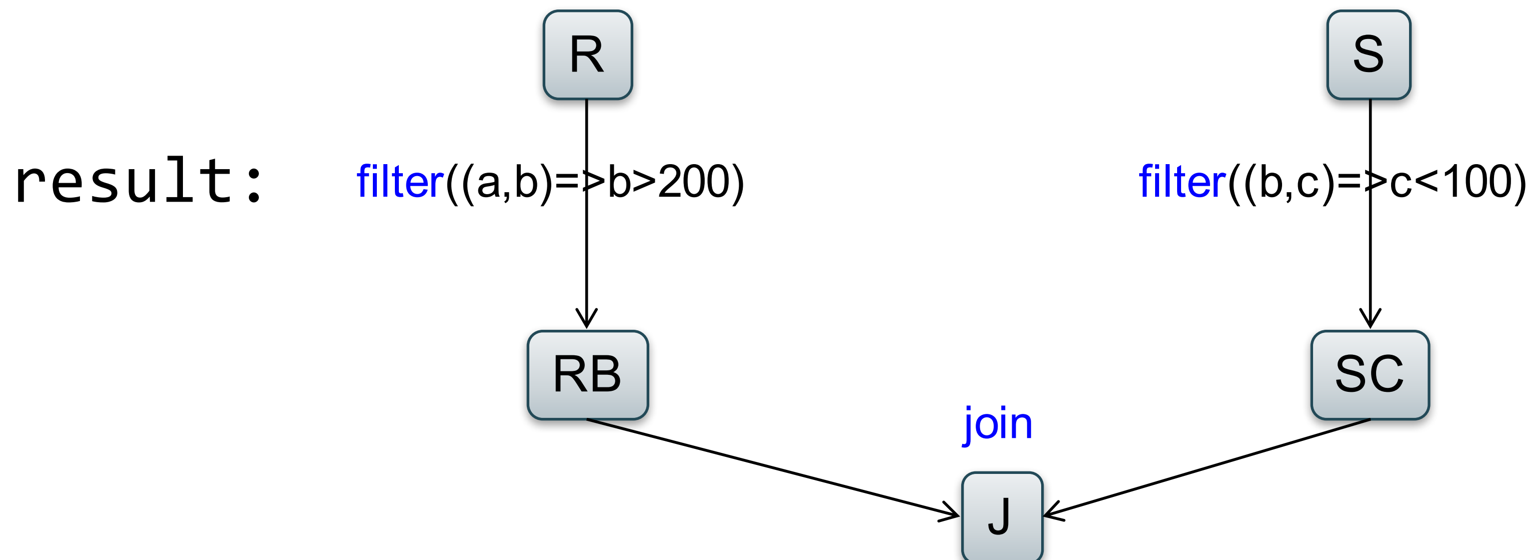
Spark can recompute the result from errors

R(A,B)
S(A,C)

```
SELECT count(*)  FROM R, S
WHERE R.B > 200 and S.C < 100  and R.A = S.A
```

Example

```
R = spark.textFile(“R.csv”).map(parseRecord).persist();
S = spark.textFile(“S.csv”).map(parseRecord).persist();
RB = R.filter((a,b) => b > 200).persist();
SC = S.filter((a,c) => c < 100).persist();
J = RB.join(SC).persist();
result = J.count();
```



Transformations:	
map(f : T => U):	RDD[T] => RDD[U]
flatMap(f: T => Seq(U)):	RDD[T] => RDD[U]
filter(f:T=>Bool):	RDD[T] => RDD[T]
groupByKey():	RDD[(K,V)] => RDD[(K,Seq[V])]
reduceByKey(F:(V,V)=> V):	RDD[(K,V)] => RDD[(K,V)]
union():	(RDD[T],RDD[T]) => RDD[T]
join():	(RDD[(K,V)],RDD[(K,W)]) => RDD[(K,(V,W))]
cogroup():	(RDD[(K,V)],RDD[(K,W)]) => RDD[(K,(Seq[V],Seq[W]))]
crossProduct():	(RDD[T],RDD[U]) => RDD[(T,U)]

Outputs 1 object per input

Output multiple objects per input

Actions:	
count():	RDD[T] => Long
collect():	RDD[T] => Seq[T]
reduce(f:(T,T)=>T):	RDD[T] => T
save(path:String):	Outputs RDD to a storage system e.g. HDFS

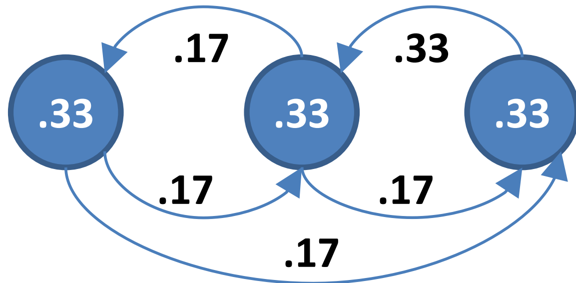
Outputs 1 object per input

An Example: PageRank

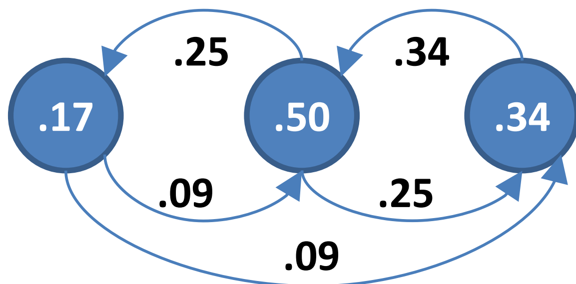
- Page Rank is an algorithm that assigns to each page a score such that pages have higher scores if more pages with high scores link to them

- Page Rank was introduced by Google, and, essentially, defined Google

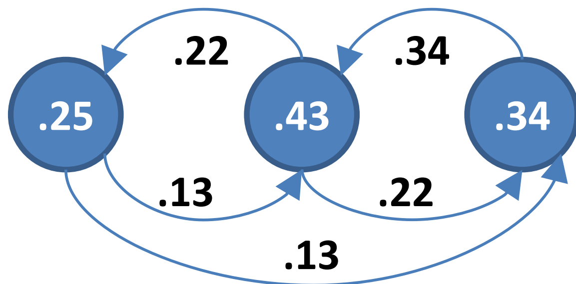
PageRank toy example



Superstep 0

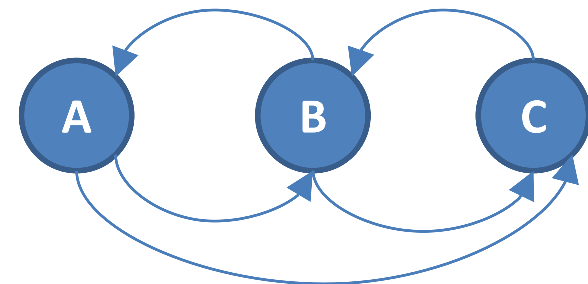


Superstep 1



Superstep 2

Input graph



http://www.slideshare.net/sscdotopen/large-scale/20