

CS 744: RESILIENT DISTRIBUTED DATASETS

Shivaram Venkataraman

Fall 2021

ADMINISTRIVIA

- Assignment 1: Due Sep 28, Tuesday at 10pm!
- Assignment 2: ML will be released Sep 29

- REMINDER: Submit your discussions
 - Within 24 hrs after end of class (11am next day)
 - Each student needs to submit

- Course project details: Next week

MOTIVATION: PROGRAMMABILITY

Most real applications require multiple MR steps

- Google indexing pipeline: 21 steps
- Analytics queries (e.g. sessions, top K): 2-5 steps
- Iterative algorithms (e.g. PageRank): 10's of steps

Multi-step jobs create spaghetti code

- 21 MR steps → 21 mapper and reducer classes

MOTIVATION: PERFORMANCE

MR only provides one pass of computation

- Must write out data to file system in-between

Expensive for apps that need to *reuse* data

- Multi-step algorithms (e.g. PageRank)
- Interactive data mining

PROGRAMMABILITY

Google MapReduce WordCount:

```
• #include "mapreduce/mapreduce.h"
•
• // User's map function
• class Splitwords: public Mapper {
• public:
•     virtual void Map(const MapInput& input)
•     {
•         const string& text = input.value();
•         const int n = text.size();
•         for (int i = 0; i < n; ) {
•             // Skip past leading whitespace
•             while (i < n && isspace(text[i]))
•                 i++;
•             // Find word end
•             int start = i;
•             while (i < n && !isspace(text[i]))
•                 i++;
•             if (start < i)
•                 Emit(text.substr(
•                     start,i-start),"1");
•         }
•     }
• };
•
• REGISTER_MAPPER(Splitwords);
•
• // User's reduce function
• class Sum: public Reducer {
• public:
•     virtual void Reduce(ReduceInput* input)
•     {
•         // Iterate over all entries with the
•         // same key and add the values
•         int64 value = 0;
•         while (!input->done()) {
•             value += StringToInt(
•                 input->value());
•             input->NextValue();
•         }
•         // Emit sum for input->key()
•         Emit(IntToString(value));
•     }
• };
•
• REGISTER_REDUCER(Sum);
•
• int main(int argc, char** argv) {
•     ParseCommandLineFlags(argc, argv);
•
•     MapReduceSpecification spec;
•     for (int i = 1; i < argc; i++) {
•         MapReduceInput* in= spec.add_input();
•         in->set_format("text");
•         in->set_filepattern(argv[i]);
•         in->set_mapper_class("Splitwords");
•     }
•
•     // Specify the output files
•     MapReduceOutput* out = spec.output();
•     out->set_filebase("/gfs/test/freq");
•     out->set_num_tasks(100);
•     out->set_format("text");
•     out->set_reducer_class("Sum");
•
•     // Do partial sums within map
•     out->set_combiner_class("Sum");
•
•     // Tuning parameters
•     spec.set_machines(2000);
•     spec.set_map_megabytes(100);
•     spec.set_reduce_megabytes(100);
•
•     // Now run it
•     MapReduceResult result;
•     if (!MapReduce(spec, &result)) abort();
•     return 0;
• }
```

APACHE SPARK PROGRAMMABILITY

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)

counts.save("out.txt")
```

APACHE SPARK

Programmability: clean, functional API

- Parallel transformations on collections
- 5-10x less code than MR
- Available in Scala, Java, Python and R

Performance

- In-memory computing primitives
- Optimization across operators



SPARK CONCEPTS

Resilient distributed datasets (RDDs)

- Immutable, partitioned collections of objects
- May be cached in memory for fast reuse

Operations on RDDs

- *Transformations* (build RDDs)
- *Actions* (compute results)

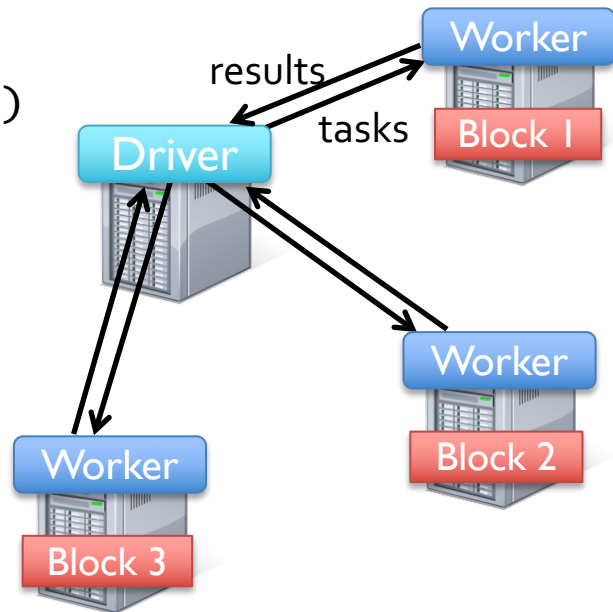
Restricted shared variables

- Broadcast, accumulators

EXAMPLE: LOG MINING

Find error messages present in log files interactively
(Example: HTTP server logs)

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

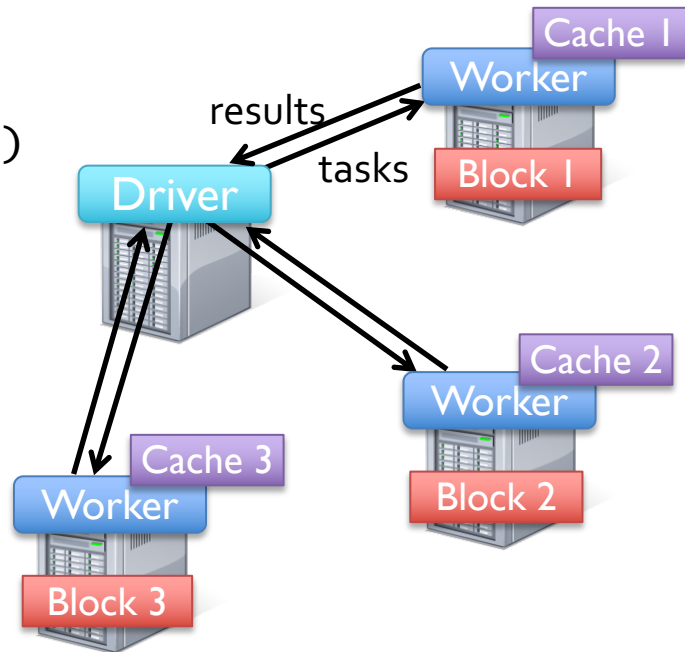


EXAMPLE: LOG MINING

Find error messages present in log files interactively
(Example: HTTP server logs)

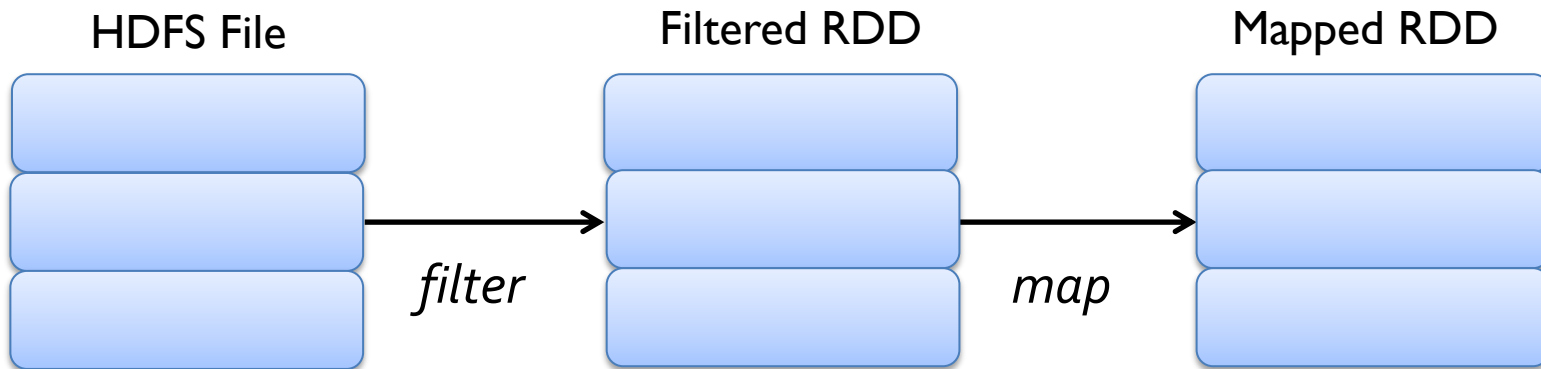
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
messages.cache()
messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
. . .
```

Result: search 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



FAULT RECOVERY

```
messages = textFile(...).filter(_.startsWith("ERROR"))  
                        .map(_.split('\t')(2))
```

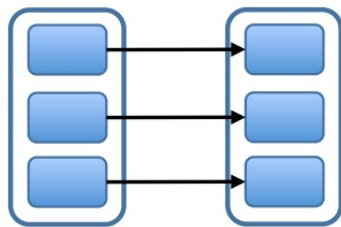


OTHER RDD OPERATIONS

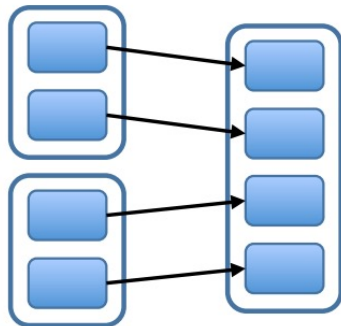
Transformations (define a new RDD)	map filter sample groupByKey reduceByKey cogroup	flatMap union join cross mapValues ...
Actions (output a result)	collect reduce take fold	count saveAsTextFile saveAsHadoopFile ...

DEPENDENCIES

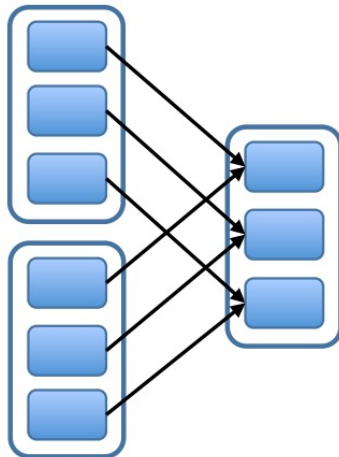
Narrow Dependencies:



map, filter

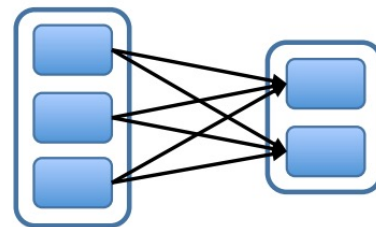


union

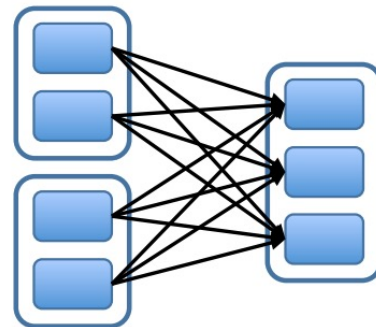


join with inputs
co-partitioned

Wide Dependencies:



groupByKey

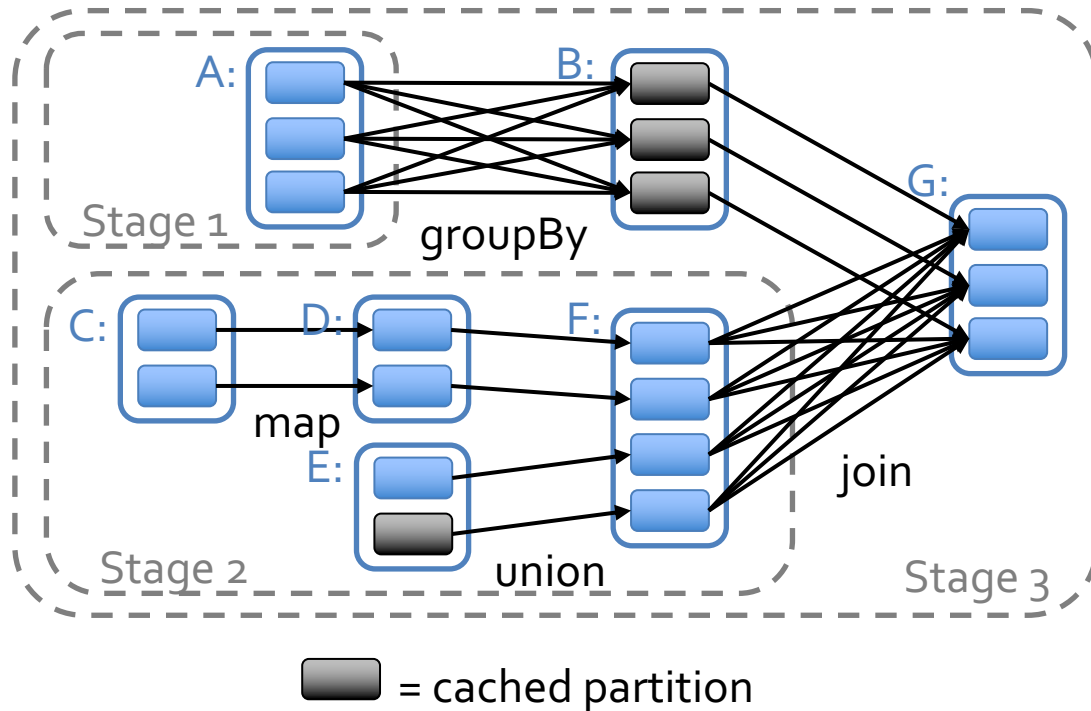


join with inputs not
co-partitioned

JOB SCHEDULER (1)

Captures RDD dependency graph

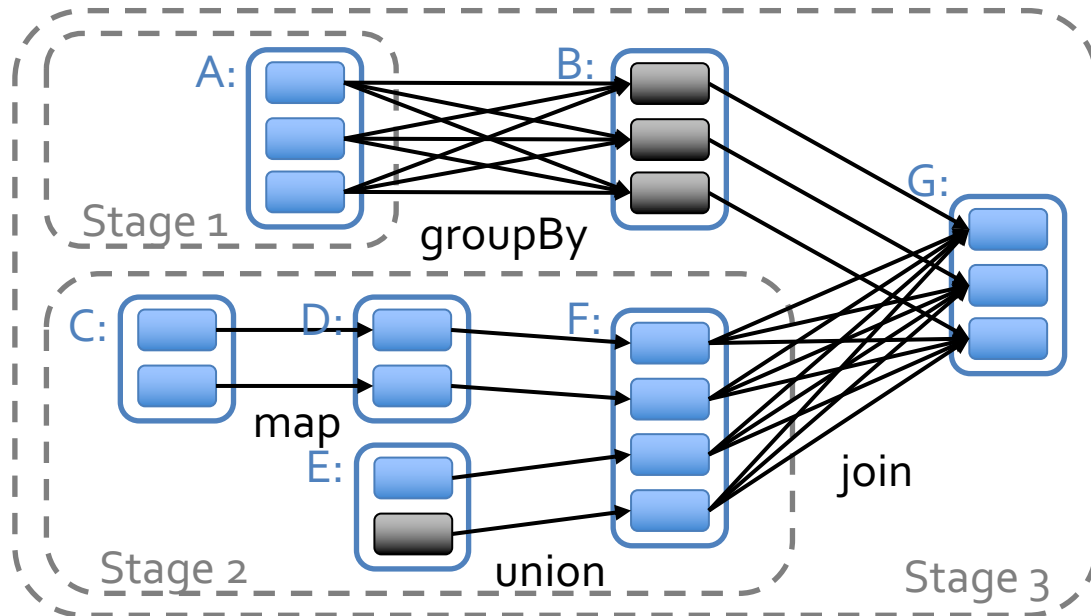
Pipelines functions into “stages”



JOB SCHEDULER (2)

Cache-aware for data reuse, locality

Partitioning-aware to avoid shuffles



SUMMARY

Spark: Generalize MR programming model

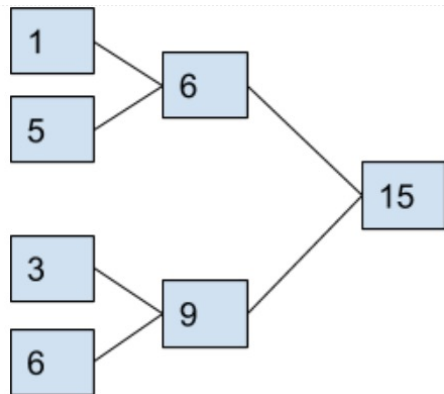
Support in-memory computations with RDDs

Job Scheduler: Pipelining, locality-aware

DISCUSSION

<https://forms.gle/nPdJYq9D4nE4gtAD9>

```
for (i <- 1 to numIters) {  
  val modelBC = sc.broadcast(model)  
  val grad = data.mapPartitions(iter => gradient(iter, modelBC.value))  
  val aggGrad = grad.reduce(case(x, y) => add(x, y))  
  model = computeUpdate(aggGrad, model)  
}
```



When would reduction trees be better than using `reduce` in Spark?
When would they not be ?

NEXT STEPS

Next week: Resource Management

- Mesos
- DRF

Assignment 1 is due soon!

CHECKPOINTING

```
rdd = sc.parallelize(1 to 100, 2).map(x → 2*x)  
rdd.checkpoint()
```