Spark Performance

Patrick Wendell
Databricks

About me

Work on performance benchmarking and testing in Spark

Co-author of spark-perf

Wrote instrumentation/UI components in Spark

This talk

Geared towards existing users

Current as of Spark 0.8.1

Outline

Part 1: Spark deep dive

Part 2: Overview of UI and instrumentation

Part 3: Common performance mistakes

databricks™

Why gain a deeper understanding?

RDD

(patrick, $24), (matei, $30), (patrick, $1), (aaron, $23), (aaron, $2), (reynold, $10), (aaron, $10)…..

```
spendPerUser = rdd
.groupByKey()
.map(lambda pair: sum
.collect()
```

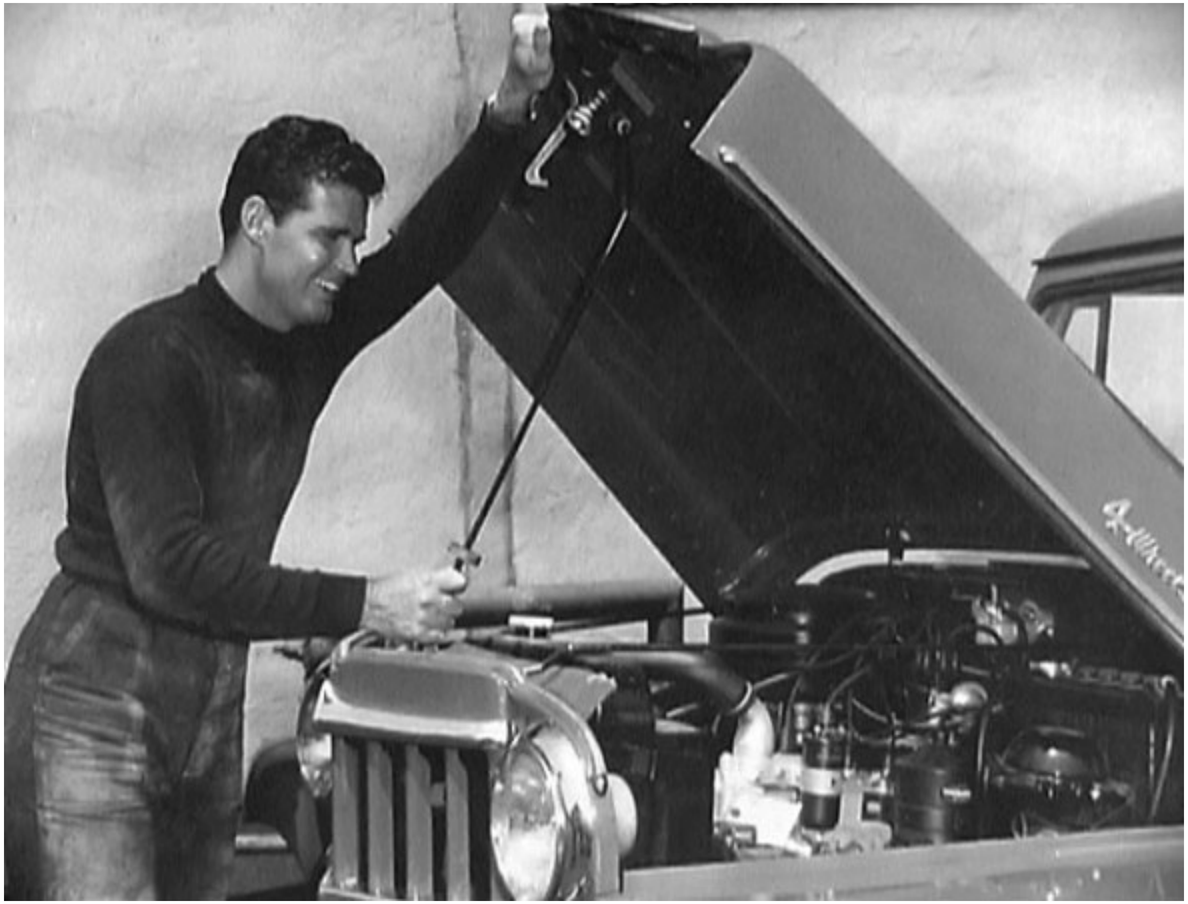
Copies all data over the network

```
spendPerUser = rdd
.reduceByKey(lambda x,
.collect()
```

Reduces locally before shuffling

databricks™

Let’s look under the hood



How Spark works

RDD: a parallel collection w/ partitions

User application creates RDDs, transforms them, and runs actions

These result in a DAG of operators

DAG is compiled into stages

Each stage is executed as a series of tasks

Example

```
sc.textFile("/some-hdfs-data")
   .map(line => line.split("\t"))
   .map(parts =>
     (parts[0], int(parts[1])))
   .reduceByKey(_ + _, 3)
   .collect()
```

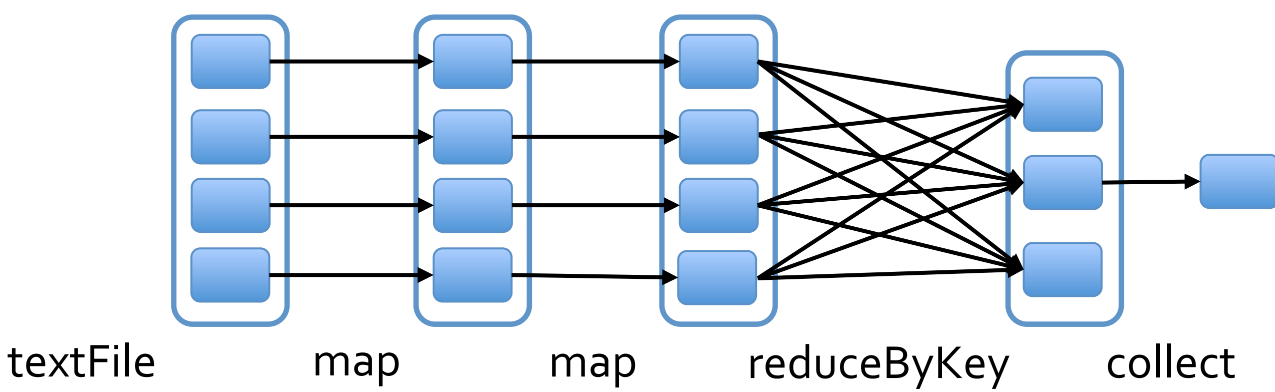
RDD[String]

RDD[List[String]]

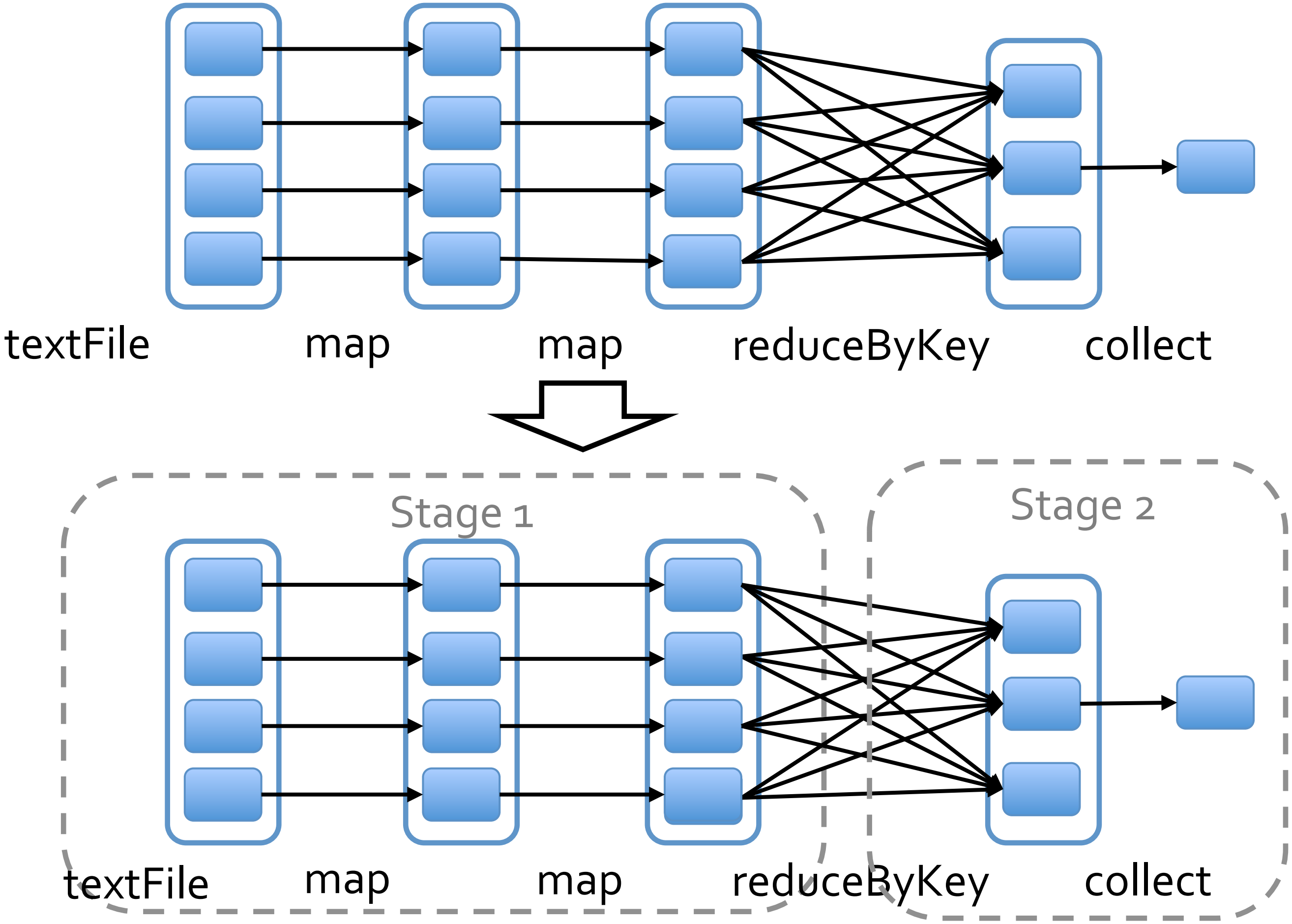
RDD[(String, Int)]

RDD[(String, Int)]

Array[(String, Int)]

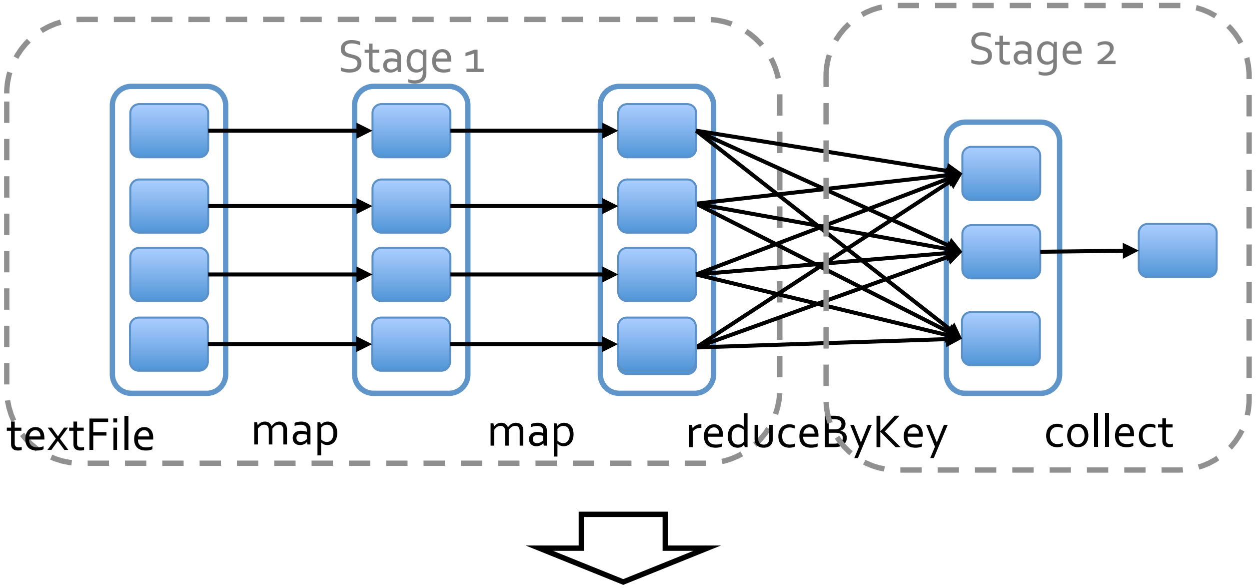


Execution Graph

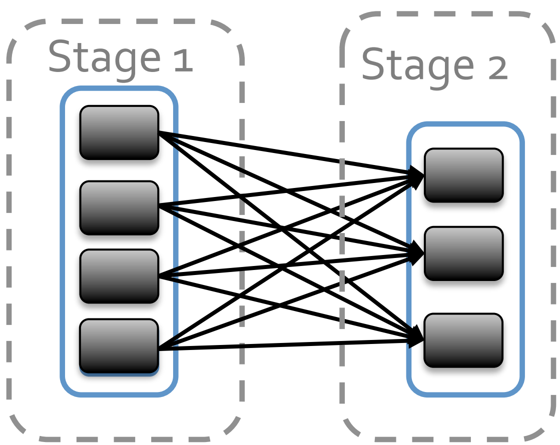


databricks™

Execution Graph



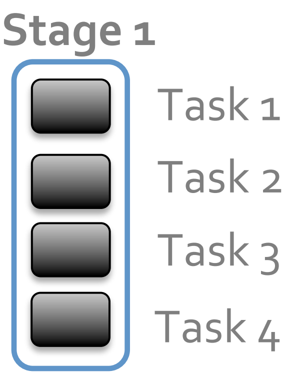
read HDFS split
apply both maps
partial reduce
write shuffle data



read shuffle data
final reduce
send result to driver

databricks™

Stage execution



Create a task for each partition in the new RDD

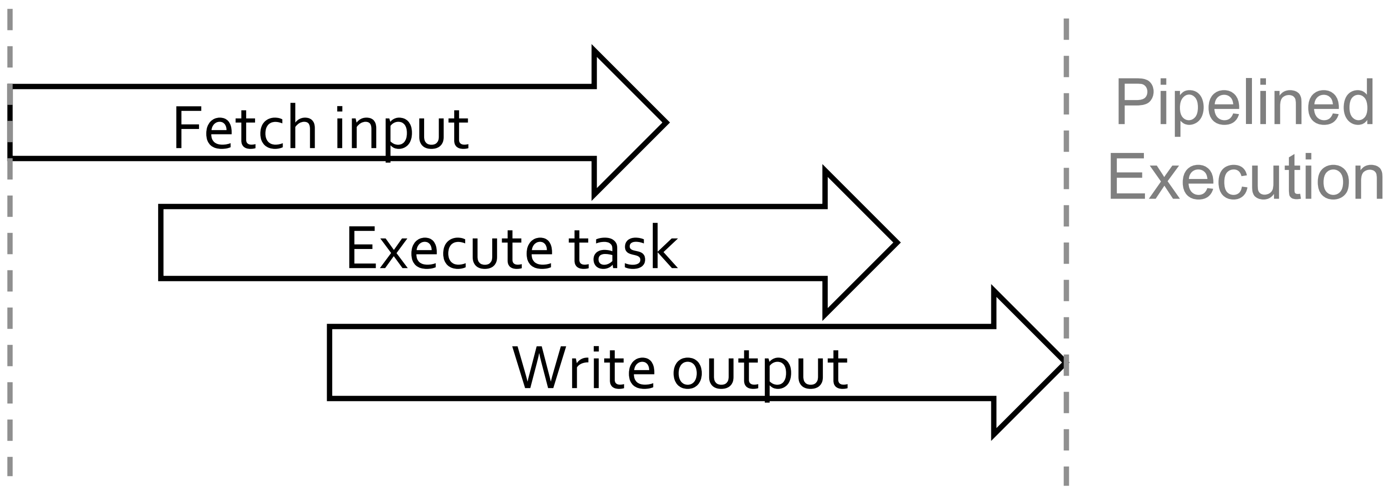
Serialize task

Schedule and ship task to slaves

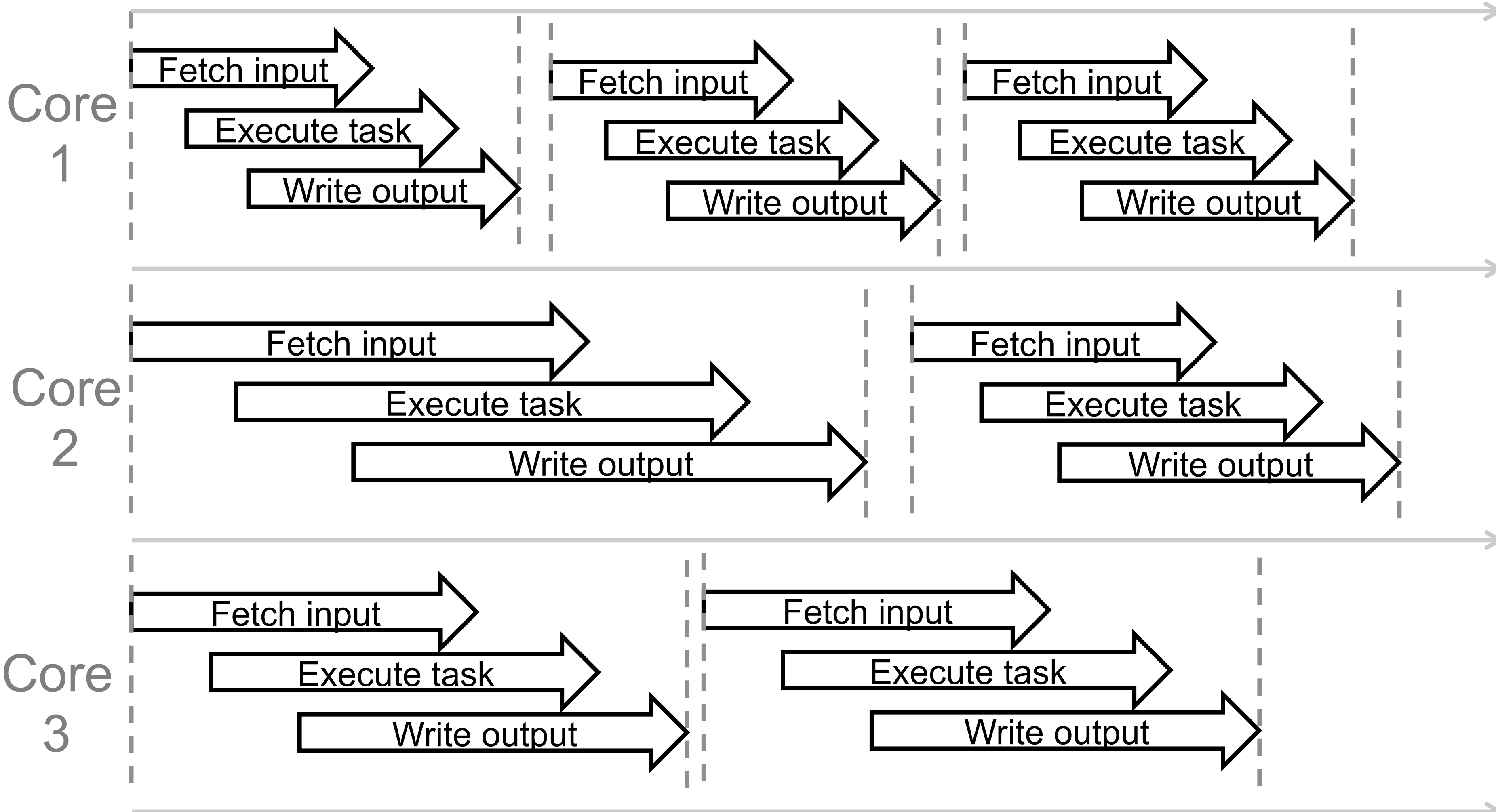
Task execution

Fundamental unit of execution in Spark

- A. Fetch input from InputFormat or a shuffle
- B. Execute the task
- C. Materialize task output as shuffle or driver result



Spark Executor



databricks™

Summary of Components

Tasks: Fundamental unit of work

Stage: Set of tasks that run in parallel

DAG: Logical graph of RDD operations

RDD: Parallel dataset with partitions

Demo of perf UI

Where can you have problems?

1. Scheduling and launching tasks
2. Execution of tasks
3. Writing data between stages
4. Collecting results

1. Scheduling and launching tasks

Serialized task is large due to a closure

```
hash_map = some_massive_hash_map()

rdd.map(lambda x: hash_map(x))
    .count_by_value()
```

Detecting: Spark will warn you! (starting in 0.9…)

Fixing

Use broadcast variables for large object

Make your large object into an RDD

Large number of "empty" tasks due to selective filter

```
rdd = sc.textFile("s3n://bucket/2013-data")
          .map(lambda x: x.split("\t"))
          .filter(lambda parts: parts[0] == "2013-10-17")
          .filter(lambda parts: parts[1] == "19:00")

rdd.map(lambda parts: (parts[2], parts[3]).reduceBy…
```

Detecting Many short-lived (< 20ms) tasks

Fixing

Use `coalesce` or `repartition` operator to shrink RDD number of partitions after filtering:

rdd.coalesce(30).map(lambda parts: (parts[2]…

databricks™

2. Execution of Tasks

Tasks with high per-record overhead

```
rdd.map(lambda x:
  conn = new_mongo_db_cursor()
  conn.write(str(x))
  conn.close())
```

Detecting: Task run time is high

Fixing

Use `mapPartitions` or `mapWith` (scala)

```
rdd.mapPartitions(lambda records:
  conn = new_mong_db_cursor()
  [conn.write(str(x)) for x in records]
  conn.close())
```

databricks™

Skew between tasks

Detecting
Stage response time dominated by a few slow tasks

Fixing
Data skew: poor choice of partition key
→ Consider different way of parallelizing the problem
→ Can also use intermediate partial aggregations

Worker skew: some executors slow/flakey nodes
→ Set spark.speculation to true
→ Remove flakey/slow nodes over time

3. Writing data between stages

Not having enough buffer cache

spark writes out shuffle data to OS-buffer cache

Detecting
tasks spend a lot of time writing shuffle data

Fixing
if running large shuffles on large heaps, allow several GB for buffer cash

rule of thumb, leave 20% of memory free for OS and caches

Not setting spark.local.dir

spark.local.dir is where shuffle files are written

ideally a dedicated disk or set of disks

spark.local.dir=/mnt1/spark,/mnt2/spark,/mnt3/spark

mount drives with noattime, nodiratime

databricks™

Not setting the number of reducers

Default behavior: inherits # of reducers from parent RDD

Too many reducers:

→ Task launching overhead becomes an issue (will see many small tasks)

Too few reducers:

→ Limits parallelism in cluster

4. Collecting results

Collecting massive result sets

sc.textFile("/big/hdfs/file/").collect()

Fixing

If processing, push computation into Spark

If storing, write directly to parallel storage

Advanced Profiling

JVM Utilities:

jstack <pid> jvm stack trace
jmap –histo:live <pid> heap summary

System Utilities:

dstat io and cpu stats
iostat disk stats
lsof –p <pid> tracks open files

Conclusion

Spark 0.8 provides good tools for monitoring performance

Understanding Spark concepts provides a major advantage in perf debugging

Questions?