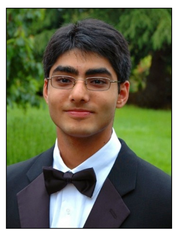
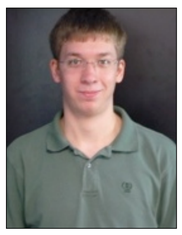
Spark Debugger

Ankur Dave, Matei Zaharia, Murphy McCauley, Scott Shenker, Ion Stoica

amplab





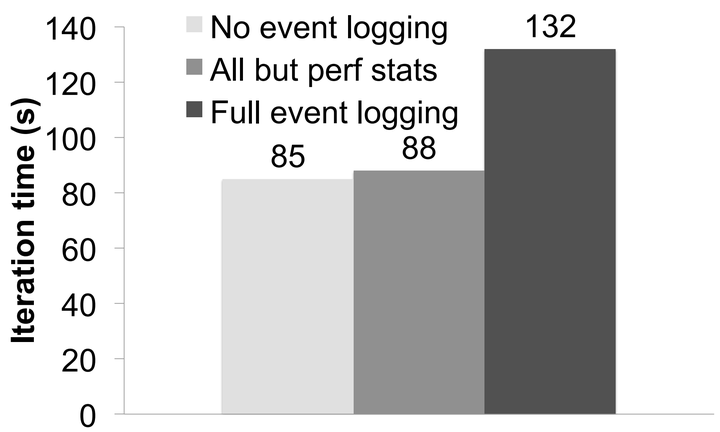
Motivation

- People want to be able to monitor and debug Spark programs
- Conventional debugging incurs too much overhead on a cluster
- Can we use the structure of Spark programs to debug and replay with minimal overhead?

Approach

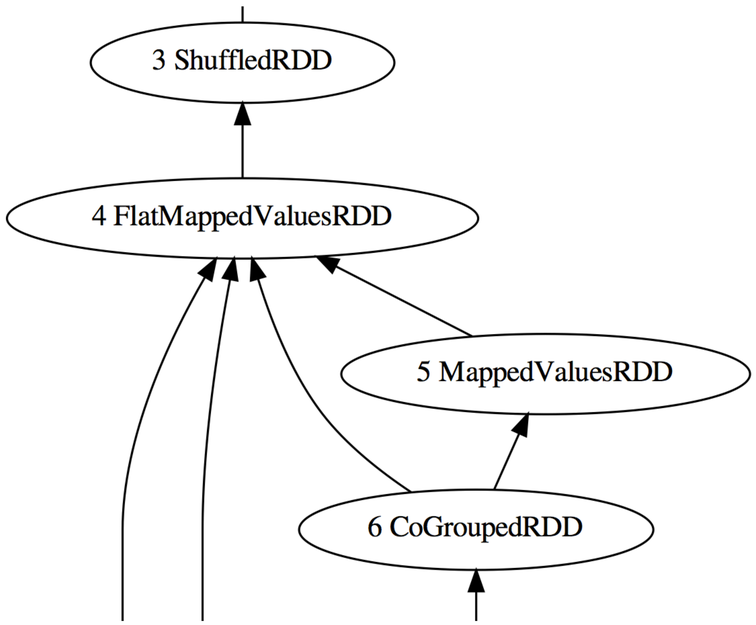
- As Spark program runs, workers report key events back to the master
- RDD lineage, checksums, shuffle order, and other events are logged
- Important events like errors are shown immediately to the user
- Later, user can reconstruct and query intermediate RDDs using the logged information, and run tasks in a conventional debugger

Performance



Capabilities

- *RDD replay and inspection:* interactively query and visualize intermediate RDDs



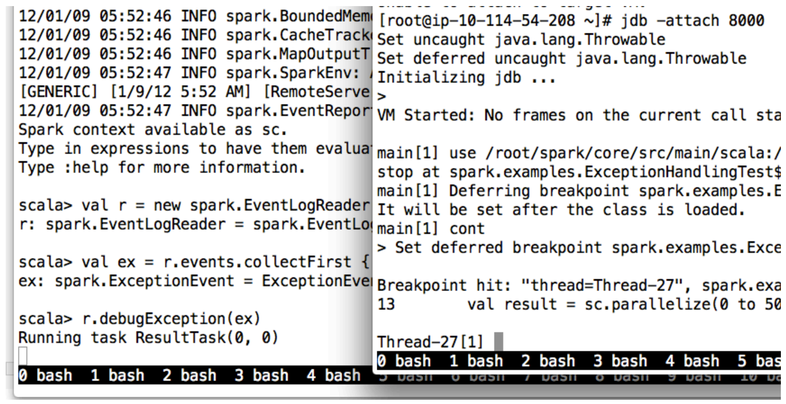
- *Exception monitoring:* aggregate and display exceptions thrown on slaves

```
01/09 03:54:02 INFO spark.SimpleJob: Starting task 0:19 as TID 19 on slave 201
01/09 03:54:06 INFO spark.SimpleJob: Lost TID 0 (task 0:0)
01/09 03:54:06 INFO spark.SimpleJob: Loss was due to java.lang.ArithmeticExcep
    at spark.examples.ExceptionHandlingTest$$anonfun$1.apply$mcII$sp(Exceptic
    at spark.examples.ExceptionHandlingTest$$anonfun$1.apply(ExceptionHandlir
    at spark.examples.ExceptionHandlingTest$$anonfun$1.apply(ExceptionHandlir
    at scala.collection.Iterator$$anon$19.next(Iterator.scala:335)
```

- *Performance monitoring:* find slow tasks caused by workload imbalance

```
scala> {
     |   println("RDD\tNormalized variance");
     |   println("---\t--------------------");
     |   r.events.collect { case rs: RuntimeStatistics if rs.mean != 0 => rs }
     |     .groupBy(_.rddId)
     |     .mapValues(xs => stdDev(xs.map(_.mean)) / mean(xs.map(_.mean)))
     |     .toList.sortBy(_._2).reverse
     |     .foreach(x => println("#" + x._1 + "\t" + x._2))
     | }
RDD     Normalized variance
---     --------------------
#4      4.611228850686498
#7      2.005886866272438
#11     1.708379166955671
#0      1.0156906223457116
#22     0.7463381659054615
#12     0.7310855018089487
#10     0.6380130541752221
```

- *Step-through debugging:* recompute a slice of an RDD in a conventional debugger



- *Serialization overhead monitoring:* calculate how much time is lost to serialization
- *Determinism verification:* ensure that user code is deterministic

Future Plans

- *Culprit determination:* automatically find which element of an RDD is causing a crash
- *GC monitoring:* determine whether GC is a problem and what type of objects are being collected
- *Cache monitoring:* determine what is being inserted and evicted from each machine's cache