Gotchas and Anti-Patterns

Michael Malak

November 5, 2014
Spark
RDDs

Image from Matei's paper
Spark

Gotchas and Anti-Patterns

Michael Malak

November 5, 2014
Fold-style aggregation for VertexRDD
Spark
AP: Ignore Partitioning

```scala
val rdd2 = rdd1.groupByKey(x => (x.substring(0, 1), x)).persist()
```

**Alternative:** group by x.substring(0, 2)
AP: Ignore GC

![Graph showing Largest Full GC Duration vs Data Size (GB) with On-Heap and Big Memory categories.]

Chart from performanceterracotta.blogspot.com

**Solution 1:** Use Tachyon

**Solution 2:** Break up node into multiple workers
AP: Ignore GC

Solution 1: Use Tachyon
Solution 2: Break up node into multiple workers
Workers Per Node

From http://hpsc-spark.github.io/

Optimal in this case: 3
AP: Ignore GC

Solution 1: Use Tachyon
Solution 2: Break up node into multiple workers
G: Sharing RDDs

(RDDs tied to SparkContext)

**Solution 1:** Tachyon and/or Succinct

**Solution 2:** "god" daemon/router w/SparkContext
G: Sharing RDDs

(RDDs tied to SparkContext)

Solution 1: Tachyon and/or Succinct
Solution 2: "god" daemon/router w/SparkContext
G: Hot-Adding Nodes to Cluster

**YARN:**
- Perform a bunch of manual steps, and then
- `su -l yarn -c "yarn rmadmin -refreshNodes"

**Mesos:**

Probably not possible based on open JIRA tickets:
- MESOS-1739: Allow slave reconfiguration on restart
- MESOS-890: Figure out a way to migrate a live Mesos cluster to a different ZooKeeper cluster
AP: Thinking Spark is "In-Memory"

Shuffle files get written to disk

Stage 1

Disk

Stage 2

From Aaron Davidson presentation at Spark Summit July 2014
At least in 1.2.0, reduce happens on node with largest shuffle file (but shuffle files still written to disk)
AP: Thinking Spark is "In-Memory"

Shuffle files get written to disk

From Aaron Davidson presentation at Spark Summit July 2014
AP: Using `groupByKey()` instead of `reduceByKey()`

```scala
sc.textFile("hdfs:/names")
  .distinct(numPartitions = 6)
  .map(name => (name.charAt(0), 1))
  .reduceByKey(_ + _)
  .collect()
```

Original:

```scala
sc.textFile("hdfs:/names")
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues { names => names.toSet.size }
  .collect()
```

From Aaron Davidson presentation at Spark Summit July 2014
AP: Using map() instead of mapPartitions()

```scala
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)

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

From Patrick Wendell's Spark Summit 2013 Presentation
AP: Going Down the Kryo Rabbit Hole

- Many groups just use Serializable instead, despite 10x performance hit
- Kryo support is much better than it used to be, though -- dozens of Jira tickets resolved in 1.0.0 and 1.1.0
- Jira tickets still outstanding:
  - SPARK-3601 Kryo NPE for output operations on Avro complex Objects even after registering
  - SPARK-3630 Identify cause of Kryo+Snappy PARSING_ERROR
val sum = sc.accumulator[Double](0.0)
val squaredSum = sc.accumulator[Double](0.0)
val values = 0 to 100000
val inputRDD = sc.makeRDD(values)
val sumRDD = inputRDD.map(value => {
  sum+= value
  squaredSum+= value*value
})
println("sum is "+sum.value+" and square sum is "+ squaredSum.value)

From blog.madhukaraphatak.com

- Above results in 0,0
- Alternative: Use mapPartitions() and forEach() within each partition
Spark Streaming
object MyFirstSparkJob {
    def main(args: Array[String]) {
        val ssc = new StreamingContext(args(0), "BeaconCount", Seconds(1))
        val parser = new JSONParser // <-- INSTANTIATED HERE

        val lines = ssc.textFileStream("beacons.txt")
        lines.map(line => parser.parse(line)) // <-- IN THE CLOSURE
        lines.foreach(line => println(line))

        ssc.start()
    }
}

From engineering.sharethrough.com

- JSONParser needs to be Serializable
- JSONParser needs to be stateless
- JSONParser cannot be outside of main()
What is Closure?

- Scala has Closure
- Scala does not have Closure with Serialization
- "Spores" library not part of even Scala 2.11
- So AMPLab wrote their own!
- Spark has serializable closure even though Scala does not
object MyFirstSparkJob {
  def main(args: Array[String]) {
    val ssc = new StreamingContext(args(0), "BeaconCount", Seconds(1))
    val parser = new JSONParser // <-- INSTANTIATED HERE

    val lines = ssc.textFileStream("beacons.txt")
    lines.map(line => parser.parse(line)) // <-- IN THE CLOSURE
    lines.foreach(line => println(line))

    ssc.start()
  }
}

- JSONParser needs to be Serializable
- JSONParser needs to be stateless
- JSONParser cannot be outside of main()
G: Trying to Do Graceful Shutdown

- Fixed in 1.0.0: SPARK-1331 Graceful shutdown of Spark Streaming computation
- Fixed in 1.0.1: SPARK-2034 KafkaInputDStream doesn't close resources and may prevent JVM shutdown
- Still open: SPARK-2892 Socket Receiver does not stop when streaming context is stopped
- Still open: SPARK-2383 With auto.offset.reset, KafkaReceiver potentially deletes Consumer nodes from Zookeeper

```java
main() {
    /* bag setup goes here */
    ssc.start();
    while (!sentinelFile.exists()) Thread.sleep(1000);
    myKafkaInputStream.stop();
    myOutputFileStream.enqueue("MagicOne")
    while ((receivedStop) { Thread.sleep(1000) }
    ssc.stop();
}

foreachRDD(rdd -> { 
    foreach(d -> { 
        if (d == "MagicOne") 
            receivedStop = true 
        else 
            kafkaProducer.send(d) 
    }) 
})
```
API

- Fixed in 1.0.0: SPARK-1331 Graceful shutdown of Spark Streaming computation
- Fixed in 1.0.1: SPARK-2034 KafkaInputDStream doesn't close resources and may prevent JVM shutdown
- Still open: SPARK-2892 Socket Receiver does not stop when streaming context is stopped
- Still open: SPARK-2383 With auto.offset.reset, KafkaReceiver potentially deletes Consumer nodes from Zookeeper
main() {
    /* DAG setup goes here */
    ssc.start()
    while (!sentinelFile.exists) {Thread.sleep(1000)}
    myKafkaInputDStream.stop()
    myQueueInputDStream.enqueue("MagicDone")
    while (!receivedStop) {Thread.sleep(1000)}
    ssc.stop()
}

foreachRDD(rdd => rdd.foreach(d => {
    if (d == "MagicDone")
        receivedStop = true
    else
        myKafkaProducer.send(d)
}))
G: Trying to Do Graceful Shutdown

- Fixed in 1.0.0: SPARK-1331 Graceful shutdown of Spark Streaming computation
- Fixed in 1.0.1: SPARK-2034 KafkaInputDStream doesn’t close resources and may prevent JVM shutdown
- Still open: SPARK-2892 Socket Receiver does not stop when streaming context is stopped
- Still open: SPARK-2383 With auto.offset.reset, KafkaReceiver potentially deletes Consumer nodes from Zookeeper
Save/Restore State

Open SPARK-3660: Initial RDD for updateStateByKey transformation

Standard `updateStateByKey()`
(keys under private control of Spark API)

def `updateStateByKey[S]`(updateFunc: (Seq[V], Option[S]) ⇒ Option[S])
  (implicit arg0: ClassTag[S]): `DStream[(K, S)]`

Alternative `updateStateByKey()` overload provided by standard API
(full access to all keys)

def `updateStateByKey[S]`(updateFunc: (Iterator[(K, Seq[V], Option[S])]) ⇒
  Iterator[(K, S)], partitioner: `Partitioner`, rememberPartitioner: Boolean)
  (implicit arg0: ClassTag[S]): `DStream[(K, S)]`
G: Overrunning Batch Duration

E.g. Batch duration of 10 seconds, but one particular batch takes 15 seconds to process

"Undefined" behavior

"Solution": Set batch size for worst case rather than average case

Batch Size Tradeoff

Latency vs. Throughput
G: No transactions

If a node deep in the DAG fails to handle incoming data, Spark Streaming has no facility to percolate that error back up the DAG (e.g. for future reprocessing or for flow control)

Major advantage of Storm over Spark Streaming

*Some reliability when using Flume*

Spark 1.1.0 SPARK-1729 Make Flume pull data from source, rather than the current push model

This makes sure that the if the Spark executor running the receiver goes down, the new receiver on a new node can still get data from Flume.

*Other data sources (Kafka? Socket?)*
G: Kafka "At Least Once" Semantics

Good News: Exactly once when Kafka nodes aren't going down

But this is a Kafka issue, not a Spark issue
G: Multiple Kafka Input Streams

Open SPARK-2388: Streaming from multiple different Kafka topics is problematic
Spark Streaming
G: Graphs are immutable

Spark 1.2.0: SPARK-2365 Add IndexedRDD, an efficient updatable key-value store

"GraphX would be the first user of IndexedRDD, since it currently implements a limited form of this functionality in VertexRDD."

"We envision a variety of other uses for IndexedRDD, including streaming updates to RDDs, direct serving from RDDs, and as an execution strategy for Spark SQL."
G: No Built-in API to Read RDF

Solution: Wait for Spark GraphX In Action mid-2015
G: INAIPL But... PageRank is patented

Join the vertices with an RDD and then apply a function from the the vertex and RDD entry to a new vertex value.

lazy val numEdges: Long
The number of edges in the graph.

lazy val numVertices: Long
The number of vertices in the graph.

lazy val outDegrees: VertexRDD[Int]
The out-degree of each vertex in the graph.

def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
Run a dynamic version of PageRank returning a graph with vertex attributes containing the PageRank and edge attributes containing the normalized edge weight.

def pickRandomVertex(): VertexId
Picks a random vertex from the graph and returns its ID.

def pregel[A](initialMsg: A, maxIterations: Int = Int.MaxValue, activeDirection: EdgeDirection = EdgeDirection.Either)(vprog: (VertexId, VD, A) ⇒ VD, sendMsg: (EdgeTriplet[VD, ED]) ⇒ Iterator[(VertexId, A)], mergeMsg: (A, A) ⇒ A)(implicit arg0: ClassTag[A]): Graph[VD, ED]
Execute a Pregel-like iterative vertex-parallel abstraction.

def staticPageRank(numIter: Int, resetProb: Double = 0.15):
G: INAIPL But... PageRank is patented

Join the vertices with an RDD and then apply a function from the the vertex and RDD entry to a new vertex value.

```scala
lazy val numEdges: Long
The number of edges in the graph.
```

```scala
lazy val numVertices: Long
The number of vertices in the graph.
```

```scala
lazy val outDegrees: VertexRDD[Int]
The out-degree of each vertex in the graph.
```

```scala
def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
Run a dynamic version of PageRank returning a graph with vertex attributes containing the PageRank and edge attributes containing the normalized edge weight.
```

```scala
def pickRandomVertex(): VertexId
Picks a random vertex from the graph and returns its ID.
```

```scala
def pregel[A](initialMsg: A, maxIterations: Int = Int.MaxValue, activeDirection: EdgeDirection = EdgeDirection.Either)(vprog: (VertexId, VD, A) ⇒ VD, sendMsg: (EdgeTriplet[VD, ED]) ⇒ Iterator[(VertexId, A)], mergeMsg: (A, A) ⇒ A)(implicit arg0: ClassTag[A]): Graph[VD, ED]
Execute a Pregel-like iterative vertex-parallel abstraction.
```

```scala
def staticPageRank(numIter: Int, resetProb: Double = 0.15):
```
G: GraphX Performance (Relative to Established Graph Tools)


"One of the reasons for PowerGraph's superior performance can be attributed to its highly optimized C++ implementation."
Shark SQL
G: Spark SQL Still Catching up to Hive (and Shark)

Fixed in 1.2.0

- SPARK-2562 DATE data type not supported
- SPARK-4154 "NOT BETWEEN" not supported
- SPARK-4120 JOIN syntax "FROM T1, T2, T3" not supported (but INNER JOIN / OUTER JOIN syntax is supported)
- SPARK-2866 ORDER BY attributes must appear in SELECT clause
- SPARK-2693 Support for UDAF Hive Aggregates like PERCENTILE
- SPARK-3807 SparkSql does not work for tables created using custom serde

Open

- SPARK-4135 Error reading Parquet file generated with SparkSQL
- SPARK-4073 Parquet+Snappy can cause significant off-heap memory usage
- SPARK-4131 INSERT OVERWRITE LOCAL (filesystem)

Other (non-Hive-like) Issues

- Fixed in 1.2.0: SPARK-3500 coalesce() and repartition() of SchemaRDD is broken
G: Performance Relative to Impala

TPC-H query times at scale factor 1000 on 20 r3.4xlarge EC2 instances

- CitusDB 3.0.2
- Impala 2.0.0
- SparkSQL 1.1.0
AP: Ignore Performance Tuning

- Partitioning
- Compression
- COMPUTE STATISTICS
- Not caching tables larger than memory
- SQLContext config options
G: Data Skipping From Shark Not Supported

(a) tuples

<table>
<thead>
<tr>
<th>time</th>
<th>id</th>
<th>event</th>
<th>category</th>
<th>publisher</th>
<th>revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:01:01</td>
<td>102</td>
<td>click</td>
<td>jeans</td>
<td>groupon</td>
<td>0.0</td>
</tr>
<tr>
<td>08:01:01</td>
<td>103</td>
<td>click</td>
<td>shirts</td>
<td>google</td>
<td>-0.5</td>
</tr>
<tr>
<td>08:01:01</td>
<td>104</td>
<td>click</td>
<td>shirts</td>
<td>groupon</td>
<td>0.0</td>
</tr>
<tr>
<td>08:01:02</td>
<td>105</td>
<td>buy</td>
<td>jeans</td>
<td>google</td>
<td>12.0</td>
</tr>
<tr>
<td>08:01:03</td>
<td>106</td>
<td>click</td>
<td>jeans</td>
<td>google</td>
<td>-0.5</td>
</tr>
<tr>
<td>08:01:04</td>
<td>107</td>
<td>buy</td>
<td>shoes</td>
<td>shoedeal</td>
<td>30.0</td>
</tr>
</tbody>
</table>

(b) features

<table>
<thead>
<tr>
<th>features</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event='buy'</code></td>
<td>50</td>
</tr>
<tr>
<td><code>product='jeans'</code></td>
<td>20</td>
</tr>
<tr>
<td><code>publisher='google'</code></td>
<td>10</td>
</tr>
<tr>
<td><code>revenue &lt; 0</code></td>
<td></td>
</tr>
</tbody>
</table>

(c) vectors

<table>
<thead>
<tr>
<th>vector</th>
<th>(F1, F2, F3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>(0,1,0)</td>
</tr>
<tr>
<td>t2</td>
<td>(0,0,1)</td>
</tr>
<tr>
<td>t3</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td>t4</td>
<td>(1,1,0)</td>
</tr>
<tr>
<td>t5</td>
<td>(0,1,1)</td>
</tr>
<tr>
<td>t6</td>
<td>(1,0,0)</td>
</tr>
</tbody>
</table>

(d) blocking

<table>
<thead>
<tr>
<th>blocking</th>
<th>(F1, F2, F3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>(0,1,0)</td>
</tr>
<tr>
<td>P2</td>
<td>(1,1,0)</td>
</tr>
<tr>
<td>P3</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td>t5</td>
<td>(0,1,1)</td>
</tr>
</tbody>
</table>

From paper Liwen Sun et al

- Shark (no skip): 33 sec
- Shark (range partition): 9.6 sec
- WARP: 2.9 sec

From AMPLab

Fixed in 1.2.0: SPARK-2961 Use statistics to skip partitions when reading from in-memory columnar data
Shark SQL
BlinkDB
G: BlinkDB Requires Spark 0.9.0

Queries with Bounded Errors and Bounded Response Times on Very Large Data

BlinkDB is a large-scale data warehouse system built on Shark and Spark and is designed to be compatible with Apache Hive. It can answer HiveQL queries up to 200-300 times faster than Hive by executing them on user-specified samples of data and providing approximate answers that are augmented with meaningful error bars. BlinkDB 0.1.0 is an alpha developer release that supports creating/deleting samples on any input table and/or materialized view and executing approximate HiveQL queries with those aggregates that have statistical closed forms (i.e., AVG, SUM, COUNT, VAR and STDEV).

BlinkDB requires:

- Scala 2.10.x
- Spark 0.9.x
MLlib
G: Not as Tunable as Mahout

Not disappointed at MLib, but quite surprised

Antoine Amend
Technical Lead Data Scientist at KPMG UK
Top Contributor

First things I’ve noticed after playing around with KMeans on MLib
- I’m not allowed to use any Distance Measure (Only Euclidean is supported)
- I’m not allowed to tune my KMeans with initial cluster centers (convergence is slower)
- I don’t get the distance associated with the matching cluster (only the cluster Id is returned)
There is no point to argue, Mahout sucks, but at least can be tuned... Anyone reaching the same conclusions?
G: Key Algorithms Missing

- SPARK-2352 Add Artificial Neural Network (ANN) to Spark
- Anomaly Detection
- Decision Tree Pruning
Spark

Gotchas and Anti-Patterns

Michael Malak

November 5, 2014