Integrating Spark at Petabyte Scale

Daniel C. Weeks
NETFLIX
### Test case XYZ

( metrics current through 10/31 )

<table>
<thead>
<tr>
<th>Display Cell</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparison Cell</td>
<td>Set All</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td># of Allocations</td>
<td>107,377</td>
<td>107,149</td>
<td>107,513</td>
<td>107,590</td>
<td>107,264</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Streaming Score</th>
<th>N.A.</th>
<th>N.A.</th>
<th>N.A.</th>
<th>N.A.</th>
<th>N.A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Accounts with &gt; 0 Hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Accounts with &gt;= 1 Hour</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Accounts with &gt;= 5 Hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Accounts with &gt;= 10 Hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Accounts with &gt;= 20 Hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANK</td>
<td>CHANGE</td>
<td>TYPE</td>
<td>ISP NAME</td>
<td>AVG SPEED (Mbps)</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>------</td>
<td>---------------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td></td>
<td>COX</td>
<td>3.62</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td></td>
<td>CABLEVISION - OPTIMUM</td>
<td>3.59</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td></td>
<td>VERIZON - FIOS</td>
<td>3.54</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>+2</td>
<td></td>
<td>CHARTER</td>
<td>3.46</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td>COMCAST</td>
<td>3.45</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>+3</td>
<td></td>
<td>BRIGHT HOUSE</td>
<td>3.42</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-2</td>
<td></td>
<td>SUDDENLINK</td>
<td>3.42</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>+1</td>
<td></td>
<td>TIME WARNER CABLE</td>
<td>3.37</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-1</td>
<td></td>
<td>MEDIACOM</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td></td>
<td>AT&amp;T - U-VERSE</td>
<td>3.20</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td></td>
<td>AT&amp;T - DSL</td>
<td>2.51</td>
<td></td>
</tr>
</tbody>
</table>
Our Biggest Challenge is Scale
Netflix Key Business Metrics

- 65+ million members
- 50 countries
- 1000+ devices supported
- 10 billion hours / quarter
Global Expansion

200 countries by end of 2016
Big Data Size

Total ~20 PB DW on S3
Read ~10% DW daily
Write ~10% of read data daily

~ 500 billion events daily

~ 350 active platform users (~100 analysts)
Architecture Overview
Data Pipelines

- Cloud apps → Suro/Kafka → Ursula
- 15 min
- Cassandra → SS Tables → Aegisthus
- Dimension Data
- Daily
- AWS S3
S3 as our DW Storage

S3 as single source of truth (not HDFS)
  • 11 9’s durability and 4 9’s availability
  • Separate compute and storage

Enables
  • Multiple heterogeneous clusters
  • Easy upgrade via r/b deployment
Spark
Apache Spark™ is a fast and general engine for large-scale data processing.
Why Spark?

- Batch jobs (Pig, Hive)
  - ETL jobs
  - Reporting and other analysis

- Interactive jobs (Presto)

- Iterative ML jobs (Spark)

- Programmatic Use Cases
Deployments @ Netflix

• Spark on Mesos
  – Self-service AMI
  – Full BDAS (Berkeley Data Analytics Stack)
  – Online streaming analytics

• Spark on YARN
  – Spark as a service
  – YARN application on Hadoop
  – Offline batch analytics
Version Support

Application

s3://.../spark-1.4.tar.gz
s3://.../spark-1.5.tar.gz
s3://.../spark-1.5-custom.tar.gz

$ spark-shell --version 1.5 ...

Configuration

s3://.../1.5/spark-0.90.conf
s3://.../h2prod/yarn-site.xml
s3://.../h2prod/core-site.xml
...

s3://.../h2prod/spark-0.90-site.xml
Multitenancy
Multitenancy
Dynamic Allocation on YARN

• Optimize for Resource Utilization

• Harness Cluster’s Scale

• Still Provide Interactive Performance
Task Execution Model

Spark Execution

Pending Tasks

Persistent

Container

Task

Task

Task

Task

Task

Traditional MapReduce

Container

Task

Container

Task

Container

Task

Container

Task

Container

Task

Container

Task

Container

Task
Dynamic Allocation
Cached Data

• Spark allows data to be cached
  • Interactive Reuse of Dataset
  • Iterative Usage (ML)

• Dynamic Allocation
  • Removes Executors when no tasks are Pending
val data = sqlContext
  .table("dse.admin_genie_job_d")
  .filter("$\text{dateint}$\geq20150601 \text{ and } $\text{dateint}$\leq20150830")
data.persist
data.count

spark.dynamicAllocation.cachedExecutorIdleTimeout
Preemption

Problem
- Spark tasks randomly fail with “executor lost” error

Cause
- YARN preemption is not graceful

Solution
- Preempted tasks shouldn’t be counted as failures but should be retried
Reading / Processing / Writing
Partition Pruning

Problem: Metadata is Big Data

- Tables with millions of partitions
- Partitions with hundreds of files each

Client processes all partitions locally
Predicate pushdown for metadata

What if your table has 1.6M partitions?
Predicate pushdown for metadata

Parser
Analyzer
Optimizer
SparkPlanner

HiveTableScans
getPartitionsByFilter()
AWS S3 Listing Optimization

Problem: Metadata is Big Data

• Tables with millions of partitions
• Partitions with hundreds of files each

Clients Take a Long Time to Launch Jobs
Input split computation

Parallelize:

mapreduce.input.fileinputformat.list-status.num-threads

- The number of threads to use list and fetch block locations for the specified input paths.

Setting this property in Spark jobs doesn’t help
File listing for partitioned table

Sequentially listing input dirs via S3N file system.
SPARK-9926, SPARK-10340

Problem
• Input split computation for partitioned Hive table on S3 is slow

Cause
• Listing files on a per partition basis is slow
• S3N file system computes data locality hints

Solution
• Bulk list partitions in parallel using AmazonS3Client.
• Bypass data locality computation for S3 objects.
S3 Bulk Listing

Partition path → HadoopRDD → ParArray[RDD] → S3listing input dirs in parallel via AmazonS3Client.

Partition path → HadoopRDD → ParArray[RDD] → S3listing input dirs in parallel via AmazonS3Client.

Partition path → HadoopRDD → ParArray[RDD] → S3listing input dirs in parallel via AmazonS3Client.

Partition path → HadoopRDD → ParArray[RDD] → S3listing input dirs in parallel via AmazonS3Client.

Input dir

Input dir

Input dir

Input dir
SELECT * FROM nccp_log WHERE dateint=20150801 and hour=0 LIMIT 10;
Hadoop Output Committer

How it works
• Each task writes output to a temp dir.
• Output committer renames first successful task’s temp dir to final destination

Challenges with S3
• S3 rename is copy and delete (non-atomic)
• S3 is eventual consistent
S3 output committer

How it works
• Each task writes output to local disk
• Output committer copies first successful task’s output to S3

Advantages
• Avoid redundant S3 copy
• Avoid eventual consistency
• Always write to new paths
Our contributions

SPARK-6018      SPARK-8355
SPARK-6662      SPARK-8572
SPARK-6909      SPARK-8908
SPARK-6910      SPARK-9270
SPARK-7037      SPARK-9926
SPARK-7451      SPARK-10001
SPARK-7850      SPARK-10340
Next Steps for Netflix Integration

Metrics

Data Lineage

Parquet Integration
THANK YOU