Spark 3.0 New Features

Spark 3.0 is coming with several important features, but here I list very few notable features only.

Spark with GPU Support

- Make Spark 3.0 GPU-aware in Spark cluster managers
- No regression on scheduler performance for normal jobs.

Read details here .

Spark Graph with Cypher Support

This is a complete package of Graph along with Property Graph and Cypher Script.

This project is inherited from morpheus & will fully support DataFrame.
Now Graph query will have its own Catalysts & it will follow similar principles as SparkSQL.

Read Details here.

Binary Files as a Data Source

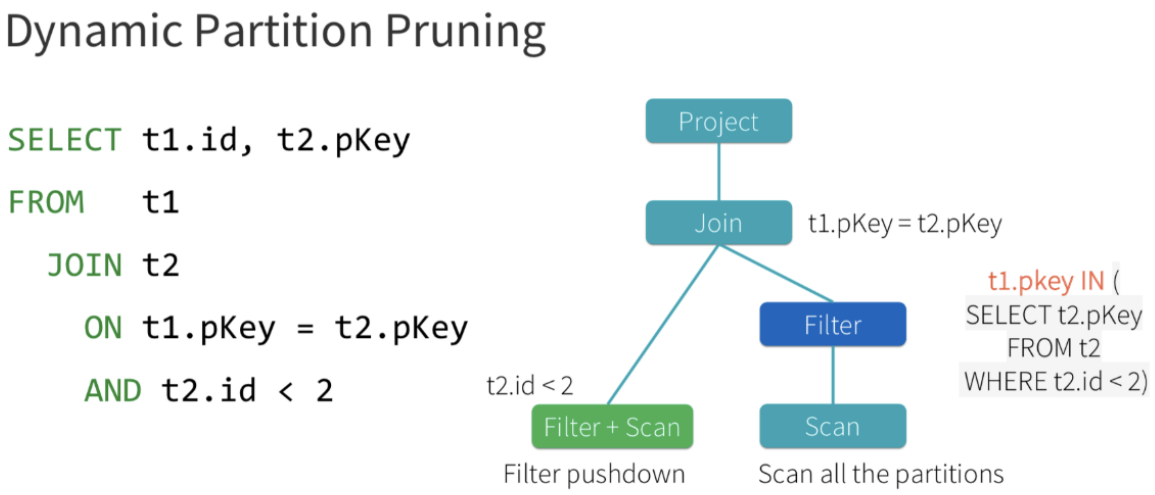
Spark 3.0 is bringing Binary Files as a core Data Source .
It would be useful to have a data source implementation for binary files, which can be used to build features to load images, audio, and videos.

```
val df = spark.read.format(BINARY_FILE).load(dir.getPath)
```

Binary Format doesn't support Write Operation & Currently it supports less than 2 GB Data file size

abhishek.create@gmail.com

Dynamic Partition Pruning



Databricks Presentation on this.

Tree Based Feature Transformation

Finally Spark ML is equipped with Decision Tree.
This is inspired by following-

http://www.herbrich.me/papers/adclicksfacebook.pdf
XGBoosting
LightGBM
Catboost

abhishek.create@gmail.com

Kafka Header Support in Structured Streaming

Kafka Header from 0.11.0 is now available with Structured Streaming.

```
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .option("includeHeaders", "true")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)", "headers")
  .as[(String, String, Map)]
```

Spark with JDK 11

Spark now fully supports JDK 11, considering end of Life of JDK 8 and JDK 9/10 is as well not that active.

JDK 11 performance is slower than JDK 8
https://issues.apache.org/jira/secure/attachment/12980684/jdk11.txt
https://issues.apache.org/jira/browse/SPARK-29173

Kubernetes Related Features

Following are few notable features(*I found them interesting*) implemented in Spark 3.0 -

- User Specific Pod Templating
- Bug Fix on OpenJDK Docker Image Supported

abhishek.create@gmail.com

- Latest Version of Kubernetes is now Supported
- spark.kubernetes.pyspark.pythonVersion can now actually be passed to Executor & python3 is now default
- Improvement on Dynamic Allocation with Kubernetes
- Daemon Thread Blocking JVM Exit automatically because of Kubernetes Client bug
- Config Requests for Driver pod
- Kubernetes support for GPU-aware scheduling
- RegisteredExecutors reload supported after ExternalShuffleService restart
- Configurable auth secret source in k8s backend
- Support automatic spark.authenticate secret in Kubernetes backend
- Subpath Mounting in Kubernetes
- Kerberos Support in Kubernetes resource manager
- Supporting emptyDir Volume/tmpfs
- More mature spark-submit with K8s

Cache Data can be Analyzed

This is a feature I always wanted and now with Spark 3.0 , its possible to analyze cached table.
Spark can analyze cached data and hold temporary column statistics for InMemoryRelation.

```
sql(
   """CACHE TABLE cachedQuery AS
    | SELECT c0, avg(c1) AS v1, avg(c2) AS v2
    | FROM (SELECT id % 3 AS c0, id % 5 AS c1, 2 AS c2 FROM range(1, 30))
    | GROUP BY c0
   """.stripMargin)

  // Analyzes one column in the cached logical plan
 sql("ANALYZE TABLE cachedQuery COMPUTE STATISTICS FOR COLUMNS v1")
)
```

Spark 3.0 has now Improved More readable Explain

Spark Explain for physical Plan was very complex to understand, but now Explain is much more readable.

Old Format:

```
*(2) Project [key#2, max(val)#15]
+- *(2) Filter (isnotnull(max(val#3)#18) AND (max(val#3)#18 > 0))
  +- *(2) HashAggregate(keys=[key#2], functions=[max(val#3)], output=[key#2, max(val)#15, max(val#3)#18])
     +- Exchange hashpartitioning(key#2, 200)
        +- *(1) HashAggregate(keys=[key#2], functions=[partial_max(val#3)], output=[key#2, max#21])
           +- *(1) Project [key#2, val#3]
              +- *(1) Filter (isnotnull(key#2) AND (key#2 > 0))
                 +- *(1) FileScan parquet default.explain_temp1[key#2,val#3] Batched: true, DataFilters: [isnotnull(key#2), (key#2 > 0)], Format: Parquet, Location: InMemoryFileIndex[file:/user/hive/warehouse/explain_temp1], PartitionFilters: [], PushedFilters: [IsNotNull(key), GreaterThan(key,0)], ReadSchema: struct<key:int,val:int>
```

New Format:

```
Project (8)
+- Filter (7)
  +- HashAggregate (6)
     +- Exchange (5)
        +- HashAggregate (4)
           +- Project (3)
              +- Filter (2)
                 +- Scan parquet default.explain_temp1 (1)


(1) Scan parquet default.explain_temp1 [codegen id : 1]
Output: [key#2, val#3]

(2) Filter [codegen id : 1]
Input     : [key#2, val#3]
Condition : (isnotnull(key#2) AND (key#2 > 0))

(3) Project [codegen id : 1]
Output    : [key#2, val#3]
Input     : [key#2, val#3]
```

abhishek.create@gmail.com

```
(4) HashAggregate [codegen id : 1]
Input: [key#2, val#3]

(5) Exchange
Input: [key#2, max#11]

(6) HashAggregate [codegen id : 2]
Input: [key#2, max#11]

(7) Filter [codegen id : 2]
Input     : [key#2, max(val)#5, max(val#3)#8]
Condition : (isnotnull(max(val#3)#8) AND (max(val#3)#8 > 0))

(8) Project [codegen id : 2]
Output    : [key#2, max(val)#5]
Input     : [key#2, max(val)#5, max(val#3)#8]
```

Dynamic Allocation without Any External Shuffle Service

Dynamic Allocation is now Possible without any External Shuffle Service & that brings Dynamic Allocation in K8s .
Currently with Spark 3.0, Dynamic Allocation is not fully available with Kuberenetes.

RobustScalar for Spark

RobustScaler is a kind of widely-used scaler, which use median/IQR to replace mean/std in StandardScaler. Scale features using statistics that are robust to outliers.

Inspired by Sklearn https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler

abhishek.create@gmail.com

Logistic Loss is now supported in Spark

Logistic Loss, a widely known Metrics for Classification Task is now available.

```
val df = sc.parallelize(labels.zip(probabilities)).map {
  case (label, probability) =>
    val prediction = probability.argmax.toDouble
    (prediction, label, probability)
}.toDF("prediction", "label", "probability")

val evaluator = new MulticlassClassificationEvaluator()
  .setMetricName("logLoss")
```

Spark New Compression Code

ZStd codec directly, we use Spark's CompressionCodec which wraps ZStd codec in a buffered stream to avoid overhead excessive of JNI call while trying to compress/decompress small amount of data.

This will lead to *Faster Performance*.

New Higher Order Functions

- Map_entries
- Map_filter
- Map_zip_with
- Transform_keys
- Transform_values
- filter(array<T>, function<T, Int, boolean>) → array<T>

abhishek.create@gmail.com

Executor Memory Metrics are now Available

Executor Metrics information will help provide insight into how executor and driver JVM memory is used, and for the different memory regions. It can be used to help determine good values for spark.executor.memory, spark.driver.memory, spark.memory.fraction, and spark.memory.storageFraction.

Currently its available with Prometheus.

```
$ bin/spark-shell --master spark://` hostname`:7077 --conf
spark.ui.prometheus.enabled=true
```

Check all available resources here.

There is still a lot more to do, I will update that separately

CSV/JSON migrated to Datasource V2

Reference Databricks
As a general computing engine, Spark can process data from various data management/storage systems, including HDFS, Hive, Cassandra and Kafka. For flexibility and high throughput, Spark defines the Data Source API, which is an abstraction of the storage layer. The Data Source API has two requirements.

1) Generality: support reading/writing most data management/storage systems.

2) Flexibility: customize and optimize the read and write paths for different systems based on their capabilities.

Check further here

[1]

[1] abhishek.create@gmail.com

abhishek.create@gmail.com