Apache Hivemall: Scalable machine learning library for Apache Hive/Spark/Pig

1). Research Engineer, Treasure Data
Makoto YUI @myui

2). Research Engineer, NTT
Takashi Yamamuro @maropu
Plan of the talk

1. Introduction to Hivemall

2. Hivemall on Spark
Hivemall entered Apache Incubator on Sept 13, 2016 🎉

hivemall.incubator.apache.org

Since then, we invited 2 contributors as new committers (a committer has been voted as PPMC). Currently, we are working toward the first Apache release on this Q2.
What is Apache Hivemall

Scalable machine learning library built as a collection of Hive UDFs

Ease-of-use  Scalable  Multi/Cross platform  Versatile

2017/5/16 Apache BigData North America ’17, Miami
Hivemall is easy and scalable ...

ML made easy for SQL developers

Born to be parallel and scalable

CREATE TABLE lr_model AS
SELECT feature, -- reducers perform model averaging in parallel
avg(weight) as weight
FROM (SELECT logress(features,label,..) as (feature,weight)
FROM train
) t -- map-only task
GROUP BY feature; -- shuffled to reducers

This query automatically runs in parallel on Hadoop

2017/5/16 Apache BigData North America '17, Miami
Hivemall is a *multi/cross-platform* ML library

Multi/Cross platform

prediction models built by Hive can be used from Spark, and conversely, prediction models built by Spark can be used from Hive
Hivemall’s Technology Stack

Machine Learning
- Hivemall
- MLlib

Query Processing
- Hive
- Pig
- SparkSQL

Parallel Data Processing Framework
- MapReduce (MRv1)
- Apache Tez DAG processing
- Apache Spark

Resource Management
- Apache YARN
- MESOS

Distributed File System Cloud Storage
- Hadoop HDFS
- Amazon S3

2017/5/16 Apache BigData North America ’17, Miami
CREATE TABLE lr_model AS
SELECT
    feature, -- reducers perform model averaging in parallel
    avg(weight) as weight
FROM (
    SELECT logress(features,label,..) as (feature,weight)
    FROM train
) t -- map-only task
GROUP BY feature; -- shuffled to reducers
Hivemall on Apache Spark Dataframe

```scala
val trainDf = 
  spark.read.format("libsvm").load("a9a.train")

trainDf.train_logregr("feature", "label")
  .groupby("feature")
  .agg("weight"->"avg")
```
Hivemall on SparkSQL

```python
context = HiveContext(sc)

context.sql(""
SELECT
    feature, avg(weight) as weight
FROM (  
    SELECT train_logregr(features, label)
    as (feature, weight)
    FROM trainTable
  ) model
GROUP BY feature
"")
```
Hivemall on Apache Pig

```java
a = load 'a9a.train'
    as (rowid:int, label:float, features:{(featurepair:chararray)});

b = foreach c generate flatten(
    logress(features, label, '-total_steps ${total_steps}')
) as (feature, weight);

c = group b by feature;

d = foreach c generate group, AVG(c.weight);
store d into 'a9a_model1';
```
Online Prediction by Apache Streaming

```scala
val testData = 
  ssc.textFileStream(...).map(LabeledPoint.parse) // Infinite stream

testData.predict { case testDf =>
  // Explode features in input streams
  val testDf_exploded = ...
  testDf_exploded
    .join(model, testDf_exploded("feature") === model("feature"), "LEFT_OUTER")
    .select("rowid", ("weight" * "value").as("value"))
    .groupby("rowid").sum("value")
    .select("rowid", sigmoid("SUM(value)"))
}```
Hivemall is a Versatile library ..

- Not only for Machine Learning
- provides a bunch of generic utility functions

Each organization has own sets of UDFs for data preprocessing

Don’t Repeat Yourself!
Don’t Repeat Yourself!
Hivemall generic functions

<table>
<thead>
<tr>
<th>Array and Map</th>
<th>Bit and compress</th>
<th>String and NLP</th>
<th>Geo Spatial</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY_CONCAT</td>
<td>TO_BITS</td>
<td>BASE91</td>
<td>TILE</td>
</tr>
<tr>
<td>ARRAY_INTERSECT</td>
<td>UNBITS</td>
<td>UNBASE91</td>
<td>MAP_URL</td>
</tr>
<tr>
<td>ARRAY_REMOVE</td>
<td>BITS_OR</td>
<td>NORMALIZE_UNICODE</td>
<td></td>
</tr>
<tr>
<td>SORT_AND_UNIQ_ARRAY</td>
<td>BITS_COLLECT</td>
<td>SPLIT_WORDS</td>
<td></td>
</tr>
<tr>
<td>SUBARRAY_ENDWITH</td>
<td>DEFLATE</td>
<td>IS_STOPWORD</td>
<td></td>
</tr>
<tr>
<td>SUBARRAY_STARTWITH</td>
<td>INFLATE</td>
<td>TOKENIZE</td>
<td></td>
</tr>
<tr>
<td>SUBARRAY</td>
<td></td>
<td>TOKENIZE_JA</td>
<td></td>
</tr>
<tr>
<td>ARRAY_AVG</td>
<td></td>
<td>TF_IDF</td>
<td></td>
</tr>
<tr>
<td>ARRAY_SUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP_GET_SUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP_TAIL_N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_MAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_ORDERED_MAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We welcome contributing your generic UDFs to Hivemall!
Map tiling functions

\[ x = \left\lfloor \frac{\text{lon} + 180}{360} \cdot 2^z \right\rfloor \]

\[ y = \left\lfloor \left( 1 - \frac{\ln \left( \tan \left( \frac{\text{lat} \cdot \pi}{180} \right) + \frac{1}{\cos \left( \text{lat} \cdot \frac{\pi}{180} \right)} \right)}{\pi} \right) \cdot 2^z - 1 \right\rfloor \]
WITH data as (  
  select 25.7724247 as lat, -80.1854473 as lon, 10 as zoom  
  union all  
  select 25.7724247 as lat, -80.1854473 as lon, 15 as zoom  
)  
select  
  map_url(lat,lon,zoom) as osm_url,  
  map_url(lat,lon,zoom,'-type googlemaps') as gmap_url,  
  tile(lat,lon,zoom) as tile_number  
from  
data;

\[
\text{Tile}(\text{lat},\text{lon},\text{zoom}) = \text{xtile}(\text{lon},\text{zoom}) + \text{ytile}(\text{lat},\text{zoom}) \times 2^n
\]
Top-k query processing

List top-2 students for each class

<table>
<thead>
<tr>
<th>student</th>
<th>class</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>student</th>
<th>class</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>a</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>60</td>
</tr>
</tbody>
</table>
Top-k query processing

List top-2 students for each class

<table>
<thead>
<tr>
<th>student</th>
<th>class</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>60</td>
</tr>
</tbody>
</table>

SELECT * FROM (  
SELECT *,  
    rank() over (partition by class order by score desc) as rank  
FROM table  
) t  
WHERE rank <= 2

RANK over() query does not finishes in 24 hours 😞  
where 20 million MOOCs classes and avg 1,000 students in each classes
Top-k query processing

List top-2 students for each class

<table>
<thead>
<tr>
<th>student</th>
<th>class</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>60</td>
</tr>
</tbody>
</table>

```sql
SELECT each_top_k(    
    2, class, score,    
    class, student
) as (rank, score, class, student)
FROM (    
    SELECT * FROM table
    DISTRIBUTION BY class SORT BY class
) t
```

EACH_TOP_K finishes in 2 hours 😊
Top-k query processing by RANK OVER()
Top-k query processing by EACH_TOP_K

- Node 1
  - distributed by class
  - Sort by class
  - each_top_k
  - OUTPUT only K items
Comparison between RANK and EACH_TOP_K

Bounded Priority Queue is utilized

Each_top_k is very efficient where the number of class is large
### List of Supported Algorithms

#### Classification
- ✓ Perceptron
- ✓ Passive Aggressive (PA, PA1, PA2)
- ✓ Confidence Weighted (CW)
- ✓ Adaptive Regularization of Weight Vectors (AROW)
- ✓ Soft Confidence Weighted (SCW)
- ✓ AdaGrad+RDA
- ✓ Factorization Machines
- ✓ RandomForest Classification

#### Regression
- ✓ Logistic Regression (SGD)
- ✓ AdaGrad (logistic loss)
- ✓ AdaDELTA (logistic loss)
- ✓ PA Regression
- ✓ AROW Regression
- ✓ Factorization Machines
- ✓ RandomForest Regression

---

**Logistic regression** is good for getting a probability of a positive class.

**Factorization Machines** is good where features are sparse and categorical ones.

---

SCW is a good first choice
Try RandomForest if SCW does not work
RandomForest in Hivemall

Ensemble of Decision Trees
Training of RandomForest

CREATE TABLE model
STORED AS SEQUENCEFILE
AS
select
    train_randomforest_classifier(features, label)
    as (model, importance, errors, tests)
from
    training;
create table predicted_vm
as
SELECT
  rowid,
  rf_ensemble(predicted) as predicted
FROM (  
  SELECT
    t.rowid,
    tree_predict(p.model, t.features, ${classification}) as predicted
  FROM
    training t
  CROSS JOIN model p
) t1
GROUP BY
  rowid
Supported Algorithms for Recommendation

**K-Nearest Neighbor**

- ✓ Minhash and b-Bit Minhash (LSH variant)
- ✓ Similarity Search on Vector Space
  (Euclid/Cosine/Jaccard/Angular)

**Matrix Completion**

- ✓ Matrix Factorization
- ✓ Factorization Machines (regression)

*each_top_k function of Hivemall is useful for recommending top-k items*
Other Supported Algorithms

Feature Engineering
✓ Feature Hashing
✓ Feature Scaling (normalization, z-score)
✓ Feature Binning
✓ TF-IDF vectorizer
✓ Polynomial Expansion
✓ Amplifier

Evaluation metrics
✓ AUC, nDCG, logloss, precision recall@K, and etc

NLP
✓ Basic English text Tokenizer
✓ Japanese Tokenizer
Feature Engineering – Feature Binning

CREATE TABLE people(
   name string, age int, sex string
);

INSERT INTO people VALUES
   ('Jacob', 20, 'Male'),
   ('Mason', 22, 'Male'),
   ('Sophia', 35, 'Female'),
   ('Ethan', 55, 'Male'),
   ('Emma', 15, 'Female'),
   ('Noah', 46, 'Male'),
   ('Isabella', 20, 'Female');

Maps quantitative variables to fixed number of bins based on quantiles/distribution

Map Ages into 3 bins

features: array<features::string>

["name#Jacob","sex#Male","age:1"]
["name#Mason","sex#Male","age:1"]
["name#Sophia","sex#Female","age:2"]
["name#Ethan","sex#Male","age:2"]
["name#Emma","sex#Female","age:0"]
["name#Noah","sex#Male","age:2"]
["name#Isabella","sex#Female","age:1"]
Feature Selection – Signal Noise Ratio

CREATE TABLE input (  
    X array<double>, -- features  
    Y array<int> -- binarized label
);

WITH snr AS (  
    -- [UDAF] snr(features::array<number>, labels::array<int>)  
    SELECT snr(X, Y) AS snr FROM input -- aggregated SNR array
)

SELECT select_k_best(X, snr, ${k}) FROM input JOIN snr;
Evaluation Metrics

```sql
select
    ndcg(t1.rec, t2.truth, t1.max_k) as ndcgk,
    ndcg(t1.rec, t2.truth, 2) as ndcg2,
    precision(t1.rec, t2.truth, t1.max_k) as precisionk,
    precision(t1.rec, t2.truth, 2) as precision2,
    recall(t1.rec, t2.truth, t1.max_k) as recallk,
    recall(t1.rec, t2.truth, 2) as recall2,
    mrr(t1.rec, t2.truth, t1.max_k) as mrrk,
    mrr(t1.rec, t2.truth, 2) as mrr2,
    average_precision(t1.rec, t2.truth, t1.max_k) as average_precisionk,
    average_precision(t1.rec, t2.truth, 2) as average_precision2,
    auc(t1.rec, t2.truth, t1.max_k) as auck,
    auc(t1.rec, t2.truth, 2) as auc2
from
    rec t1
join
    truth t2 on (t1.userid = t2.userid);
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>@k (all of rankedList)</th>
<th>@2</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG</td>
<td>0.7039180890341349</td>
<td>0.6131471927654585</td>
</tr>
<tr>
<td>Precision</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Recall</td>
<td>0.6666666666666666</td>
<td>0.333333333333333333</td>
</tr>
<tr>
<td>MRR</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>MAP</td>
<td>0.5555555555555555</td>
<td>0.333333333333333333</td>
</tr>
<tr>
<td>AUC</td>
<td>0.75</td>
<td>1.0</td>
</tr>
</tbody>
</table>
### Other Supported Features

#### Anomaly Detection
- ✓ Local Outlier Factor (LoF)
- ✓ ChangeFinder

#### Change Point Detection
- ✓ ChangeFinder
- ✓ Singular Spectrum Transformation

#### Clustering / Topic models
- ✓ Online mini-batch LDA
- ✓ Online mini-batch PLSA
Anomaly/Change-point Detection by ChangeFinder

Efficient algorithm for finding change point and outliers from timeseries data


```sql
select
time,
    -- x is double or array<double>
    changefinder(x) as (anomaly_score, changepoint_score)
    -- changefinder(x),"-outlier_threshold 30 -changepoint_threshold 15"
    as (anomaly_score, changepoint_score, is_anomaly, is_changepoint)
from
timeseries
order by
time asc
```
Efficient algorithm for timeseries data


```
select
    time,
    -- x is double or array<dc
    changefinder(x) as (anomal
    -- changefinder(x),"-outli
    as (anomaly_score, chang
from
    timeseries
order by
    time asc
```
SELECT
  time,
  -- x is double or array<double>
  -- sst(x) AS res
  sst(x, "-th 0.005") AS res
FROM
  twitter_timeseries
ORDER BY time ASC

Online mini-batch LDA

```
-- fitting

select
  label, word, avg(lambda) as lambda
from (  
  select
    train_lda(feature, "-topic 2 -iter 20")
    as (label, word, lambda)
  from data
) t1

group by label, word;

-- prediction

select
  t.docid,
  lda_predict(
    t.word, t.value, m.label, m.lambda,
    "-topic 2"
  ) as probabilities
from
  test t
  JOIN lda_model m ON (t.word = m.word)

group by
  t.docid;
```

<table>
<thead>
<tr>
<th>docid</th>
<th>probabilities (sorted by probabilities)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[&quot;label&quot;:0,&quot;probability&quot;:0.875],[&quot;label&quot;:1,&quot;probability&quot;:0.125]</td>
</tr>
<tr>
<td>2</td>
<td>[&quot;label&quot;:1,&quot;probability&quot;:0.9375],[&quot;label&quot;:0,&quot;probability&quot;:0.0625]</td>
</tr>
</tbody>
</table>
Other new features in development

- XGBoost Integration
- Sparse vector support in RandomForests
- Docker support (for evaluation)
- Field-aware Factorization Machines
- Generalized Linear Model
  - Optimizer framework including ADAM
  - L1/L2 regularization
Hivemall on Spark
Who am I

• Takeshi Yamamuro
• NTT corp. in Japan
• OSS activities
  • Apache Hivemall PPMC
  • Apache Spark contributor

Takeshi Yamamuro
@maropu

Research engineer - interested in DBs, compression, hardware-aware algorithms (e.g., SIMD, NUMA, GPU, ...), Apache Spark, Apache Hivemall, ....
What’s Spark?

• Distributed data analytics engine, generalizing Map Reduce
What’s Spark?

• 1. Unified Engine
  • support end-to-end APs, e.g., MLlib and Streaming

• 2. High-level APIs
  • easy-to-use, rich optimization

• 3. Integrate broadly
  • storages, libraries, …
What’s Hivemall on Spark?

• Hivemall wrapper for Spark
  • Wrapper implementations for DataFrame/SQL
  • + some utilities for easy-to-use in Spark

• The wrapper makes you...
  • run most of Hivemall functions in Spark
  • try examples easily in your laptop
  • improve some function performance in Spark
Why’s Hivemall on Spark?

• Hivemall already has many fascinating ML algorithms and useful utilities
  • + High barriers to add newer algorithms in MLlib

MLlib-specific Contribution Guidelines

While a rich set of algorithms is an important goal for MLLib, scaling the project requires that maintainability, consistency, and code quality come first. New algorithms should:

• Be widely known
• Be used and accepted (academic citations and concrete use cases can help justify this)
• Be highly scalable
• Be well documented
• Have APIs consistent with other algorithms in MLLib that accomplish the same thing
• Come with a reasonable expectation of developer support.

https://cwiki.apache.org/confluence/display/SPARK/Contributing+to+Spark
Current Status

• Most of Hivemall functions supported in Spark v2.0 and v2.1

- To compile for Spark v2.1

```
$ git clone https://github.com/apache/incubator-hivemall
$ cd incubator-hivemall
$ mvn package -Pspark-2.1-DskipTests
$ ls target/*spark*
```

```
target/hivemall-spark-2.1_2.11-XXX-with-dependencies.jar
...```
Current Status

• Most of Hivemall functions supported in Spark v2.0 and v2.1

- To compile for Spark v2.0

$ git clone https://github.com/apache/incubator-hivemall
$ cd incubator-hivemall
$ mvn package -Pspark-2.0-DskipTests
$ ls target/*spark*
target/hivemall-spark-2.0_2.11-XXX-with-dependencies.jar
...
4 Step Example

• 1. Fetch training and test data
• 2. Load these data in Spark
• 3. Build a model
• 4. Do predictions
1. Fetch training and test data

• E2006 tfidf regression dataset

```bash
```
2. Load data in Spark

// Download Spark-v2.1 and launch a spark-shell with Hivemall
$ <HIVEMALL_HOME>/bin/spark-shell

// Create DataFrame from the bzip’d libsvm-formatted file
scala> val trainDf = spark.read.format("libsvm").load("E2006.train.bz2")

scala> trainDf.printSchema
root
 |-- label: double (nullable = true)
 |-- features: vector (nullable = true)
Load in parallel because bzip2 is splittable

Load in parallel because bzip2 is splittable
3. Build a model - DataFrame

scala> paste:
val modelDf = trainDf.train_logregr("features", "label")
  .groupBy("feature")
  .agg("weight" -> "avg")
3. Build a model - SQL

```scala
scala> trainDf.createOrReplaceTempView("TrainTable")
scala> paste:
val modelDf = sql(""")
   | SELECT feature, AVG(weight) AS weight
   | FROM (  
   |   SELECT train_logregr(features, label)  
   |     AS (feature, weight)  
   |     FROM TrainTable  
   |   )  
   | GROUP BY feature
""".stripMargin)
```
4. Do predictions - DataFrame

```scala
scala> paste:
val df = testDf.select(rowid(), "$features")
  .explode_vector("features")
  .cache

# Do predictions
df.join(modelDf, df("feature") == model("feature"), "LEFT_OUTER")
  .groupBy("rowid")
  .avg(sigmoid(sum("weight" * "$value")))
```
4. Do predictions - SQL

scala> modelDf.createOrReplaceTempView("ModelTable")
scala> df.createOrReplaceTempView("TestTable")
scala> paste:
sql("",
| SELECT rowid, sigmoid(value * weight) AS predicted
| FROM TrainTable t
| LEFT OUTER JOIN ModelTable m
| ON t.feature = m.feature
| GROUP BY rowid
"").stripMargin)
Add New Features in Spark

• Top-K Join
  • Join two relations and compute top-K for each group

Use Spark vanilla APIs

```scala
scala> paste:
val topkDf = leftDf.join(rightDf, "group" :: Nil, "INNER")
  .select(leftDf("group"), (leftDf("x") + rightDf("y")).as("score"))
  .withColumn(
    "rank",
    rank().over(partitionBy("group").orderBy("score".desc))
  )
  .where("rank" <= topK)
```
Add New Features in Spark

• Top-K Join
  • Join two relations and compute top-K for each group

Use Spark vanilla APIs

scala> paste:
val topkDf = leftDf.join(rightDf, "group" :: Nil, "INNER")
  .select(leftDf("group"), (leftDf("x") + rightDf("y")).as("score")
  .withColumn(
    "rank", rank().over(partitionBy("group").orderBy("score".desc))
  )
  .where("rank" <= topK)

1. Join leftDf with rightDf
2. Sort data in each group
3. Filter top-K data
Add New Features in Spark

• Top-K Join
  • Join two relations and compute top-K for each group

Use a fused API implemented in Hivemall

```scala
scala> paste:
val topkDf = leftDf.top_k_join(
  lit(topK), rightDf, leftDf("group") === rightDf("group"),
  (leftDf("x") + rightDf("y")).as("score")
)
```
Add New Features in Spark

• How **top_k_join** works?

<table>
<thead>
<tr>
<th>group</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
leftDf

<table>
<thead>
<tr>
<th>group</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
rightDf
Add New Features in Spark

• How \texttt{top\_k\_join} works?

Compute top-K rows by using a priority queue

K-length priority queue
Add New Features in Spark

• How `top_k_join` works?

Compute top-K rows by using a priority queue

K-length priority queue

leftDf  Only join top-K rows

rightDf
Add New Features in Spark

• Codegen’d top_k_join for fast processing
  • Spark internally generates Java code from a built physical plan, and compiles/executes it

Spark Planner (Catalyst) Overview
Add New Features in Spark

- Codegen’d **top_k_join** for fast processing
  - Spark internally generates Java code from a built physical plan, and compiles/executes it

```java
long count = 0;
for (ss_item_sk in store_sales) {
    if (ss_item_sk == 1000) {
        count += 1;
    }
}
```
Add New Features in Spark

• Codegen’d `top_k_join` for fast processing
  • Spark internally generates Java code from a built physical plan, and compiles/executes it

```scala
scala> topkDf.explain
== Physical Plan ==  ‘*’ in the head means a codegen’d plan
*ShuffledHashJoinTopK 1, [group#10], [group#27]
  :- Exchange hashpartitioning(group#10, 200)
  :  +- LocalTableScan [group#10, x#11]
  +- Exchange hashpartitioning(group#27, 200)
    +- LocalTableScan [group#27, y#28]
```
Add New Features in Spark

• Benchmark Result

```scala
TestUtils.benchmark("top-k join query") {
  /**
   * Java HotSpot(TM) 64-Bit Server VM 1.8.0_31-b13 on MacOS X 10.10.2
   * Intel(R) Core(TM) i7-4578U CPU @ 3.00GHz
   *
   * top-k join (k=3): Best/Avg Time(ms) Rate(M/s) Per Row(ns) Relative
   *--------------------------------------------------------------------------------------------
   * join + rank                65959 / 71324        0.0      503223.9      1.0X
   * join + each_top_k          66093 / 78864        0.0      504247.3      1.0X
   * top_k_join                 5013 / 5431         0.0      38249.3       13.2X
   */
```

~13 times faster than vanilla APIs!!
Add New Features in Spark

• Support more useful functions
  • Spark only implements naive and basic functions in terms of usability and maintainability
  • e.g.,
    • flatten - flatten a nested schema into flat one
    • from_csv/to_csv – interconversion of CSV strings and structured data with schemas
    • ...
  • See more in the Hivemall user guide
    • https://hivemall.incubator.apache.org/userguide/spark/misc/misc.html
Improve Some Functions in Spark

- Spark has overheads to call Hive UD*Fs
  - Hivemall heavily depends on them

- ex.1) Compute a sigmoid function

```scala
scala> val sigmoidFunc = (d: Double) => 1.0 / (1.0 + Math.exp(-d))
scala> val sparkUdf = functions.udf(sigmoidFunc)
scala> df.select(sparkUdf("value"))
```
Improve Some Functions in Spark

- Spark has overheads to call Hive UD*Fs
  - Hivemall heavily depends on them

- ex.1) Compute a sigmoid function

```scala
scala> val hiveUdf = HivemallOps.sigmoid
scala> df.select(hiveUdf("value"))
```
Improve Some Functions in Spark

• Spark has overheads to call Hive UD*Fs
  • Hivemall heavily depends on them

• ex.1) Compute a sigmoid function

```scala
TestUtils.benchmark("closure/exprs/spark-udf/hive-udf") {
/**
 * Java HotSpot(TM) 64-Bit Server VM 1.8.0_31-b13 on Mac OS X 10.10.2
 * Intel(R) Core(TM) i7-4578U CPU @ 3.00GHz
 */

* sigmoid functions: Best/Avg Time(ns) Rate(M/s) Per Row(ns) Relative
* Exprs
* closure 7722 / 8342 3.4 294.6 1.0X
* Exprs 7708 / 8173 3.4 294.0 1.0X
* Exprs 7963 / 8350 3.3 303.8 1.0X
* Exprs 13977 / 14050 1.9 533.2 0.6X
*/
```
Improve Some Functions in Spark

• Spark has overheads to call Hive UD*Fs
  • Hivemall heavily depends on them

• ex.2) Compute top-k for each key group

```scala
scala> paste:
df.withColumn(
  "rank",
  rank().over(Window.partitionBy("key").orderBy("score".desc))
) .where("rank" <= topK)
```
Improve Some Functions in Spark

- Spark has overheads to call Hive UD*Fs
  - Hivemall heavily depends on them

- ex.2) Compute top-k for each key group

```scala
scala> df.each_top_k(topK, "key", "score", "value")
```

- Fixed the overhead issue for each_top_k
  - See pr#353: “Implement EachTopK as a generator expression” in Spark
Improve Some Functions in Spark

• Spark has overheads to call Hive UD*Fs
  • Hivemall heavily depends on them

• ex.2) Compute top-k for each key group

```scala
TestUtils.benchmark("top-k query") { 
  /**
   * top-k (k=100):       Best/Avg Time(ms)  Rate(M/s)  Per Row(ns)  Relative
   */
  * rank                   62748 / 62862       0.4        2393.6       1.0X
  * each_top_k (hive-udf) 41421 / 41736       0.6        1580.1       1.5X
  * each_top_k (exprs)    15793 / 16394       1.7        602.5        4.0X
  */
```

~4 times faster than rank!!
Support 3rd Party Library in Spark

**dmllc** supports **under development**
- fast implementation of the gradient tree boosting
- widely used in Kaggle competitions

**This integration will make you...**
- load built models and predict in parallel
- build multiple models in parallel for cross-validation
Conclusion and Takeaway

Hivemall is a *multi/cross-platform* ML library providing a collection of machine learning algorithms as Hive UDFs/UDTFs.

We welcome your contributions to Apache Hivemall 😊
Any feature request or questions?