UTILIZING ACCELERATORS TO SPEED UP ETL, ML, AND DL APPLICATIONS

Jason Lowe and Robert Evans, 05/19/2020
AGENDA

Accelerated ETL

Accelerated SQL/Dataframe

Accelerated Shuffle

What's Next
ACCELERATED ETL?
Can a GPU make an elephant fast?

TPCx—BB Like Benchmark Results (10TB Dataset, Two Nodes DGX-2 Cluster)*

Environment: Two DGX-2 (96 CPU Cores, 1.5TB Host memory, 16 V100 GPUs, 512 GB GPU Memory)

* Not official or complete TPCx-BB runs (ETL power only).
MODERN ML/DL WORKFLOW

Data Sources -> Ingest -> Data Store

Load -> Transform

CPU Compute

Training

GPU Compute

Model Training
APACHE SPARK 2.X

DISTRIBUTED, SCALE-OUT DATA SCIENCE AND AI APPLICATIONS

APACHE SPARK COMPONENTS
- Spark
- SQL/DF
- GraphX
- MLlib
- Streaming

ACCELERATED ML/DL FRAMEWORKS
- XGBoost
- TensorFlow
- PyTorch
- Horovod

SPARK 2.x CORE

CLUSTER MANAGEMENT/DEPLOYMENT (YARN, K8S, Standalone)

CPU Infrastructure
SPARK 3.X IS A UNIFIED AI PLATFORM

DISTRIBUTED, SCALE-OUT DATA SCIENCE AND AI APPLICATIONS

END-TO-END APACHE SPARK 3.0 PIPELINE

APACHE SPARK COMPONENTS
- Spark SQL/DF
- GraphX
- Streaming
- MLlib

RAPIDS Accelerator for Apache Spark

ACCELERATED ML/DL FRAMEWORKS
- XGBoost
- TensorFlow
- PyTorch
- Horovod

SPARK 3.0 CORE

CLUSTER MANAGEMENT/DEPLOYMENT (YARN, K8S, Standalone)

GPU-Accelerated Infrastructure
ETL + ML/DL WORKFLOW

Data Sources → Ingest → Load → Transform → Data Store → Model Training → GPU Compute
Anonymized 7-day clickstream dataset (1 TB)
Convert high cardinality string categorical data to contiguous integer ids
DLRM github repo has scripts for this out of the box
DLRM ON CRITEO DATASET (PAST)

ETL & Training Run Time for CPU & GPU
CRITEO DATASET (1TB)

* Extrapolated couldn't convince anyone to wait that long
DLRM ETL ON CRITEO DATASET (PRESENT)

Spark ETL for CRITEO DATASET (1TB)

<table>
<thead>
<tr>
<th>Time (Hours)</th>
<th>Spark ETL (96 core CPU)</th>
<th>Spark ETL (1 - V100)</th>
<th>Spark ETL (8 - V100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1</td>
<td></td>
<td>2.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>
DLRM END-TO-END ON CRITERIO DATASET (PRESENT)

Spark ETL + Training for Criteo Dataset (1TB)

- 160x faster than original
- 48x faster than CPU (4% the cost)
- 10x faster than typical (1/6th the cost)

<table>
<thead>
<tr>
<th></th>
<th>Time (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original CPU (1 Core for ETL, 96 Core CPU for Training)</td>
<td>144.0</td>
</tr>
<tr>
<td>Spark CPU (96 Core for ETL &amp; Training)</td>
<td>45.0</td>
</tr>
<tr>
<td>Spark CPU (96 Core for ETL &amp; Spark GPU (1-V100 Training))</td>
<td>0.7</td>
</tr>
<tr>
<td>Spark GPU (8-V100 for ETL &amp; 1-V100 Training)</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- Training: 45.0
- ETL: 144.0

Spark ETL + Training for Criteo Dataset (1TB)

Compute Engine:
- CPU + GPU MLP
- GPU (100-200)
- NVCC (V100)
“The more you buy, the more you save.”

— Jensen Huang, GTC 2018
RAPIDS ACCELERATOR FOR APACHE SPARK (PLUGIN)

DISTRIBUTED SCALE-OUT SPARK APPLICATIONS

APACHE SPARK CORE

- Spark SQL API
- DataFrame API
- Spark Shuffle

if gpu_enabled(operation, data_type)
call-out to RAPIDS
else
execute standard Spark operation

RAPIDS Accelerator for Spark

- Custom Implementation of Spark Shuffle
- Optimized to use RDMA and GPU-to-GPU direct communication

JNI bindings
Mapping From Java/Scala to C++

RAPIDS C++ Libraries

CUDA

UCX Libraries
No Code Changes

(None)

Same SQL and Dataframe code.

```python
spark.conf.set("spark.rapids.sql.enabled", "true")

start = time.time()
spark.sql(""
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1993-07-01'
    and o_orderdate < date '1993-07-01' + interval '3' month
    and exists (select *
        from
        lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority
).show()
time.time() - start
```
WHAT WE SUPPORT
and growing...

!  ^  concat  double  input_file_block_length  locate  nanvl  rand*  sinh  ucase
%  abs  cos  e  input_file_block_start  log  negative  regexp_replace  smallint  upper
&  acos  cosh  exp  input_file_name  log1  not  replace  spark_partition_id  window
*  expm1  first  int  log2  nullif  rint  sqrt  year
+  asin  count  first_value  int2  lower  nvl  rollup  string  |  limit
and  cube  first  isan  lcase  like  ln  posexplode*
+  count  floor  isnotnull  lower  mean  or  pow  position
-  atan  from_unixtime  isnull  min  pi  pow  power
/  avg  float  last  minute  position  power  power
<  bigint  floor  last_value  mod  position  pow  power
<=  boolean  floor  last_value  mod  position  pow  pow
<=>  cast  from_unixtime  last_value  mod  position  pow  pow
=  cbrt  hour  lcase  monotone_increase_id  month
>=  ceil  hour  lcase  month  month
coalesce  ifnull  in  initcap  month
degrees  ifnull  input_file_name  input_file_block_length
int
like
ln
log
log10
log1p
mean
max
min
mod
month
negative
not
now
nullif
growing

IS THIS A SILVER BULLET?

NO

Small amounts of data
   Few hundred MB per partition for GPU
Highly cache coherent processing
Data Movement
   Slow I/O (networking, disks, etc.)
   Going back and forth to the CPU (UDFs)
Shuffle
Limited GPU Memory
BUT IT CAN BE AMAZING

What the SQL plugin excels at

High cardinality joins
High cardinality aggregates
High cardinality sort
Window operations (especially on large windows)
Complicated processing
Transcoding (Writing Parquet and ORC is hard, reading CSV is hard)
HOW DOES IT WORK
SPARK SQL & DATAFRAME COMPILATION FLOW

QUERY

SELECT product_id, ds, max(price) - min(price) AS range FROM bar GROUP BY product_id, ds

bar.groupBy(
  col("product_id"),
  col("ds"))
  .agg(
    max(col("price")) -
    min(col("price")).alias("range"))
SPARK SQL & DATAFRAME COMPILATION FLOW

**QUERY**

```sql
SELECT product_id, ds, 
max(price) - min(price) AS range 
FROM bar GROUP BY 
product_id, ds
```

**GPU PHYSICAL PLAN**

```
bar.groupBy(
  col("product_id"),
  col("ds"))
  .agg(
    max(col("price")) - 
    min(col("price")).alias("range"))
```

**RAPIDS SQL Plugin**

- DataFrame
- Logical Plan
- Physical Plan
- GPU Physical Plan
- RDD[ColumnarBatch]
- RDD[InternalRow]
SPARK SQL & DATAFRAME COMPILATION FLOW

CPU PHYSICAL PLAN

GPU PHYSICAL PLAN

1. Read Parquet File
2. Convert to Row Format
3. First Stage Aggregate
4. Shuffle Exchange
5. Combine Shuffle Data
6. Second Stage Aggregate
7. Write Parquet File
8. Convert to Row Format
ETL TECHNOLOGY STACK

- Dask cuDF
- cuDF, Pandas
- Python
- Cython

- Spark dataframes, Scala, PySpark
- Java
- JNI bindings

- RAPIDS cuDF C++
- Apache Arrow

- CUDA Libraries

- CUDA
ACCELERATED SHUFFLE
SPARK SHUFFLE
Data Exchange Between Stages

Stage 1
Task 0  Task 1  Task 2

Stage 2
Task 0  Task 1
SPARK SHUFFLE
Cpu-Centric Data Movement
ACCELERATED SPARK SHUFFLE

GPU-Centric Data Movement

- CPU
- GPU 1
- GPU 0
- Network
- Local Storage

PCI-e Bus
- RDMA
- GPU Direct Storage
- NVLink
ACCELERATED SPARK SHUFFLE

Shuffling Spilled Data

Host Memory

PCI-e Bus

CPU

GPU 1

GPU 0

Network

Local Storage

RDMA
UCX LIBRARY
Unified Communication X

Abstracts communication transports

Selects best available route(s) between endpoints
TCP, RDMA, Shared Memory, GPU

Zero-copy GPU memory transfers over RDMA

RDMA requires network support (IB or RoCE)

http://openucx.org
ACCELERATED SHUFFLE RESULTS

Inventory Pricing Query

<table>
<thead>
<tr>
<th></th>
<th>Query Duration in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>228</td>
</tr>
<tr>
<td>GPU</td>
<td>45</td>
</tr>
<tr>
<td>GPU+UCX</td>
<td>8.4</td>
</tr>
</tbody>
</table>
ACCELERATED SHUFFLE RESULTS
ETL for Logistical Regression Model

<table>
<thead>
<tr>
<th></th>
<th>Query Duration in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1556</td>
</tr>
<tr>
<td>GPU</td>
<td>172</td>
</tr>
<tr>
<td>GPU+UCX</td>
<td>79</td>
</tr>
</tbody>
</table>
WHAT’S NEXT?
WHAT’S NEXT

Open Source/Spark 3.0 Release
Nested types Arrays, Structs, and Maps
Decimal type
More operators

COMING SOON

GPU Direct Storage
Time zone support for timestamps (only UTC for now)
Higher order functions
UDFs

FURTHER OUT
WHERE TO GET MORE INFO

Learn more about the RAPIDS Accelerator for Apache Spark

Visit: NVIDIA.com/Spark

Please use the “contact us” to get in touch with NVIDIA’s Spark team

Listen to how Adobe Email Marketing Intelligent Services leverages the RAPIDS Accelerator & Spark 3.0 on Databricks

Upcoming Spark+AI Summit Sessions on GPU support for Apache Spark 3.0:

Deep Dive into GPU Support in Apache Spark 3.x

Scalable Acceleration of XGBoost Training on Apache Spark GPU Clusters

Preview of Spark 3.0 GPU Features: NVIDIA.com/Spark-Book

QUESTIONS
FAQS

Q: What are the minimum requirements?

A: The RAPIDS accelerator requires:

- Apache Spark 3.0
- RAPIDS cudf 0.14
- CUDA 10.1 or later
- NVIDIA GPU with Pascal architecture or later
- Ubuntu 16.04+ or CentOS 7+
FAQS

Q: Do all cluster nodes require GPUs?
A: All Spark executors running with the RAPIDS accelerator require their own GPU.
   The Spark driver process does not require a node with a GPU.

Q: Can I run more than one executor per GPU?
A: No, there must be a one-to-one mapping between Spark executors and GPUs.
   You can run more than one concurrent task per executor.
FAQS

Q: Will the RAPIDS accelerator work in the cloud?
A: Yes, if the VM environment meets the minimum requirements.

Q: Will the RAPIDS accelerator be available for Apache Spark 2.x?
A: No. The columnar processing APIs added in Apache Spark 3.0 are required.

Q: How can I tell if an operation is being accelerated?
A: Accelerated operations appear in the query explanation and SQL UI.
RAPIDS ACCELERATOR CONFIGURATION

spark.rapids.sql.enabled is the master enable

spark.rapids.sql.explain enables logging of operations not accelerated

spark.rapids.sql.concurrentGpuTasks controls concurrent task count per GPU
SPARK ACCELERATOR-AWARE SCHEDULING

Tracking JIRA: SPARK-24615

Request executor and driver resources (GPU, FPGA, etc.)

Resource discovery

Specify task resources

API to determine assigned resources

YARN, Kubernetes, and Standalone
SPARK ACCELERATOR-AWARE SCHEDULING

Sample Command-Line

./bin/spark-shell --master yarn --executor-cores 2 \
   --conf spark.driver.resource.gpu.amount=1 \
   --conf spark.driver.resource.gpu.discoveryScript=/opt/spark/getGpuResources.sh \
   --conf spark.executor.resource.gpu.amount=2 \
   --conf spark.executor.resource.gpu.discoveryScript=./getGpuResources.sh \
   --conf spark.task.resource.gpu.amount=1 \
   --files examples/src/main/scripts/getGpusResources.sh
SPARK STAGE LEVEL SCHEDULING

Tracking JIRA: SPARK-27495

Specify task resource requirements per RDD operation

Dynamically allocates containers to meet resource requirements

Schedules tasks on appropriate containers