Bringing Data to Life
Data management and Visualization Techniques

Benika Hall
Rob Harrison
Corporate Model Risk

March 16, 2018

Together we’ll go far
Introduction

- Benika Hall
  Analytic Consultant
  Wells Fargo - Corporate Model Risk

- Rob Harrison
  Analytic Consultant
  Wells Fargo - Corporate Model Risk
Presentation Outline

- Business Model of the Financial Industry
- About the Data
  - Financial Data Flow Overview
  - Complexities of Data
  - Ingestion
  - Preparation
  - Pre-processing
  - Data Exploration
- Available Solutions
- Regression Analysis with GPU databases
  - Fannie Mae Example
Business Model of the Financial Industry

- What do banks do?

- Comparison to other industries

- Financial Industry Business Model
  - Provide financial services
  - Manage your money
  - Working with data!
INFUSION STATE:
As data progresses through each stage through the lifecycle, it is infused with value from raw low value human/machine level to highly refined analytical output.

It is here that we apply machine learning algorithms to infuse value into the data.
Complexities of Data

- Format
- Sensitivity
- Volume
- Variety
- Lineage
- Governance
- Multi-sourced
- Velocity
- Context
Phases of Data Management & Analysis

- **Ingestion** – pulling data from source system into local storage environment
- **Preparation** – data merging based on context and analytical needs
- **Pre-processing** – feature engineering, missing value imputation, outlier detection, categorical feature encoding, binning, etc.
Data Ingestion

- **Ingestion** – pulling data from source system into local storage environment
  - Sqoop
  - Kafka
  - SAS Connectors
  - NiFi
Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import sequential datasets from mainframe</td>
<td>Satisfies the growing need to move data from mainframe to HDFS</td>
</tr>
<tr>
<td>Import direct to ORCFiles</td>
<td>Improved compression and light-weight indexing for improved query performance</td>
</tr>
<tr>
<td>Data imports</td>
<td>Moves certain data from external stores and EDWs into Hadoop to optimize cost-effectiveness of combined data storage and processing</td>
</tr>
<tr>
<td>Parallel data transfer</td>
<td>For faster performance and optimal system utilization</td>
</tr>
<tr>
<td>Fast data copies</td>
<td>From external systems into Hadoop</td>
</tr>
<tr>
<td>Efficient data analysis</td>
<td>Improves efficiency of data analysis by combining structured data with unstructured data in a schema-on-read data lake</td>
</tr>
<tr>
<td>Load balancing</td>
<td>Mitigates excessive storage and processing loads to other systems</td>
</tr>
</tbody>
</table>

YARN coordinates data ingest from Apache Sqoop and other services that deliver data into the Enterprise Hadoop cluster.
Apache Kafka is an open-source stream processing software platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

Kafka has four core APIs:

- The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.
- The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The **Streams API** allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.
SAS Connectors

- SAS/ACCESS Interface to Hadoop
- SAS Data Loader
NiFi

Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.
NiFi (continued)

- **Systems fail:** Networks fail, disks fail, software crashes, people make mistakes.

- **Data access exceeds capacity to consume:** Sometimes a given data source can outpace some part of the processing or delivery chain - it only takes one weak-link to have an issue.

- **Boundary conditions are mere suggestions:** You will invariably get data that is too big, too small, too fast, too slow, corrupt, wrong, or in the wrong format.

- **What is noise one day becomes signal the next:** Priorities of an organization change - rapidly. Enabling new flows and changing existing ones must be fast.

- **Systems evolve at different rates:** The protocols and formats used by a given system can change anytime and often irrespective of the systems around them. Dataflow exists to connect what is essentially a massively distributed system of components that are loosely or not-at-all designed to work together.

- **Compliance and security:** Laws, regulations, and policies change. Business to business agreements change. System to system and system to user interactions must be secure, trusted, accountable.

- **Continuous improvement occurs in production:** It is often not possible to come even close to replicating production environments in the lab
Data Preparation

- Data Preparation -> requires data exploration!
  - GPU Databases
  - Hive
  - Hive LLAP
  - Spark
  - Alluxio
  - Druid
Hive & LLAP

- Live Long and Process, LLAP provides a hybrid execution model. It consists of a long-lived daemon which replaces direct interactions with the HDFS DataNode, and a tightly integrated DAG-based framework. Functionality such as caching, pre-fetching, some query processing and access control are moved into the daemon. Small/short queries are largely processed by this daemon directly, while any heavy lifting will be performed in standard YARN containers.

Hive 2 with LLAP: Architecture Overview

© 2017 Wells Fargo Bank, N.A. All rights reserved. For public use.
Apache Spark is a lightning-fast **unified analytics engine** for big data and machine learning. It was originally developed at UC Berkeley in 2009.

It provides in-memory computing capabilities to deliver speed, a generalized execution model to support a wide variety of applications, and Java, Scala, and Python APIs for ease of development.

**Supported languages:**
- R
- SQL
- Python
- Scala
- Java
Alluxio holds a unique place in the big data ecosystem, residing between **storage** systems such as Amazon S3, Apache HDFS or OpenStack Swift and **computation** frameworks and applications such as Apache Spark or Hadoop MapReduce and provides the central point of access with a memory centric design.
Druid

- Druid is an open-source data store designed for sub-second queries on real-time and historical data.
Data Pre-processing

- Feature engineering, missing value imputation, outlier detection, categorical feature encoding, binning, etc.

- Languages:
  - Python
  - R
  - SAS
  - Scala

- Why PySpark?
  - Leverage python
  - Big data
  - Flexibility
Apache Arrow - GOAI

- What is it?

Advantages of a Common Data Layer

- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (e.g., Parquet-to-Arrow reader)

- How does this differ from Parquet? Alluxio?
Apache Arrow

- Columnar In-Memory Storage
Apache Arrow Example

- Example of usage:

```python
from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark.sql import Row
import time

#Configure spark session
conf = SparkConf().setAppName("Spark Arrow 032318")
conf.setMaster("local")
spark = SparkSession.builder.enableHiveSupport().config(conf=conf).getOrCreate()

#Load data
df_spark_default = spark.sql("select * from default.fannie_mae_mgd_1mm_txt limit 600000")

%time df_original = df_spark_default.toPandas()
CPU times: user 6.56 s, sys: 624 ms, total: 7.18 s
Wall time: 11.2 s

#Read into Arrow
table = pa.Table.from_pandas(df_original)

#Write into into parquet table format in Arrow
pq.write_table(table, 'example.parquet')

#Read into arrow data frame
df_parquet_arrow = pq.read_table('example.parquet')

%time df_arrow = df_parquet_arrow.to_pandas()
CPU times: user 289 ms, sys: 75.6 ms, total: 365 ms
Wall time: 323 ms
```
Data Exploration w/GPU Technology

- Overview and examples of each of the following:
  - Immerse
  - Superset
  - Reveal
Superset

World's Bank Data

Region Filter
- region
  - Select [region]
- country_name
  - Select [country_name]

Growth Rate

World's Population

7.24G

Rural Breakdown

Most Populated Countries
- country_name
  - China: 1.36G
  - India: 1.30G
  - United States: 319M

© 2017 Wells Fargo Bank, N.A. All rights reserved. For public use.
Regression Analysis using GPU

- Example of running regression analysis using GPU for large-scale data and calculating variable importance:
  - Fannie Mae data of 1.8 billion rows
  - Gradient Boosting Trees
  - Hyper-parameter tuning
  - Determine Variable Importance
# GPU Specifications

<table>
<thead>
<tr>
<th>NVIDIA-SMI 384.81</th>
<th>Driver Version: 384.81</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Name</td>
<td>Persistence-M</td>
</tr>
<tr>
<td>Fan</td>
<td>Temp</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0 Tesla P100-SXM2... Off</td>
<td>00000002:01:00.0 Off</td>
</tr>
<tr>
<td>N/A</td>
<td>29C</td>
</tr>
</tbody>
</table>
Variable Importance using GBM

<table>
<thead>
<tr>
<th>variable</th>
<th>relative_importance</th>
<th>scaled_importance</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>month_on_book</td>
<td>67.9615631</td>
<td>1.0</td>
<td>0.4004045</td>
</tr>
<tr>
<td>bin_fico</td>
<td>41.4487762</td>
<td>0.6098856</td>
<td>0.2442009</td>
</tr>
<tr>
<td>bin_dti</td>
<td>38.9857483</td>
<td>0.5736441</td>
<td>0.2296896</td>
</tr>
<tr>
<td>purpose</td>
<td>7.5061255</td>
<td>0.1104466</td>
<td>0.0442233</td>
</tr>
<tr>
<td>bin_oltv</td>
<td>7.0943146</td>
<td>0.1043872</td>
<td>0.0417971</td>
</tr>
<tr>
<td>prop_typ</td>
<td>3.5422773</td>
<td>0.0521218</td>
<td>0.0208698</td>
</tr>
<tr>
<td>occ_stat</td>
<td>2.0115166</td>
<td>0.0295979</td>
<td>0.0118511</td>
</tr>
<tr>
<td>fthb_flg</td>
<td>1.1819650</td>
<td>0.0173917</td>
<td>0.0069637</td>
</tr>
</tbody>
</table>
## Performance comparison using GPU vs. CPU

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to load data (1.5 million rows)</td>
<td>1.5 hours</td>
<td>10 secs</td>
</tr>
<tr>
<td>Time to run model</td>
<td>2 hours</td>
<td>50 secs</td>
</tr>
<tr>
<td>Hyper-parameter tuning</td>
<td>4.5 hours</td>
<td>200 secs</td>
</tr>
<tr>
<td>Total</td>
<td>8 hours</td>
<td>260 secs</td>
</tr>
</tbody>
</table>
Summary

What have we covered?

- Financial Data Overview
- Data Ingestion
- Data Preparation
- Current Solutions
- Data visualization
- Examples of Machine Learning using GPUs
Thank you!