Adaptive Executive Layer with Pentaho Data Integration
An Introduction to AEL and the AEL Spark Engine

Jonathan Jarvis
Senior Solutions Engineer / Engineering Services
June 26th, 2018
Agenda

AEL Overview

AEL Spark Engine

Word Count Demonstration

Best Practices for AEL Spark

Q & A
AEL Overview
The Adaptive Execution Layer (AEL)

- Develop Once, Choose the Execution Engine
  - Easily develop transformations in PDI’s drag-and-drop design environment
  - Switch between execution engines to fit data volume and transformation complexity
  - Utilize emerging technologies without being a Java, Scala, or Python developer
How to Use AEL

1. Create a new run configuration
2. Specify execution engine properties
3. Invoke from PDI by selecting the desired run configuration
AEL Spark Engine
Apache Spark chosen for the first AEL engine implementation

Simplicity of PDI unleashes the power of Spark

With Pentaho 8.1, AEL Spark supports these Hadoop distributions:
- CDH 5.13
- HDP 2.6
- EMR 5.9
- MapR 5.2

Runs in Spark local or with YARN resource management
AEL Spark Reference Architecture

Hadoop Cluster

Spark/Hadoop Processing Nodes

Spark Executors

Hadoop/Spark Compatible Storage Cluster

HDFS
Amazon S3
etc...

AEL Spark Engine (Spark Driver)

AEL Spark Daemon on Edge Node

PDI Clients
AEL Spark Setup

- Build an AEL Spark daemon
  - Use script packaged with PDI
  - Custom tailor build to include desired step plugins
- Move the package to an edge node and install in HDFS
- Configure the AEL Spark daemon’s properties
- Start the AEL Spark daemon on an edge node
Spark processes data in **partitions** within **executor** processes distributed across a cluster:

- Large web access log file broken into partitioned input splits

```
2018-01-01 00:00:00T000 GET
2018-01-01 00:00:00T001 GET
...
2018-01-01 01:00:00T000 GET

2018-01-02 00:00:00T000 GET
2018-01-02 00:00:00T001 GET
...
2018-01-02 02:00:00T000 GET
...

2018-05-31 00:00:00T000 GET
2018-05-31 00:00:00T001 GET
...
2018-05-31 23:00:00T000 GET
```
PDI Step Implementations with AEL Spark

- **PDI Steps**
  - **Kettle Implementation**
    - **Distributable**: Step code used across all executors in parallel
    - **Non-Distributable**: Data partitions are coalesced and processed by a single executor
  - **Overridden Spark Implementation**: Step metadata supplies configuration to use Spark’s built-in distributed processing capabilities
Distributable Steps

- **Kettle Step Implementation**
  - Distributed to Spark executors
  - Entire stream is processed as data partitions

- **Steps that do not hold state between rows**
  - Calculator
  - Split field to rows
  - String operations
  - Value Mapper
  - ...
Non Distributable Steps

- Some steps currently do not support parallel execution
  - Steps where “Number of copies” would be left at one
  - Overridden Spark implementations can provide distributed functionality

- AEL protectively adds a coalesce(1)
  - Steps work with AEL Spark
  - Data processed on single executor thread
  - Produce correct results
  - Controlled by the `forceCoalesceSteps` list in `org.pentaho.pdi.engine.spark.cfg`
# Steps with Overridden Spark Implementations

## Input/Output
- Text file input/output
- Hadoop File Input/Output
- Avro Input/Output
- Parquet Input/Output
- ORC Input/Output

## ETL/Analysis
- Filter rows
- Group by / Memory Group by
- Merge Join
- Sort rows
- Stream lookup
- Unique rows / Unique rows (HashSet)

## Streaming
- Get records from stream
- Kafka Consumer
- MQTT Consumer

## Subtransformation
- Transformation Executor
- Get rows from / Copy rows to result
- Abort
Word Count Demonstration
Best Practices for AEL Spark Transformations
Use Lookup Instead of Merge Join

When?
- You want to perform an INNER, LEFT, or RIGHT join
- One stream has a small set of rows that can fit in memory
- The small stream is accessed by a set of fields that form a unique key

Why?
- The Lookup step has an overridden AEL Spark implementation
- Utilizes Spark’s broadcast feature to send the small stream to each executor
- Join can be done without moving larger stream around cluster network
Remove Unnecessary Sort rows Steps

- **When?**
  - Your transformation is being setup exclusively for AEL Spark
  - Transformation has **Merge Join, Unique Rows, or Group by**

- **Why?**
  - The overridden Spark step implementations do not require the inputs to be sorted
  - No Sort rows step is executed, which requires a network data shuffle/transfer

Use **Unique rows (HashSet)** or **Memory Group by** for transformations to work in both Pentaho local (Kettle) and AEL Spark
Configure the Spark History Server

- **When?**
  - You want to trace through Spark execution for tuning
  - View Spark application run/job history

- **How?**
  - Set `sparkEventLogEnabled` to true
  - Configure the `sparkEventLogDir` location, found on Spark History Server UI

- **Where?**
  - The PDI AEL Spark daemon’s `application.properties` file
Tune with Parameters, Variables, or Properties

Why?
- You want to tune a Spark setting (e.g. executor memory) for a certain transformation or for all transformations launched.

How?
- All “spark.” properties in `application.properties`, transformation variables or parameters will be forwarded to the Spark driver’s configuration.
- Precedence:
  1. Transformation Parameters
  2. Transformation Variables
  3. Daemon’s `application.properties` file
Re-use Spark Sessions During Development

- **Why?**
  - During development, this is useful to tweak transformations and re-execute
    - The daemon keeps the driver and executors alive without an active transformation

- **Production?**
  - **No**: This consumes resources that may be useful for other cluster tasks
  - **No**: Reduces traceability in the Spark History Server

- **How?**
  - Add `KETTLE_AEL_PDI_DAEMON_CONTEXT_REUSE=true` to `kettle.properties` on the development client machine (not the daemon)
Control Partitioning with Data Preparation

- **When?**
  - Data is typically dropped in HDFS to process with a batch AEL Spark transformation

- **How?**
  - Create files in the ingest directory that correspond to the number of desired partitions to process

- **Why?**
  - Spark’s parallelism is dictated by the number of files input and their split points
Caution Using External Resources

- **When?**
  - A transformation has a step that utilizes an external resource, like a REST Client or Database lookup step

- **Why?**
  - Spark executors could be executing the step code on many threads of many executors
  - Chance of a self-inflicted Denial of Service attack
Caution Using Coalesced Steps

- **When?**
  - A transformation has a step that is on the `forceCoalesceSteps` list

- **Why?**
  - All data must be processed by a single thread of a single executor

- **Tip**
  - If the transformation allows, try to use these steps after summary aggregations, filtering or pruned data, or on smaller data streams
Questions
Thank You