Apache Spark & Parquet in AWS

Thaer Khawaja

Hothead Games

✉ tkhawaja@hotheadgames.com

April 25, 2017

Table of Contents

1 Case Study

2 Tech Overview

3 ETL Integration

4 Documentation

5 Dev Horizon

Case Study

1 Case Study

2 Tech Overview

3 ETL Integration

4 Documentation

5 Dev Horizon

Case Study - New Game, Unexpected Data

- First game to encode and log in-mission gameplay event data
  - decodes into processable data inflates data up to $\sim 30x$ the size
  - contains all game events
    - damage taken per character, where and when
    - enemies killed
    - abilities used
    - "ghost" actions (team AI kills, actions, maneuvers, etc.)
    - and more ...
  - valuable for gameplay parameter tunings
- Analytics data warehouse (Redshift) not a suitable intermediary
  - postprocessing summary required on intermediary decoded data would cause heavy load
  - would impact response time on actively used cluster by analysts and data scientists
  - inflated data size would require us to unnecessarily provision cluster upwards
  - postprocessed summaries still need to be on Redshift (for now)

Tech Overview

1 Case Study

2 Tech Overview

- Apache Spark
- Apache Parquet
- ETL Pipeline

3 ETL Integration

4 Documentation

5 Dev Horizon

Spark - Cluster Computing Framework

Highlights:

- Supports batch and “real-time” (mini-batch) data processing
  - Guarantees no duplicate processed output
- AWS EMR-friendly (runs on Hadoop YARN)
- Commercially available Databricks solution (runs on Mesos)
- “All the rage” framework with large community support

Lowlights:

- Not a true real-time streaming framework
  - Not an issue for our use case
  - Seasoned alternative: Apache Storm with Trident
  - New alternatives: Apache Flink and Apache Apex
- Databricks DBU units cost $$$
  - One r3.8xlarge instance = 8 DBUs

Parquet - Fancy File Format

Highlights:

- Column-oriented format
  - Space-efficient data encodings
  - Queries only the data that is being ask for
  - Supports predicate pushdown
- Really good at dealing with wide and nested JSON data
- Automated schema discovery
  - Not magical and breaks on complex loosely structured schemas
- Integrated support with Apache Spark (and others)
- Benchmarks put Parquet as best performer for our use case

Lowlights:

- ORC seems to outperform on flatter data schemas
  - Possibly because Parquet doesn't have bloom filters (PARQUET-41)

Current ETL Pipeline Tech Stack

- **S3** - Data lake
- **Redshift** - Data warehouse
- **EMR** - ETL workhorse
- **Data Pipeline** - Workflow scheduler
- **Kinesis** - Data streams
- **Python** - Code glue, MapReduce, and business logic

ETL Integration

1 Case Study

2 Tech Overview

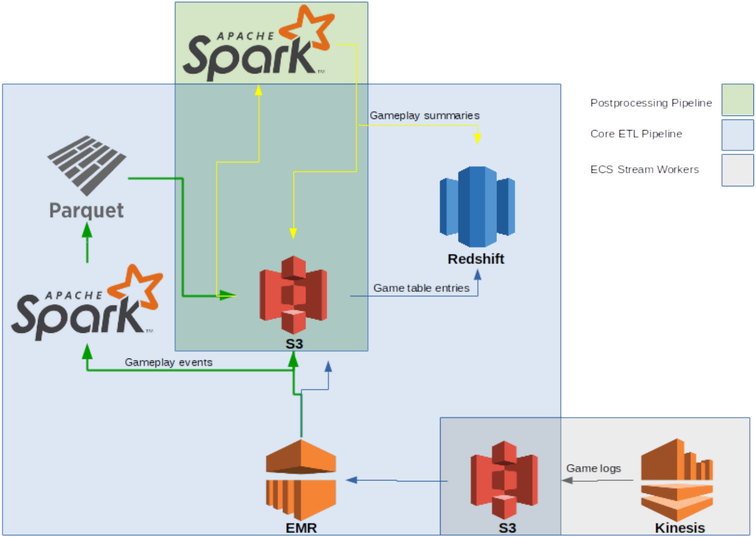
3 ETL Integration

- Diagram
- Parquet Conversion
- Spark Submit

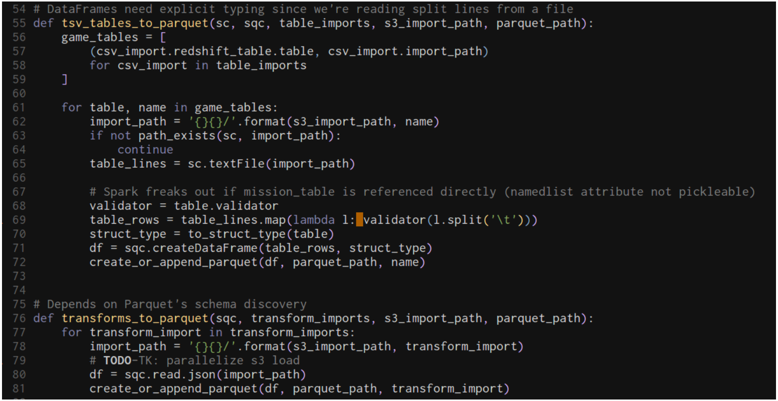
4 Documentation

5 Dev Horizon

ETL Integration - Diagram

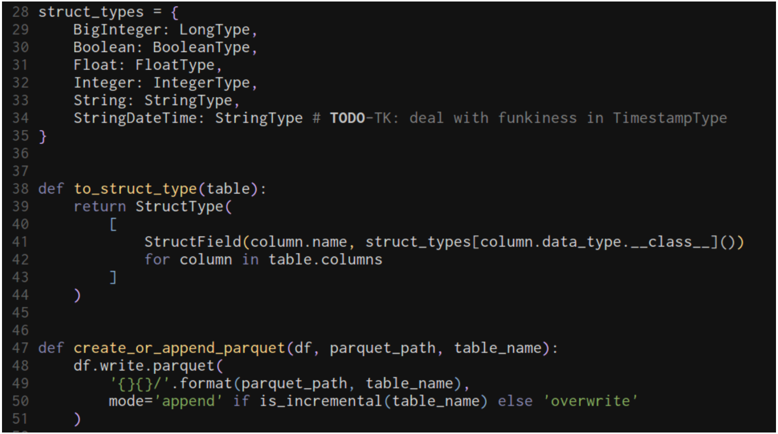


Parquet Conversion - TSV and JSON



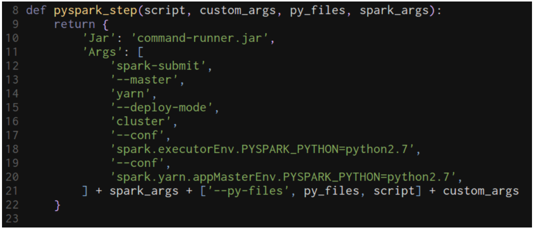
```python
# DataFrames need explicit typing since we're reading split lines from a file
def tsv_tables_to_parquet(sc, sqc, table_imports, s3_import_path, parquet_path):
    game_tables = [
        (csv_import.redshift_table.table, csv_import.import_path)
        for csv_import in table_imports
    ]

    for table, name in game_tables:
        import_path = '{}{}/'.format(s3_import_path, name)
        if not path_exists(sc, import_path):
            continue
        table_lines = sc.textFile(import_path)

        # Spark freaks out if mission_table is referenced directly (namedlist attribute not pickleable)
        validator = table.validator
        table_rows = table_lines.map(lambda l:validator(l.split('\t')))
        struct_type = to_struct_type(table)
        df = sqc.createDataFrame(table_rows, struct_type)
        create_or_append_parquet(df, parquet_path, name)


# Depends on Parquet's schema discovery
def transforms_to_parquet(sqc, transform_imports, s3_import_path, parquet_path):
    for transform_import in transform_imports:
        import_path = '{}{}/'.format(s3_import_path, transform_import)
        # TODO-TK: parallelize s3 load
        df = sqc.read.json(import_path)
        create_or_append_parquet(df, parquet_path, transform_import)
```

Parquet - Tables & Write Mode

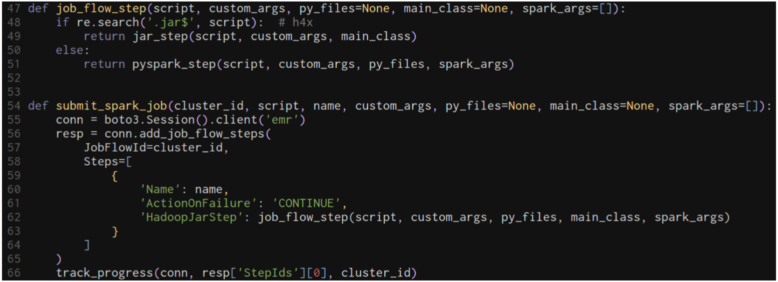


```python
struct_types = {
    BigInteger: LongType,
    Boolean: BooleanType,
    Float: FloatType,
    Integer: IntegerType,
    String: StringType,
    StringDateTime: StringType # TODO-TK: deal with funkiness in TimestampType
}


def to_struct_type(table):
    return StructType(
        [
            StructField(column.name, struct_types[column.data_type.__class__]())
            for column in table.columns
        ]
    )


def create_or_append_parquet(df, parquet_path, table_name):
    df.write.parquet(
        '{}{}/'.format(parquet_path, table_name),
        mode='append' if is_incremental(table_name) else 'overwrite'
    )
```

Spark Submit - On EMR



```python
def pyspark_step(script, custom_args, py_files, spark_args):
    return {
        'Jar': 'command-runner.jar',
        'Args': [
            'spark-submit',
            '--master',
            'yarn',
            '--deploy-mode',
            'cluster',
            '--conf',
            'spark.executorEnv.PYSPARK_PYTHON=python2.7',
            '--conf',
            'spark.yarn.appMasterEnv.PYSPARK_PYTHON=python2.7',
        ] + spark_args + ['--py-files', py_files, script] + custom_args
    }

```



```python
def job_flow_step(script, custom_args, py_files=None, main_class=None, spark_args=[]):
    if re.search('.jar$', script):  # h4x
        return jar_step(script, custom_args, main_class)
    else:
        return pyspark_step(script, custom_args, py_files, spark_args)


def submit_spark_job(cluster_id, script, name, custom_args, py_files=None, main_class=None, spark_args=[]):
    conn = boto3.Session().client('emr')
    resp = conn.add_job_flow_steps(
        JobFlowId=cluster_id,
        Steps=[
            {
                'Name': name,
                'ActionOnFailure': 'CONTINUE',
                'HadoopJarStep': job_flow_step(script, custom_args, py_files, main_class, spark_args)
            }
        ]
    )
    track_progress(conn, resp['StepIds'][0], cluster_id)
```

Documentation

1 Case Study

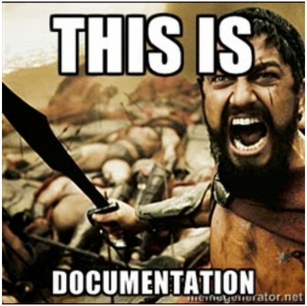
2 Tech Overview

3 ETL Integration

4 Documentation

5 Dev Horizon

Documentation



PySpark docs: `https://spark.apache.org/docs/2.1.0/api/python/pyspark.html`
Boto3 docs: `https://boto3.readthedocs.io/en/latest/`

Dev Horizon

1 Case Study

2 Tech Overview

3 ETL Integration

4 Documentation

5 Dev Horizon

Dev Horizon - Upcoming Changes

- Add a Parquet copy of our data lake
  - ad-hoc queries for a given user or set of users
  - Partitioning by day/week/month
  - integration tests with randomly sampled data
- Leverage Spark Streaming to improve ETL data freshness
- Migrate post-processing queries from Redshift to Spark
- Alternatives away from Redshift as a data warehouse (???)
  - AWS Athena supports ANSI SQL and has JDBC driver
  - Spark SQL is mature and more testing-friendly

Questions?