Contents

Project Overview

- **Environment**: Spark on EMR Cluster
- **Input**: 100G CSV; Output: 5B rows in Redshift, ~10T storage
- **Functionality**: Calculate history & daily inventory and other metrics
- **Challenge:**
  - Calculated metrics take sequential computation each day
  - Calculation based on big-sized Item Level tables
- **Legacy Implementation:**
  - Multiple table join
  - Fulfill and expand raw data as

No. of Location * No. of Item * No. of Date * No. of Metrics

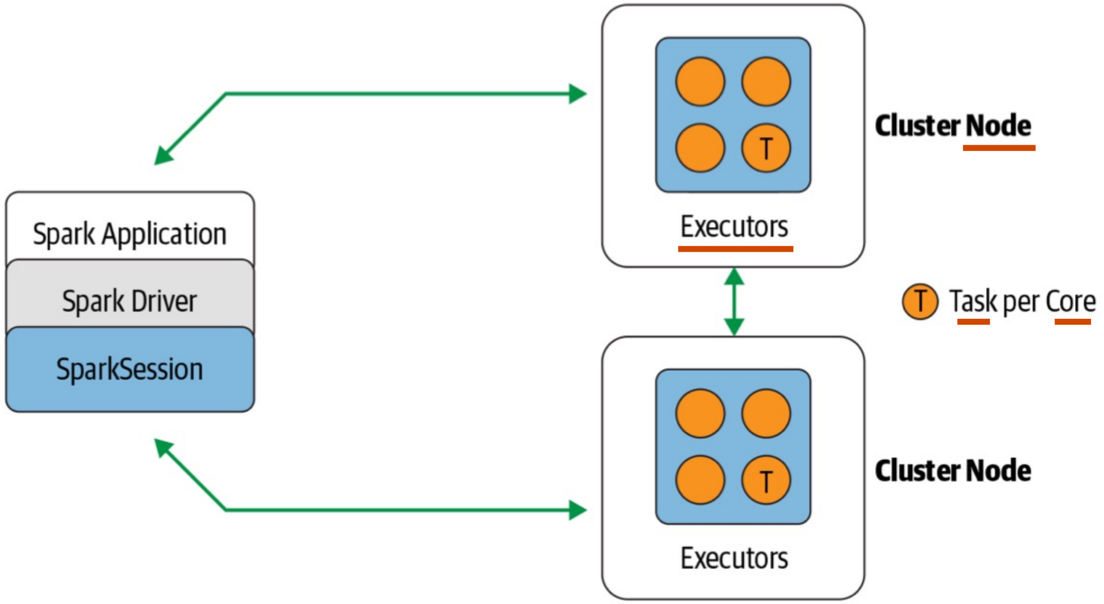
Refactored design

Index	Legacy code	Refactored design	Benefits
data size/shuffle key	>>129416i * 365d * 27m * 200l	129416i * 28d * 27m	>24000x
I/O	500T	10T	50x
exec time	8 hours	6 hours	1.2x
total cost (USD/month)	350k	10k	35x

Legacy code	Refactored design	Benefits
8 EMR Notebooks	Packaged Spark app with CICD and tests	Automation and collaboration
Default Spark properties	Fine-tuned Spark properties	Reduced cost
Excessive IO and checkpoints	End to end script with reusable components	Development velocity
Monolithic CSV as input	Partitioned CSV as input	Parallelism and scalability
Mysterious code, redundant logic	Refactored code and logic	Code efficiency

Spark performance tuning and optimization

Spark properties and Spark application architecture



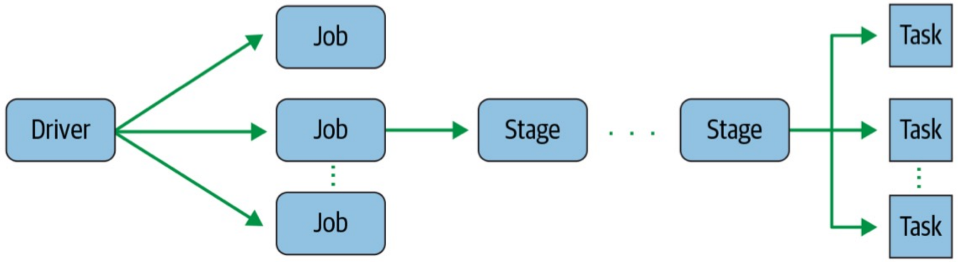
Learning Spark - Lightning-Fast Data Analytics, 2nd Edition, page 50

- *Default Spark properties*
- *EMR default Spark properties*
- *Best practices for* ***successfully managing memory*** *for Apache Spark applications on Amazon EMR*
  - EMR memory calculation helper program

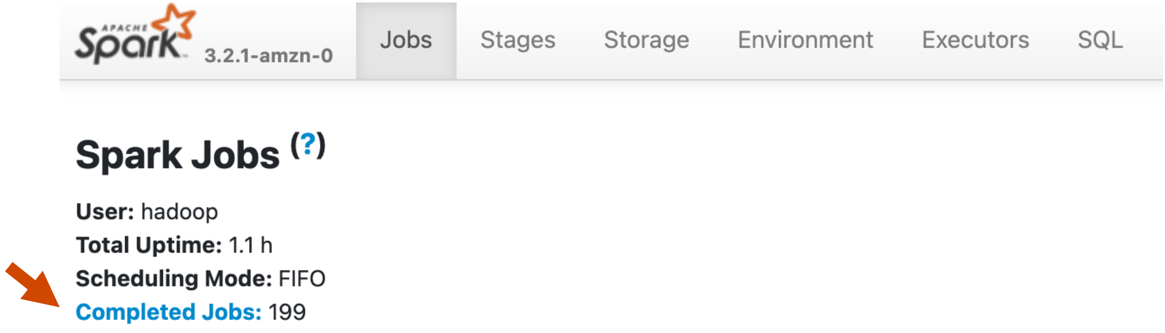
Spark performance tuning and optimization

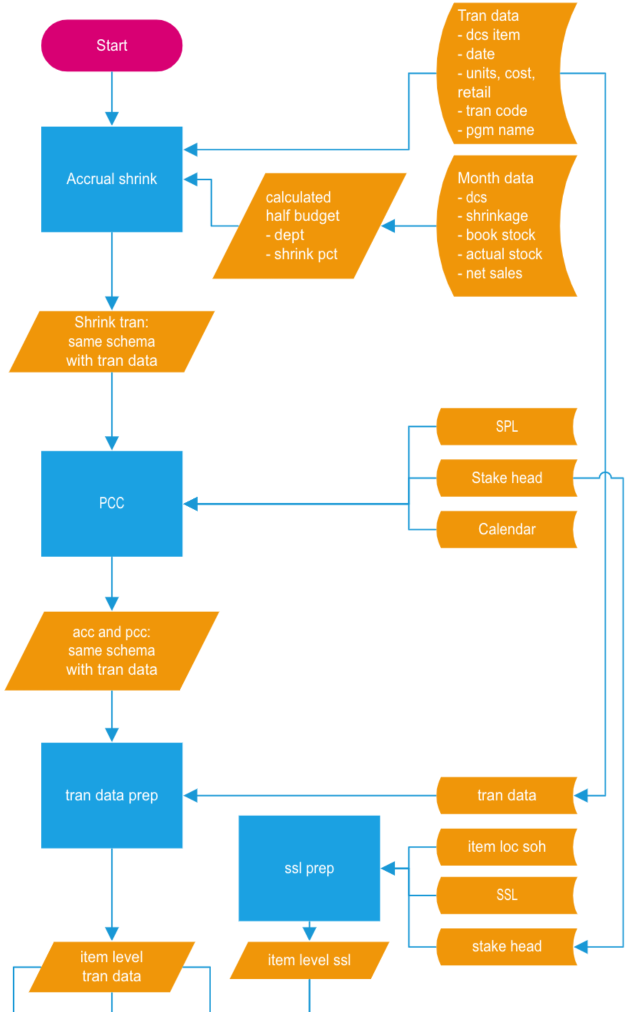
Spark UI: Jobs

- Mind the usage of Spark actions and thus the number of jobs



Learning Spark - Lightning-Fast Data Analytics, 2nd Edition, page 51





Spark performance tuning and optimization

Spark UI: Stage and Task

- Take shuffle action as little as possible. E.g., Broadcast Join
  - `spark.sql.autoBroadcastJoinThreshold`
- Shuffle data as few as possible
  - E.g., The order of joins

- The number of tasks: shouldn't < or >> max tasks in parallel
  - To adjust the number of tasks:
    - Coalesce / Repartition
    - `spark.sql.shuffle.partitions`
    - `spark.sql.adaptive.coalescePartitions.parallelismFirst`
- The running time of tasks: should be balanced
  - To rebalance skewed partitions: repartition (max tasks in parallel)
- GC, Memory spill
  - To manage memory:
    - Adjust partition size
    - `spark.executor.memory`
    - `df.unpersist()`

Spark 3.2.1-amzn-0 | Jobs | Stages | Storage | Environment | Executors | SQL

Details for Stage 511 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 2.5 h
Locality Level Summary: Process local: 200
Shuffle Read Size / Records: 38.2 GiB / 1555965936
Shuffle Write Size / Records: 40.1 GiB / 1458928201
Spill (Memory): 1047.1 GiB
Spill (Disk): 35.5 GiB
Associated Job Ids: 153

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	20 s	42 s	45 s	48 s	1.1 min
GC Time	0.0 ms	0.1 s	0.3 s	0.7 s	2 s
Spill (memory)	0.0 B	5.3 GiB	5.3 GiB	5.3 GiB	6.4 GiB
Spill (disk)	0.0 B	184.3 MiB	184.4 MiB	184.5 MiB	325.3 MiB
Shuffle Read Size / Records	195.3 MiB / 7772955	195.5 MiB / 7777916	195.5 MiB / 7779894	195.6 MiB / 7781707	195.7 MiB / 7787165
Shuffle Write Size / Records	205 MiB / 7289074	205.1 MiB / 7292929	205.1 MiB / 7294761	205.2 MiB / 7296306	225.9 MiB / 7300972

Showing 1 to 6 of 6 entries

- **Aggregated Metrics by Executor**

Tasks (200)

Show 20 entries Search:

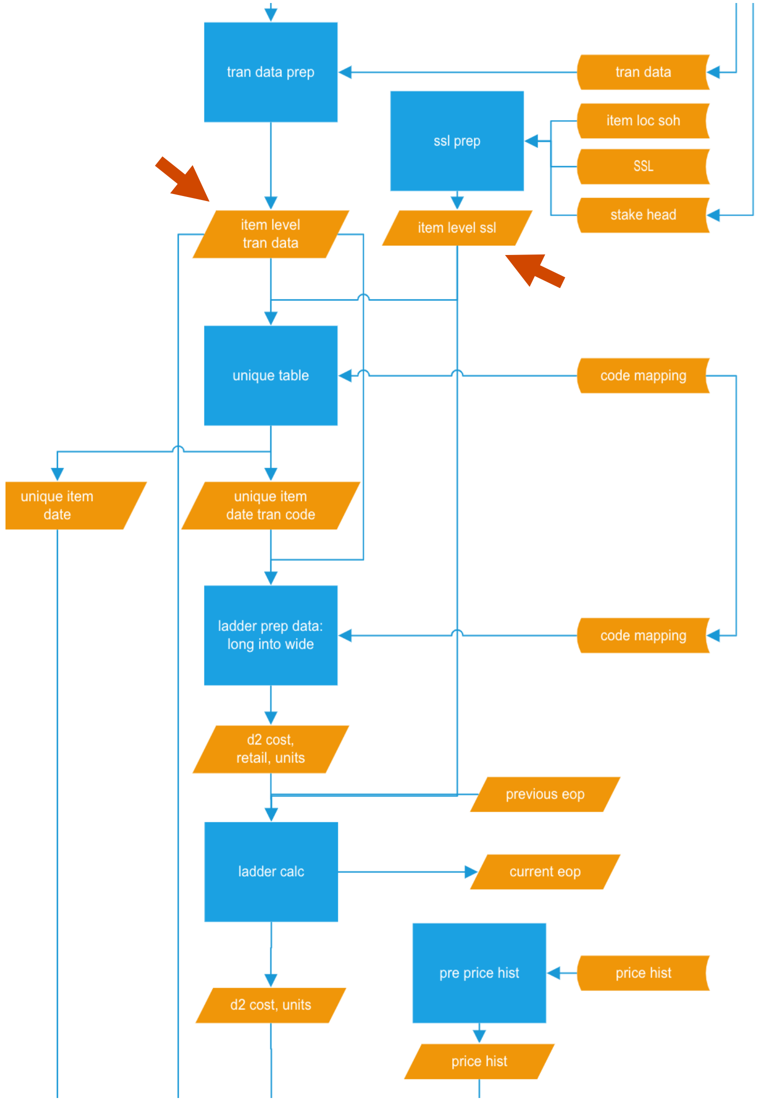
Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Shuffle Write Size / Records	Shuffle Read Size / Records	Spill (Memory)	Spill (Disk)	Errors
0	21662	0	SUCCESS	PROCESS_LOCAL	2	ip-172-24-35-234.us-west-2.compute.internal	stderr stdout	2023-03-02 13:35:31	51 s	0.9 s	205.1 MiB / 7293944	195.5 MiB / 7779020	5.3 GiB	184.4 MiB	
1	21663	0	SUCCESS	PROCESS_LOCAL	6	ip-172-24-35-234.us-	stderr	2023-03-	47 s	1 s	205.1 MiB /	195.5 MiB /	5.3 GiB	184.4	

Spark performance tuning and optimization

Code-level design choices

- I/O Parallelization
  - Use parquet. Partition size: ~ tens of MB
  - EMR cluster IP = Number of nodes + 1
- Filter as early as possible
- Use API; avoid UDF
- Cache

```
joined_df = df1.join(df2, 'join_key', 'left')
# joined_df.cache()
# joined_df.count()
new_df_1 = joined_df.withColumn('new_column_1', F.lit(1))
new_df_2 = joined_df.withColumn('new_column_2', F.lit(2))
new_df_1.count(), new_df_2.count()
```



Optimization checklist

- Designing stage
  - Partition IO; adopt parquet if possible
  - Streamline code logic: transformations and actions
  - Calculate Spark properties based on estimated workload
- Building stage
  - Filter as early as possible
  - Shuffle as few as possible
  - Use API; avoid UDF
  - Cache if needed
- Testing stage
  - Fine-tune Spark properties
  - The number of tasks: shouldn't < or >> max tasks in parallel
  - The size of partitions: ~ tens of MB output partition
  - The running time of tasks: should be balanced
  - Look for GC or memory spill
- Other optimization techniques such as AQE, Bucketing, Checkpoint

References

Spark - The Definitive Guide, Chapter 19: Performance Tuning

Learning Spark - Lightning-Fast Data Analytics, Chapter 7: Optimizing and Tuning Spark Applications