Experience of optimizing SparkSQL when migrating from MPP DBMS

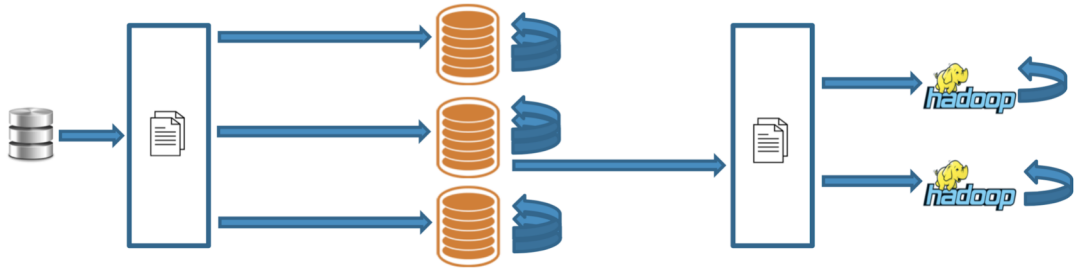
Yucai Yu, Yuming Wang, Chenxiao Mao
eBay, Data Services and Solutions

#SAISExp11

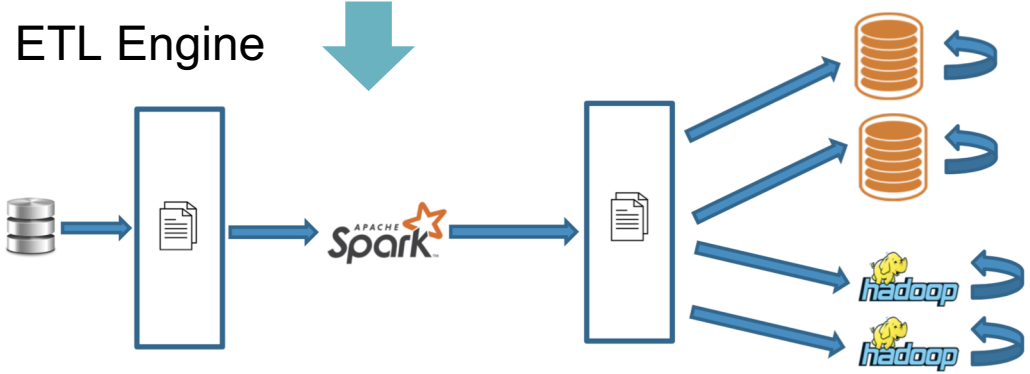
About Us

- Spark / Hadoop Contributors
- eBay Data Services and Solutions team
- Responsible for enterprise data warehouse / data lake development and optimization

Migrating from MPP DBMS



Spark as Core ETL Engine

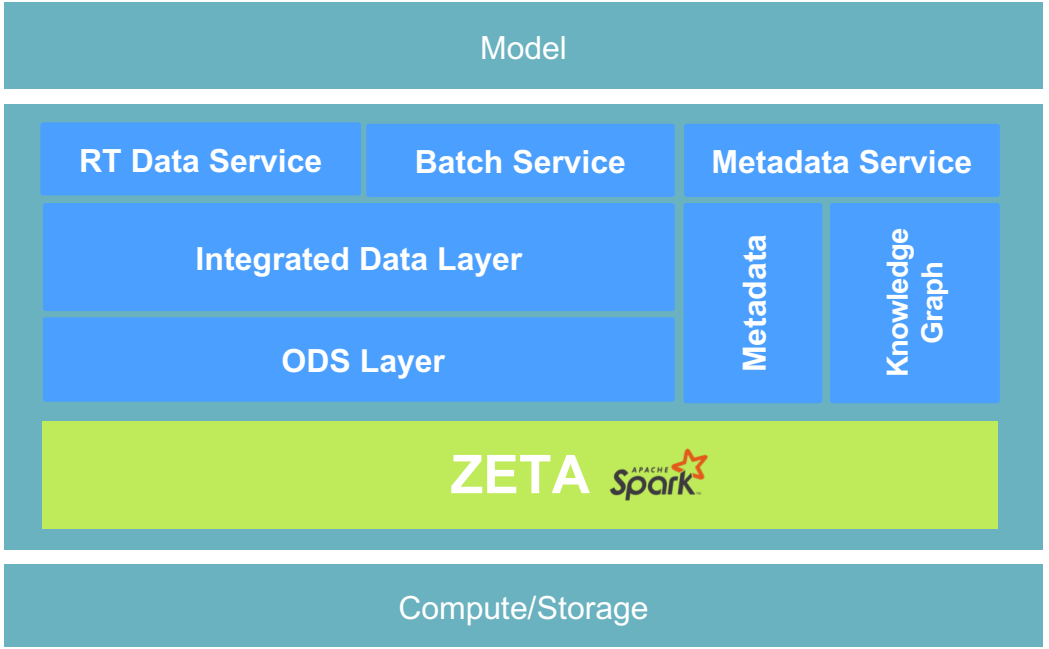


Spark as DW Processing Engine

DS
(Data Science)

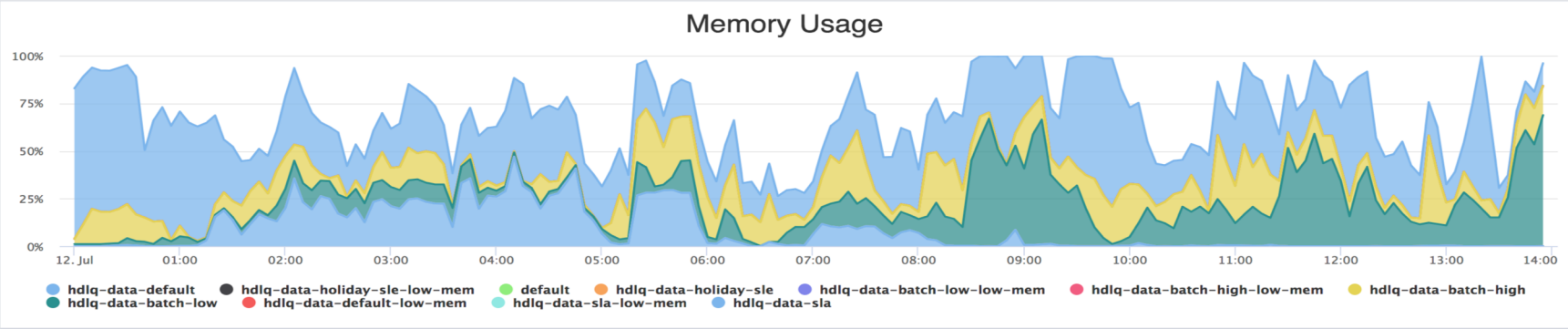
DW
(Data Warehouse)

DI
(Data Infrastructure)



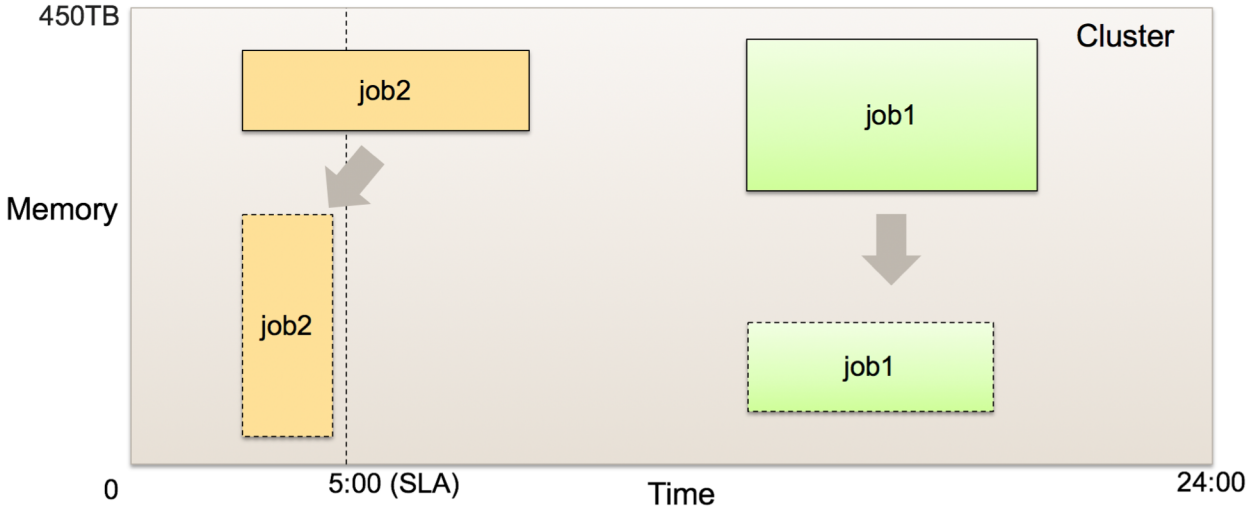
Memory Capacity Challenge

- 5K Target tables
- 20K Intermediate/Working tables
- 30PB Compressed data
- 60PB Relational data processing every day



Optimization Objectives

- Increase overall throughput: MB-Seconds (Memory x Duration)
- Reduce response time for critical jobs: Duration



Spark Core Optimization

Adaptive Execution

- Dynamically optimize execution plan

Indexed Bucket

- Optimize data layout

Adaptive Execution Background

- Initial by Databricks since 2015 @SPARK-9850
- At 2017, new design & implementation @SPARK-23128
- Main contributors from Intel and eBay.
  - Carson Wang
  - Yucai Yu
  - Chenzhao Guo
  - Cheng Hao
  - Yuanjian Li
  - Yuming Wang
  - JkSelf
  - Windpiger
  - Chenxiao Mao

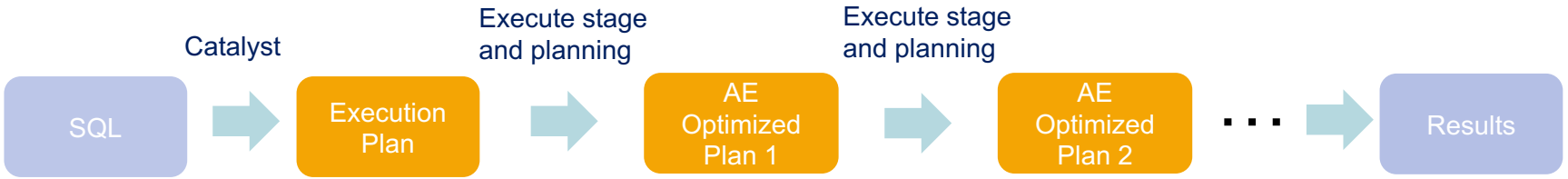
What's Adaptive Execution?

Original Spark

Even with CBO, hard to get optimal execution plans

Spark With Adaptive Execution

Oracle has already supported AE in their latest products



Major Cases

- Simplify and Optimize configuration
- Optimize join strategy
- Handle skewed join

Dive Into Top Memory Intensive Queries

Improper user configuration actually lead to many memory issue

- Migration timeline is very tight
- Tuning is very time consuming

Query 1: shuffle.partition is too big

- shuffle.partition = 5000, each task processes only 20MB data.
- Many schedule overhead, IO overhead etc.

Tasks: Succeeded/Total	Shuffle Read
5000/5000	170.3 GB
5000/5000	113.5 GB
5000/5000	85.6 GB
5000/5000	84.9 GB

113.5 GB / 5000 = 23.2 MB

85.6 GB / 5000 = 17.5 MB

Query 2: shuffle.partition is too small

- shuffle.partition = 600
- Lots of GC overhead, the tasks run slow

Details for Stage 2 (Attempt 0)

Total Time Across All Tasks: 70.2 h
Locality Level Summary: Process local: 600
Shuffle Read: 820.0 GB / 63726663959

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 600 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	5.9 min	6.6 min	7.0 min	7.4 min	8.4 min
Scheduler Delay	70 ms	83 ms	94 ms	0.1 s	0.2 s
Task Deserialization Time	97 ms	0.2 s	0.3 s	0.3 s	0.7 s
GC Time	25 s	38 s	43 s	48 s	59 s
Result Serialization Time	0 ms	1 ms	1 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	1 ms	2 ms	0.6 s
Shuffle Read Size / Records	1399.0 MB / 106186128	1399.4 MB / 106205910	1399.5 MB / 106210924	1399.6 MB / 106216378	1399.9 MB / 106234720
Shuffle Remote Reads	1218.0 MB	1399.3 MB	1399.4 MB	1399.5 MB	1399.8 MB

Duration: 7.0 min

GC: 43s

Query 2: memory per core is too big

- CPU usage is low: each core has 20GB memory

spark.executor.memory	40g
spark.executor.cores	2
spark.sql.shuffle.partitions	600

40 GB / 2 = 20 GB

Details for Stage 2 (Attempt 0)

Total Time Across All Tasks: 70.2 h
Locality Level Summary: Process local: 600
Shuffle Read: 820.0 GB / 63726663959

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 600 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	5.9 min	6.6 min	7.0 min	7.4 min	8.4 min
Scheduler Delay	70 ms	83 ms	94 ms	0.1 s	0.5 s
Task Deserialization Time	97 ms	0.2 s	0.3 s	0.3 s	0.7 s
GC Time	25 s	38 s	43 s	48 s	59 s
Result Serialization Time	0 ms	1 ms	1 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	1 ms	2 ms	0.6 s
Shuffle Read Size / Records	1399.0 MB / 106186128	1399.4 MB / 106205910	1399.5 MB / 106210924	1399.6 MB / 106216378	1399.9 MB / 106234720
Shuffle Remote Reads	1218.0 MB	1399.3 MB	1399.4 MB	1399.5 MB	1399.8 MB

Cluster-wide Configuration

- Best memory per core is 5GB

❑ eBay has separate storage nodes and computer nodes

❑ Each computer node has 384GB memory and 64 cores

❑ CPU is the most expensive, max the CPU utilization: 384 / 64 = 5GB

Cluster-wide Configuration

- Roughly estimated data size per core (shuffle stage) is 100MB

❑ Spark UMM 5 GB * 60% = 3GB = 3072 MB

❑ Shuffle data's compression rate is 5x – 30x, suppose 15x

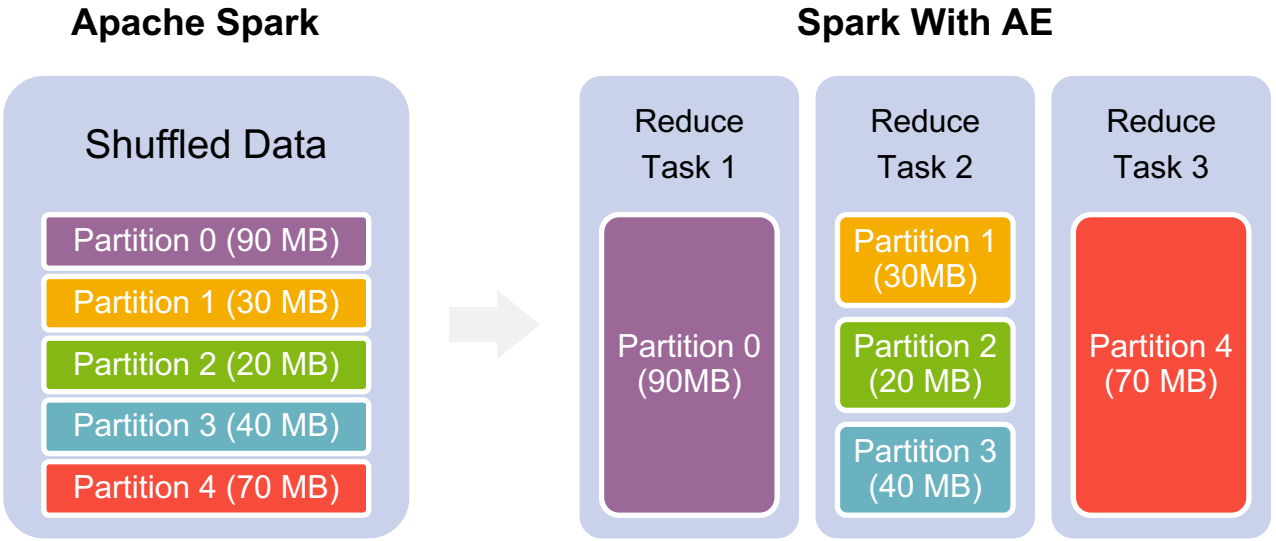
❑ Temporary space for algorithm (e.g. radix sort), suppose 2x

❑ **5GB * 60% / 15 / 2 ~ 102.4 MB**: less spill, GC etc.

Configuration with AE

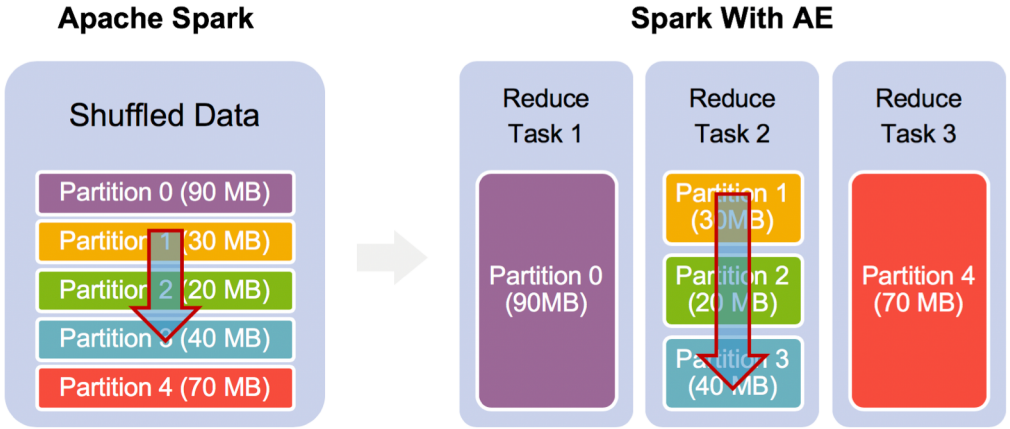
SPARK-23128

- In shuffle write: use big shuffle partition number to split data into small blocks.
- In shuffle read: AE packs small blocks into **100MB**.



But…No Free Launch !

- SPARK-9853: Optimize shuffle fetch of contiguous partition IDs **1.3x**
- SPARK-22537: Aggregation of map output statistics on driver faces single point bottleneck, **5s** delay when shuffle.partition > 10000



Unified Configuration with AE

- spark.executor.memory=20GB
- spark.executor.cores=4 // memory per core 5GB
- spark.sql.adaptive.maxNumPostShufflePartitions=10000
- spark.sql.adaptive.shuffle.targetPostShuffleInputSize=100MB

less spill, less GC, less schedule overhead

less human configuration

Query 1: 1.7x MB-Seconds improvement

Apache Spark

MB-Seconds: 7,661,564,130

Tasks: Succeeded/Total
5000/5000
5000/5000
5000/5000
5000/5000

5000

Spark With AE

MB-Seconds: 4,540,665,342

Tasks: Succeeded/Total	Shuffle Read
2500/2500	170.3 GB
2500/2500	113.5 GB
1112/1112	85.6 GB
1250/1250	84.9 GB

2500

1112

Query 2: 6x MB-Seconds improvement

- Enable AE: no GC issue, **1.4x** MB-Seconds improvement
- Less memory in new configuration solution

Details for Stage 4 (Attempt 0)

Total Time Across All Tasks: 50.8 h
Locality Level Summary: Process local: 10000
Shuffle Read: 1011.6 GB / 63312287767

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 10000 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	15 s	17 s	17 s	19 s	33 s
Scheduler Delay	1 ms	3 ms	4 ms	5 ms	0.5 s
Task Deserialization Time	6 ms	8 ms	9 ms	11 ms	0.5 s
GC Time	17 ms	0.1 s	0.2 s	0.2 s	0.7 s
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	44 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	1 ms	31 ms	1.0 s
Shuffle Read Size / Records	103.5 MB / 6323691	103.6 MB / 6329881	103.6 MB / 6331224	103.6 MB / 6332565	103.7 MB / 6337742
Shuffle Remote Reads	90.3 MB	103.5 MB	103.6 MB	103.6 MB	103.7 MB

Duration: 17s

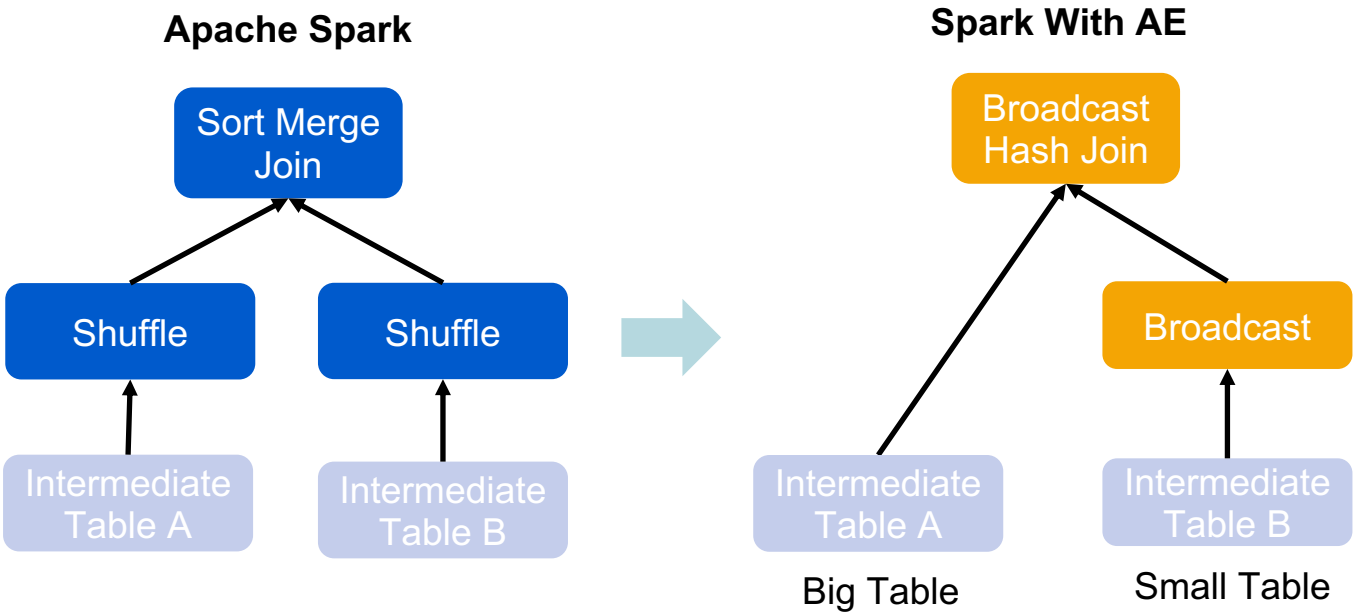
GC: 0.2s

Major Cases

- Simplify and Optimize configuration
- Optimize join strategy
- Handle skewed join

Optimize Join Strategy – Overview

- Based on runtime intermediate table size, AE builds best join plan.



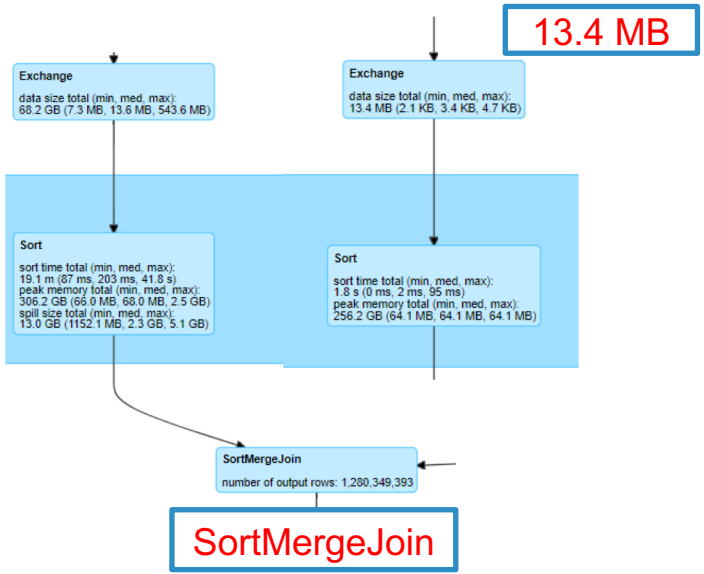
Optimize Join Strategy – How it works

- Enable stats estimation for physical plan
- Allow shuffle readers to request data from just one mapper
- Optimize SortMergeJoin to BroadcastHashJoin at runtime

Optimize Join Strategy – Case

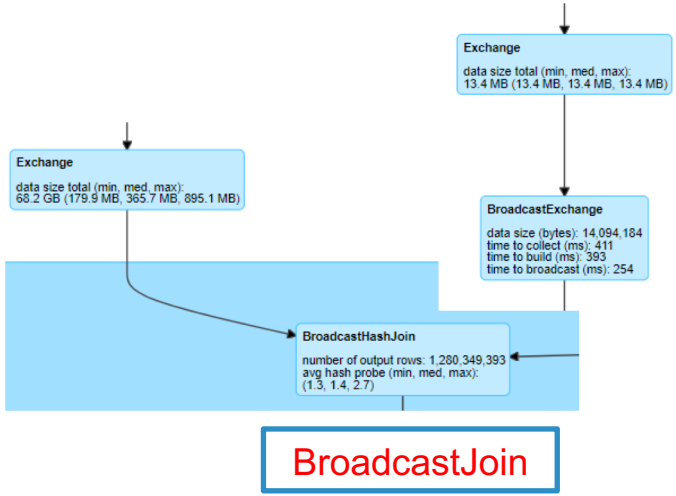
Apache Spark

MB-Seconds: 75,203,212,678



Spark With AE

MB-Seconds: 17,926,197,353



Optimize Join Strategy – Case

4.2x MB-Seconds improvement

Apache Spark

4.7 min	4096/4096			10.3 GB	11.3 GB
7.4 min	4096/4096			10.3 GB	11.6 GB
8.3 min	4096/4096			10.7 GB	11.8 GB

Spark With AE

22 s	162/162			9.4 GB	11.3 GB
19 s	176/176			9.4 GB	11.4 GB
21 s	176/176			9.7 GB	11.5 GB

Major Cases

- Simplify and Optimize shuffle partitions
- Optimize join strategy
- Handle skewed join

Handle Skewed Join – Challenge

- Common issue, some partitions data are extremely larger than others.

Summary Metrics for 5600 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	15 ms	2 s	3 s	5 s	4.0 min
Scheduler Delay	0 ms	2 ms	2 ms	3 ms	1 s
Task Deserialization Time	3 ms	5 ms	6 ms	7 ms	0.4 s
GC Time	0 ms	0 ms	0.2 s	0.3 s	35 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	3 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	44 ms	0.2 s	0.6 s	31 s
Shuffle Read Size / Records	0.0 B / 0	3.5 MB / 374415	4.9 MB / 747052	6.9 MB / 1783859	172.8 MB / 283732661
Shuffle Remote Reads	0.0 B	3.4 MB	4.7 MB	6.6 MB	166.6 MB
Shuffle Write Size / Records	0.0 B / 0	109.0 B / 3	132.0 B / 4	158.0 B / 6	262.0 B / 13
Shuffle spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	13.3 GB
Shuffle spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	68.2 MB

Avg: 3 s

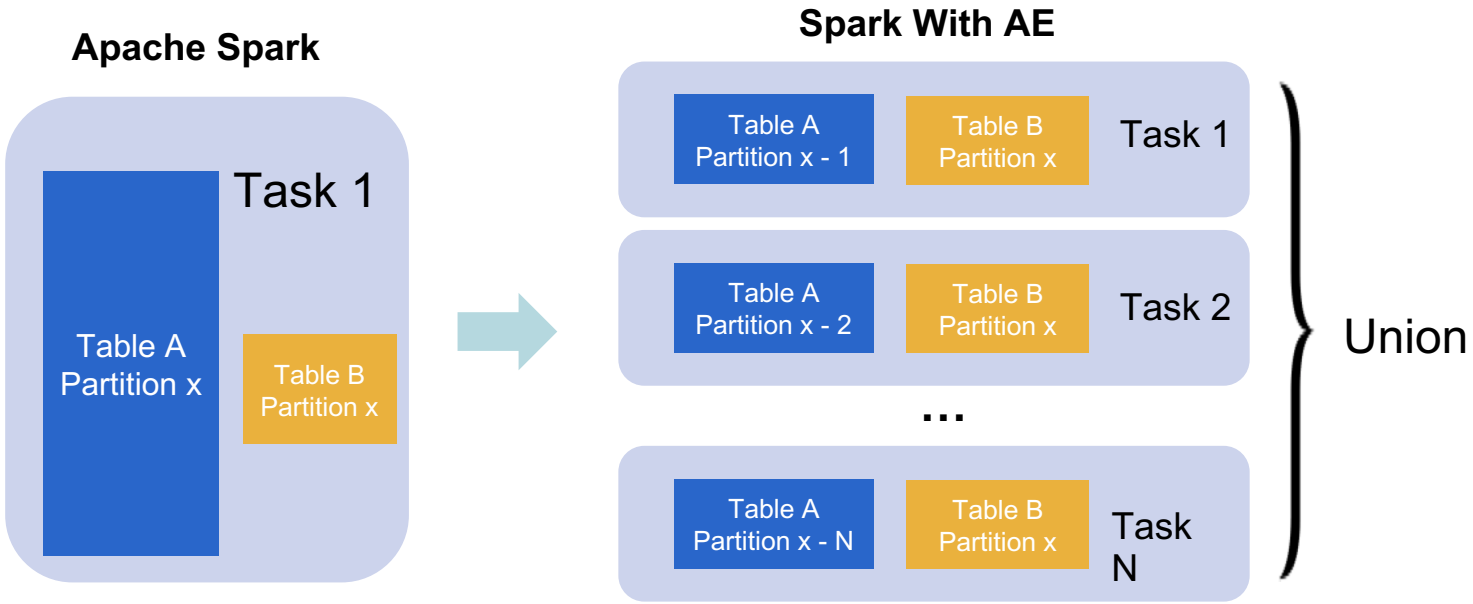
Max Time: 4.0 min

Avg: 5 MB

Max Size: 172 MB

Handle Skewed Join – How it works

- AE dynamically detect the skewed task, and increases parallelism.



Handle Skewed Join – Case

1.6x run time reduced

Summary Metrics for 3148 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	32 ms	31 s	45 s	1.1 min	35 min
GC Time	0 ms	0.1 s	0.3 s	2 s	1.9 min
Shuffle Read Size / Records	0.0 B / 0	21.4 MB / 15105344	27.6 MB / 23656956	30.1 MB / 76802640	1024.1 MB / 2762311594
Shuffle Write Size / Records	0.0 B / 0	146.0 B / 4	369.0 B / 8	630.0 B / 13	1420.0 B / 26
Shuffle spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	130.3 GB
Shuffle spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	908.7 MB

27MB

1024MB

Apache Spark

35 min

2017/11/13 23:53:32	35 min	3148/3148			112.7 GB	1299.2 KB

Spark with AE

22 min

2017/11/14 22:33:59	22 min	4317/4317			140.3 GB	1312.8 KB

Bucket Support in AE

- Bucket table read support: avoid shuffle for the bucketed table when shuffle.partition > bucket number

- Bucket table write support: AE should respect bucket number when writing to the bucket table

Production Benchmark

In 40 production queries:

- 3.09x average MB-Seconds improvement
- 61% improve +200%
- 83% improve +50%

SQL	AE (MB-seconds)	Production (MB-seconds)	Production/AE
b_zeta_spark/129	221871376	6823561697	30.7546
b_tns/001	1571054013	43623855787	27.7673
b_zeta_spark/142	1734111586	18927912838	10.9150
b_zeta_spark/145	1699741075	15611515463	9.1846
b_ctlg/310	2867233423	25477859567	8.8859
b_lstg/007	1324843534	9662286042	7.2932
b_zeta_spark/147	2888996062	18581014216	6.4317
b_zeta_spark/120	6334618240	37592253460	5.9344
b_vertical/001	21537803	115318465	5.3542
b_attr/005	2970198864	15433374244	5.1961
b_zeta_spark/130	4809816657	22544640826	4.6872
b_tns/003	670181076	3103304323	4.6305
b_ctlg/312	5518098418	18581014216	3.3673
b_tns/021	41850727	137391370	3.2829
b_ctlg/318	2849932867	9152171383	3.2114
b_attr/021	2085364872	6604722482	3.1672
b_attr/020	1576877191	4522695497	2.8681
b_lstg/005	3804430972	10638915453	2.7965

Spark Core Optimization

Adaptive Execution

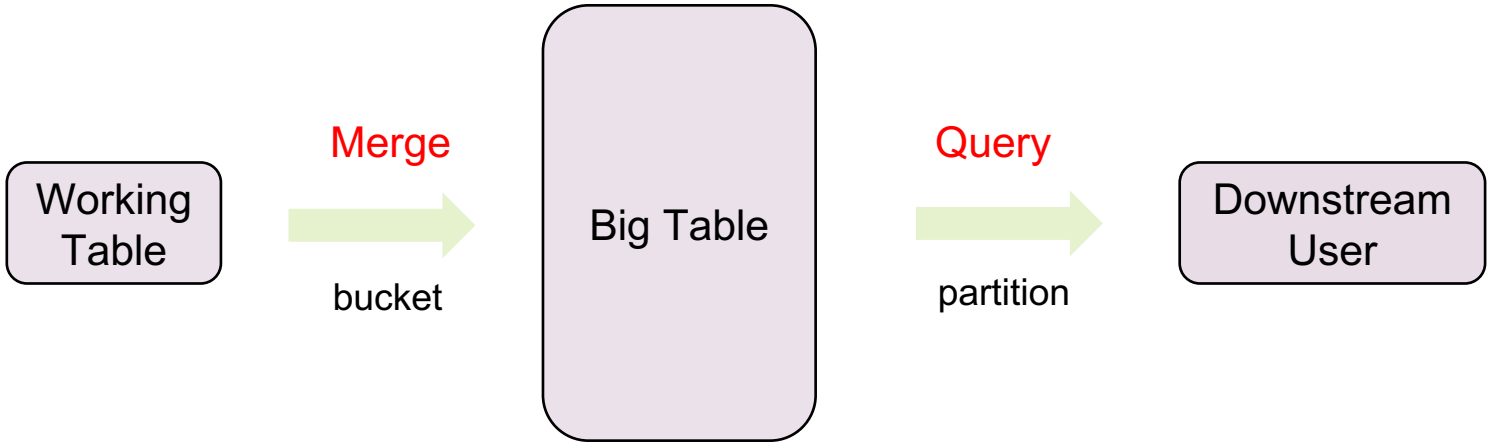
- Dynamically optimize execution plan

Indexed Bucket

- Optimize data layout

Why Indexed Bucket

Challenge: how to optimize query and merge at the same time

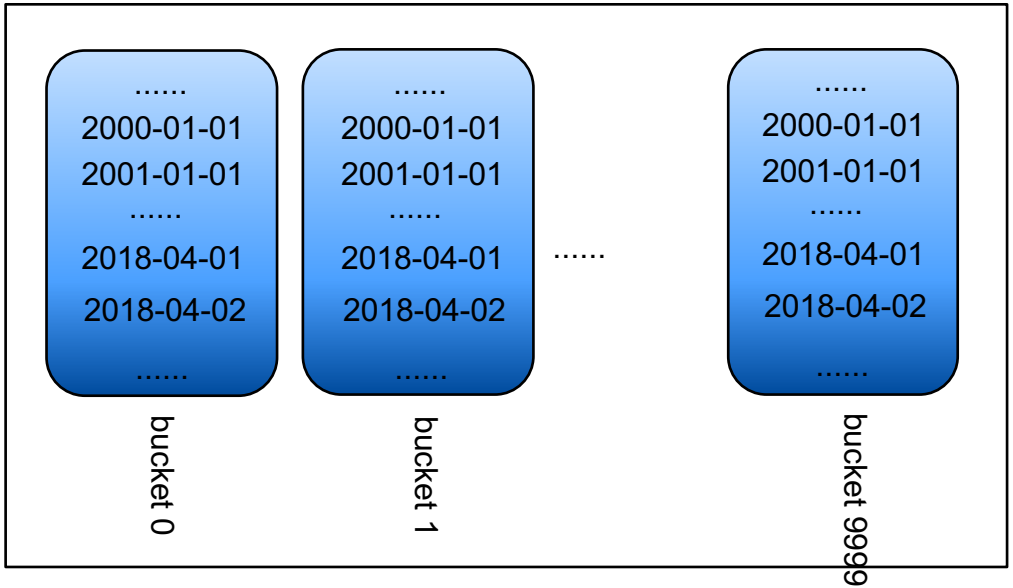


Bucket + Partition: too many small files

Indexed Bucket – How it works

```
CREATE TABLE dw_lstg_item (
    item_id DECIMAL(18,0),
    auct_end_dt DATE,
    ...
)
USING parquet
CLUSTERED BY (item_id)
SORTED BY (auct_end_dt)
INTO 10000 BUCKETS;

INSERT INTO TABLE dw_lstg_item
SELECT * FROM xxx
DISTRIBBUTE BY item_id;
```



- bucket by item_id + sort by auct_end_dt (reduces files number)
- data is sorted with parquet's min-max index for filtering performance

Query Improvements – Predicate Pushdown

[SPARK-25419] Improvement parquet predicate pushdown

- [SPARK-23727] Support Date type
- [SPARK-24549] Support Decimal type
- [SPARK-24718] Support Timestamp type
- [SPARK-24706] Support Byte type and Short type
- [SPARK-24638] Support StringStartsWith predicate
- [SPARK-17091] Support IN predicate

Query Improvements – Predicate Pushdown

Bucket table

Total Time Across All Tasks: 262.4 h
Locality Level Summary: Any: 1705; Node local: 7481;
Input Size / Records: 19.5 TB / 61595886872
Output: 17.6 GB / 15542438

Indexed bucket table

Total Time Across All Tasks: 10.6 h
Locality Level Summary: Any: 27; Node local: 9117;
Input Size / Records: 579.8 GB / 1735868108
Output: 17.6 GB / 15542438

```
SELECT
   lstg.item_site_id,
   ...
FROM
   dw_lstg_item lstg
WHERE
   lstg.auct_end_dt =  CAST('2018-04-19') AS DATE);
```

24.7x Improvement

Query Improvements – Enhance Bucket Join

```
SELECT
  lstg.item_site_id,
  h_categ.meta_categ_id,
  lstg.auct_end_dt,
  v.item_vrtn_id,
  v.price_amt,
  v.qty,
  v.sold_qty,
  v.dlt_sts_vn_ind,
  v.item_id
FROM
  dw_lstg_item lstg,
  dw_category_hierarchy h_categ
  lstg_item_vrtn v
WHERE
  lstg.auct_end_dt =  CAST(SUBSTR('2018-04-19 00:00:00',1,10) AS DATE)
  AND lstg.item_id = v.item_id
  AND lstg.auct_end_dt = v.auct_end_dt
  AND lstg.categ_key = h_categ.categ_key
  AND lstg.item_site_id IN (SELECT site_id FROM dw_dlp_site_lkp)
ORDER BY 1,2,3;
```

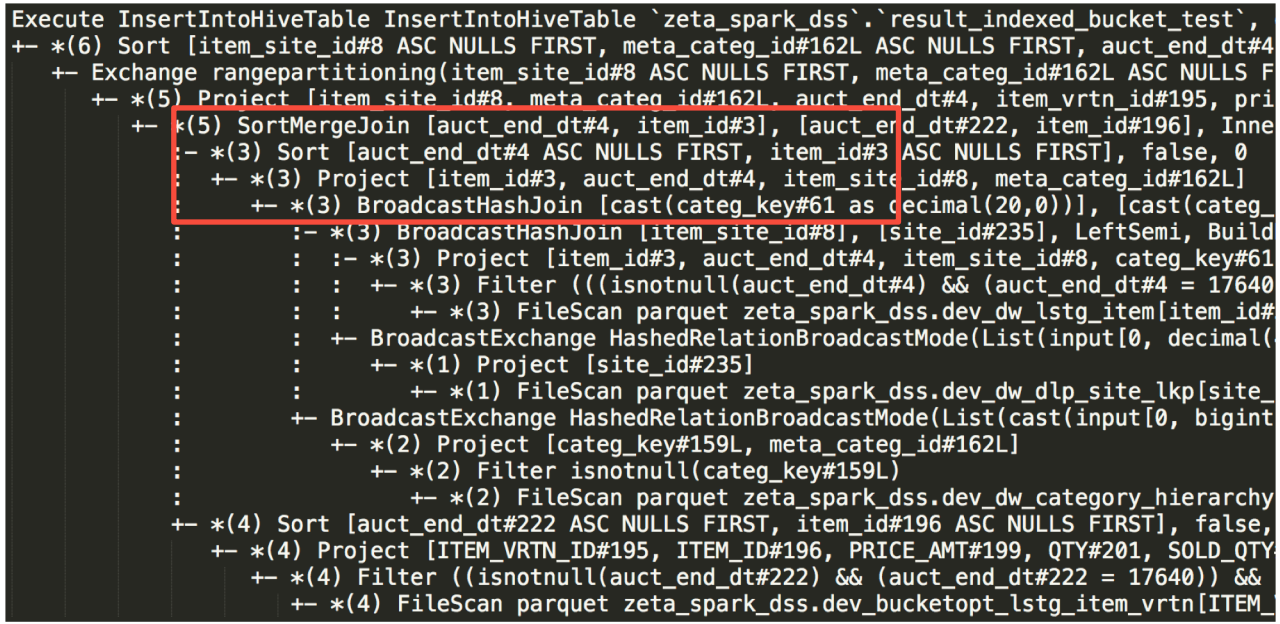
Query Improvements – Enhance Bucket Join

Apache Spark, unnecessary shuffle

```
Execute InsertIntoHiveTable InsertIntoHiveTable `zeta_spark_dss`.`result_indexed_bucket_test`,
+- *(8) Sort [item_site_id#8 ASC NULLS FIRST, meta_categ_id#162L ASC NULLS FIRST, auct_end_dt#4
   +- Exchange rangepartitioning(item_site_id#8 ASC NULLS FIRST, meta_categ_id#162L ASC NULLS F
      +- *(7) Project [item_site_id#8, meta_categ_id#162L, auct_end_dt#4, item_vrtn_id#195, pri
         +- *(7) SortMergeJoin [auct_end_dt#4, item_id#3], [auct_end_dt#222, item_id#196], Inne
            :- *(4) Sort [auct_end_dt#4 ASC NULLS FIRST, item_id#3 ASC NULLS FIRST], false, 0
            :  +- Exchange hashpartitioning(auct_end_dt#4, item_id#3, 10000)
            :     +- *(3) Project [item_id#3, auct_end_dt#4, item_site_id#8, meta_categ_id#162L
            :        +- *(3) BroadcastHashJoin [cast(categ_key#61 as decimal(20,0))], [cast(cat
            :           :- *(3) BroadcastHashJoin [item_site_id#8], [site_id#235], LeftSemi, Bu
            :           :  :- *(3) Project [item_id#3, auct_end_dt#4, item_site_id#8, categ_key
            :           :  :  +- *(3) Filter (((isnotnull(auct_end_dt#4) && (auct_end_dt#4 = 17
            :           :  :     +- *(3) FileScan parquet zeta_spark_dss.dev_dw_lstg_item[item_
            :           :  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, decim
            :           :     +- *(1) Project [site_id#235]
            :           :        +- *(1) FileScan parquet zeta_spark_dss.dev_dw_dlp_site_lkp[si
            :           +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, big
            :              +- *(2) Project [categ_key#159L, meta_categ_id#162L]
            :                 +- *(2) Filter isnotnull(categ_key#159L)
            :                    +- *(2) FileScan parquet zeta_spark_dss.dev_dw_category_hierar
            +- *(6) Sort [auct_end_dt#222 ASC NULLS FIRST, item_id#196 ASC NULLS FIRST], false,
               +- Exchange hashpartitioning(auct_end_dt#222, item_id#196, 10000)
                  +- *(5) Project [ITEM_VRTN_ID#195, ITEM_ID#196, PRICE_AMT#199, QTY#201, SOLD_
                     +- *(5) Filter ((isnotnull(auct_end_dt#222) && (auct_end_dt#222 = 17640))
                        +- *(5) FileScan parquet zeta_spark_dss.dev_bucketopt_lstg_item_vrtn[IT
```

Query Improvements – Enhance Bucket Join

Spark With IB

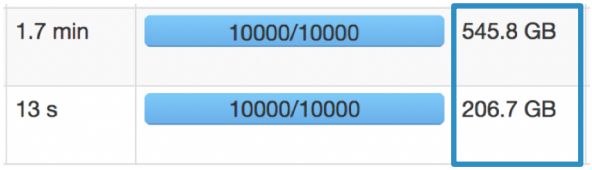


[SPARK-24087] Avoid shuffle when join keys are a super-set of bucket keys

Query Improvements – Comprehensive Case

Apache Spark

2,892,165,122 MB-Seconds



Spark With IB

944,326,656 MB-Seconds

3.06x MB-Seconds improvement

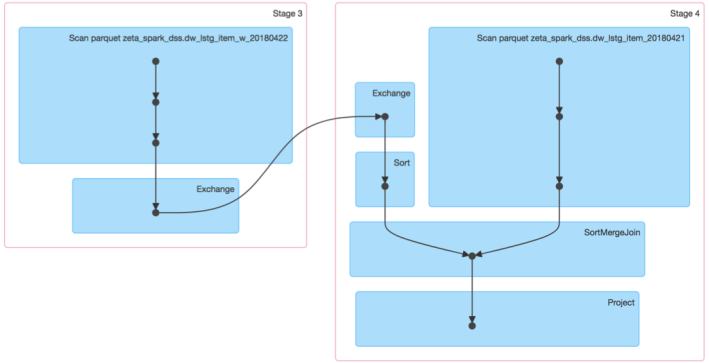
```
SELECT
   lstg.item_site_id,
   ...
FROM
   dw_lstg_item lstg,
   dw_category_hierarchy h_categ
   lstg_item_vrtn v
WHERE
   lstg.auct_end_dt =  CAST(SUBSTR('2018-04-19 00:00:00',1,10) AS DATE)
   AND lstg.item_id = v.item_id
   AND lstg.auct_end_dt = v.auct_end_dt
   AND lstg.categ_key = h_categ.categ_key
   AND lstg.item_site_id IN (SELECT site_id FROM dw_dlp_site_lkp)
ORDER BY 1,2,3;
```

Merge Performance Evaluation

```
INSERT OVERWRITE TABLE dw_lstg_item_merge partition (dt = '${UOW_TO_DATE}')
SELECT
    case when w.item_id is not null then w.item_id else t.item_id end,
    case when w.item_id is not null then coalesce(w.auct_end_dt,current_date) else
t.auct_end_dt end
    ......
FROM
dw_lstg_item t
FULL OUTER JOIN dw_lstg_item_w w ON w.item_id = t.item_id
```

Merge Performance Evaluation

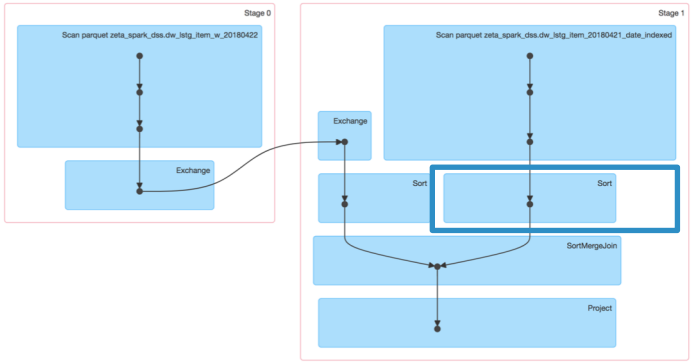
Apache Spark



Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2018/04/25 08:19:18	50 min	10000/10000 (13 failed) (1	16.7 TB		650.4 GB	

Total Time Across All Tasks: 2381.6 h

Spark With IB



Additional Sort

Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2018/04/25 07:25:28	50 min	10000/10000 (11 failed)	16.8 TB		650.2 GB	

Total Time Across All Tasks: 2356.7 h

Duration & Total Task Time:
no significant degradation

* 1000 executors, 5 cores per executor, 5000 tasks run in parallel

Indexed Bucket Summary

- Brings significant benefits
  - Reduce execution time
  - Improve memory efficiency
- Especially useful for the downstream to avoid full table scan on huge and hot tables.

More Than AE/IB …

Make optimization production ready

- Integration Test Suite: TPCDS 100TB, eBay production queries
- Rolling Upgrade
- Auto Data Quality Check

Important Improvement

Case-insensitive field resolution

SPARK-25132 Case-insensitive field resolution when reading from Parquet
SPARK-25175 Field resolution should fail if there's ambiguity for ORC native reader
SPARK-25207 Case-insensitive field resolution for filter pushdown when reading Parquet

Critical Issue

SPARK-24076 very bad performance when shuffle.partition = 8192
SPARK-25084 "distribute by" on multiple columns may lead to codegen issue
SPARK-25368 Incorrect constraint inference returns wrong result

Thanks

#SAISExp11