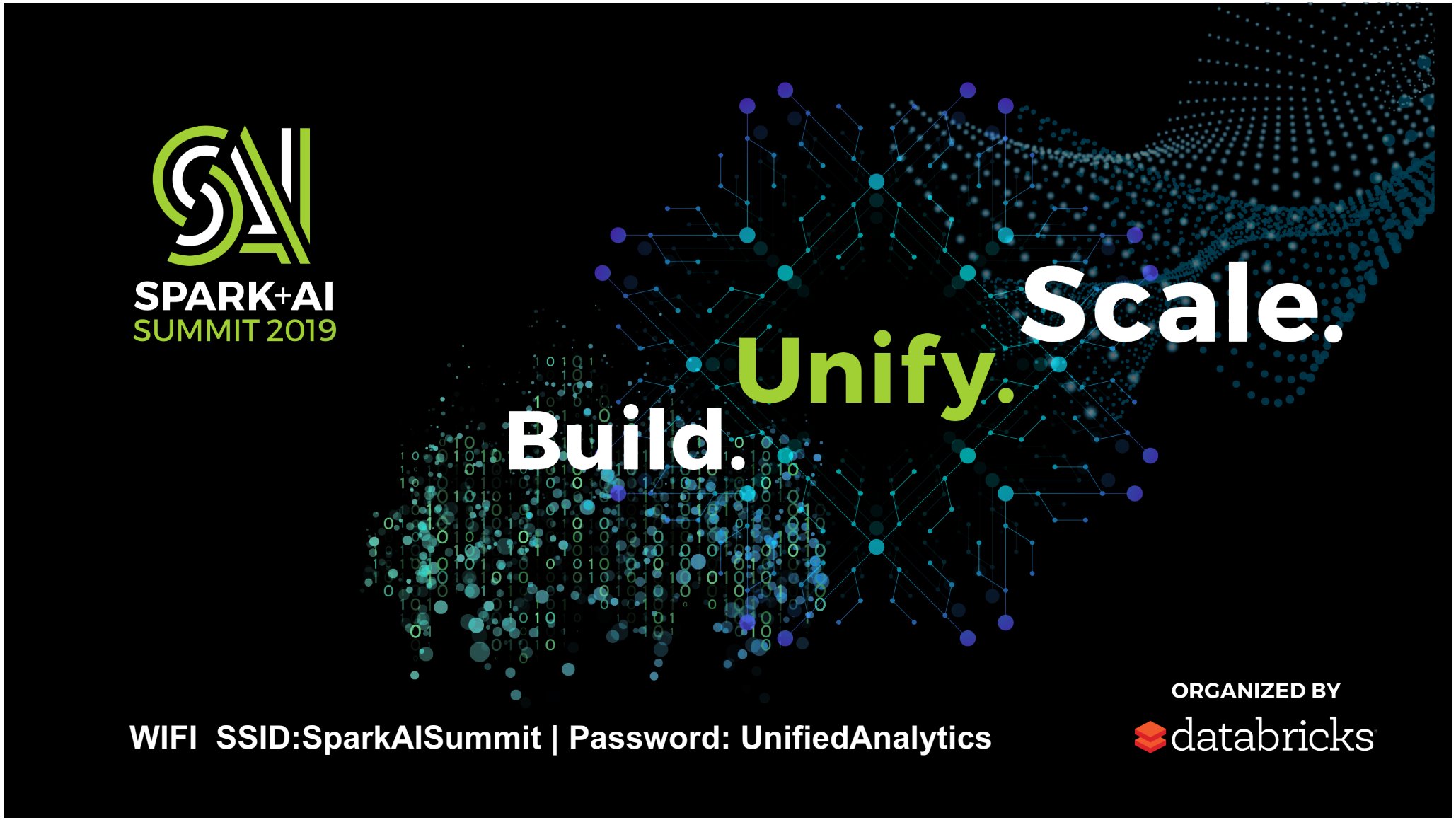

SPARK+AI
SUMMIT 2019
Build.
Unify.
Scale.
WIFI SSID:SparkAISummit | Password: UnifiedAnalytics
ORGANIZED BY
databricks

Spark Core – Proper Optimization

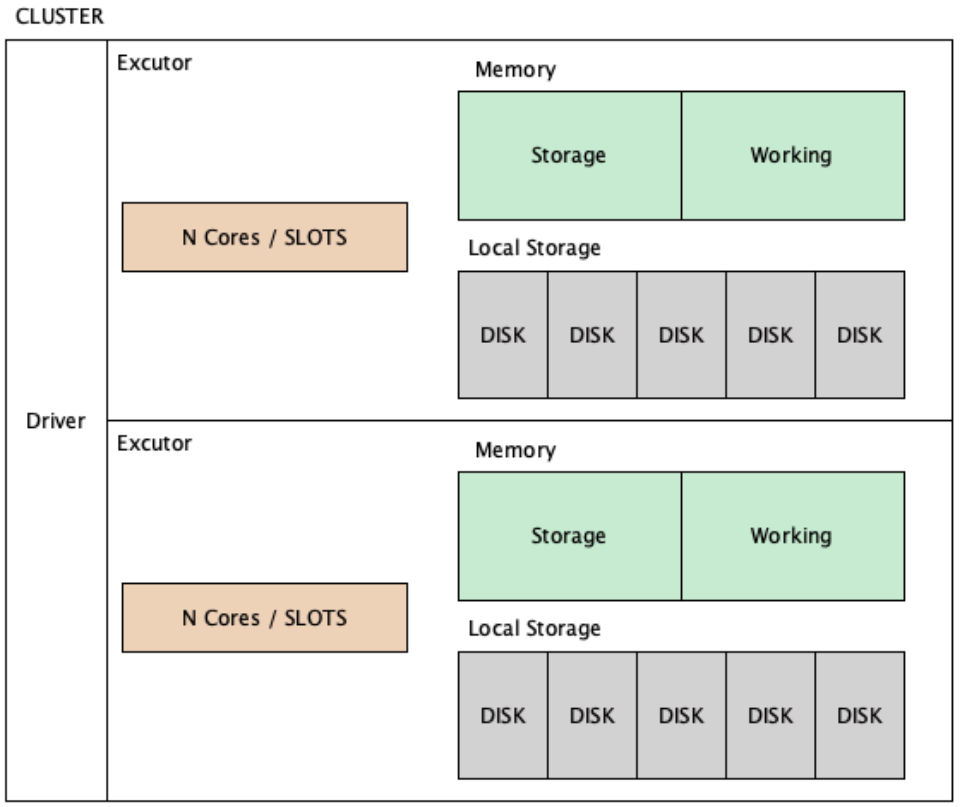
Daniel Tomes, Databricks

#UnifiedAnalytics #SparkAISummit

Talking Points

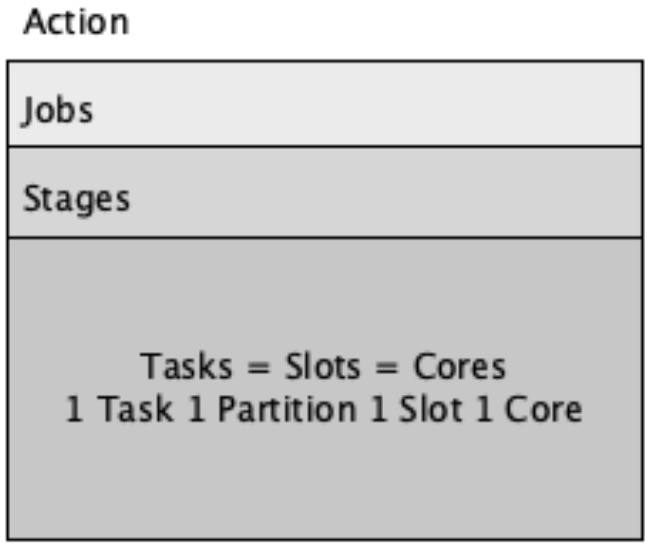
- Spark Hierarchy
- The Spark UI
- Rightsizing & Optimizing
- Advanced Optimizations

Spark Hierarchy



Spark Hierarchy

- Actions are eager
  - Made of transformations (lazy)
    - narrow
    - wide (requires shuffle)
  - Spawn jobs
    - Spawn Stages
      - Spawn Tasks
        - » Do work & utilize hardware



Navigating The Spark UI

DEMO

Understand Your Hardware

- Core Count & Speed
- Memory Per Core (Working & Storage)
- Local Disk Type, Count, Size, & Speed
- Network Speed & Topology
- Data Lake Properties (rate limits)
- Cost / Core / Hour
  - Financial For Cloud
  - Opportunity for Shared & On Prem

Get A Baseline

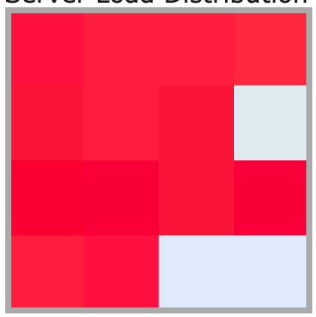
- Is your action efficient?
  - Long Stages, Spills, Laggard Tasks, etc?
- CPU Utilization
  - GANGLIA / YARN / Etc
  - Tails

Goal

CPUs Total:	**216**
Hosts up:	**14**
Hosts down:	**0**

Current Load Avg (15, 5, 1m):
34%, 79%, 125%
Avg Utilization (last hour):
0%

Server Load Distribution



Minimize Data Scans (Lazy Load)

- Data Skipping
  - HIVE Partitions
  - Bucketing
    - Only Experts – Nearly Impossible to Maintain
  - Databricks Delta Z-Ordering
    - What is It
    - How To Do It

Cmd 8

```
1  val master_no_lazy = ss_sub
2    .join(dt_sub.withColumnRenamed("d_date_sk", "ss_sold_date_sk"), Seq("ss_sold_date_sk"))
3    .join(item.withColumnRenamed("i_item_sk", "ss_item_sk"), Seq("ss_item_sk"))
4    .join(inv.withColumnRenamed("inv_item_sk", "ss_item_sk"), Seq("ss_item_sk"))
```

No Lazy Loading

Cmd 8

```
val master_no_lazy = ss_sub
  .join(dt_sub.withColumnRenamed("d_date_sk", "ss_sold_date_sk"), Seq("ss_sold_date_sk"))
  .join(item.withColumnRenamed("i_item_sk", "ss_item_sk"), Seq("ss_item_sk"))
  .join(inv.withColumnRenamed("inv_item_sk", "ss_item_sk"), Seq("ss_item_sk"))
```

Simple

▾Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	7127041251392214025	master_no_lazy.write.mode("overwrite").format("... save at command-885910:1 +details (kill)	2019/03/31 17:11:16	6.2 h	0/200 (96 running)			58.7 GB	

Extra Shuffle Partitions

▾Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	3026732521940153177	spark.conf.set("spark.sql.shuffle.partitions", ... save at command-885910:2 +details (kill)	2019/03/31 23:30:54	50 min	0/2016 (96 running)			5.8 GB	

With Lazy Loading

Cmd 12

```
val master_lazy = ss_sub
  .join(dt_sub
          .filter(year('d_date).between(2000,2001)),
        'd_date_sk === 'ss_sold_date_sk)
  .join(item.withColumnRenamed("i_item_sk", "ss_item_sk"), Seq("ss_item_sk"))
  .join(inv,
        'inv_item_sk === 'ss_item_sk &&
        'inv_date_sk === 'ss_sold_date_sk)
```

Details for Stage 21 (Attempt 0)

Total Time Across All Tasks: 12.9 h
Locality Level Summary: Process local: 200
Output: 21.8 GB / 3112776340
Shuffle Read: 53.9 GB / 1883116073
Shuffle Spill (Memory): 552.0 GB
Shuffle Spill (Disk): 67.4 GB

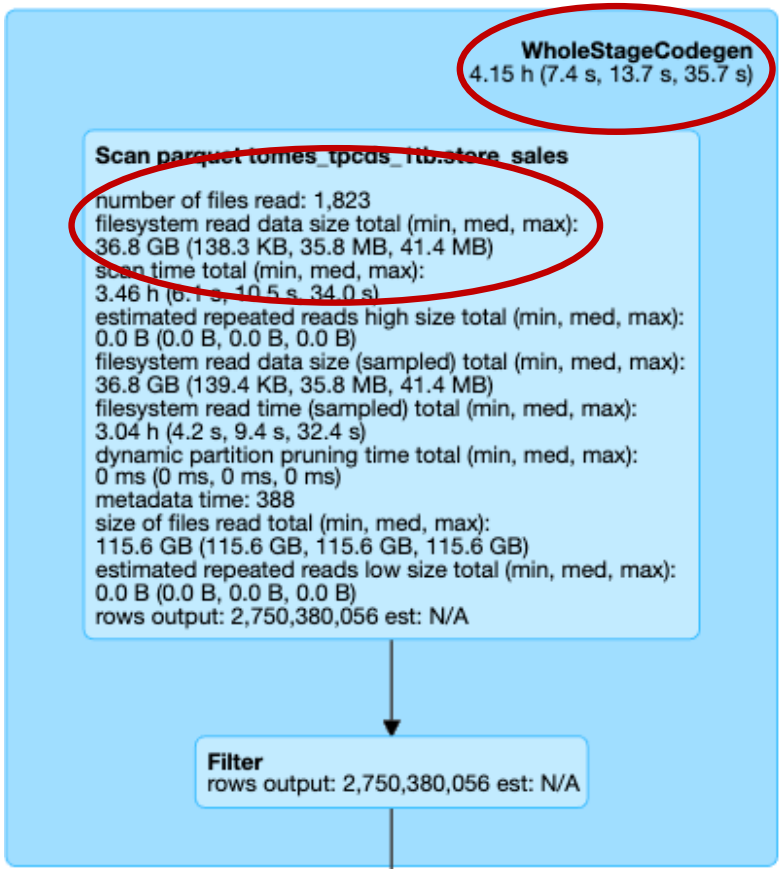
▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1.7 min	3.8 min	3.9 min	4.2 min	4.6 min
GC Time	2 s	7 s	11 s	23 s	30 s
Output Size / Records	111.0 MB / 15333760	111.6 MB / 15524100	111.7 MB / 15566840	111.9 MB / 15608700	114.0 MB / 15771320
Shuffle Read Size / Records	275.0 MB / 9379740	275.8 MB / 9405412	276.1 MB / 9415012	276.4 MB / 9425519	277.7 MB / 9458918
Shuffle spill (memory)	0.0 B	2.9 GB	2.9 GB	2.9 GB	2.9 GB
Shuffle spill (disk)	0.0 B	358.9 MB	359.7 MB	360.1 MB	360.6 MB

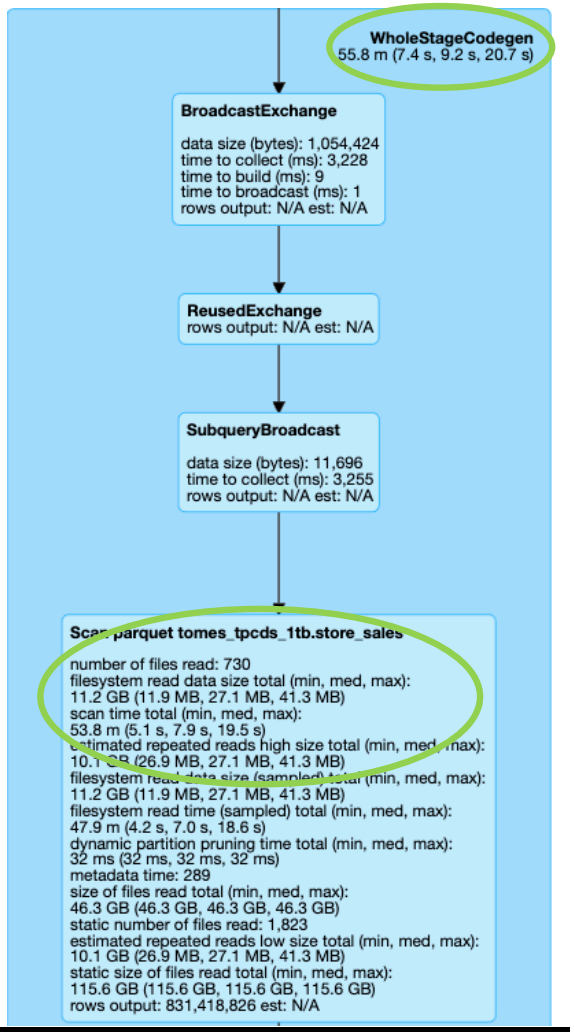
▾Comp

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:17:4	8.5 min	200/200		21.8 GB	53.9 GB	
20	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:16:15	27 s	131/4	3.8 GB			8.6 GB
19	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:16:15	1.3 min	452/452	14.7 GB			45.4 GB



Without Partition Filter

Shrink Partition Range Using a Filter on HIVE Partitioned Column



With Partition Filter

Partitions – Definition

Each of a number of portions into which some operating systems divide memory or storage

~~HIVE PARTITION == SPARK PARTITION~~

Spark Partitions – Types

- Input
  - Controls - Size
    - spark.default.parallelism (don't use)
    - spark.sql.files.maxPartitionBytes (mutable)
      - assuming source has sufficient partitions
- Shuffle
  - Control = Count
    - spark.sql.shuffle.partitions
- Output
  - Control = Size
    - Coalesce(n) to shrink
    - Repartition(n) to increase and/or balance (shuffle)
    - df.write.option("maxRecordsPerFile", N)

Partitions – Shuffle – Default

Partitions – Right Sizing – Shuffle – Master Equation

- Largest Shuffle Stage
  - Target Size <= 200 MB/partition
- Partition Count = Stage Input Data / Target Size
  - Solve for Partition Count

EXAMPLE
Shuffle Stage Input = 210GB
x = 210000MB / 200MB = 1050
spark.conf.set("spark.sql.shuffle.partitions", 1050)
BUT -> If cluster has 2000 cores
spark.conf.set("spark.sql.shuffle.partitions", 2000)

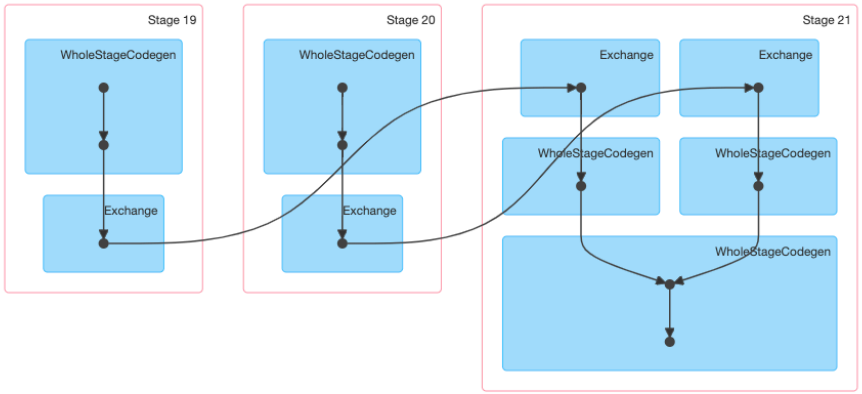
Cluster Spec
96 cores @ 7.625g/core
3.8125g Working Mem
3.8125g Storage Mem

Stage 21 -> Shuffle Fed By Stage 19 & 20
THUS
Stage 21 Shuffle Input = 45.4g + 8.6g == 54g
Default Shuffle Partition == 200 == 54000mb/200parts =~ 270mb/shuffle part

Details for Job 11

Status: SUCCEEDED
Associated SQL Query: 95
Job Group: 238012049243624904_7941856623374133785_21481c00c02f4ff2aee8f2d8342a3ea5
Completed Stages: 3

▸ Event Timeline
▾DAG Visualization



Details for Stage 21 (Attempt 0)

Total Time Across All Tasks: 12.9 h
Locality Level Summary: Process local: 200
Output: 21.8 GB / 3112776340
Shuffle Read: 53.9 GB / 1883116073
Shuffle Spill (Memory): 552.0 GB
Shuffle Spill (Disk): 67.4 GB

▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Spills

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1.7 min	3.8 min	3.9 min	4.2 min	4.6 min
GC Time	2 s	7 s	11 s	23 s	30 s
Output Size / Records	111.0 MB / 15333760	111.6 MB / 15524100	111.7 MB / 15566840	111.9 MB / 15608700	114.0 MB / 15771320
Shuffle Read Size / Records	275.0 MB / 9379740	275.8 MB / 9405412	276.1 MB / 9415012	276.4 MB / 9425519	277.7 MB / 9458918
Shuffle spill (memory)	0.0 B	2.9 GB	2.9 GB	2.9 GB	2.9 GB
Shuffle spill (disk)	0.0 B	358.9 MB	359.7 MB	360.1 MB	360.6 MB

▾Completed Stages (3)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:17:34	8.5 min	200/200		21.8 GB	53.9 GB	
20	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:16:15	27 s	131/131	3.8 GB			8.6 GB
19	238012049243624904	master_lazy_item_bc.write.format("parquet").sav... save at command-885895:1 +details	2019/03/31 14:16:15	1.3 min	452/452	14.7 GB			45.4 GB

Cluster Spec
96 cores @ 7.625g/core
3.8125g Working Mem
3.8125g Storage Mem

```
1 spark.conf.set("spark.sql.shuffle.partitions", 480)
```

Completed Stages (3)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
37	238012049243624904	spark.conf.set("spark.sql.shuffle.partitions", ... save at command-885895:2 +details	2019/03/31 14:53:45	7.8 min	480/480		19.9 GB	54.0 GB	
36	238012049243624904	spark.conf.set("spark.sql.shuffle.partitions", ... save at command-885895:2 +details	2019/03/31 14:52:25	25 s	131/131	3.8 GB			8.7 GB
35	238012049243624904	spark.conf.set("spark.sql.shuffle.partitions", ... save at command-885895:2 +details	2019/03/31 14:52:25	1.3 min	452/452	14.7 GB			45.3 GB

480 shuffle partitions – WHY?
Target shuffle part size == 100m
p = 54g / 100m == 540
540p / 96 cores == 5.625
96 * 5 == 480

If p == 540 another 60p have to be loaded and processed after first cycle is complete

Details for Stage 37 (Attempt 0)

Total Time Across All Tasks: 11.8 h
Locality Level Summary: Process local: 480
Output: 19.9 GB / 3112776340
Shuffle Read: 54.0 GB / 1883116073

- DAG Visualization
- Show Additional Metrics
- Event Timeline

NO SPILL

Summary Metrics for 480 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1.2 min	1.5 min	1.5 min	1.6 min	1.8 min
GC Time	0.8 s	2 s	2 s	3 s	12 s
Output Size / Records	41.5 MB / 6362420	42.2 MB / 6456400	42.5 MB / 6484840	42.7 MB / 6515940	43.4 MB / 6601140
Shuffle Read Size / Records	114.3 MB / 3895329	115.0 MB / 3917334	115.2 MB / 3923186	115.4 MB / 3929412	115.9 MB / 3948584

Input Partitions – Right Sizing

- Use Spark Defaults (128MB) unless…
  - Increase Parallelism
  - Heavily Nested/Repetitive Data
  - Generating Data – i.e. Explode
  - Source Structure is not optimal (upstream)
  - UDFs

```
spark.conf.set("spark.sql.files.maxPartitionBytes", 16777216)
```

Cmd 48

```
1 spark.conf.set("spark.sql.files.maxPartitionBytes", 134217728) 128mb
2 val master_source_10p = spark.read.parquet("/tmp/tomes/scratch/SAIS19/10p_source")
3 master_source_10p.rdd.partitions.size
```

▸ (1) Spark Jobs

▸ master_source_10p: org.apache.spark.sql.DataFrame = [ss_item_sk: integer, ss_customer_sk: integer ... 33 m

```
master_source_10p: org.apache.spark.sql.DataFrame = [ss_item_sk: int, ss_customer_sk: in
res34: Int = 90
```

```
1 spark.conf.set("spark.sql.files.maxPartitionBytes", 1024 * 1024 * 16) 16mb
2 val master_source_10p = spark.read.parquet("/tmp/tomes/scratch/SAIS19/10p_source")
3 master_source_10p.rdd.partitions.size
```

▸ (1) Spark Jobs

▸ master_source_10p: org.apache.spark.sql.DataFrame = [ss_item_sk: integer, ss_customer_sk: integer ... 33 r

```
master_source_10p: org.apache.spark.sql.DataFrame = [ss_item_sk: int, ss_customer_sk: i
res36: Int = 710
```

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
3	7427136891458074341	spark.conf.set("spark.sql.files.maxPartitionByt... save at command-899638:4 +details	2019/04/23 13:49:35	4.6 min	90/90	11.9 GB	24.1 GB

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2.9 min	3.8 min	4.4 min	4.5 min	4.5 min
GC Time	1 s	1 s	1 s	2 s	2 s
Input Size / Records	105.9 MB / 26344585	138.9 MB / 35511851	139.0 MB / 35544755	139.1 MB / 35576807	139.4 MB / 35647533
Output Size / Records	213.8 MB / 26344585	280.5 MB / 35511851	281.1 MB / 35544755	281.5 MB / 35576807	282.6 MB / 35647533

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	6 s	1.1 min	1.5 min	1.6 min	2.5 min
GC Time	48 ms	3 s	4 s	5 s	6 s
Input Size / Records	11.0 MB / 0	26.9 MB / 4414822	27.0 MB / 4441791	27.1 MB / 4471397	27.4 MB / 4537083
Output Size / Records	0.0 B / 0	25.1 MB / 4414822	25.4 MB / 4441791	25.6 MB / 4471397	26.2 MB / 4537083

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
1	7427136891458074341	spark.conf.set("spark.sql.files.maxPartitionByt... save at command-899639:4 +details	2019/04/23 13:46:34	2.9 min	710/710	18.6 GB	17.4 GB

Output Partitions – Right Sizing

- Write Once -> Read Many
  - More Time to Write but Faster to Read
- Perfect writes limit parallelism
  - Compactions (minor & major)

Write Data Size = 14.7GB

Desired File Size = 1500MB

Max write stage parallelism = 10

96 – 10 == 86 cores idle during write

Only 10 Cores Used

▾Completed Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
113	9020956004722677074	val item_bc = broadcast(item.withColumnRenamed(... save at command-885938:10 +details	2019/03/31 19:51:10	14 min	10/10	18.9 GB	14.3 GB		

Cmd 21

```
1  getSizeInfo(s"$s3Prefix/store_sales_2y_10p")
```

▸ (4) Spark Jobs

```
Table store_sales_2y_10p has 10 files and average 1.5304093662 gb with stddev of 250 mb
```

Command took 0.61 seconds -- by daniel.tomes@databricks.com at 3/31/2019, 2:21:49 PM on Tomes_Base

Average File Size == 1.5g

All 96 Cores Used

▾Completed Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
97	9020956004722677074	val item_bc = broadcast(item.withColumnRenamed(... save at command-885971:10 +details	2019/03/31 19:43:41	1.8 min	96/96	15.1 GB	14.3 GB		

Cmd 22

```
1  getSizeInfo(s"$s3Prefix/store_sales_2y_96p")
```

▸ (4) Spark Jobs

```
Table store_sales_2y_10p has 96 files and average 0.15996809254166666 gb with stddev of 35 mb
```

Command took 2.00 seconds -- by daniel.tomes@databricks.com at 3/31/2019, 3:46:45 PM on Tomes_Base

Average File Size == 0.16g

Output Partitions – Composition

- df.write.option("maxRecordsPerFile", n)
- df.coalesce(n).write…
- df.repartition(n).write…
- df.repartition(n, [colA, …]).write…
- spark.sql.shuffle.partitions(n)
- df.localCheckpoint(…).repartition(n).write…
- df.localCheckpoint(…).coalesce(n).write…

Partitions – Why So Serious?

- Avoid The Spill
- Maximize Parallelism
  - Utilize All Cores
  - Provision only the cores you need

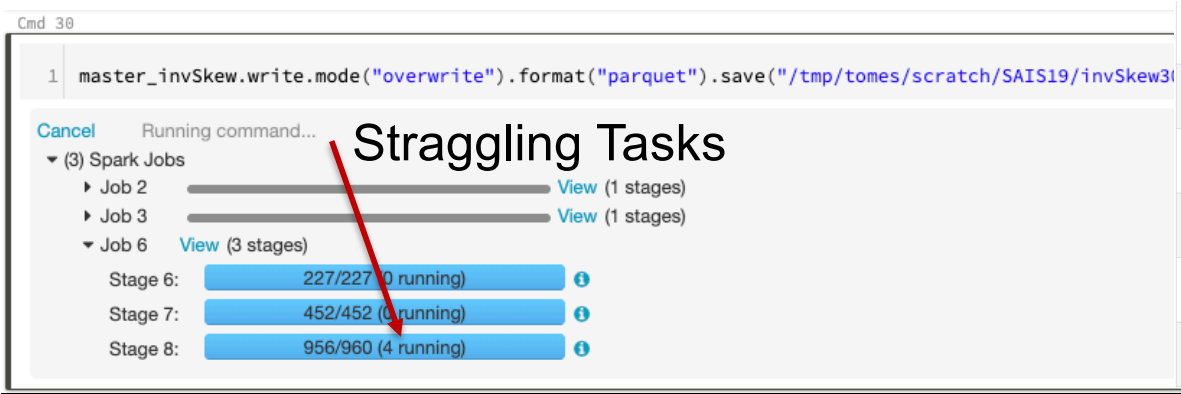
Advanced Optimizations

- Finding Imbalances
- Persisting
- Join Optimizations
- Handling Skew
- Expensive Operations
- UDFs
- Multi-Dimensional Parallelism

Balance

- Maximizing Resources Requires Balance
  - Task Duration
  - Partition Size
- SKEW
  - When some partitions are significantly larger than most

Input Partitions
Shuffle Partitions
Output Files
Spills
GC Times



Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Output Size / Records	Shuffle Read Size / Records	Shuffle Spill (Memory
387	1661	0	RUNNING	PROCESS_LOCAL	4	10.126.236.32 stdout stderr	2019/03/31 23:09:51	0 ms	15 min	0.0 B / 0	665.0 MB / 44831864	672.0 MB
433	1707	0	RUNNING	PROCESS_LOCAL	5	10.126.243.84 stdout stderr	2019/03/31 23:10:24	0 ms	12 min	0.0 B / 0	666.7 MB / 44808085	672.0 MB
547	1821	0	RUNNING	PROCESS_LOCAL	2	10.126.251.232 stdout stderr	2019/03/31 23:11:24	0 ms	12 min	0.0 B / 0	665.7 MB / 44810914	672.0 MB
654	1928	0	RUNNING	PROCESS_LOCAL	1	10.126.243.152 stdout stderr	2019/03/31 23:12:38	0 ms	13 min	0.0 B / 0	666.5 MB / 44816824	672.0 MB
0	1274	0	SUCCESS	PROCESS_LOCAL	2	10.126.251.232 stdout stderr	2019/03/31 23:07:09	52 s	2 s	16.2 MB / 3214560	321.2 MB / 1960754	0.0 B

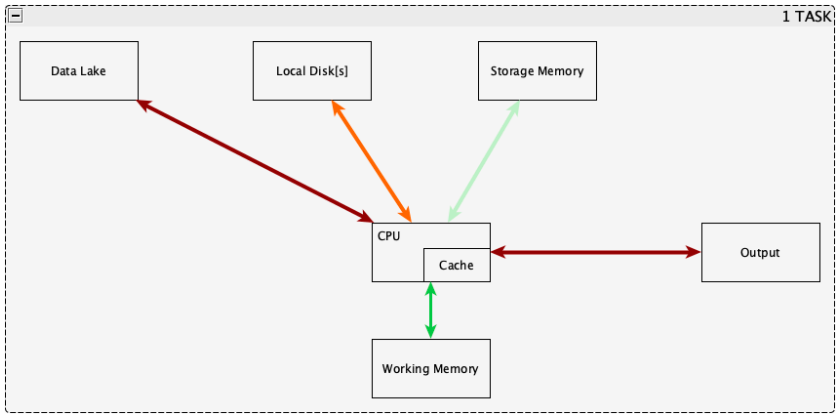
75th percentile ~ 2m recs
max ~ 45m recs
stragglers take > 22X longer IF no spillage
With spillage, 100Xs longer

Summary Metrics for 956 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	40 s	47 s	52 s	3.7 min
GC Time	0.3 s	0.8 s	1 s	4 s	3.2 min
Output Size / Records	0.0 B / 0	16.1 MB / 3219340	16.3 MB / 3242100	16.5 MB / 3263540	17.0 MB / 3323340
Shuffle Read Size / Records	317.2 MB / 1940846	320.7 MB / 1957433	321.6 MB / 1961754	322.4 MB / 1965647	666.7 MB / 44831864
Shuffle spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	672.0 MB
Shuffle spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	91.5 MB

Minimize Data Scans (Persistence)

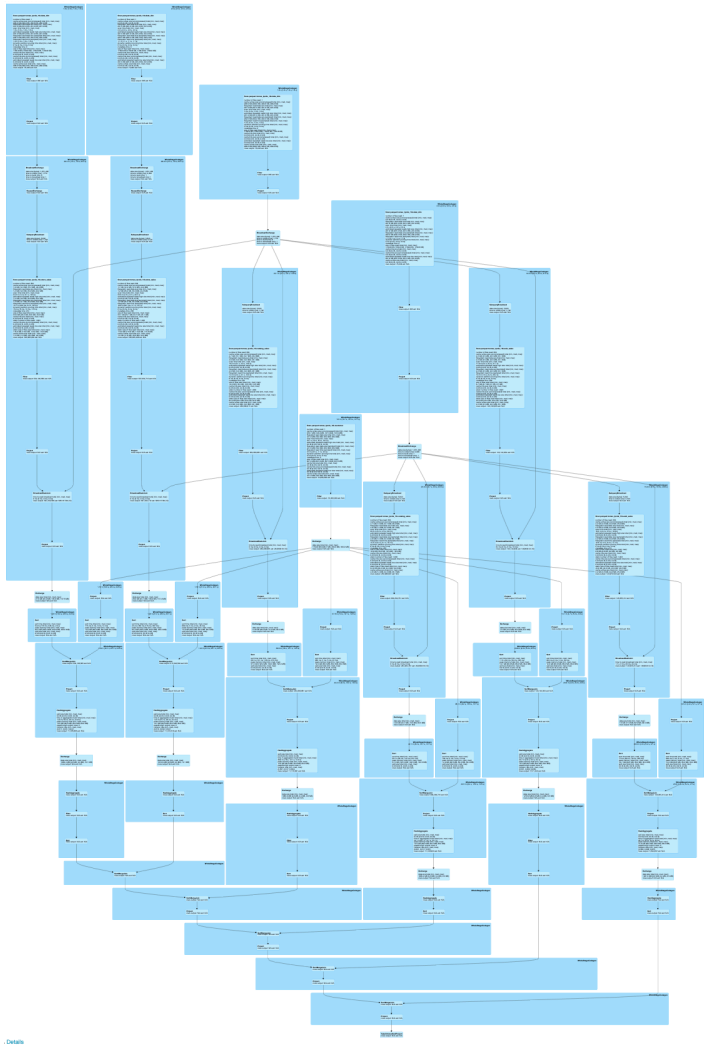
- Persistence
  - Not Free
- Repetition
  - SQL Plan



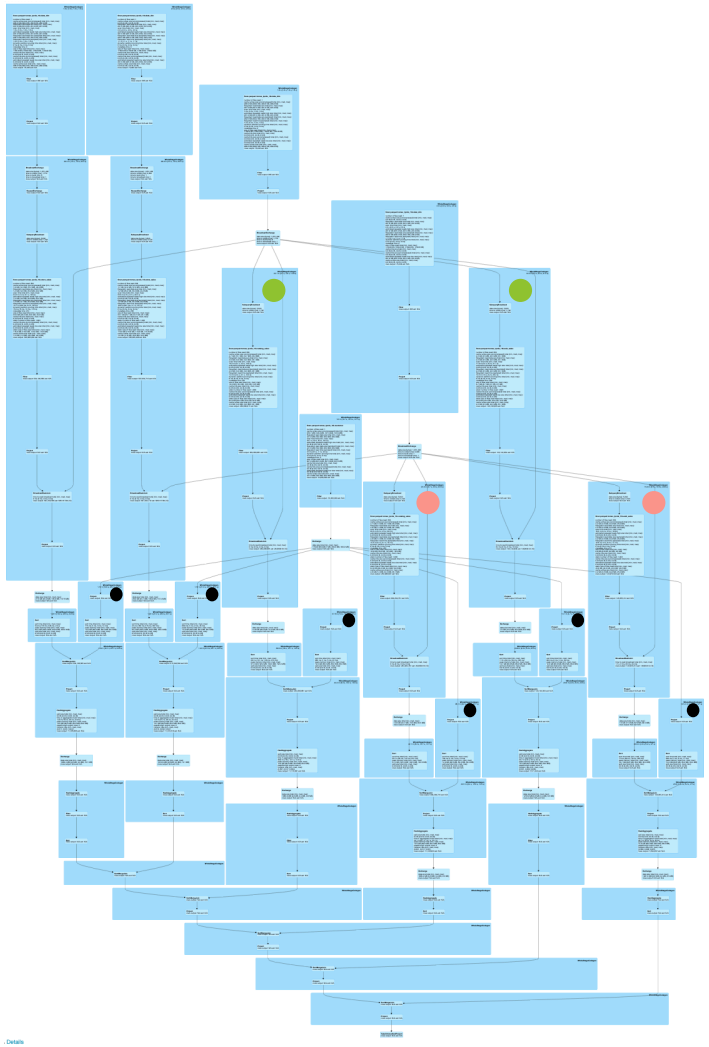
df.cache == df.persist(StorageLevel.MEMORY_AND_DISK)

- Types
  - Default (MEMORY_AND_DISK)
    - Deserialized
  - Deserialized = Faster = Bigger
  - Serialized = Slower = Smaller
  - _2 = Safety = 2X bigger
  - MEMORY_ONLY
  - DISK_ONLY

Don't Forget To Cleanup!
df.unpersist



TPCDS Query 4



Minimize Data Scans (Delta Cache)

CACHE SELECT column_name[, **column_name**, ...] **FROM** [db_name.]**table_name** [**WHERE** boolean_expression]

Jobs Stages Storage Environment Executors SQL JDBC/ODBC Server

Storage

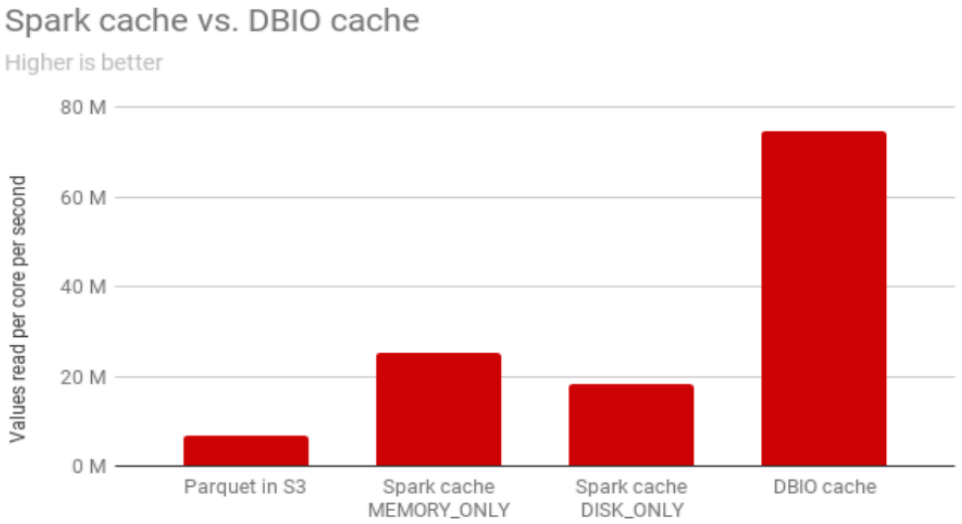
Parquet IO Cache

Host	Disk Usage	Max Disk Usage Limit	Percent Disk Usage	Metadata Cache Size	Max Metadata Cache Size Limit	Percent Metadata Usage
10.231.249.153	168.1 GB	1855.5 GB	9 %	5.8 MB	37.1 GB	0 %
10.231.250.190	175.4 GB	1855.5 GB	9 %	6.0 MB	37.1 GB	0 %
Total	343.5 GB	3.6 TB	9 %	11.8 MB	74.2 GB	0 %

Data Read from External Filesystem	Data Read from IO Cache	Data Written to IO Cache	Estimated Size of Repeatedly Read Data	Cache Metadata Manager Peak Disk Usage
260.9 GB	52.3 GB	260.9 GB	53.1 GB (16 %) - 53.1 GB (16 %)	4.3 KB

Hot Data Auto Cached
Super Fast
Relieves Memory Pressure

spark.databricks.io.cache.enabled true
spark.databricks.io.cache.maxDiskUsage 50g
spark.databricks.io.cache.maxMetaDataCache 1g
spark.databricks.io.cache.compression.enabled false



Why So Fast?

HOW TO USE
AWS - i3s – On By Default
AZURE – Ls-series – On By Default

Join Optimization

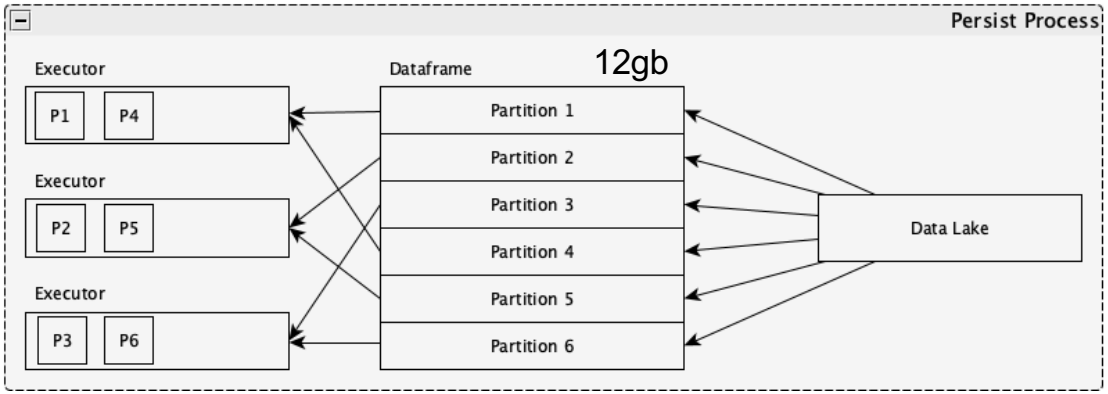
- SortMergeJoins (Standard)
- Broadcast Joins (Fastest)
- Skew Joins
- Range Joins
- BroadcastedNestedLoop Joins (BNLJ)

Join Optimization

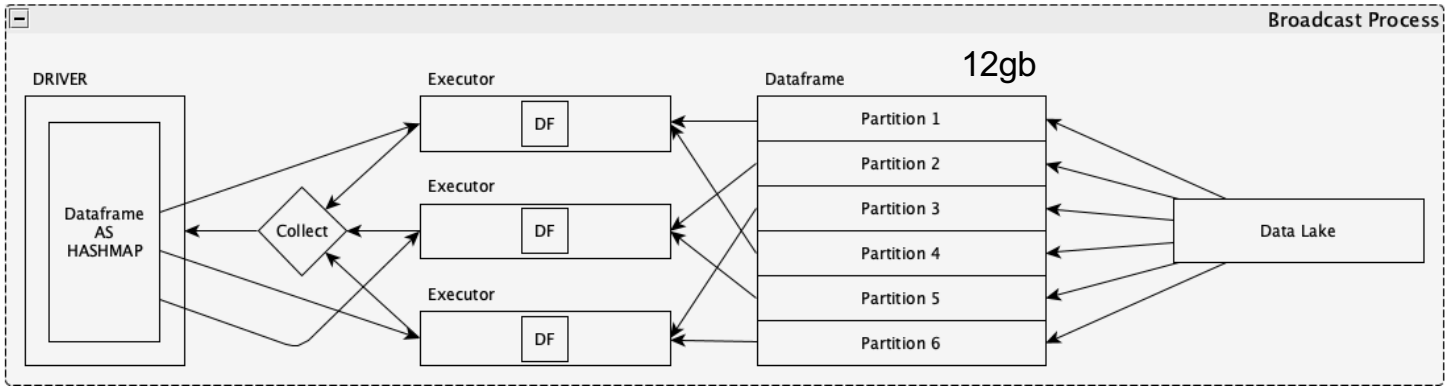
- SortMerge Join – Both sides are lrage
- Broadcast Joins – One side is small
  - Automatic If:

    (one side < *spark.sql.autoBroadcastJoinThreshold*) (default 10m)
  - Risks
    - Not Enough Driver Memory
    - DF > *spark.driver.maxResultSize*
    - DF > Single Executor Available Working Memory
  - Prod – Mitigate The Risks
    - Validation Functions

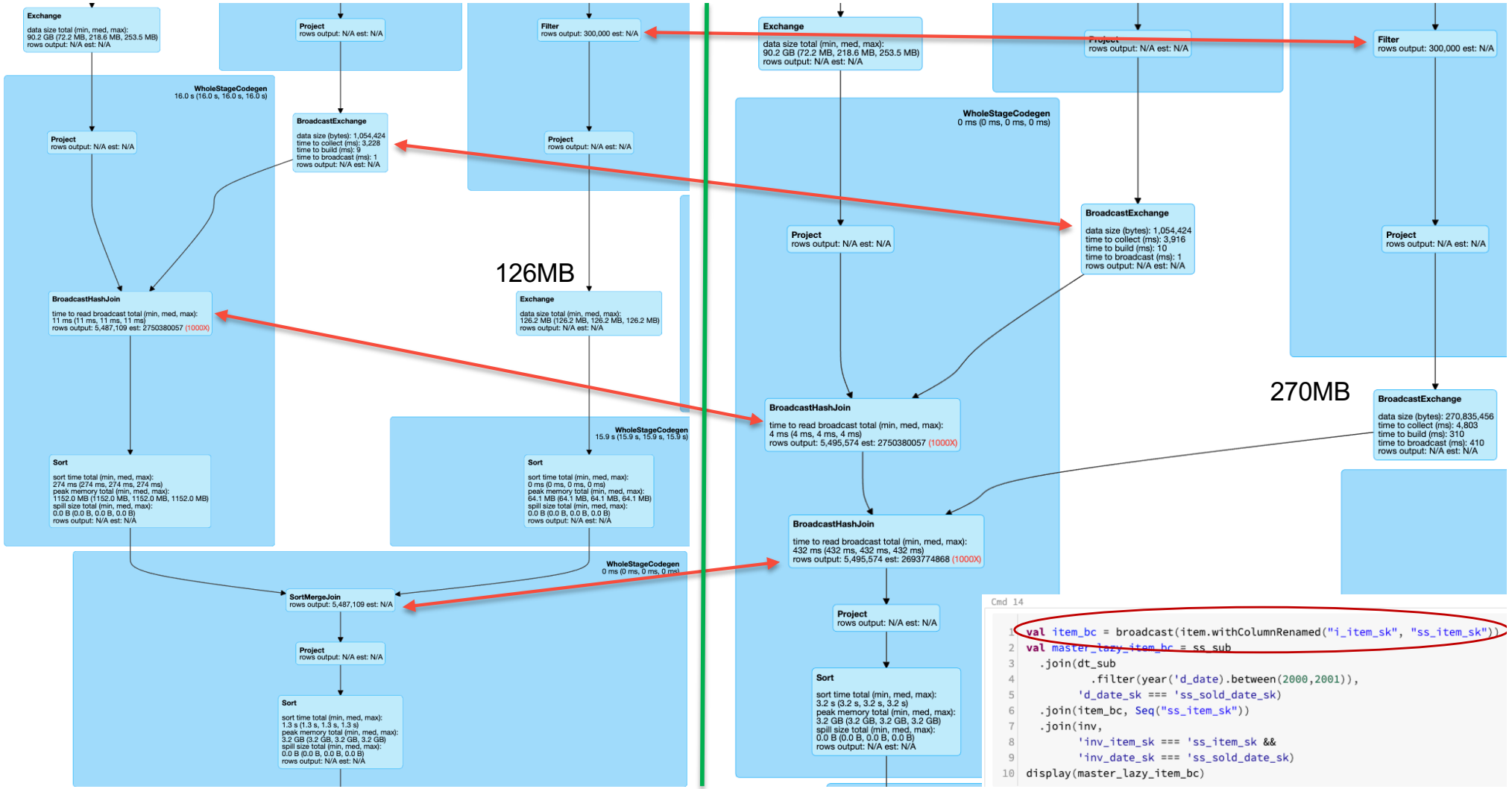
Persistence Vs. Broadcast



Attempt to send compute to the data



Data availability guaranteed ->
each executor has entire dataset


Exchange
data size total (min, med, max):
90.2 GB (72.2 MB, 218.6 MB, 253.5 MB)
rows output: N/A est: N/A
Project
rows output: N/A est: N/A
Filter
rows output: 300,000 est: N/A
WholeStageCodegen
16.0 s (16.0 s, 16.0 s, 16.0 s)
BroadcastExchange
data size (bytes): 1,054,424
time to collect (ms): 3,228
time to build (ms): 9
time to broadcast (ms): 1
rows output: N/A est: N/A
126MB
BroadcastHashJoin
time to read broadcast total (min, med, max):
11 ms (11 ms, 11 ms, 11 ms)
rows output: 5,487,109 est: 2750380057 (1000X)
Sort
sort time total (min, med, max):
274 ms (274 ms, 274 ms, 274 ms)
peak memory total (min, med, max):
1152.0 MB (1152.0 MB, 1152.0 MB, 1152.0 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
rows output: N/A est: N/A
WholeStageCodegen
15.9 s (15.9 s, 15.9 s, 15.9 s)
Sort
sort time total (min, med, max):
0 ms (0 ms, 0 ms, 0 ms)
peak memory total (min, med, max):
64.1 MB (64.1 MB, 64.1 MB, 64.1 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
rows output: N/A est: N/A
WholeStageCodegen
0 ms (0 ms, 0 ms, 0 ms)
SortMergeJoin
rows output: 5,487,109 est: N/A
Project
rows output: N/A est: N/A
Sort
sort time total (min, med, max):
1.3 s (1.3 s, 1.3 s, 1.3 s)
peak memory total (min, med, max):
3.2 GB (3.2 GB, 3.2 GB, 3.2 GB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
rows output: N/A est: N/A
Exchange
data size total (min, med, max):
126.2 MB (126.2 MB, 126.2 MB, 126.2 MB)
rows output: N/A est: N/A
Exchange
data size total (min, med, max):
90.2 GB (72.2 MB, 218.6 MB, 253.5 MB)
rows output: N/A est: N/A
Project
rows output: N/A est: N/A
Filter
rows output: 300,000 est: N/A
WholeStageCodegen
0 ms (0 ms, 0 ms, 0 ms)
Project
rows output: N/A est: N/A
BroadcastExchange
data size (bytes): 1,054,424
time to collect (ms): 3,916
time to build (ms): 10
time to broadcast (ms): 1
rows output: N/A est: N/A
Project
rows output: N/A est: N/A
270MB
BroadcastExchange
data size (bytes): 270,835,456
time to collect (ms): 4,803
time to build (ms): 310
time to broadcast (ms): 410
rows output: N/A est: N/A
BroadcastHashJoin
time to read broadcast total (min, med, max):
4 ms (4 ms, 4 ms, 4 ms)
rows output: 5,495,574 est: 2750380057 (1000X)
BroadcastHashJoin
time to read broadcast total (min, med, max):
432 ms (432 ms, 432 ms, 432 ms)
rows output: 5,495,574 est: 2693774868 (1000X)
Project
rows output: N/A est: N/A
Sort
sort time total (min, med, max):
3.2 s (3.2 s, 3.2 s, 3.2 s)
peak memory total (min, med, max):
3.2 GB (3.2 GB, 3.2 GB, 3.2 GB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
rows output: N/A est: N/A
Cmd 14
1 val item_bc = broadcast(item.withColumnRenamed("i_item_sk", "ss_item_sk"))
2 val master_lazy_item_bc = ss_sub
3 .join(dt_sub
4 .filter(year('d_date).between(2000,2001)),
5 'd_date_sk === 'ss_sold_date_sk)
6 .join(item_bc, Seq("ss_item_sk"))
7 .join(inv,
8 'inv_item_sk === 'ss_item_sk &&
9 'inv_date_sk === 'ss_sold_date_sk)
10 display(master_lazy_item_bc)

From 6h and barely started
TO 8m → Lazy Loading
TO 2.5m → Broadcast

Cmd 31

```
1  val master_lazy = ss_sub
2    .join(dt_sub
3            .filter(year('d_date).between(2000,2001)),
4          'd_date_sk === 'ss_sold_date_sk)
5    .join(broadcast(item.withColumnRenamed("i_item_sk", "ss_item_sk")), Seq("ss_item_sk"))
6    .join(inv,
7          'inv_item_sk === 'ss_item_sk &&
8          'inv_date_sk === 'ss_sold_date_sk)
```

▾Completed Stages (3)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
36	4260198380812558228	master_lazy.write.format("parquet").mode("overw... save at command-885896:1 +details	2019/04/19 17:42:45	2.5 min	200/200		21.8 GB	55.6 GB	
35	4260198380812558228	master_lazy.write.format("parquet").mode("overw... save at command-885896:1 +details	2019/04/19 17:42:02	43 s	452/452	14.7 GB			47.1 GB
34	4260198380812558228	master_lazy.write.format("parquet").mode("overw... save at command-885896:1 +details	2019/04/19 17:42:02	21 s	261/261	3.8 GB			8.6 GB

SQL Example:
SELECT /*+ BROADCAST(customers) */ * **FROM** customers, orders **WHERE** o_custId = c_custId

Skew Join Optimization

Cmd 5

```
1  val skewedVals = invWSkew.withColumn("key", concat('inv_item_sk, 'inv_date_sk))
2    .groupBy("key", "inv_item_sk", "inv_date_sk")
3    .agg(count('key).alias("cnt"))
4    .orderBy('cnt.desc)
```

Cmd 6

```
1  display(skewedVals)
```

(1) Spark Jobs
Job 43 View (Stages: 2/2)
Stage 76: 229/229
Stage 77: 480/480

key	inv_item_sk	inv_date_sk	cnt
924052452275	92405	2452275	42872883
924042452275	92404	2452275	42857263
924072452275	92407	2452275	42857248
924062452275	92406	2452275	42854886
924032452275	92403	2452275	42853282
924012452275	92401	2452275	42852880
924022452275	92402	2452275	42851558
565032452341	56503	2452341	20
1196112452341	119611	2452341	20
2926412452341	292641	2452341	20
2283662452341	228366	2452341	20

- OSS Fix - Salting
  - Add Column to each side with random int between 0 and spark.sql.shuffle.partitions – 1 to both sides
  - Add join clause to include join on generated column above
  - Drop temp columns from result
- Databricks Fix (Skew Join)

```
val skewedKeys = List("id1", "id200", "id-99")
df.join(
  skewDF.hint("SKEW", "skewKey", skewedKeys),
  Seq(keyCol), "inner")
```

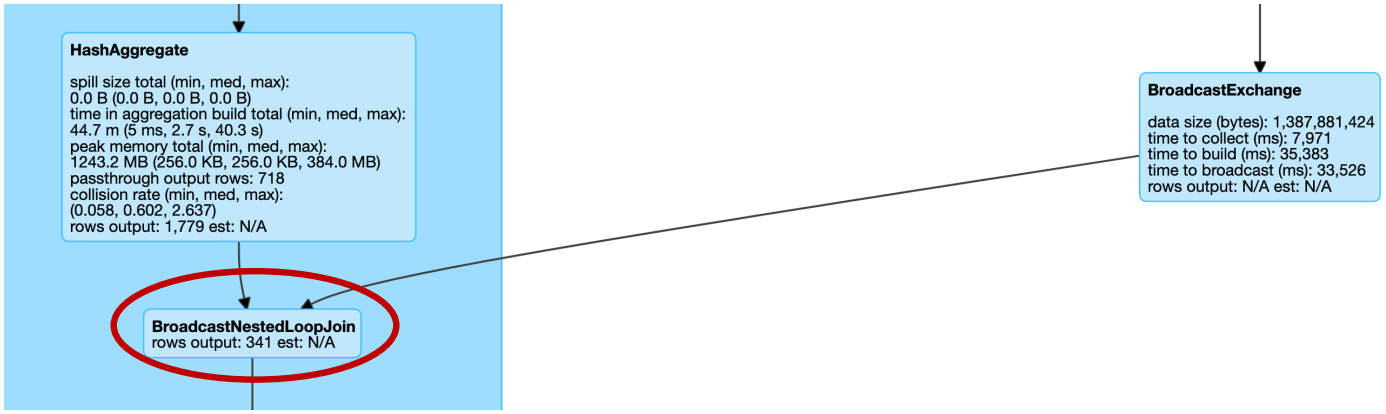
Skewed Aggregates

df.groupBy(“city”, “state”).agg(<f(x)>).orderBy(col.desc)

val saltVal = random(0, spark.conf.get(org...shuffle.partitions) - 1)

```
df.withColumn(“salt”, lit(saltVal))
      .groupBy(“city”, “state”, “salt”)
      .agg(<f(x)>)
      .drop(“salt”)
      .orderBy(col.desc)
```

BroadcastNestedLoopJoin (BNLJ)



```sql
SELECT distinct sys_cd_val , PI_NUM FROM  retail_sales
WHERE loadTS IN (SELECT MAX(loadTS)  FROM  retail_sales )
AND PI_NUM NOT IN (SELECT distinct PI_13 FROM  product_master)
AND sys_cd_val  <> 'PWG'
```

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
47	3925260204415598680	SELECT distinct SRC_SYS_CD , PIN_NUM FROM edl.... collectResult at OutputAggregator.scala:136 +details	2019/04/22 18:56:06	7.1 min	557/563 (6	7.8 GB				Job 17 cancelled part of cancelled job 3925260204415598680_50868365920

```sql
SELECT distinct sys_cd_val , PI_NUM
FROM  retail_sales
WHERE loadTS IN (
  SELECT MAX(loadTS)  FROM  retail_sales
  )
AND NOT exists (
  SELECT 1
  FROM  product_master sub
  where sub.PI_13 <=> a.PI_NUM
  )
AND sys_cd_val  <> 'PWG'
```

ubmitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
019/04/22 19:04:34	0.1 s	1/1			7.1 KB	
019/04/22 19:04:06	28 s	200/200			674.0 MB	1062.5 KB
019/04/22 19:03:5	16 s	569/569	6.6 GB			148.8 MB
019/04/22 19:03:50	14 s	12/12				525.2 MB

Range Join Optimization

- Range Joins Types
  - Point In Interval Range Join
    - Predicate specifies value in one relation that is between two values from the other relation
  - Interval Overlap Range Join
    - Predicate specifies an overlap of intervals between two values from each relation

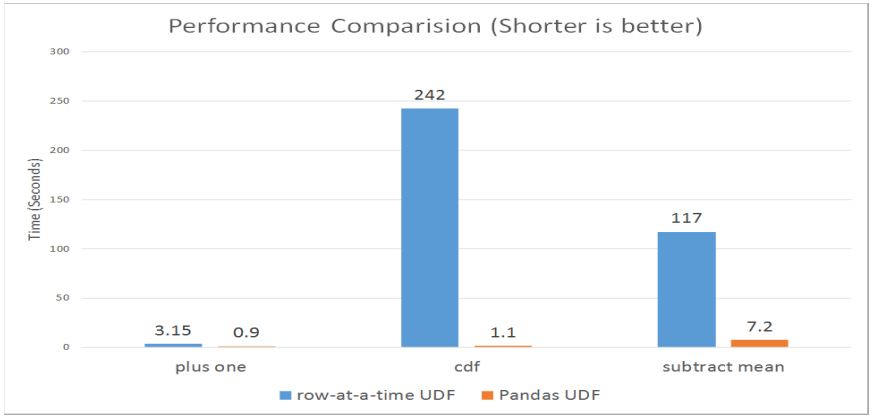
REFERENCE

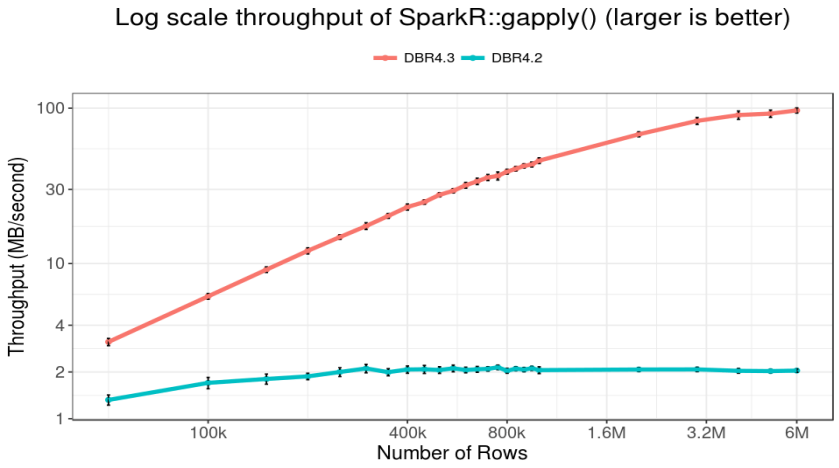
Omit Expensive Ops

- Repartition
  - Use Coalesce or Shuffle Partition Count
- Count – Do you really need it?
- DistinctCount
  - use approxCountDistinct()
- If distincts are required, put them in the right place
  - Use dropDuplicates
  - dropDuplicates BEFORE the join
  - dropDuplicates BEFORE the groupBy

UDF Penalties

- Traditional UDFs cannot use Tungsten
  - Use org.apache.spark.sql.functions
  - Use PandasUDFs
    - Utilizes Apache Arrow
  - Use SparkR UDFs





Advanced Parallelism

- Spark's Three Levels of Parallelism
  - Driver Parallelism
  - Horizontal Parallelism
  - Executor Parallelism

```scala
val origHist_1y = spark.read.table(origHistTable)

spark.conf.set("spark.sql.shuffle.partitions", 500)

private val taskSupport = new ForkJoinTaskSupport(new ForkJoinPool(parallelism))
case class monthPart(year: Int, month: Int)

private val parts: ArrayBuffer[monthPart] = ArrayBuffer[monthPart]()
Range(2018,2019).toArray.foreach(yr => {
 Range(1,13).toArray.foreach(mnth => parts.append(monthPart(yr, mnth)))
})

val parMonth = parts.toArray.par

parMonth.tasksupport = taskSupport
parMonth.foreach(part => {
  try {
    origHist_1y.filter('end_yr === part.year && 'end_mnth === part.month)
      .write.format("delta")
      .mode("append")
      .partitionBy("end_cptr_dt")
      .save(histPerfPath_1y)
  } catch {
    case e: Throwable => println(s"ERROR: Failed on ${part.year}-${part.month} with, $e")
  }
  println(s"Completed ${part.year}-${part.month}")
})
```

Summary

- Utilize Lazy Loading (Data Skipping)
- Maximize Your Hardware
- Right Size Spark Partitions
- Balance
- Optimized Joins
- Minimize Data Movement
- Minimize Repetition
- Only Use Vectorized UDFs

QUESTIONS