Урок 4. Оптимизация

Партиционирование

Optimization: partitioning

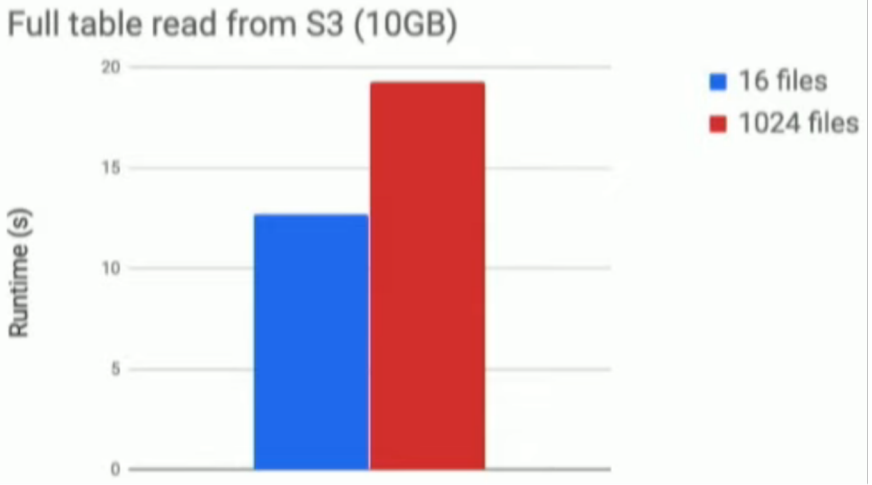
- Embed predicates in directory structure

```
df.write.partitionBy("date").parquet(...)

./example_parquet_file/date=2019-10-15/...
./example_parquet_file/date=2019-10-16/...
./example_parquet_file/date=2019-10-17/part-00000-...-475b15e2874d.c000.snappy.parquet
```

...

Optimization: avoid many small files

- For every file
  - Set up internal data structures
  - Instantiate reader objects
  - Fetch file
  - Parse Parquet metadata



Optimization: avoid few huge files

- Also avoid having huge files!
- `SELECT count(*)` on 250GB dataset
  - 250 partitions (~1GB each)
    - 5 mins
  - 1 huge partition (250GB)
    - **1 hour**
- Footer processing not optimized for speed...

https://stackoverflow.com/questions/52043874/limiting-maximum-size-of-dataframe-partition

Bucketing in Spark

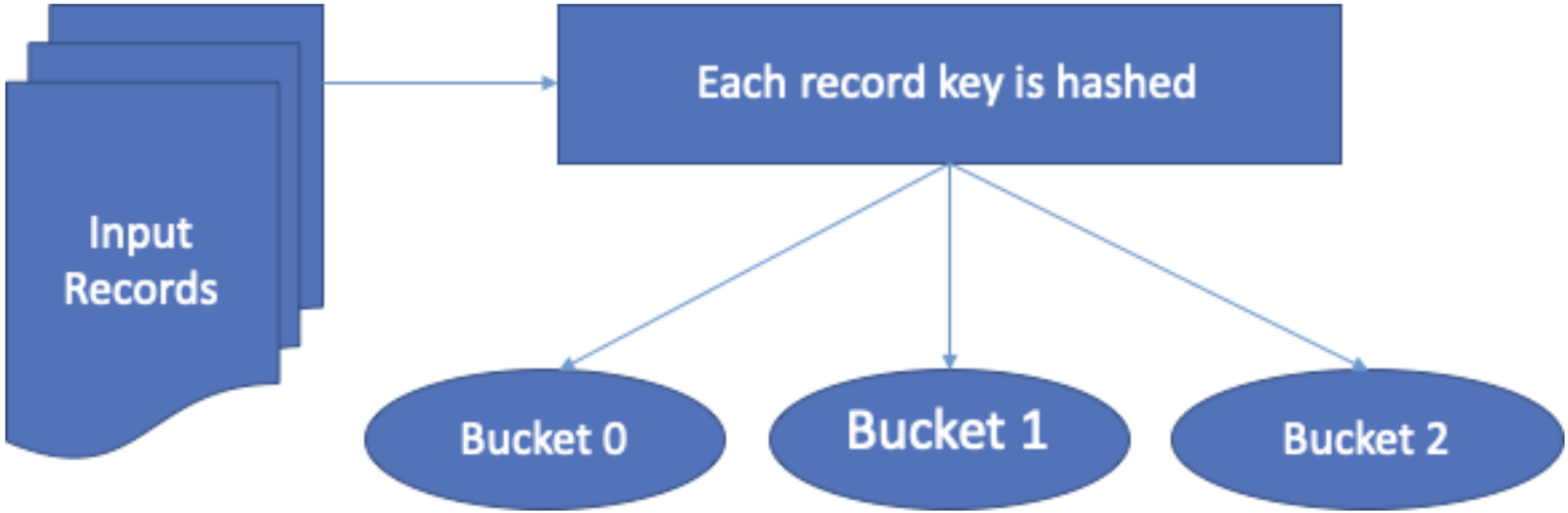
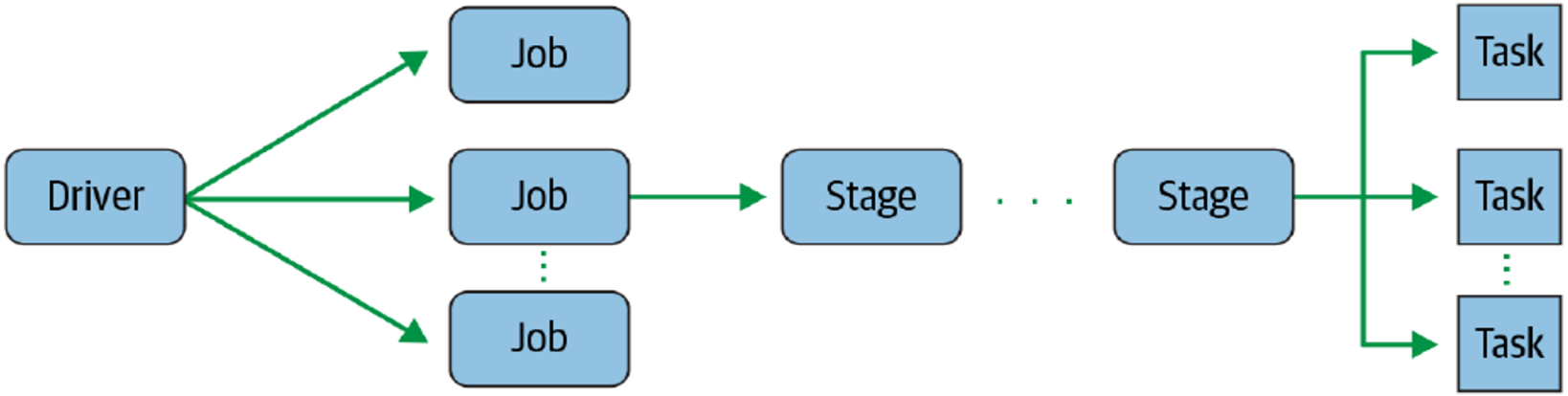


Figure 1.1

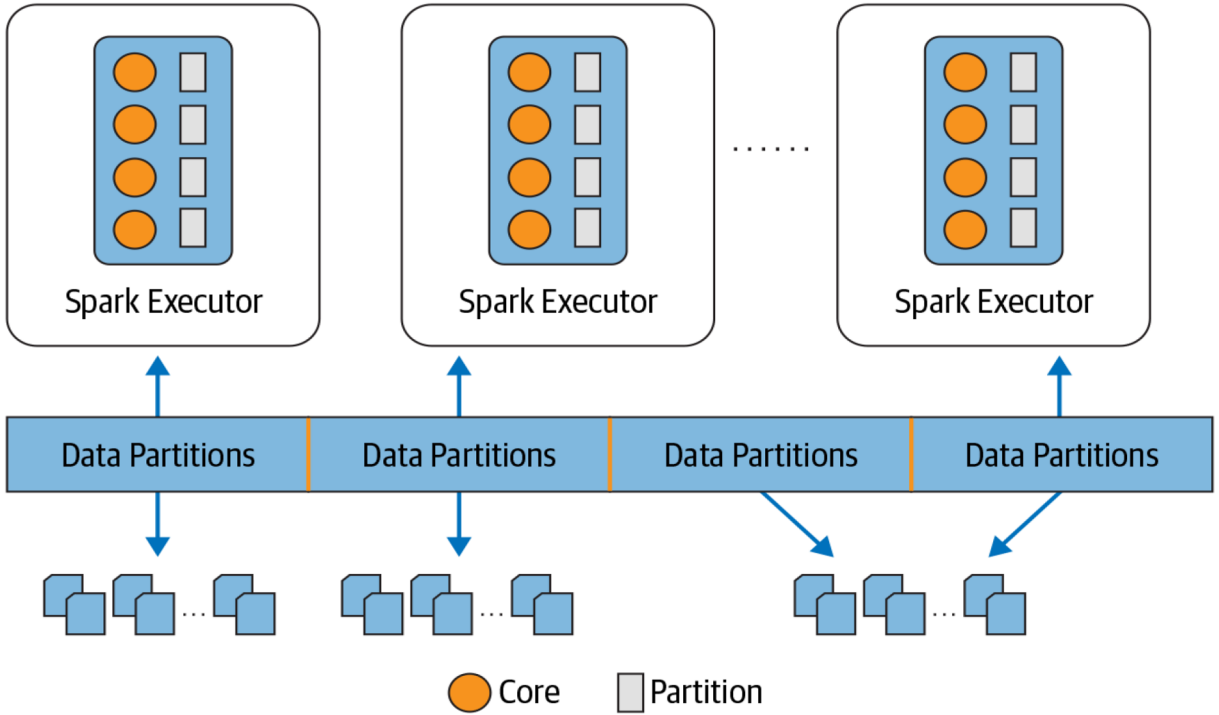
Spark jobs, stage, task

shuffle occurs between every two stages



Spark executors, partitions

Relationship of Spark tasks, cores, partitions



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)

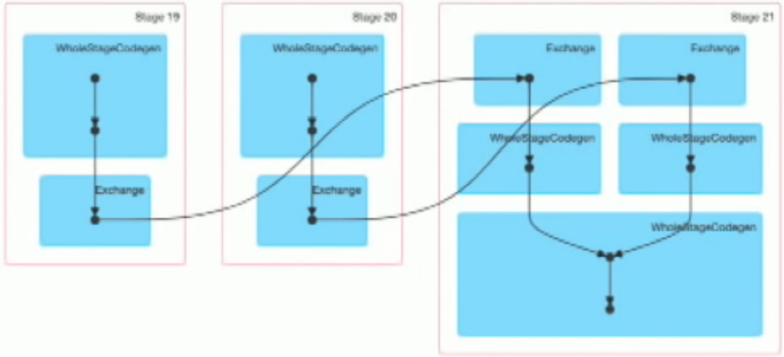
Cluster Spec
96 cores @ 7.625g/core
3.8125g Working Mem
3.8125g Storage Mem

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: 2380120492438249O4_7941856823374133785_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

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 / 19333760	111.8 MB / 19524100	111.7 MB / 19566840	111.9 MB / 19606700	114.0 MB / 19771320
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	2380120492438249O4	master_lazy_item_bc.write.format("parquet").sav... save at command-885885:1 +details	2019/03/31 14:17:34	8.5 min	200/200		21.8 GB	53.9 GB	
20	2380120492438249O4	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	2380120492438249O4	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

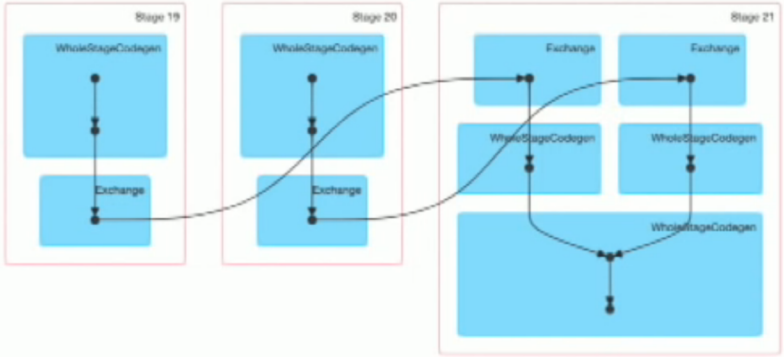
Cluster Spec
96 cores @ 7.625g/core
3.8125g Working Mem
3.8125g Storage Mem

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: 2380120492438249O4_7941856823374133785_21481c00c02f4ff2aee8f2d8342a3ee5
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 / 1853116073
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 / 19333760	111.8 MB / 19524100	111.7 MB / 19566840	111.9 MB / 19808700	114.0 MB / 19771320
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	2380120492438249O4	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	2380120492438249O4	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	2380120492438249O4	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

Data skew

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

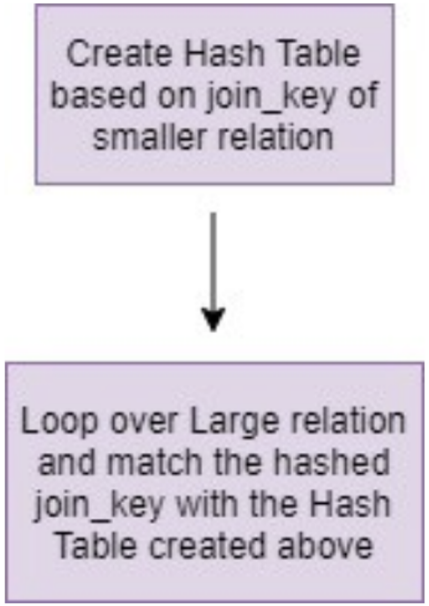
Skewed Aggregates

```
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)
```

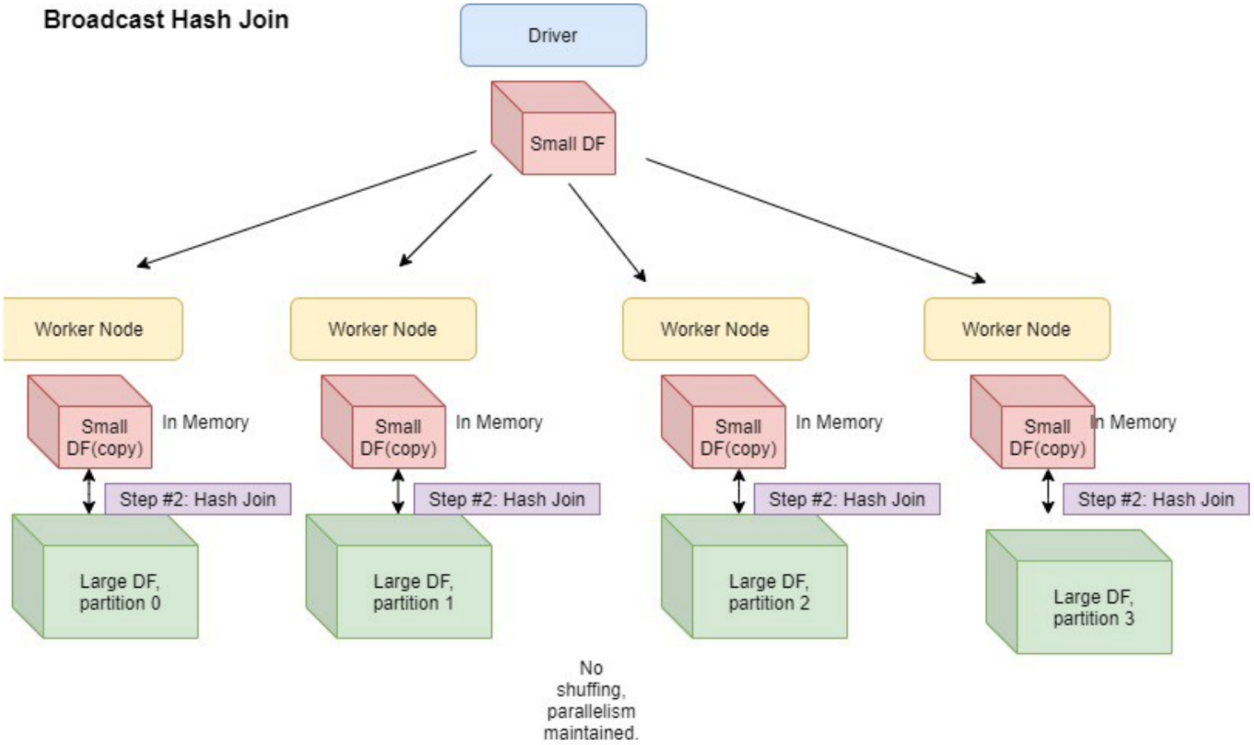
Join Optimization

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

Broadcast Hash Join

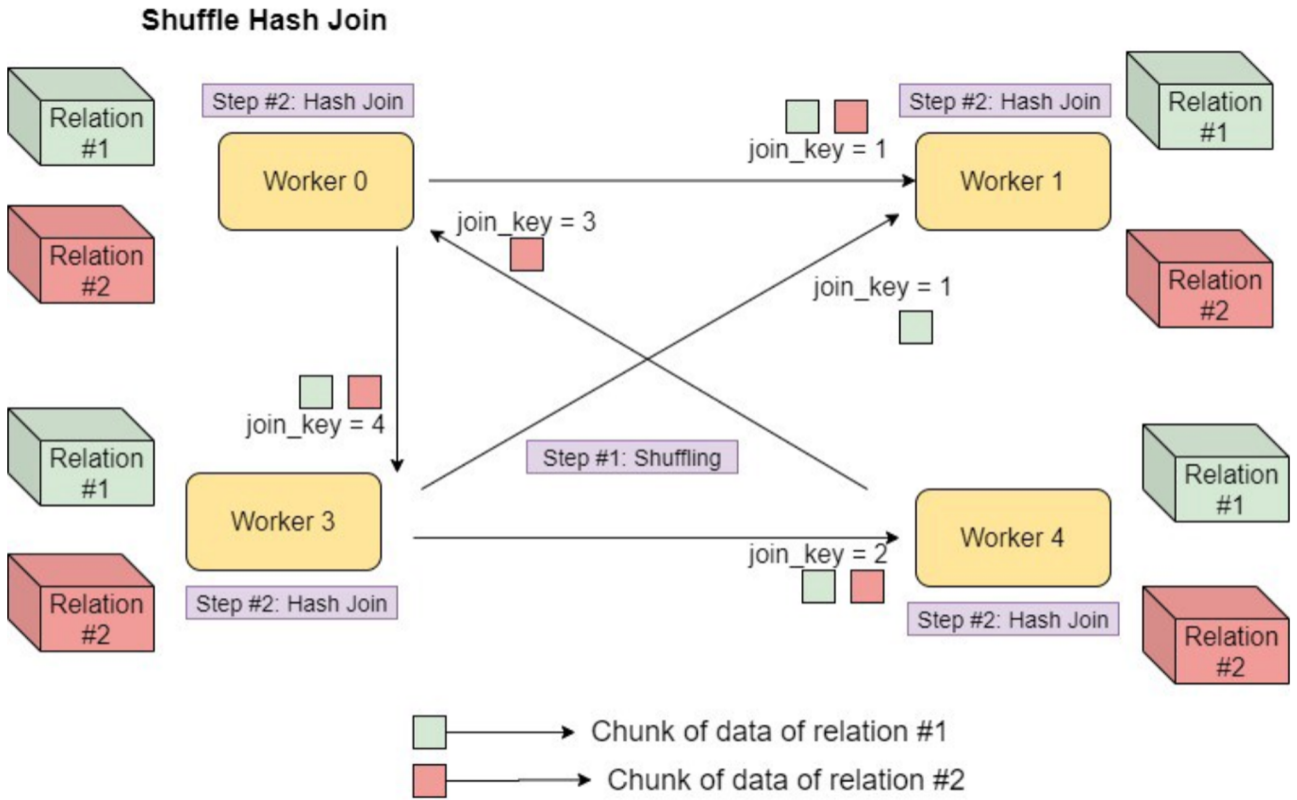


Hash Join

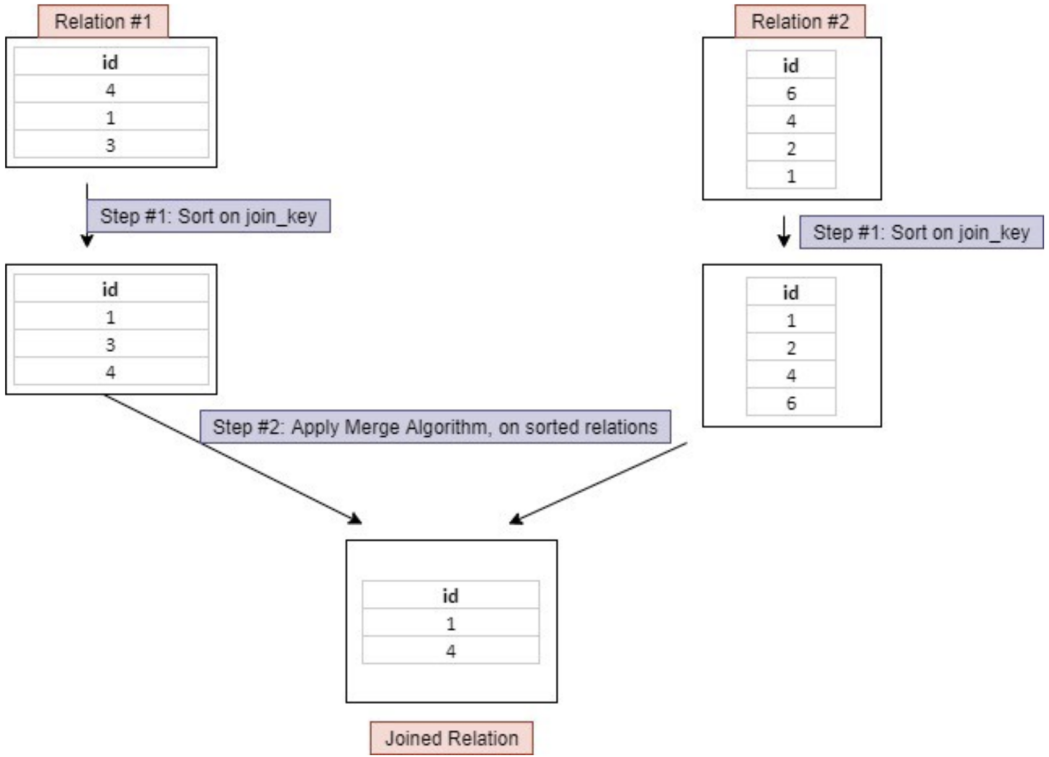


Broadcast Hash Join

Shuffle Hash Join

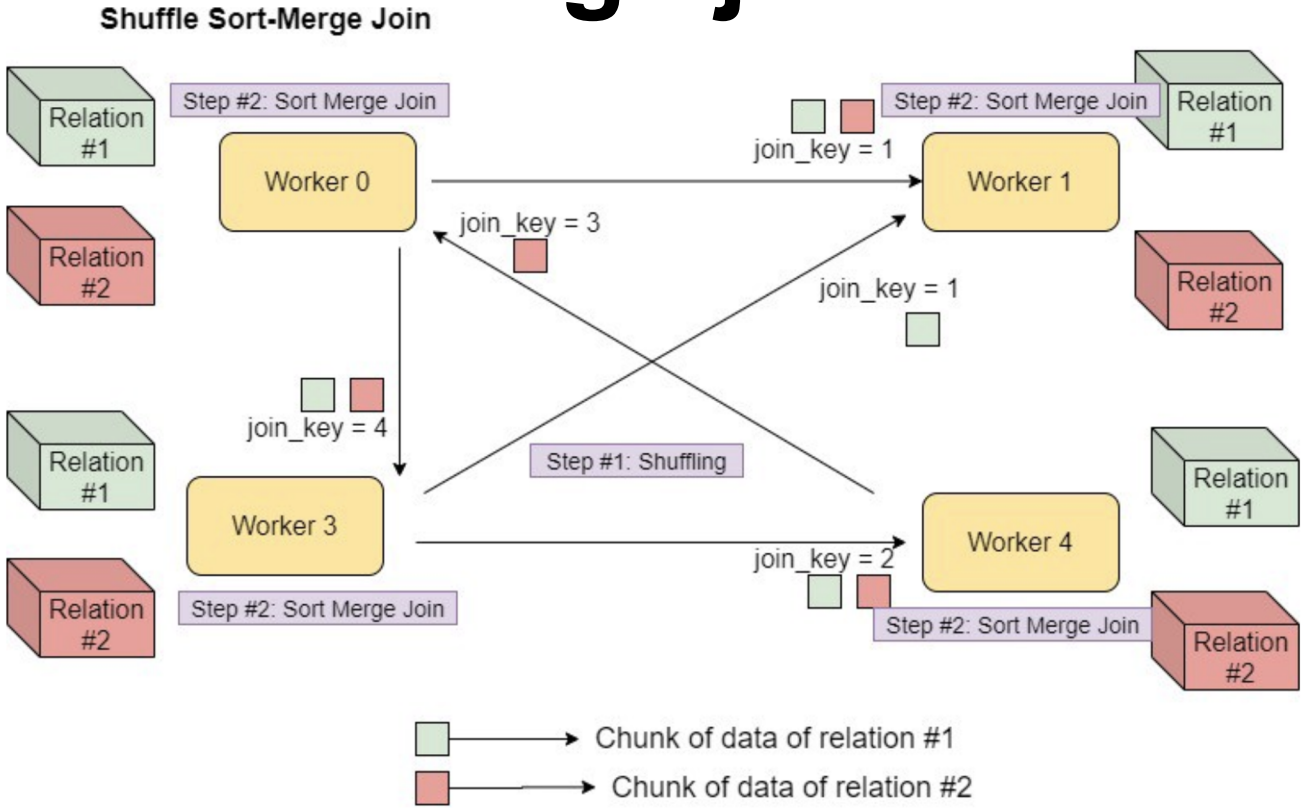


Sort merge Join



Sort Merge Join

Shuffle sort-merge join



Join Optimization

- SortMerge Join – Both sides are large
- 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

Bucketing in Spark

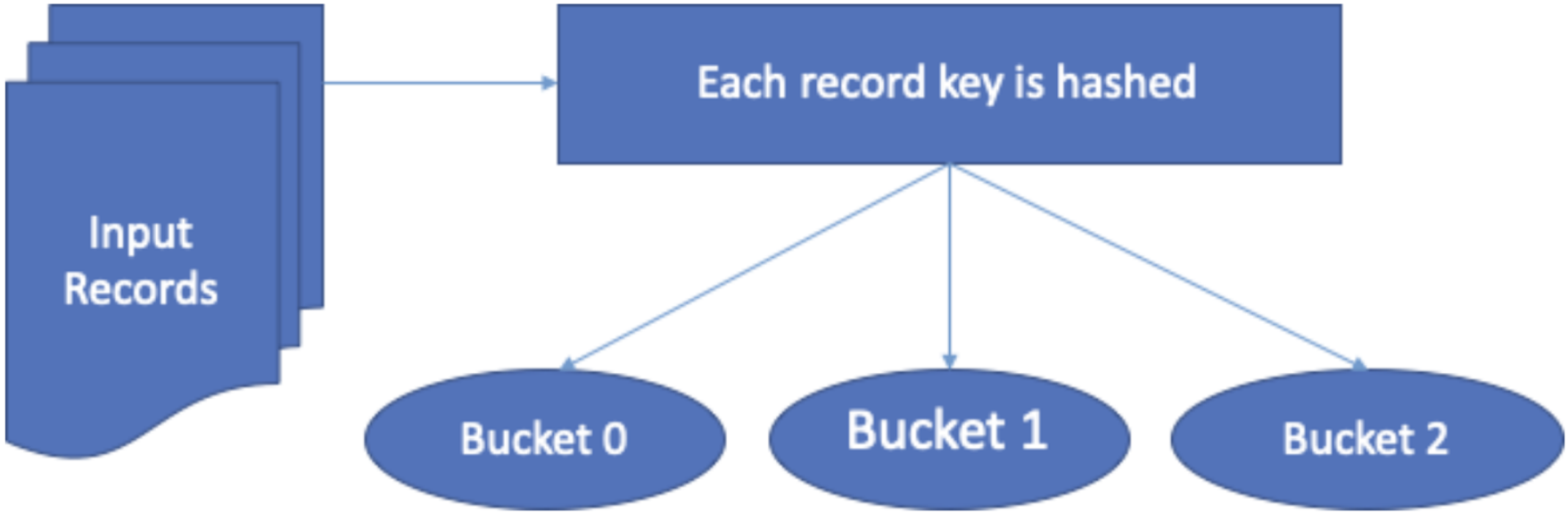
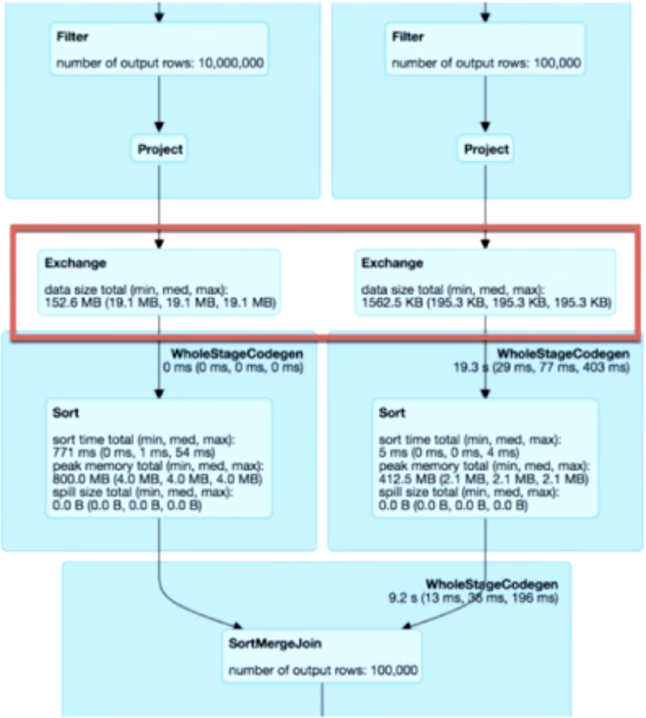


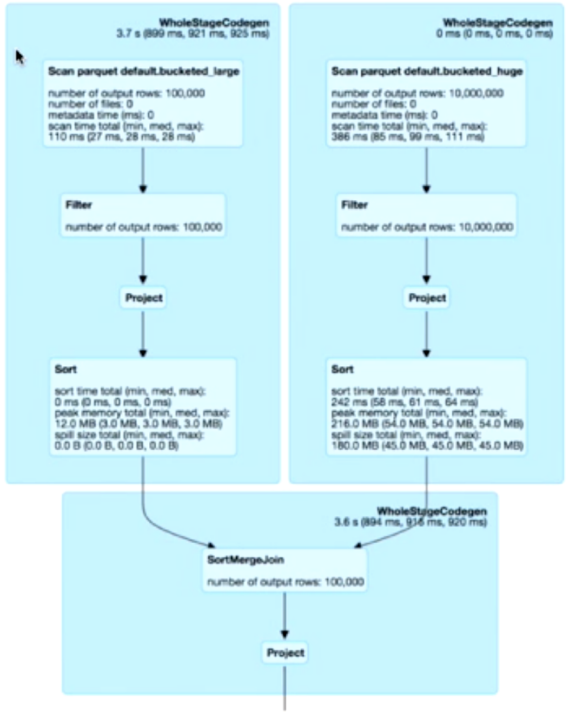
Figure 1.1

DATAFRAMEWRITER.BUCKETBY EXAMPLE

```
// Creating bucketed tables
import org.apache.spark.sql.SaveMode
large.write
  .bucketBy(4, "id")          // <-- bucketing
  .sortBy("id")               // <-- optional sorting
  .mode(SaveMode.Overwrite)
  .saveAsTable("bucketed_large")
huge.write
  .bucketBy(4, "id")          // <-- bucketing
  .sortBy("id")               // <-- optional sorting
  .mode(SaveMode.Overwrite)
  .saveAsTable("bucketed_huge")
```

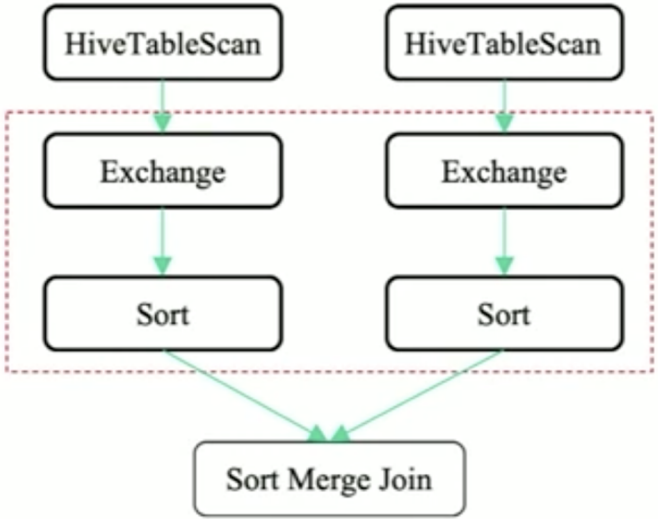
SORTMERGEJOINS (BEFORE AND AFTER)





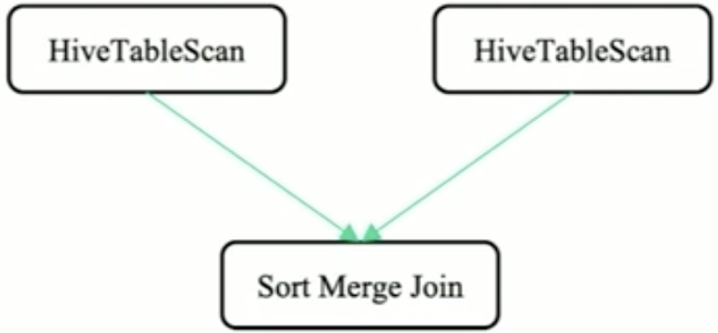
Bucketing Optimizations at ByteDance

outputPartitioning: UnknownPartitioning
outputOrdering: Nil



requireChildDistribution: HashClusteredDistribution(id, n, Murmur3Hash)
requireChildOrdering: SortOrder(id)

outputPartitioning: HashPartitioning(id, n, HiveHash)
outputOrdering: SortOrder(id)



requireChildDistribution: HashClusteredDistribution(id, n, HiveHash)
requireChildOrdering: SortOrder(id)

DESCRIBE EXTENDED

Use **DESCRIBE EXTENDED** SQL command to know whether a table is bucketed or not

```
val describeSQL = sql("DESCRIBE EXTENDED bucketed_large")
describeSQL.show(numRows = 50, truncate = false)
```

DESCRIBE EXTENDED

```
scala> describeSQL.show(numRows = 50, truncate = false)
+----------------------------+----------------------------------------------------------------------+-------+
|col_name                    |data_type                                                             |comment|
+----------------------------+----------------------------------------------------------------------+-------+
|id                          |bigint                                                                |null   |
|                            |                                                                      |       |
|# Detailed Table Information|                                                                      |       |
|Database                    |default                                                               |       |
|Table                       |bucketed_large                                                        |       |
|Owner                       |jacek                                                                 |       |
|Created Time                |Thu Oct 04 10:42:05 BST 2018                                          |       |
|Last Access                 |Thu Jan 01 01:00:00 GMT 1970                                          |       |
|Created By                  |Spark 2.3.2                                                           |       |
|Type                        |MANAGED                                                               |       |
|Provider                    |parquet                                                               |       |
|Num Buckets                 |4                                                                     |       |
|Bucket Columns              |[`id`]                                                                |       |
|Sort Columns                |[`id`]                                                                |       |
|Table Properties            |[transient_lastDdlTime=1538646125]                                    |       |
|Statistics                  |414404 bytes                                                          |       |
|Location                    |file:/Users/jacek/dev/oss/spark/spark-warehouse/bucketed_large        |       |
|Serde Library               |org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe                    |       |
|InputFormat                 |org.apache.hadoop.mapred.SequenceFileInputFormat                      |       |
|OutputFormat                |org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat             |       |
|Storage Properties          |[serialization.format=1]                                              |       |
+----------------------------+----------------------------------------------------------------------+-------+
```

Ссылки

https://www.youtube.com/watch?v=1j8SdS7s_NY

https://www.youtube.com/watch?v=7cvaH33S7uc

https://www.youtube.com/watch?v=daXEp4HmS-E

https://www.youtube.com/watch?v=dv7IIYuQOXI

https://youtu.be/99fYi2mopbs

catalyst-optimizer
deep-dive-into-spark-sqls-catalyst-optimizer
high-performance-spark/ch04.html
towardsdatascience.com/the-art-of-joining-in-spark
https://itnext.io/handling-data-skew-in-apache-spark-9f56343e58e8
https://blog.clairvoyantsoft.com/optimize-the-skew-in-spark-e523c6ee18ac