

# How to improve cube building and query performance

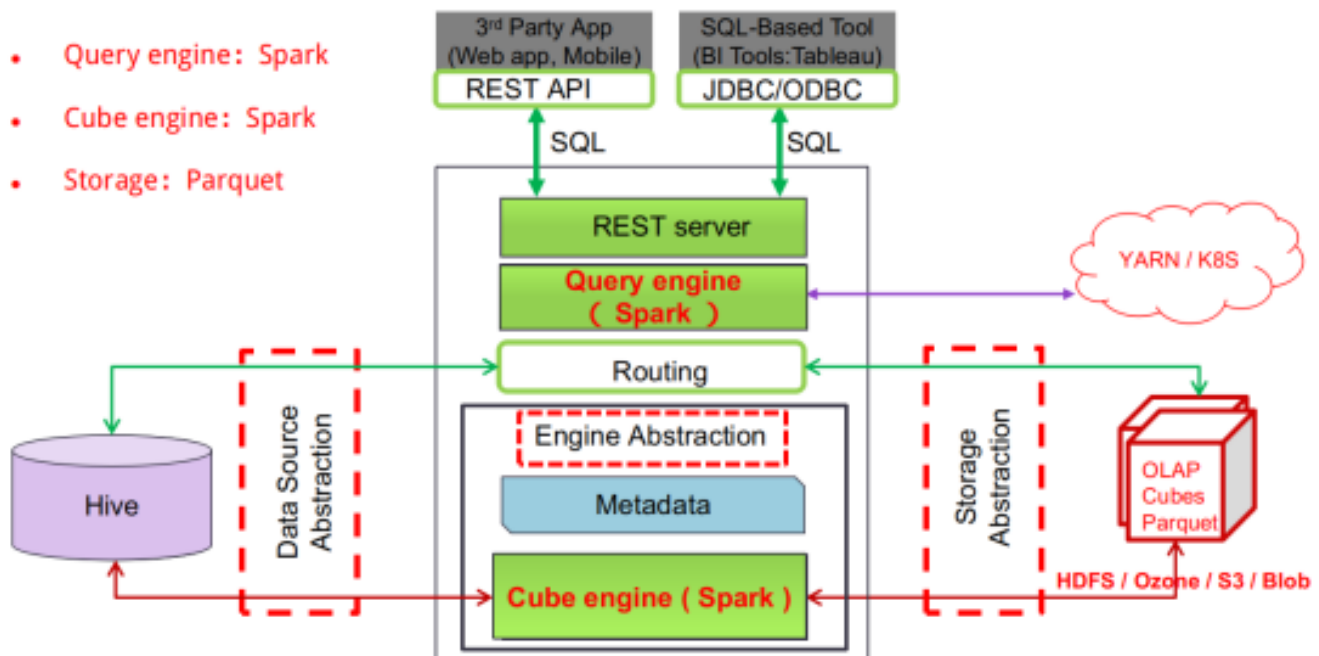
- [Background](#)
- [Cube building performance tuning](#)
  - [Use proper Spark resources and configurations to build cube data](#)
    - [Automatically setting spark configurations](#)
    - [Manually setting spark configurations \(if needed\)](#)
  - [Global dictionary building performance tuning](#)
  - [Snapshot tables building performance tuning](#)
- [Query performance tuning](#)
  - [Reduce small or uneven parquet files](#)
    - [The strategy to check whether needs to repartition](#)
    - [The relevant configurations](#)
    - [How to set the above configurations properly](#)
  - [Use shard by column to prune parquet files](#)
    - [How to use shard by column](#)
    - [Recommendations](#)
  - [Use sort by columns to filter data quickly when reading parquet files](#)
  - [Pack a number of small files into a single partition](#)
  - [Enable off-heap](#)
  - [Set different configurations for each query](#)
- [Reference](#)

## Release History

ID	Date	Author	Comment
1	2020-11	Zhichao Zhang	Tuning guide for 4.0.0-alpha,
2	2022-5	Shaofeng Shi	Update for 4.0.1

## Background

Kylin 4 is a [major architecture upgrade](#) version, as the picture shows below, both the cube building engine and query engine use spark as calculation engine, and cube data is stored in parquet files instead of HBase.



The build and query performance tuning is very different from Kylin 3 tuning([http://kylin.apache.org/docs/howto/howto\\_optimize\\_build.html](http://kylin.apache.org/docs/howto/howto_optimize_build.html)). This article will introduce how to improve cube build and query performance in Kylin 4, including some tuning ways which will be made by Kylin 4 automatically.

# Cube building performance tuning

In Kylin 4, there are two steps in the cube building job:

1. The first step is detecting how many source files will be built as cube data;
2. The second one is to build the snapshot tables (if needed), generate the global dictionary (if needed) and build cube data as parquet files.

In the second step, all calculations are operations with a relatively heavy load, so except using "Joint" or "Hierarchy" on dimensions to reduce the number of cuboids (refers to the section '**Reduce combinations**' in [http://kylin.apache.org/docs/tutorial/cube\\_build\\_performance.html](http://kylin.apache.org/docs/tutorial/cube_build_performance.html)), it's also very important to use proper spark resources and configurations to build cube data. There are **3 key points** in this section to improve cube building performance.

## Use proper Spark resources and configurations to build cube data

Assume your Spark application runs on YARN, the relevant configurations are as below:

Key	Description
spark.executor.instances	The number of executors for spark application.
spark.executor.cores	The number of cores to use on each executor. The value of 'spark.executor.instances' * 'spark.executor.cores' is the maximum parallelism when running the cube building job.
spark.executor.memory	Amount of memory to use per executor process. Generally speaking, the ratio of core to memory is 1:4, for example, if you set 'spark.executor.cores' to 4, and then set 'spark.executor.memory' to 16G.
spark.executor.memoryOverhead	The amount of off-heap memory to be allocated per executor. This is the memory that accounts for things like VM overheads, interned strings, other native overheads, etc. This tends to grow with the executor size (typically 6-10%).
spark.sql.shuffle.partitions	Configures the number of partitions to use when shuffling data for joins or aggregations, the default value is 200. A larger value requires more CPU resources, while a smaller value requires more memory resources.
spark.sql.files.maxPartitionBytes	The maximum number of bytes to pack into a single partition when reading files, the default value is 128M. If there are many small files in source tables (Hive source), the spark will automatically pack a number of small files into a single partition to avoid too many small tasks.

## Automatically setting spark configurations

You can set these configurations with a 'kylin.engine.spark-conf.' prefix in 'kylin.properties' file; for example: 'kylin.engine.spark-conf.spark.executor.instances'. Then Kylin 4 will use them to allocate spark resources for the cube building job.

Similar to the tuning in spark + parquet, you may find out some problems through the Spark UI and change some configurations to improve performance, there are many articles describing how to improve the performance in spark + parquet, such as <http://spark.apache.org/docs/2.4.6/sql-performance-tuning.html> and <http://spark.apache.org/docs/2.4.6/tuning.html>.

If you don't know how to set these configurations properly, Kylin 4 will use the below allocation rules to automatically set spark resources and configurations, all spark resources and configurations are set according to **the maximum file size of source files** and **whether the cube has accurate count distinct measure**, this is the reason why we need to detect how many source files which will be built in the first step. You can see these allocation rules in the class 'SparkConfHelper':

- **ExecutorMemoryRule**

If  $\{the\ maximum\ file\ size\} \geq 100G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memory' to 20G;

If  $\{the\ maximum\ file\ size\} \geq 100G$  or  $\{the\ maximum\ file\ size\} \geq 10G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memory' to 16G;

If  $\{the\ maximum\ file\ size\} \geq 10G$  or  $\{the\ maximum\ file\ size\} \geq 1G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memory' to 10G;

If  $\{the\ maximum\ file\ size\} \geq 1G$  or  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memory' to 4G;

Otherwise set 'spark.executor.memory' to 1G.

- **ExecutorCoreRule**

If  $\{the\ maximum\ file\ size\} \geq 1G$  or  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.cores' to 5;

Otherwise set 'spark.executor.cores' to 1.

- **ExecutorOverheadRule**

If  $\{the\ maximum\ file\ size\} \geq 100G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memoryOverhead' to 6G, so in this case, the memory of per executor is  $20G + 6G = 26G$ ;

If  $\{the\ maximum\ file\ size\} \geq 100G$  or  $\{the\ maximum\ file\ size\} \geq 10G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memoryOverhead' to 4G;

If  $\{the\ maximum\ file\ size\} \geq 10G$  or  $\{the\ maximum\ file\ size\} \geq 1G$  and  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memoryOverhead' to 2G;

If  $\{the\ maximum\ file\ size\} \geq 1G$  or  $\{exist\ accurate\ count\ distinct\}$ , then set 'spark.executor.memoryOverhead' to 1G;

Otherwise set 'spark.executor.memoryOverhead' to 512M.

- **ExecutorInstancesRule**

The steps to set 'spark.executor.instances' are as follows:

1. Get the value of required cores, the default value is 1;
2. Get the value of configuration 'kylin.engine.base-executor-instance' as basic executor instances, default value is 5;
3. According to the number of the cuboids, calculate the required number of executor instances:  $\{calculateExecutorInsByCuboidSize\}$ . The configuration of the calculation strategy is 'kylin.engine.executor-instance-strategy', default value is '100,2,500,3,1000,4', which means if the number of the cuboids is greater and equal than 100, the factor is 2, and then the number of executor instances is  $\{basic\ executor\ instances\} * \{factor\} = 10$ , if greater and equal than 500, the factor is 3, and so on.
4. Get the available memory and cores count of the default pool from yarn:  $\{availableMem\}$  and  $\{availableCore\}$ ;
5. Get the sum memory value after applying 'ExecutorOverheadRule' and 'ExecutorMemoryRule':  $\{executorMem\} = \{spark.executor.memory\} + \{spark.executor.memoryOverhead\}$ ;
6. Get the cores count after applying 'ExecutorCoreRule':  $\{executorCore\}$ ;
7. According to  $\{availableMem\}$ ,  $\{availableCore\}$ ,  $\{executorCore\}$  and  $\{executorMem\}$ , calculate the maximum executor instances count which can request from yarn:  $\{queueAvailableInstance\} = \text{Math.min}(\{availableMem\} / \{executorMem\}, \{availableCore\} / \{executorCore\})$ ; The purpose of this step is to avoid applying for more than the available resources on yarn.
8. Get the final executor instances count:  $\{executorInstance\} = \text{Math.max}(\text{Math.min}(\{calculateExecutorInsByCuboidSize\}, \{queueAvailableInstance\}), \{kylin.engine.base-executor-instance\})$ ;
9. Set 'spark.executor.instances' to  $\{executorInstance\}$ ;

- **ShufflePartitionsRule**

1. Set 'spark.sql.shuffle.partitions' to the value of  $\text{max}(2, \{the\ maximum\ file\ size\ in\ MB\} / 32)$ ;

After applying all rules above, you can find some log messages in 'kylin.log' file as below:

```
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.executor.memory = 4GB.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: count distinct = true.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.executor.cores = 5.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.executor.memoryOverhead = 1GB.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.executor.instances = 5.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.yarn.queue = default.
[Scheduler 1683494526 Job b89bc963-feb9-401a-be0a-2970987892ae-394] job.NSparkExecu
utils.SparkConfHelper:103 : Auto set spark conf: spark.sql.shuffle.partitions = 2.
```

## Manually setting spark configurations (if needed)

Based on the values of automatically adjusted configurations by Kylin, if there are still some cube building performance issues, you can appropriately change the values of these configurations to have a try, for example:

- If you observe from the Spark UI that there is a **serious GC phenomenon** in some tasks, or find a large number of executor lost or fetch failure errors, you can change the value of these two configurations to increase the memory per executor:
  - spark.executor.memory=
  - spark.executor.memoryOverhead=

The general adjustment strategy is to increase the value by 2 times. If the problem is solved, you can decrease it appropriately to avoid wasting resources. After increasing the memory per executor, if there is still a serious memory problem, you can consider adjusting 'spark.executor.cores' to 1, this adjustment can make a single task exclusive memory per executor and the execution efficiency is relatively low, but it can be done in this way to avoid build failure.

- If you observe from the Spark UI that there are a large number of tasks that need to be scheduled for multiple rounds (each round eats all cores), you can change the value of these two configurations to increase the cores count of the spark application:
  - spark.executor.cores=
  - spark.executor.instances=

The general adjustment strategy is to increase the value by 2 times. If the problem is solved, you can decrease it appropriately to avoid wasting resources.

- If there are some **executor lost or fetch failure errors**, and just because the number of reducers during shuffling is too small, or the data is skewed, you can try to increase the value of '[spark.sql.shuffle.partitions](#)'.
- If you observe from the Spark UI that the duration time of the job is more than the sum duration time of stages, this means that the core resources are not enough and there are many jobs are waiting for core resources to be scheduled:

111	build 139103 from parent 147295 <a href="#">parquet at ParquetStorage.scala:28</a>	2020/11/04 06:50:12	3.0 min	2/2	<div><div>806/806</div></div>
-----	---	---------------------	---------	-----	-------------------------------

The duration time of this job is 3.0 min, but the sum duration time of stages is 17s + 2s = 19s, the stage 204 waited more than 2.0 min to be scheduled.

- Completed Stages (2)								
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
204	build 139103 from parent 147295 <a href="#">parquet at ParquetStorage.scala:28</a>	2020/11/04 06:52:56	17 s	<div><div>806/806</div></div>		25.5 MB	75.6 MB	
203	build 139103 from parent 147295 <a href="#">parquet at ParquetStorage.scala:28</a>	2020/11/04 06:50:12	2 s	<div><div>2/2</div></div>	21.0 MB			75.6 MB

In this case, you need to increase the cores count of the spark application.

## Global dictionary building performance tuning

If the cube has accurate "count distinct" measures, Kylin 4.0 will build the global dictionary for these measure columns in the second step based on Spark for distributed encoding processing, which reduces the pressure on a single machine node, and can break the limit of the maximum integer of the global dictionary, please refer to the detail design article: <https://cwiki.apache.org/confluence/display/KYLIN/Global+Dictionary+on+Spark> . There is one configuration about tuning on global dictionary building:

[kylin.dictionary.globalV2-threshold-bucket-size](#) (default value is 500000)

Reducing the value of this configuration can reduce the amount of data in a single partition to build the global dictionary and speed up the dictionary building.

## Snapshot tables building performance tuning

If there are some snapshot tables to be built, Kylin 4.0 will build them parallelly in the second step, because the default value of the configuration '[kylin.snapshot.parallel-build-enabled](#)' is true, which will speed up the snapshot tables building.

On the other hand, you can reduce the value of configuration '[kylin.snapshot.shard-size-mb](#)' (default value is 128MB) to increase the parallelism when building snapshot tables. According to the size of the source table, make sure the number of the building tasks is within 3 times the number of cores of the spark cube building application.

## Query performance tuning

In Kylin 4.0, the query engine (called **SparderContext**) uses spark as the calculation engine too, it's a real distributed query engine, especially for complex queries, the performance will be better than Apache Calcite. However, there are still many key performance points that need to be optimized.

In addition to setting proper calculation resources mentioned above, it also includes reducing small or uneven files, setting proper partitions, and pruning parquet files as many as possible. Kylin 4.0 and Spark provide some optimization strategies to improve query performance.

## Reduce small or uneven parquet files

Reading too many small files or a few too big files when querying will lead to low performance, in order to avoid this problem, Kylin 4.0 will repartition parquet files according to the following strategy to reduce small or uneven parquet files when building cube data as parquet files.

### The strategy to check whether needs to repartition

If the following conditions are met

1. If this cuboid has shard by column;
2. The average size of parquet files which have saved < the value of configuration '[kylin.storage.columnar.repartition-threshold-size-mb](#)' && the number of parquet files is bigger than 1; This condition is to avoid too many small files;

3. The number of parquet files < (the total row count of parquet files / 'kylin.storage.columnar.shard-rowcount' \* 0.75), if this cuboid has accurate count distinct measure, use 'kylin.storage.columnar.shard-countdistinct-rowcount' instead of 'kylin.storage.columnar.shard-rowcount'; This condition is to avoid uneven files;

If meet the one of the conditions above, it will do repartition, the number of the partitions is calculated by this way:

$\text{\$fileLengthRepartitionNum} = \text{Math.ceil}(\text{\$(the parquet files size in MB)} / \text{\$(kylin.storage.columnar.shard-size-mb)})$

$\text{\$rowCountRepartitionNum} = \text{Math.ceil}(\text{\$(the total row count of parquet files)} / \text{\$(kylin.storage.columnar.shard-rowcount)})$

If this cuboid has accurate count distinct measure, use 'kylin.storage.columnar.shard-countdistinct-rowcount' instead of 'kylin.storage.columnar.shard-rowcount'.

The number of the partitions is :

$\text{Math.ceil}((\text{\$fileLengthRepartitionNum} + \text{\$rowCountRepartitionNum}) / 2)$

## The relevant configurations

Key	Default value	Description
kylin.storage.columnar.shard-size-mb	128MB	The max size of each parquet file for shard by column, in MB.
kylin.storage.columnar.shard-rowcount	2500000	Each parquet files should contain at most 2.5 million rows.
kylin.storage.columnar.shard-countdistinct-rowcount	1000000	Since that Bitmap has a bigger size, so we can specify the max row count for cuboids containing Bitmap. By default, it contains at most 1.0 million rows.
kylin.storage.columnar.repartition-threshold-size-mb	128MB	The max size of each parquet file, in MB.

## How to set the above configurations properly

You can use this command to find the repartition info messages in the kylin.log file after building cube data:



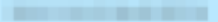

```
grep "Before repartition, cuboid" logs/kylin.log
```

```
[root@c3be274dee41 apache-kylin-4.0.0-alpha-bin-hadoop2]# grep "Before repartition, cuboid" logs/kylin.log
883 INFO [thread-build-cuboid-262143] utils.Repartitioner:123 : Before repartition, cuboid[262143] has 806999 row, 33046123 bytes and 2 files.
Partition count calculated by file size is 4, calculated by row count is 1614, final is 809.
115 INFO [thread-build-cuboid-147455] utils.Repartitioner:123 : Before repartition, cuboid[147455] has 806637 row, 22507487 bytes and 2 files.
Partition count calculated by file size is 3, calculated by row count is 1614, final is 809.
992 INFO [thread-build-cuboid-139263] utils.Repartitioner:123 : Before repartition, cuboid[139263] has 806605 row, 22495509 bytes and 2 files.
Partition count calculated by file size is 3, calculated by row count is 1614, final is 809.
968 INFO [thread-build-cuboid-147375] utils.Repartitioner:123 : Before repartition, cuboid[147375] has 804576 row, 22008173 bytes and 2 files.
Partition count calculated by file size is 3, calculated by row count is 1610, final is 807.
512 INFO [thread-build-cuboid-147295] utils.Repartitioner:123 : Before repartition, cuboid[147295] has 804545 row, 22057959 bytes and 2 files.
Partition count calculated by file size is 3, calculated by row count is 1610, final is 807.
551 INFO [thread-build-cuboid-147443] utils.Repartitioner:123 : Before repartition, cuboid[147443] has 805109 row, 22317027 bytes and 2 files.
Partition count calculated by file size is 3, calculated by row count is 1611, final is 807.
```

According to the log messages, you can find that the final number of partitions is too large, this will impact the building performance and query performance, after increasing the value of configuration 'kylin.storage.columnar.shard-rowcount' or 'kylin.storage.columnar.shard-countdistinct-rowcount' and rebuilding again, the log messages are shown below:

```
045 INFO [thread-build-cuboid-262143] utils.Repartitioner:123 : Before repartition, cuboid[262143] has 806999 row, 33046123 bytes and 2 files
Partition count calculated by file size is 4, calculated by row count is 1, final is 3.
594 INFO [thread-build-cuboid-147455] utils.Repartitioner:123 : Before repartition, cuboid[147455] has 806637 row, 22507487 bytes and 2 files
Partition count calculated by file size is 3, calculated by row count is 1, final is 2.
393 INFO [thread-build-cuboid-147443] utils.Repartitioner:123 : Before repartition, cuboid[147443] has 805109 row, 22317027 bytes and 2 files
Partition count calculated by file size is 3, calculated by row count is 1, final is 2.
091 INFO [thread-build-cuboid-147295] utils.Repartitioner:123 : Before repartition, cuboid[147295] has 804545 row, 22057959 bytes and 2 files
Partition count calculated by file size is 3, calculated by row count is 1, final is 2.
680 INFO [thread-build-cuboid-147375] utils.Repartitioner:123 : Before repartition, cuboid[147375] has 804576 row, 22008173 bytes and 2 files
Partition count calculated by file size is 3, calculated by row count is 1, final is 2.
```

The final number of partitions was reduced a lot: 809 to 3, and the time of cube building was reduced a lot too: 58 mins to 24 mins:

BUILD CUBE - kylin_sales_cube_large_shard_rowcount - 20120101000000_20120301000000 - GMT+08:00	kylin_sales_cube_large_shard_rowcount	100%		24.58 mins	Action ▾	
BUILD CUBE - kylin_sales_cube_small_shard_rowcount - 20120101000000_20120301000000 - GMT+08:00	kylin_sales_cube_small_shard_rowcount	100%		57.90 mins	Action ▾	

And the query performance is improved too:

```

Duration: 1.699
Project: learn_kylin
Realization Names: [CUBE[name=kylin_sales_cube_small_shard_rowcount]]
Cuboid Ids: [262143]
Total scan count: 206573
Total scan files: 58
Total metadata time: 0ms
Total spark scan time: -1ms
Total scan bytes: 40883846
Result row count: 11
Storage cache used: false
Is Query Push-Down: false
Is Prepare: false
Used Spark pool: lightweight_tasks
Trace URL: null
Message: null

```

The query time from the cube which has a too large number of partitions is 1.7s, and the query engine scanned 58 files.

```

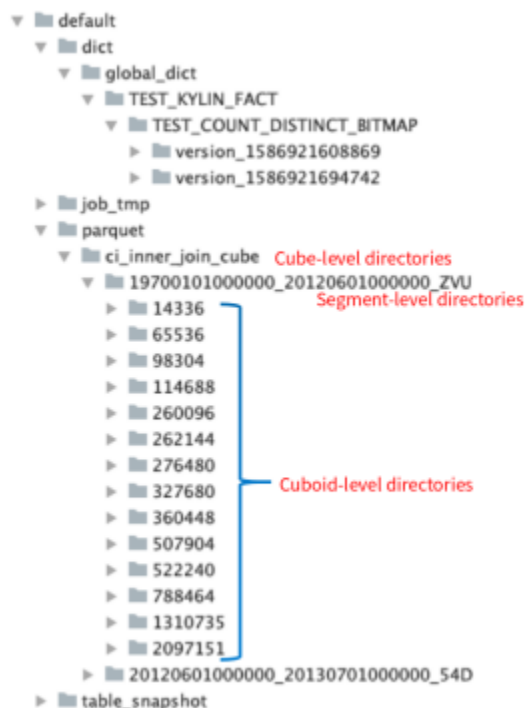
Duration: 0.414
Project: learn_kylin
Realization Names: [CUBE[name=kylin_sales_cube_large_shard_rowcount]]
Cuboid Ids: [262143]
Total scan count: 807730
Total scan files: 4
Total metadata time: 0ms
Total spark scan time: 201ms
Total scan bytes: 33291100
Result row count: 11
Storage cache used: false
Is Query Push-Down: false
Is Prepare: false
Used Spark pool: lightweight_tasks
Trace URL: null
Message: null

```

But the query time from a cube that has a proper number of partitions is 0.4s, and the query engine only scanned 4 files.

## Use shard by column to prune parquet files

In Kylin 4.0, the directory structure of parquet file storage is as follows:



When querying, the query engine can filter out the segment-level directories through **the date partition column**, and filter out the cuboid-level directories through **the hit cuboid**, but at this time, if there are still many parquet files in the cuboid-level directories, you can use **shard by column** to further prune parquet files.

## How to use shard by column

From Cube Designer Advanced Setting Rowkeys in Kylin UI, you can specify a shard by column when creating a cube:

New Aggregation Group+

### Rowkeys ⓘ

**Important:** Dimension's position(Rowkey) has impact on query performance. You can drag and drop to adjust the sequence. In short, put filtering dimension before non-filtering dimension, and put high cardinality dimension before low cardinality dimension.

ID	Column	Shard By
1	KYLIN_SALES.PART_DT	true
2	KYLIN_SALES.BUYER_ID	false
3	KYLIN_SALES.SELLER_ID	false
4	KYLIN_SALES.TRANS_ID	false
5	KYLIN_SALES.LEAF_CATEG_ID	false
6	KYLIN_CATEGORY_GROUPINGS.META_CATEG_NAME	false
7	KYLIN_CATEGORY_GROUPINGS.CATEG_LVL2_NAME	false
8	KYLIN_CATEGORY_GROUPINGS.CATEG_LVL3_NAME	false
9	BUYER_ACCOUNT.ACCOUNT_BUYER_LEVEL	false

After specifying a shard by column, it will repartition parquet files by this column when building cube data (If you do not specify, repartition is done with all columns).

When querying with this shard by column as a filter condition, the query engine will prune parquet files according to the value of shard by column, for example:



```
[root@ec3be274dee41 admin]# hdfs dfs -ls /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_non_shardby/20120101000000_20120301000000_M6F/131084/
20/11/04 01:57:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 1 root supergroup 5062608 2020-11-04 01:12 /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_non_shardby/20120101000000_20120301000000_M6F/131084/part-00000-1f7a3a0a-a0fc-436a-b573-556a57eddeaa-c000.snappy.parquet
-rw-r--r-- 1 root supergroup 5386207 2020-11-04 01:12 /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_non_shardby/20120101000000_20120301000000_M6F/131084/part-00001-1f7a3a0a-a0fc-436a-b573-556a57eddeaa-c000.snappy.parquet
[root@ec3be274dee41 admin]#
[root@ec3be274dee41 admin]#
[root@ec3be274dee41 admin]#
[root@ec3be274dee41 admin]# hdfs dfs -ls /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_shardby/20120101000000_20120301000000_EY2/131084/
20/11/04 01:57:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 1 root supergroup 4504472 2020-11-04 01:41 /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_shardby/20120101000000_20120301000000_EY2/131084/part-00000-d65811dd-d0fe-4977-ba8d-531907b8cf86-c000.snappy.parquet
-rw-r--r-- 1 root supergroup 5730090 2020-11-04 01:41 /kylin4_metadata/kylin4/learn_kylin/parquet/kylin_sales_cube_shardby/20120101000000_20120301000000_EY2/131084/part-00001-d65811dd-d0fe-4977-ba8d-531907b8cf86-c000.snappy.parquet
```

There are two parquet files in each of the cuboid 131084 directories of these two cubes: `kylin_sales_cube_non_shardby` and `kylin_sales_cube_shardby`;

```
Project: learn_kylin
Realization Names: [CUBE[name=kylin_sales_cube_non_shardby]]
Cuboid Ids: [131084]
Total scan count: 2360
Total scan files: 2
Total metadata time: 0ms
Total spark scan time: 50ms
Total scan bytes: 10448815
Result row count: 1
Storage cache used: false
Is Query Push-Down: false
```

Querying from cube '`kylin_sales_cube_non_shardby`' which doesn't specify shard by column is scanning 2 files.

```
Project: learn_kylin
Realization Names: [CUBE[name=kylin_sales_cube_shardby]]
Cuboid Ids: [131084]
Total scan count: 1040
Total scan files: 1
Total metadata time: 0ms
Total spark scan time: 14ms
Total scan bytes: 4504472
Result row count: 1
Storage cache used: false
Is Query Push-Down: false
```

Querying from cube '`kylin_sales_cube_shardby`' which specifies shard by column is only scanning 1 file.

## Recommendations

- Currently, it only supports the following filtering operations with shard by column in SQL query to prune parquet files:
  - Equality
  - In
  - InSet
  - IsNull
- Because it only supports specifying **one** shard by column for one cube currently, it's better to use a column that has high cardinality and often is used as a filter condition, such as a mandatory dimension column. If the specified shard by column is not a mandatory dimension, there are some cases where the cuboid cannot use this shard by column; for example, the specified shard by column is A, but the columns of one cuboid are B, C, D.

## Use sort by columns to filter data quickly when reading parquet files



When you create a cube, you can specify the order of the dimension columns, and when saving cube data, the first of the dimension columns for each cuboid will be used to do the sort operation. The purpose is to filter out unwanted data as much as possible through the min-max index of the parquet file when querying with the sort by column.

From Cube Designer Advanced Setting Rowkeys in Kylin UI, you can drag the columns to adjust the order:

### Rowkeys

**Important:** Dimension's position(Rowkey) has impact on query performance. You can drag and drop to adjust the sequence. In short, put filtering dimension before non-filtering dimension, put high cardinality dimension before low cardinality dimension.

ID	Column	Shard By
1	KYLIN_SALES.SELLER_ID	false
2	KYLIN_SALES.PART_DT	true
3	KYLIN_SALES.BUYER_ID	false
4	KYLIN_SALES.TRANS_ID	false
5	KYLIN_SALES.LEAF_CATEG_ID	false
6	KYLIN_CATEGORY_GROUPINGS.META_CATEG_NAME	false
7	KYLIN_CATEGORY_GROUPINGS.CATEG_LVL2_NAME	false
8	KYLIN_CATEGORY_GROUPINGS.CATEG_LVL3_NAME	false
9	BUYER_ACCOUNT.ACCOUNT_BUYER_LEVEL	false
10	SELLER_ACCOUNT.ACCOUNT_SELLER_LEVEL	false
11	BUYER_ACCOUNT.ACCOUNT_COUNTRY	false

For example: if the cuboid includes these three columns: BUYER\_ID, TRANS\_ID, LEAF\_CATEG\_ID, then it will sort data in one partition by the BUYER\_ID column when saving this cuboid data.

**Notes:** Currently Apache Spark 2.4.6 which Kylin 4.0 used only supports filtering out unwanted data through the min-max index of RowGroup in parquet files, which means that if there are some RowGroups in one parquet file, Spark will filter out unwanted data by the min-max index of RowGroup, but if one parquet file only includes one RowGroup, the filter doesn't take effect.

## Pack a number of small files into a single partition

When there are many small files in some segments which had been built, you can set the configuration '[spark.sql.files.maxPartitionBytes](#)' (default value is 128MB) to a proper value, which will let the spark engine pack some small files into a single partition and avoid to need too many small tasks, for example:

```
Project: learn_kylin
Realization Names: [CUBE[name=kylin_sales_cube_non_shardby]]
Cuboid Ids: [131084]
Total scan count: 2360
Total scan files: 2
Total metadata time: 0ms
Total spark scan time: 749ms
Total scan bytes: 10448815
Result row count: 59
Storage cache used: false
Is Query Push-Down: false
```

This query scanned 2 parquet files but it used one task to handle these two files:

0	lightweight_tasks	sparder collect at ResultPlan.scala:108	+details	2020/11/04 13:56:23	2 s	1/1	3 MB		440.5 KB
---	-------------------	--	----------	---------------------	-----	-----	------	--	----------

On the other hand, if there are enough resources, you can reduce the value of configuration 'spark.sql.files.maxPartitionBytes' to increase the parallel tasks, but it also needs to reduce the value of configuration 'spark.hadoop.parquet.block.size' (default value is 128MB) when building cube data, because the smallest split unit of parquet files is RowGroup and configuration 'spark.hadoop.parquet.block.size' indicates the maximum size of one RowGroup for parquet.

### Enable off-heap

Spark can directly operate the off-heap memory to reduce unnecessary memory overhead, as well as frequent GC, and improve processing performance.

Relevant configurations:

Key	Description
spark.memory.offHeap.enabled	Set to 'true', use off-heap memory for spark shuffle e.g.
spark.memory.offHeap.size	indicates the size of off-heap memory.

### Set different configurations for each query

Currently, all queries share one Spark Session, which means that all of them share the same configurations, but each query has different scenarios and could be optimized by different configurations. Therefore, we plan to clone a thread-level SparkSession for each query to set different configurations, and then execute the query, such as configuration 'spark.sql.shuffle.partitions', set this configuration to different values according to the amount of data obtained by each query to achieve the optimal query performance.

### Reference

- [http://kylin.apache.org/docs/tutorial/cube\\_build\\_job.html](http://kylin.apache.org/docs/tutorial/cube_build_job.html)
- [http://kylin.apache.org/docs/howto/howto\\_optimize\\_build.html](http://kylin.apache.org/docs/howto/howto_optimize_build.html)
- <http://spark.apache.org/docs/2.4.6/tuning.html>
- <https://cwiki.apache.org/confluence/display/KYLIN/Global+Dictionary+on+Spark>