Apache CarbonData技术原理及使用介绍

2018-09 蔡强

内容大纲

- 1. 项目背景
- 2. CarbonData 技术原理
- 3. CarbonData 使用介绍
- 4. What’s coming in 1.5

企业中包含多种数据应用，从商业智能、批处理到机器学习

Report & Dashboard

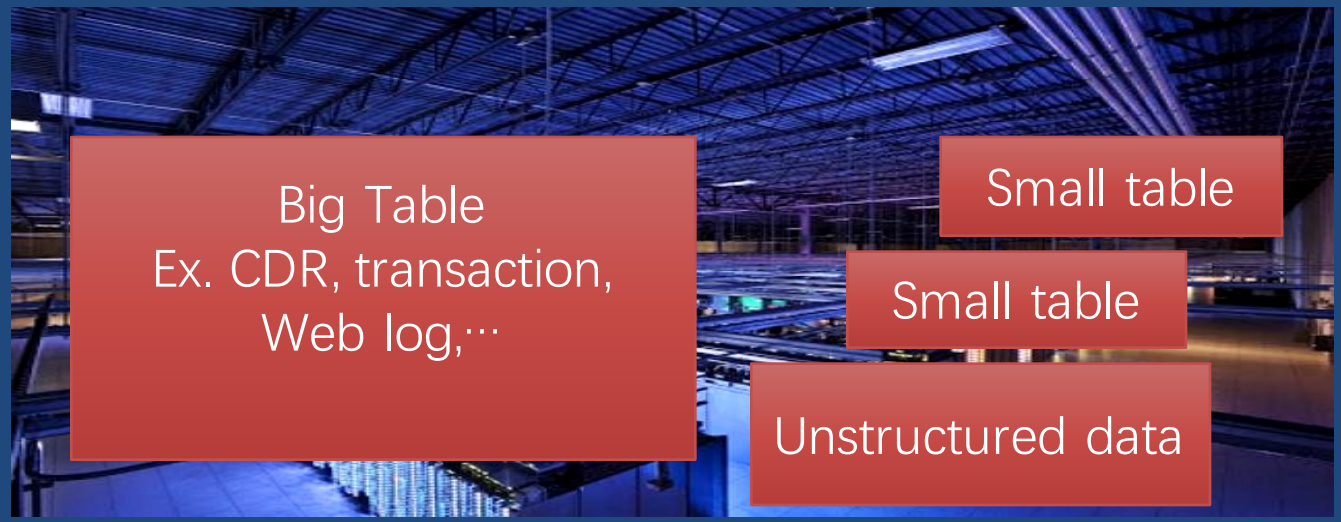
OLAP & Ad-hoc

Batch processing

Machine learning

Realtime Analytics

data



来自数据的挑战

- Data Size 数据规模
  - Single Table >10 B　单表大于100亿行
  - Fast growing　快速增长
  - Nested data structure for complex object　数据结构复杂

- Multi-dimensional 数据维度多
  - Every record > 100 dimensions　分析的维度超过100
  - Add new dimension occasionally　维度不断增长
  - Billion level high cardinality　不同值范围在亿级别

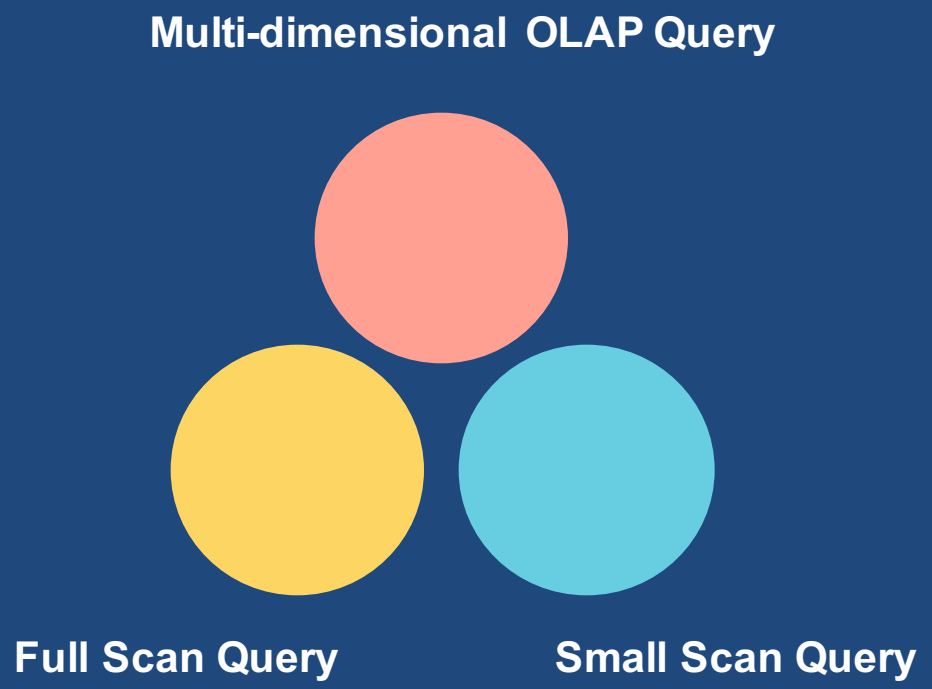
来自应用的挑战

- Enterprise Integration
  - SQL 2003 Standard Syntax
  - BI integration, JDBC/ODBC

企业应用集成

- Flexible Query
  - Any combination of dimensions
  - OLAP Vs Detail Record
  - Full scan Vs Small scan
  - Precise search & Fuzzy search

灵活查询
无固定模式



How to choose storage?

如何构建大数据统一存储平台？

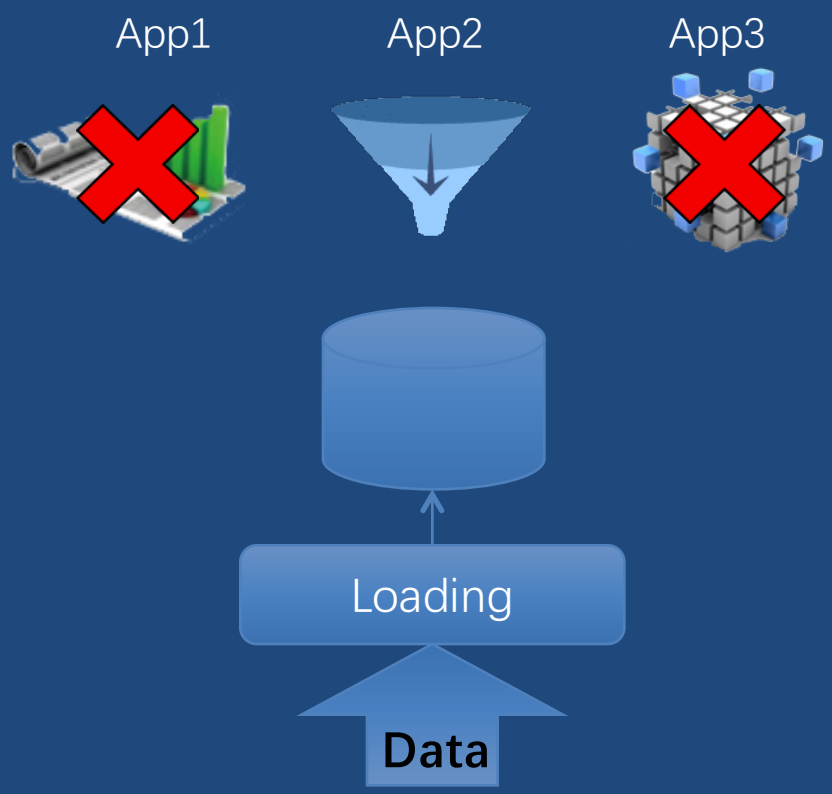
How to choose storage 当前各种大数据方案分析

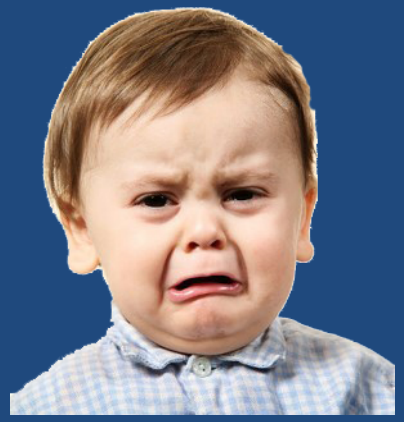
1. NoSQL Data(适合实时应用，不适合分析型应用)
   - 只支持单列key value查询 <5ms
   - 不支持标准SQL
2. MPP Data(适合中小规模数据分析)
   - Shared-nothing架构，并行计算
   - 不支持大集群 <100节点，没有容错，扩展能力有上限，不能与大数据生态集成
3. Cube Data(适合BI类MOLAP应用)
   - 预聚合，查询快
   - 但数据膨胀大，支持维度少，不支持查明细数据
4. Search Engine Data(适合文本类分析)
   - 通过索引快速找到数据
   - 数据膨胀大2-4倍，不支持标准SQL
5. SQL on Hadoop
   - 聚焦计算引擎的分布式扫描
   - 存储效率不高

架构师的苦恼：不同应用不同数据存储，如果让数据存储统一？

Choice 1: Compromising

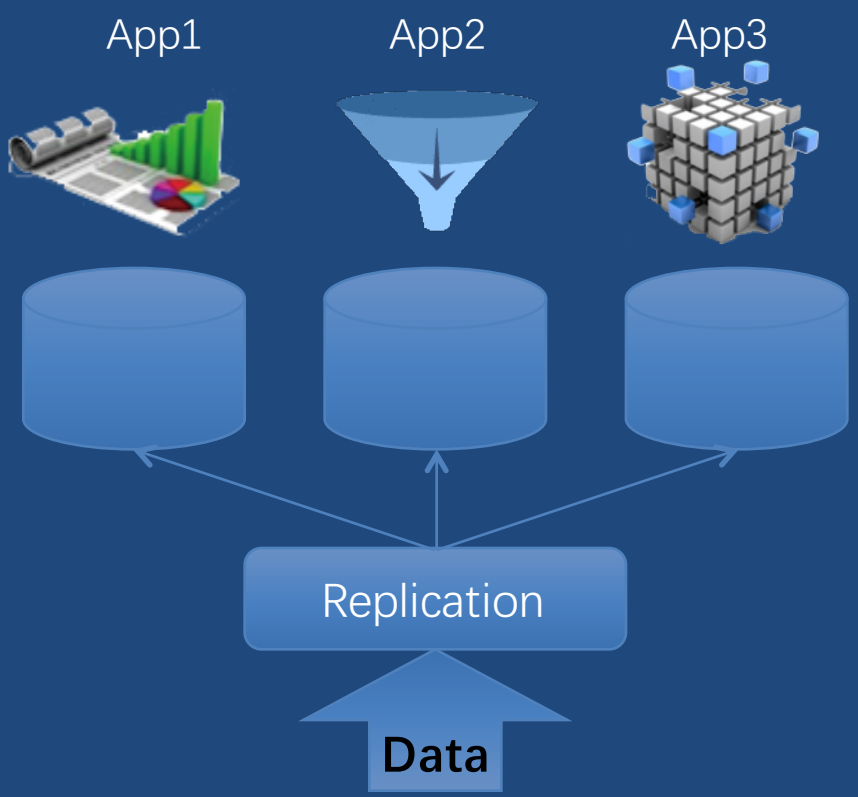
做出妥协，只满足部分应用





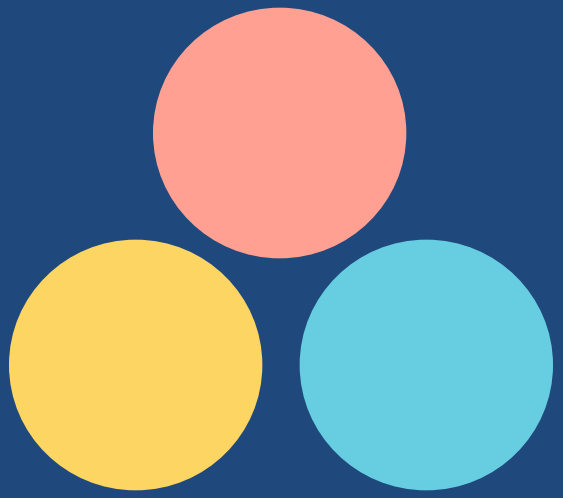
Choice 2: Replicating of data

复制多份数据，满足所有应用



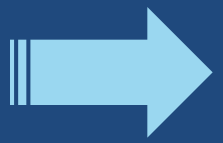
CarbonData目标：
一份数据满足多种业务需求，与大数据生态无缝集成

Multi-dimensional OLAP Query

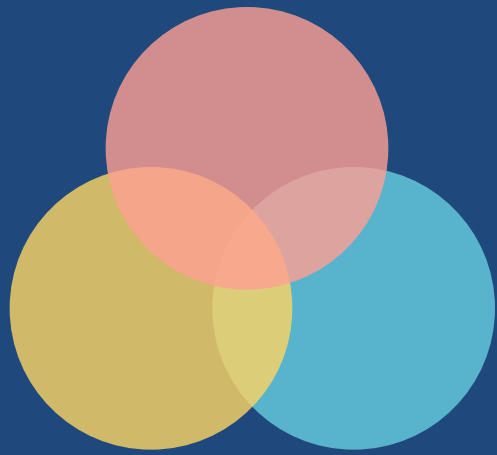


Full Scan Query

Small Scan Query



CarbonData: Unified Storage



一份数据满足多种分析场景
详单过滤，海量数仓，数据集市，…

内容大纲

- 2. CarbonData 技术原理
  - CarbonData File 文件格式
  - CarbonData Table
  - DataMap 介绍

CarbonData设计思路

- **数据统一存储**：一份数据支持多种业务场景，减少数据孤岛和冗余，通过数据共享产生更大价值。
- **大集群**：区别于以往的单机系统，用户希望新的大数据存储方案能应对日益增多的数据，随时可以通过增加资源的方式横向扩展，无极扩容。
- **易集成**：提供标准接口，新的大数据方案与企业已采购的工具和IT系统要能无缝集成，支撑老业务快速迁移。与大数据生态软件能无缝集成。
- **高性能**：数据分析要求越来越高效、实时。
- **开放生态**：通过开源开放，让更多的客户和合作伙伴的数据连接在一起，发挥更大的价值，与当前大数据生态无缝集成。

CarbonData File文件格式

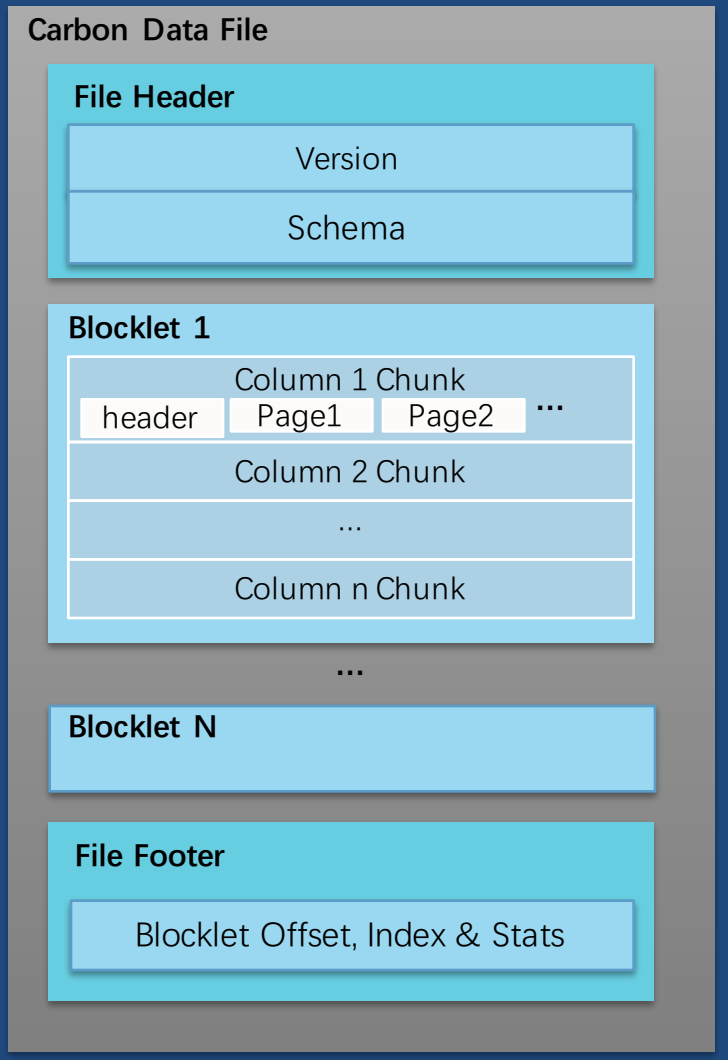
包含索引的列式文件格式

CarbonData File文件格式

- 数据布局
  - Block：一个HDFS文件
  - Blocklet：文件内的列存数据块，是最小的IO读取单元
    - Column chunk: Blocklet内的列数据
    - Page：Column chunk内的数据页，是最小的解码单元

- 元数据信息
  - Header：Version，Schema
  - Footer： Blocklet Offset, Index & 文件级统计信息

- 内置索引和统计信息
  - Blocklet索引：B Tree start key, end key
  - Blocklet级和Page级统计信息：min，max等



CarbonData File

- 压缩编码
  - RLE, 本地字典编码, 全局字典编码
  - 自适应编码
  - Snappy, Zstd

- 数据类型

  SMALLINT, INT/INTEGER, BIGINT, DOUBLE, DECIMAL
  TIMESTAMP, DATE
  STRING, CHAR, VARCHAR
  ARRAY, STRUCT, MAP
  BOOLEAN, BINARY

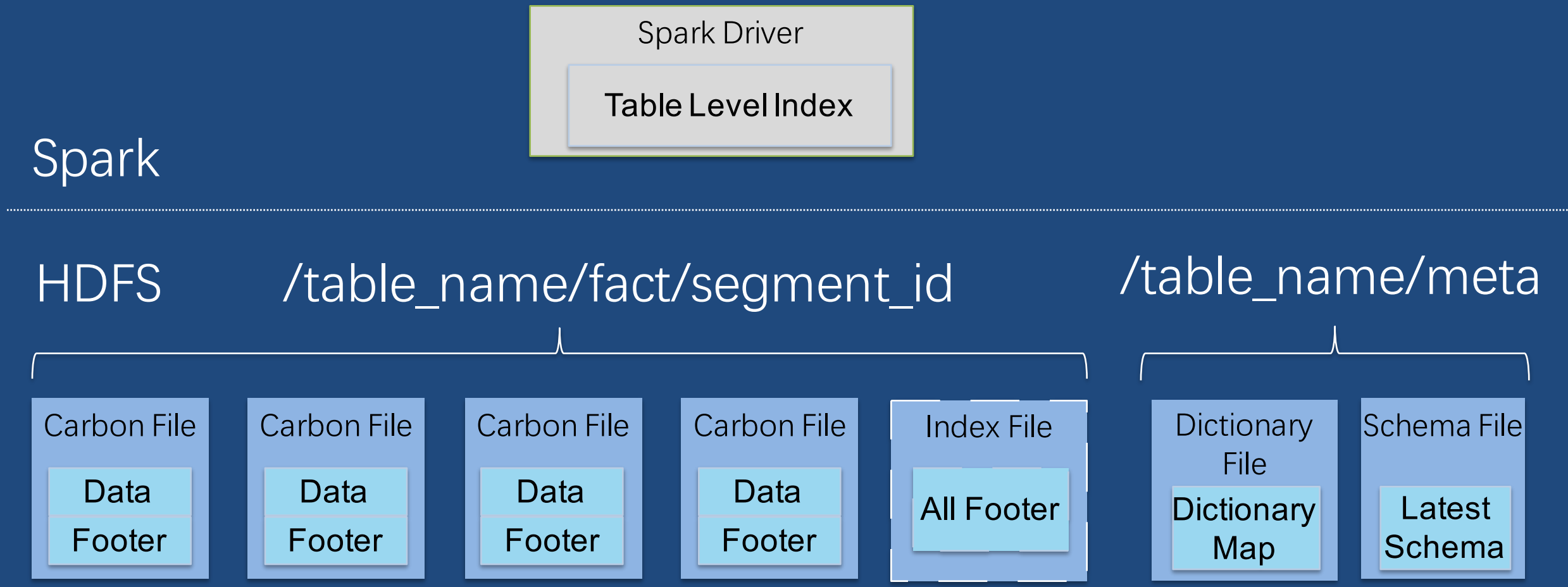
CarbonData Table

支持索引的、对CarbonData File的原子化读写操作

CarbonData Table

- 支持Segment级的读写操作的数据一致性和原子性。
- 一次Load/Insert对应生成一个Segment
- 每个Segment 包含数据和元数据: CarbonData File和Index文件
- 不同的Segment可以有不同的文件格式
  - 流式入库的Streaming Segment采用了写优化的文件格式
  - 历史数据的Batch Segment采用读优化的文件格式
  - 将支持更多其他格式，例如: CSV，Parquet

CarbonData Table Layout

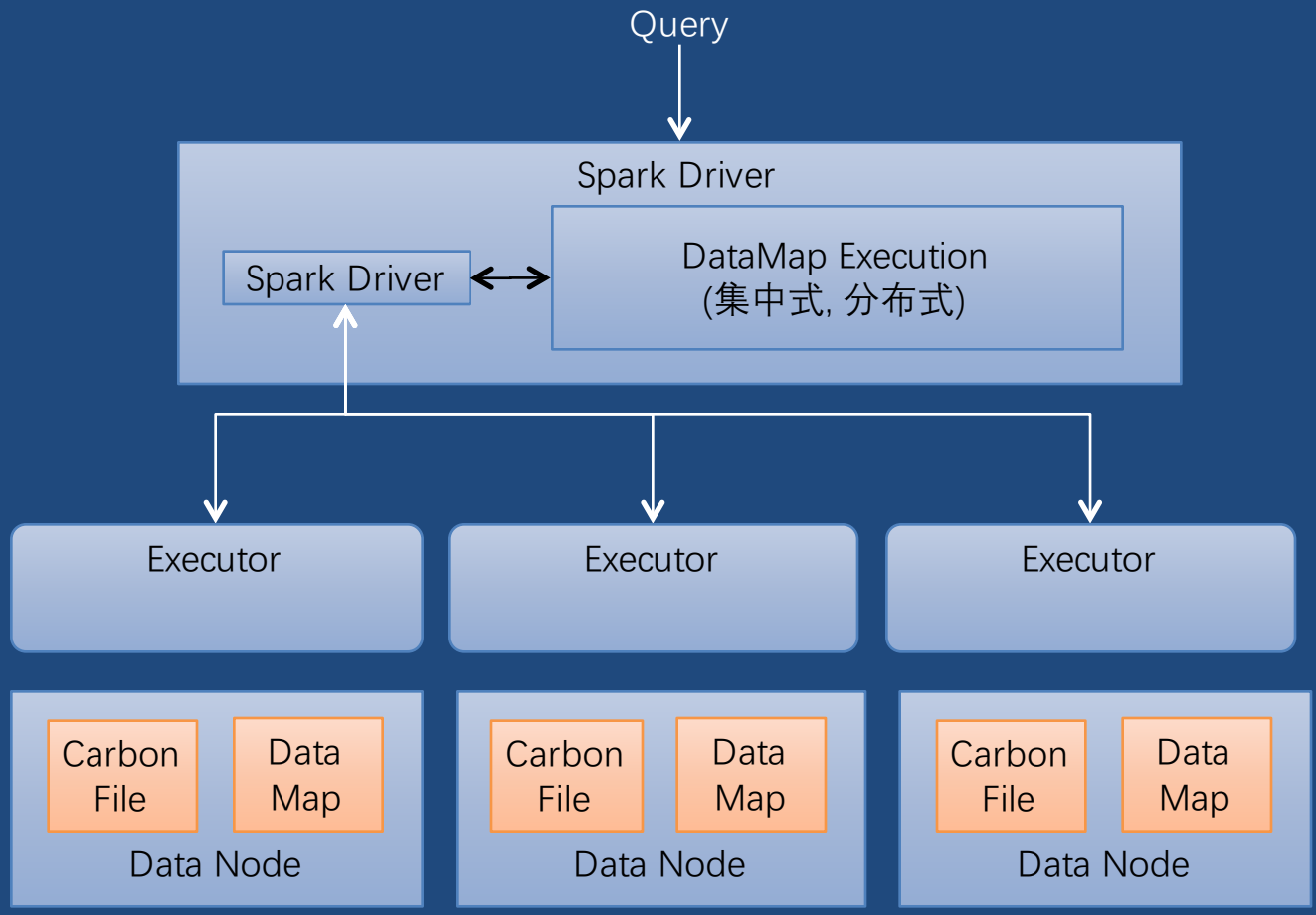


CarbonData DataMap

数据地图

DataMap执行模型

- 下推filter和projection到DataMap
- DataMap可以是集中式或者分布式的（避免大内存问题）
- DataMap可以缓存在内存或者存储在磁盘上
- 主要包括Index DataMap和MV DataMap



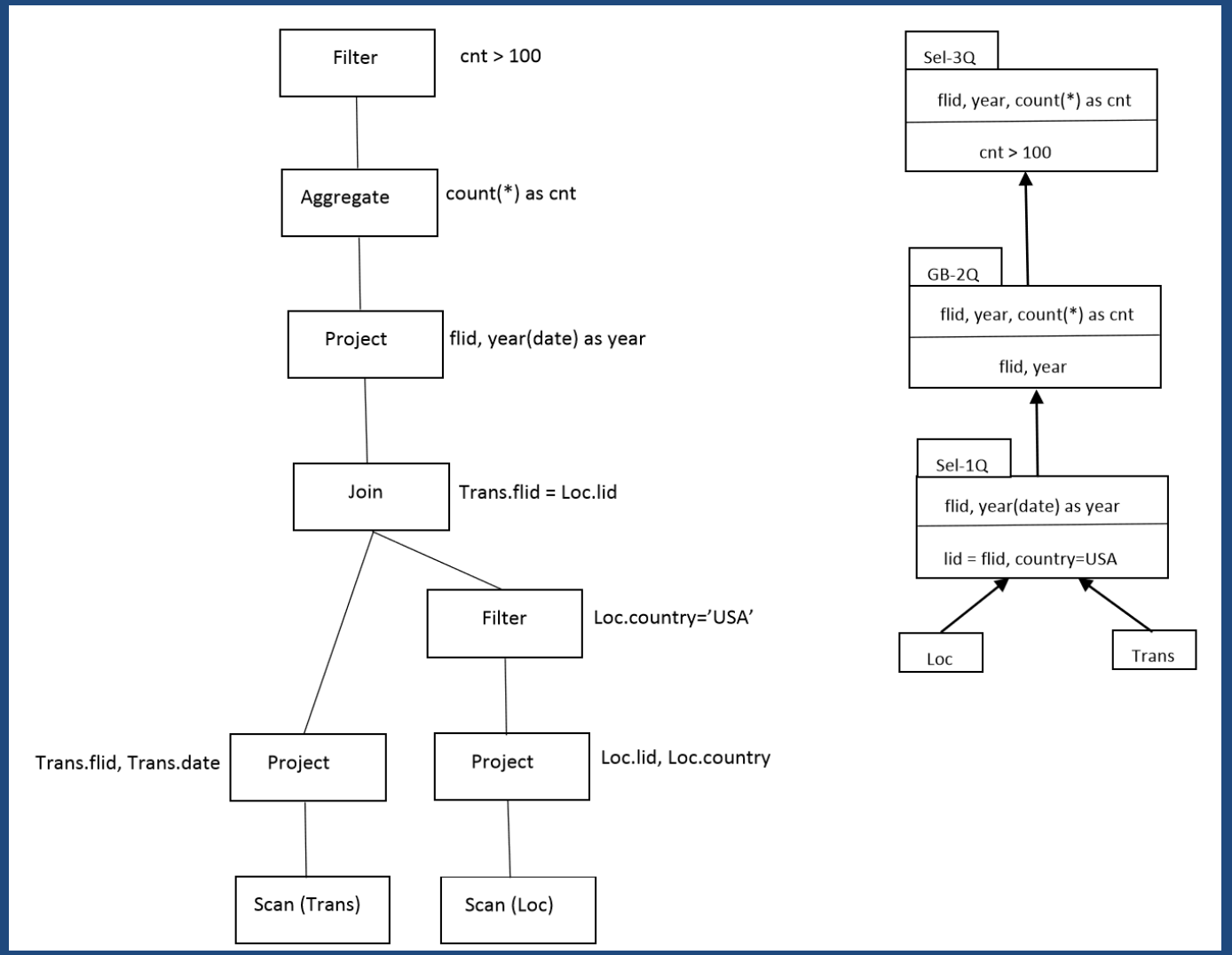
Index DataMap

- 主索引
  - B-Tree
  - MinMax

- 二级索引
  - B-Tree：适合一般OLAP多维分析
  - BloomFilter：适合id类字段过滤，如手机号，终端ID，车牌等

MV DataMap

- 物化视图可以进行预汇聚、Join等操作，用数据入库时间换取查询时间
- 入库方式：
  - 立即入库：入库主表时自动入库MV
  - 延迟入库：用户手动操作
- 查询时会将查询语句转换为Modular Plan的查询计划，将原SQL改写为针对MV的SQL
  - 当命中多个MV时，自动选取代价最小的MV
- 当前支持projection, filter, groupby, join, having等语法



内容大纲

- 3. CarbonData 使用介绍
  - CarbonData File 读写
  - CarbonData Table 操作
  - DataMap 使用
  - 流式入库介绍

CarbonData File读写

使用CarbonData SDK读写CarbonData文件

CarbonData File 读写

```
CarbonWriter writer = CarbonWriter.builder()
                .outputPath(path)
                .buildWriterForCSVInput(new Schema(fields));
writer.write(row);
writer.close;

CarbonReader reader = CarbonReader
                .builder(path, "_temp")
                .projection(strings)
                .build();
reader.readNextRow();
reader.close();
```

CarbonData Table操作

与SparkSQL深度集成

新建CarbonData表

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type , ...)]
  STORED AS carbondata
  [TBLPROPERTIES (property_name=property_value, ...)]
  [LOCATION 'path']
```

*如何正确配置*Table Properties，提升查询性能？

配置Table Properties

- SORT_COLUMNS
  - 默认按所有维度列在schema中的顺序对数据做row排序
  - 详单过滤：频繁使用的过滤条件列，越靠前的列过滤效果越好
  - OLAP分析：查询使用的列，按基数由低到高排序，提升压缩率，降IO
- COLUMN_META_CACHE
  - 默认会缓存所有列的Min/Max值
  - 建议：查询中使用的过滤条件列，可以减少索引对内存的使用
- CACHE_LEVEL
  - Block：默认值，占用较少的内存，降低部分driver侧过滤性能
  - Blocklet：更加精准的driver侧过滤，占用较多的内存

配置Table Properties

- DICTIONARY_INCLUDE
  - 低基数String列，提升压缩率
  - 延迟解码，可以提升Group by效率
- LOCAL_DICTIONARY
  - Blocklet级别的字典编码，可以节省存储空间，提升查询速度
- SORT_SCOPE
  - NO_SORT, BATCH_SORT, LOCAL_SORT, GLOBAL_SORT

数据导入

使用SQL

```
LOAD DATA [LOCAL] INPATH 'folder path' [OVERWRITE]
INTO TABLE tablename OPTIONS(...)

INSERT INTO TABLE tablennme  select ... FROM table1
```

使用Dataframe

```
df.write
   .format(“carbondata")
   .options("tableName“, “t1"))
   .mode(SaveMode.Overwrite)
   .save()
```

- 符合事务的原子性，一致性
- 支持多并发数据导入
- 支持并发导入和查询

查询数据

使用SQL

```
SELECT project_list FROM t1
WHERE cond_list
GROUP BY columns
ORDER BY columns
```

使用Dataframe

```
df = sparkSession.read
        .format(“carbondata”)
        .option(“tableName”, “t1”)
        .load(“path_to_carbon_file”)

df.select(…).show
```

- Same SQL syntax and DataFrame API as in SparkSQL
- 支持并发导入，更新，合并和查询。

数据更新/删除

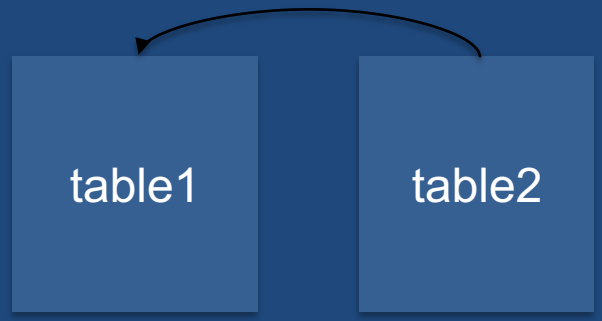
修改表1的一列

```
UPDATE table1 A
SET （A.REVENUE） = （A.REVENUE – 10）
WHERE A.PRODUCT = ‘phone’
```

phone, ~~70~~ 60
car,100
phone, ~~30~~ 20

使用表2的数据修改表1的两列

```
UPDATE table1 A
SET (A.PRODUCT, A.REVENUE) = (
    SELECT PRODUCT, REVENUE
            FROM table2 B
    WHERE B.CITY = A.CITY AND B.BROKER = A.BROKER)
WHERE A.DATE BETWEEN ‘2017-01-01’ AND ‘2017-01-31’
```



删除表1的部分记录

```
DELETE FROM table1 A
WHERE A.CUSTOMERID = ‘123’
```

~~123,~~
~~abc~~
456, jkd

Segment管理

- 展示t1表的segment列表

```
SHOW SEGMENTS FOR TABLE t1 LIMIT 100
```

- 按segment id 删除指定segment

```
DELETE FROM TABLE t1 WHERE SEGMENT.ID IN (98, 97, 30)
```

- 按segment加载时间删除segment

```
DELETE FROM TABLE t1 WHERE SEGMENT.STARTTIME BEFORE '2017-06-01 12:05:06'
```

- 指定需要查询的segment

```
SET carbon.input.segments.default.t1 = 100, 99
```

数据合并

Minor合并

```
ALTER TABLE table_name COMPACT 'MINOR'
```

Major合并

```
ALTER TABLE table_name COMPACT MAJOR'
```

自定义

```
ALTER TABLE table_name COMPACT 'CUSTOM' WHERE SEGMENT.ID IN (2,3,4)
```

合并后清除SEGMENTS文件及目录

```
CLEAN FILES FOR TABLE carbon_table
```

分区表

新建分区表

```
CREATE TABLE sale(id string, quantity int ...)
    PARTITIONED BY (country  string, state string)
    STORED AS carbondata
```

静态分区加载数据

```
LOAD DATA LOCAL INPATH ‘folder path’ INTO TABLE sale PARTITION (country = 'US', state = 'CA')

INSERT INTO TABLE sale PARTITION (country = 'US', state = 'AL')
SELECT <columns list excluding partition columns> FROM another_sale
```

动态分区加载数据

```
LOAD DATA LOCAL INPATH ‘folder path’ INTO TABLE

INSERT INTO TABLE sale
SELECT <columns list including partition columns> FROM another_sale
```

DataMap使用

加速CarbonData Table查询

新建DataMap

```
CREATE DATAMAP [IF NOT EXISTS] datamap_name
  [ON TABLE main_table]
  USING "datamap_provider“ [WITH DEFERRED REBUILD]
DMPROPERTIES ('key'='value', ...)
  AS SELECT statement
```

- bloomfilter
- preaggregate
- mv

启用DataMap

```
SET carbon.datamap.visible.dbName.tableName.dataMapName = true
```

禁用DataMap

```
SET carbon.datamap.visible.dbName.tableName.dataMapName = false
```

BloomFilter DataMap

```
CREATE DATAMAP [IF NOT EXISTS] datamap_name
   ON TABLE main_table
   USING 'bloomfilter'
DMPROPERTIES (
  'INDEX_COLUMNS'='city, name',
  'BLOOM_SIZE'='640000',
  'BLOOM_FPP'='0.00001',
  'BLOOM_COMPRESS'='true')
```

- Index_columns: 索引的列
- Bloom_size: 位数组大小
- Bloom_fpp: 期望的误判率
- Bloom_compress: 是否压缩索引

Pre-aggregate DataMap

```
CREATE DATAMAP agg_sales  ON TABLE sales
  USING "preaggregate"
AS
  SELECT country, sex, sum(quantity), avg(price)
  FROM sales
  GROUP BY country, sex
```

- 支持的聚合函数：
  SUM，AVG，MAX，MIN，COUNT
- 只支持单表的聚合操作

MV DataMap

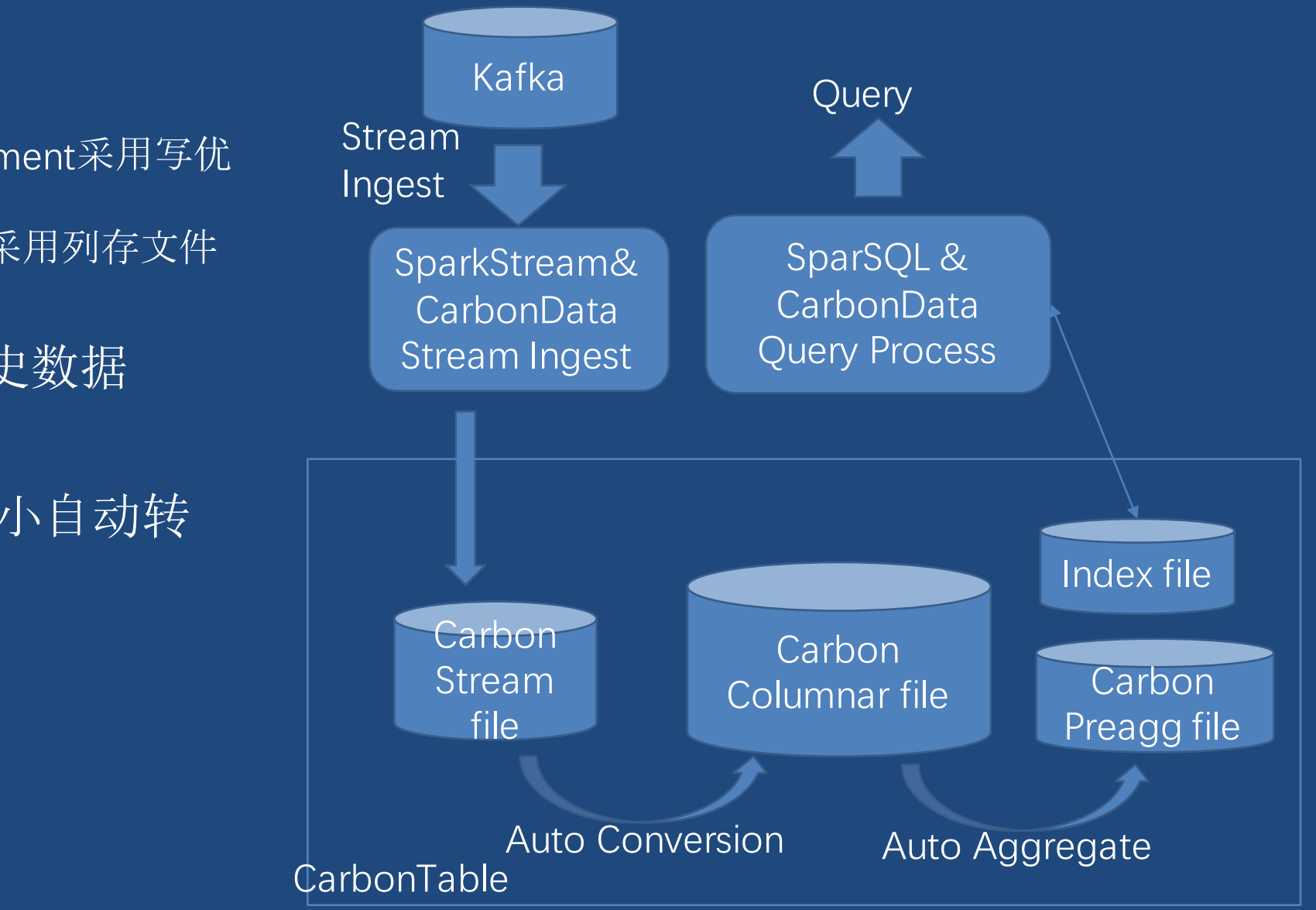
```
CREATE DATAMAP simple_agg_with_join
  USING 'mv'
AS
  SELECT
    id,address, sum(age)
 FROM mainTable INNER JOIN dimtable
 ON mainTable.name=dimtable.name
 GROUP BY id ,address
```

Streaming Ingestion

流式入库，准实时查询

流式入库

- 混合格式
  - 近期数据：streaming segment采用写优化的文件格式
  - 历史数据: batch segment采用列存文件
- 可以查询近期数据和历史数据
- 按streaming segment大小自动转换为batch segment
- 支持预聚合



Stream SQL

- 新建kafka source表

```
CREATE TABLE source (sensor_Id string, …)
STOREAD AS 'carbondata'
TBLPROPERTIES (
  'streaming'='source',
  'format'='kafka',
  'kafka.bootstrap.servers'='host1:port1,…',
  'subscribe'='topic1')
```

- 新建carbon sink表

```
CREATE TABLE sink (sensor_Id string, …)
STOREAD AS 'carbondata'
TBLPROPERTIES (streaming='sink')
```

- 新建流式入库作业

```
CREATE STREAM stream ON TABLE sink
STMPROPERTIES (
"trigger'='ProcessingTime',
'interval'='10 seconds')
AS
    SELECT * FROM source
    WHERE temperature > 30.0
```

- 停止作业

```
DROP STREAM stream
```

总结：CarbonData的能力

- 数据管理：
- 增量入库，建立索引：**由用户权衡**入库时间、索引粒度和查询性能
- 流式入库：可与Kafka, Spark, HBase集成，实现准实时分析
- 批量更新：支持**快速更新**事实表或维表，闲时做数据合并
- 查询：
- 通过DataMap索引，减少IO：**适合ad-hoc查询**，任意维度组合查询场景
- 向量化处理，提升扫描性能：适合全表扫描、**汇总分析**场景
- 基于**物化视图**和CBO的查询优化：BI分析类场景加速
- 大规模：
- 计算与存储分离：支持从GB到PB大规模数据，**十万亿数据秒级响应**
- 部署：
- 与大数据生态无缝集成，**利用云存储和Hadoop集群**

APACHE CarbonData

- Apache Incubator Project since June, 2016
- Apache releases
  - >10 stable releases
  - Coming release: **1.5.0, 2018-09**
- Contributors:

Compute

Storage

4. What's coming in 1.5

- 1.5.0
  - Support Spark File Format
  - Improve performance on S3
  - Spark 2.3 and Hadoop 3.1
  - Complex datatype Enhancement
  - Stream format support min/max index
- 1.5.1
  - OLAP on demand
  - MV
    - Incremental load
    - DataMap choosing
  - Presto enhancement

欢迎参与Apache CarbonData社区

- website: http://carbondata.apache.org
- Code: https://github.com/apache/carbondata
- JIRA: https://issues.apache.org/jira/browse/CARBONDATA
- Mail list: dev@carbondata.apache.org, user@carbondata.apache.org

- 欢迎在Maillist上提问，共同探讨和开发CarbonData 2.0新特性