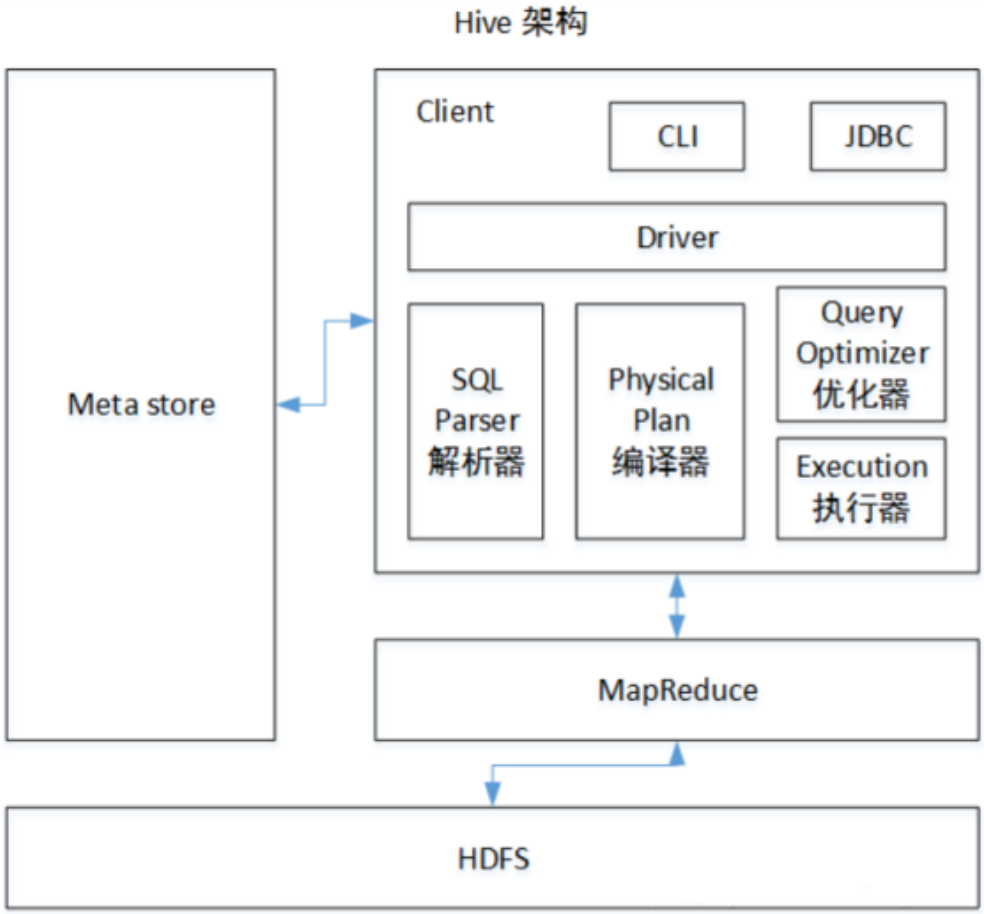
Hive介绍

提供了一种SQL(结构化查询)语言，可将结构化的数据文件映射为一张表，查询存储在HDFS上的数据或其他在HDFS上的文件系统，如HBase，MapR-FS、Cassandra

```
优点：
    1、操作接口采用类SQL语法，能快速开发
    2、避免去学MapReduce，减少学习成本
    3、支持用户自定义函数
    4、处理大数据便捷
缺点：
    1、执行延迟比较高，自动生成的MapReduce作业比较慢
    2、表达能力有限，体现在迭代式算法无法表达、MapReduce数据处理流程限制，无法实现效率更高的算法
    3、不支持记录级别的更新、插入、删除操作。
```

Hive架构（重点）



用户接口（Client）：CLI、JDBC/ODBC、WEBUI

元数据（Meta store）：表名、表所属数据库、表的拥有者、列/分区字段、表的类型（内、外部表）、表的数据所在目录等

驱动器Driver

```
1、解析器：将SQL字符串转换成抽象语法树AST，一般都用第三方工具库完成，比如antlr
        对AST进行语法分析，例如表、字段是否存在，SQL语义是否有误
2、编译器：将AST编译成逻辑执行计划
3、优化器：将逻辑执行计划进行优化
4、执行器：将逻辑执行计划转化成物理计划，例如MR/Spark
```

Hive内外部表（重点）

内部表（managed table）、外部表（external table）：是否被external修饰

```
内部表数据存储的位置是hive.metastore.warehouse.dir（默认是/user/hive/warehouse）
外部表数据存储位置是自己规定（如果没有LOCATION，在）在HDFS上的/user/hive/warehouse下以外部表的表名创建一个文件夹

内部表的数据由Hive自身管理
外部表的数据由HDFS管理

创建表：
    创建内部表时，数据将移动到数据仓库指向的路径
    创建外部表时，仅记录数据所在路径
删除表：
    删除内部表时，元数据和数据一起被删除
    删除外部表时，只删除元数据
```

外部表的优点：

1、外部表不会加载到Hive的默认仓库，减少了数据的传输，同时还能和其他外部表共享数据

2、使用外部表，Hive不会修改源数据，不用担心数据损坏或丢失

Hive建表语句

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement]
```

(1)CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXISTS 选项来忽略这个异常。
(2)EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时可以指定一个指向实际数据的路径（LOCATION），在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。
(3)COMMENT 为表和列添加注释
(4)PARTITIONED BY 创建分区表
(5)CLUSTERED BY 创建分桶表
(6)SORTED BY 不常用，对桶中的一个或多个列另外排序
(7)ROW FORMAT DELIMITED [FIELDS TERMINATED BY char][COLLECTION ITEMS TERMINATED BY char][MAP KEYS TERMINATED BY char][LINES TERMINATED BY char]SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value,property_name=property_value, ...)] 用户在建表的时候可以自定义SerDe或者使用自带的SerDe，如果没有指定ROW FORMAT或者ROW FORMAT DELIMITED，将会使用自带的SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的SerDe,Hive通过SerDe确定表的具体的列的数据。SerDe是Serialize/Deserilize的简称，hive使用Serde进行行对象的序列与反序列化。
(8)STORED AS 指定存储文件类型，常用的存储文件类型，常用的存储文件类型：SEQUENCEFILE(二进制序列文件)、TEXTFILE(文本)、RCFILE(列式存储格式文件)如果文件数据是纯文本，可以使用STORED AS TEXTFILE，如果数据需要压缩，使用STORED AS SEQUENCEFILE
(9)LOCATION 指定表在HDFS上的存储位置
(10)AS 后跟查询语句，根据查询结果创建表
(11)LIKE 允许用户复制现有的表结构，但是不复制数据

Hive数据倾斜以及解决方案（重点）

1、什么是数据倾斜?

数据倾斜主要表现在，map/reduce程序执行时，reduce节点大部分执行完毕，但是有一个或者几个reduce节点运行很慢，导致整个程序的处理时间很长。

这是因为某一个key的条数比其他key多很多，这条key节点所处理的数据量比其他节点大很多，从而导致某几个节点迟迟运行不完。

2、数据倾斜的原因

关键词	情形	后果
Join	其中一个表较小，但是key集中	分发到某一个或几个Reduce上的数据远高于平均值
大表与大表，但是分桶的判断字段0值或空值过多	这些空值都由一个reduce处理，非常慢	
group by	group by维度过小，某值的数量过多	处理某值的reduce非常耗时
Count Distinct	某特殊值过多	处理此特殊值的reduce非常耗时

原因：

- key分布不均匀
- 业务数据本身的特性
- 建表时考虑不周
- 某些SQL语句本身就有数据倾斜

现象：

- 任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。
- 单一reduce的记录数与平均记录数差异过大，通常可能达到3倍甚至更多。最长时长远大于平均时长。

3、数据倾斜的解决方案

1）参数调节

```
hive.map.aggr = true
```

Map端部分聚合，相当于Combiner

```
hive.groupby.skewindata = true
```

有数据倾斜的时候进行负载均衡，当选型设定为true，生成的查询计划会有两个MR Job。

第一个MR Job中，Map的输出结果会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果。这样处理的结果是相同的Group By Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；

第二个MR Job再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key被分布到同一个Reudce中），最后完成最终的聚合操作。

2）SQL语句调节

```
如何Join：
关于驱动表的选取，选用Join key分布最均匀的表最为驱动表，做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小。
大小表Join：
使用map join让小的维度表（1000条以下的记录条数）先进内存。在map端完成reduce

大表Join大表：
```

把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

count distinct大量相同特殊值：
count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果+1

group by维度过小：
采用sum() group by的方式来替换count(distinct)完成计算

特殊情况特殊处理：
在业务逻辑优化效果的一般的情况下，可以将倾斜的数据单独拿出来处理，最后union回去。

Hive的自定义函数

Hive自带了一些函数，比如max/min等，但是数量有限。当Hive提供的内置函数无法满足业务处理需求时，可通过考虑用户自定义函数，用户自定义函数有以下三种：

- UDF（User-Defined-Function）：一进一出
- UDAF（User-Defined Aggregation Function）：聚集函数，多进一出（类似count、max）
- UDTF（User-Defined Table-Generating Functions）：一进多出（类似lateral view exploed()）

完成步骤

1）导入依赖
2）创建一个类，继承于Hive自带的UDF
3）打成jar包上传到linux服务器
4）将jar包添加到hive的classpath
5）创建临时函数与开发好的java class关联

Hive的sort by、distribute by、cluser by、order by区别

- sort by：不是全局排序，数据在进入reducer前完成排序
- distribute by：按照指定的字段对数据进行划分输出到不同的reduce中
- cluster by：除了具有distribute by的功能之外兼具sort by
- order by：对输入做全局排序，只有一个reducer（多个reducer无法保证全局有序）

Hive分区和分桶的区别

1、定义上

分区
Hive的分区使用HDFS的子目录功能实现，每一个子目录包含了分区对应的列名和每一列的值。
Hive的分区方式：Hive实际是存储在HDFS上的抽象，一个分区名对应一个目录名，子分区名就是子目录名，并不是一个实际字段
注意：partitioned by自己中定义的列是表中正式的列（分区列），但是数据文件内不包含这些列

分桶
分桶表时在表或者分区表的基础上，进一步对表进行组织，Hive使用对分桶所用的值；
进行hash，用hash结果除以桶的个数做取余运算的方式来分桶，保证每个桶中都有数据

2、数据类型上

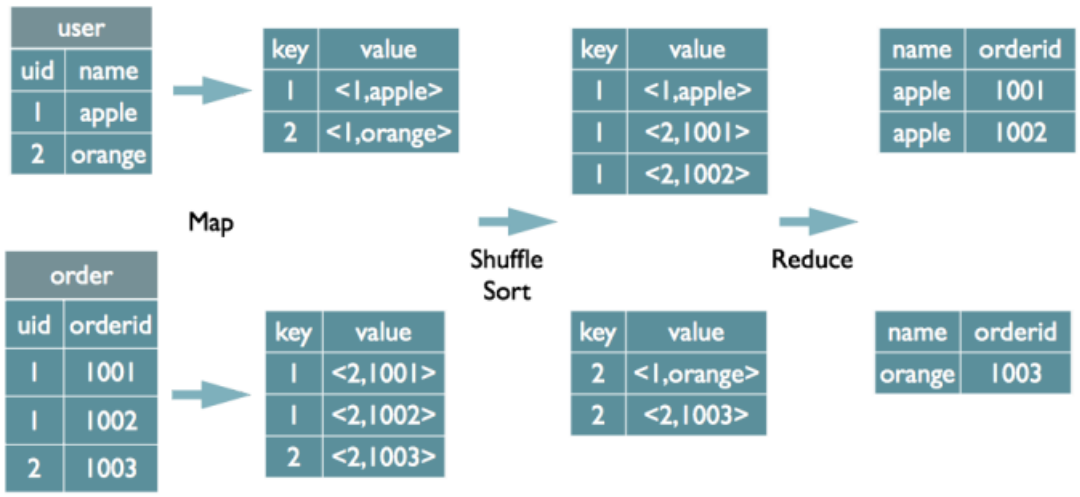
分区是非随机分割数据库：按照列的值分割，易造成数据倾斜；对应不同的文件夹（粗粒度）
分桶随机分割数据库：按照列的哈希函数分割，相对平均；对应不同的文件（细粒度），能获得比分区更高的查询处理效率

注意：普通表（内部表、外部表）、分区表这三个都是对应HDFS上的目录，桶表对应目录里的文件。

HQL转化为MR的过程

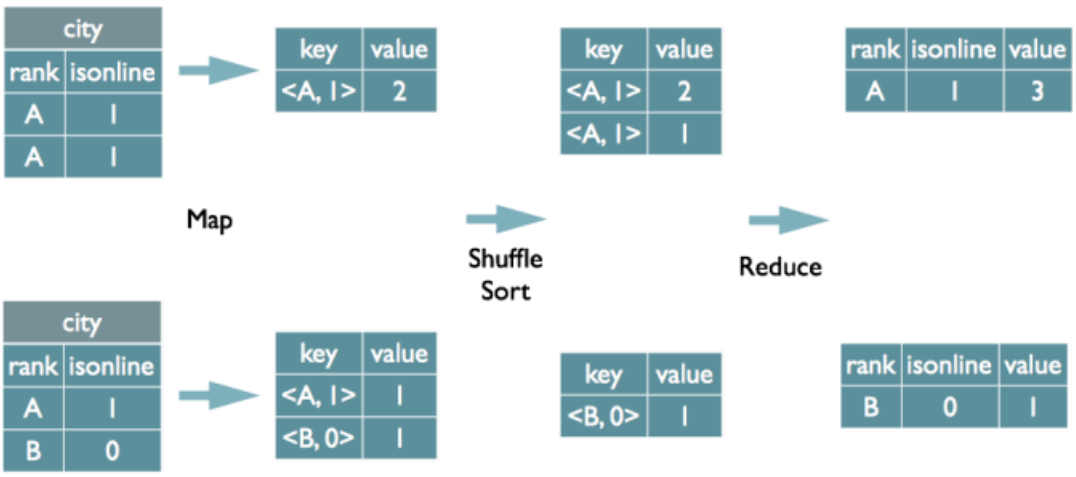
1、Join的实现原理

```
select u.name,o.orderid from order o join uuser u on o.uid = u.uid;
```



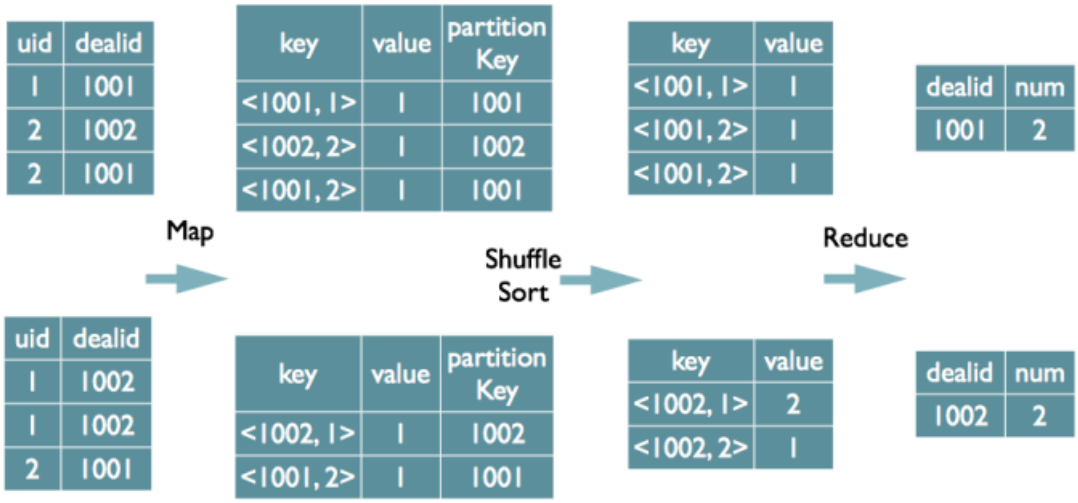
2、Group By实现原理

```
select rank,isonline,count(*) from group by rank,isonline
```



3、Distinct实现原理

```
select dealid,count(distinct uid) num from order group by dealid;
```



SQL转化为MapReduce过程

1、Antlr定义SQL的语法规则，完成SQL词法、语法解析、将SQL转化为抽象语法树AST Tree
2、遍历AST Tree，抽象出查询的基本组成单元QueryBlock
3、遍历QueryBlock，翻译为执行操作树OperatorTree
4、逻辑层优化器进行OperatorTree变换，合并不必要的ReduceSinkOperator，减少shuffle数据量
5、遍历OperatorTree，翻译为MR任务
6、物理层优化器进行MR任务的变化，生成最终的执行计划

Hive的存储引擎和计算引擎

1、计算引擎

MapReduce、Tez、Spark计算引擎

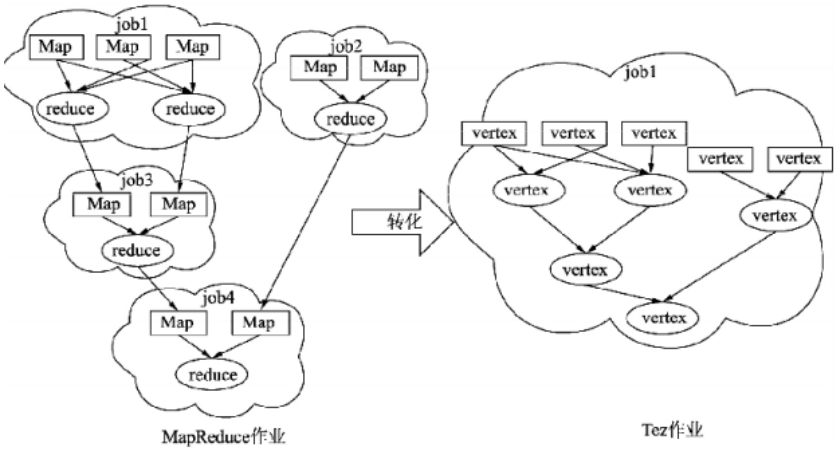
```
set hive.execution.engine=mr/spark/tez
```

Tez计算引擎

Apache Tez是进行大规模数据处理且支持DAG作业的计算框架，源于MR框架，允许不同类型的作业在一个集群中运行
将原有的Map、Reduce两个操作简化为一个概念——Vertex，将原有的计算节点拆分成多个部分：Vertex Input、Vertex Output、Sorting、Shuffling、Merging。计算节点之间的数据通信被统称为Edge

Tez也使用Yarn作为资源调度。Tez on Yarn将作业提交到AMPoolServer的服务上，AMPoolServer事先存放了启动ApplicationMaster的服务。当用户提交一个作业上来，AMPoolServer从中挑选一个ApplicationMaster用于管理用户提交上来的作业

Tez运行一次发送整个查询计划，实现应用程序动态规划，从而使框架能够更智能分配资源，并通过各个阶段流水线传输数据（MRR）



Spark计算引擎

```
Apache Spark专门为大规模数据处理而设计的快速、通用支持DAG作业的计算引擎
```

2、存储引擎

Hive的文件存储格式（存储引擎）有四种：TEXTFILE、SEQUENCEFILE、PARQUET、ORC

```
EXTFILE：
    按行存储；不支持块压缩
    磁盘开销大，加载数据的速度最高
SEQUENCEFILE：
    按行存储
    Hadoop API提供的一种二进制文件，以<key,value>的形式序列化到文件中
PARQUET：
    按列存储
    压缩比较低，查询效率低
RCFILE：
    数据按行分块，每块按列存储
    同一行的数据位于同一节点，元组重构开销很低
    能够利用列维度的数据压缩，并且能跳过不必要的列读取
ORCFILE：
    RCFILE的改良版本，使用了索引
    提高了hive读、写和处理数据的能力
```

Join的操作原理

Hive中的Join可分为Common Join、Map Join

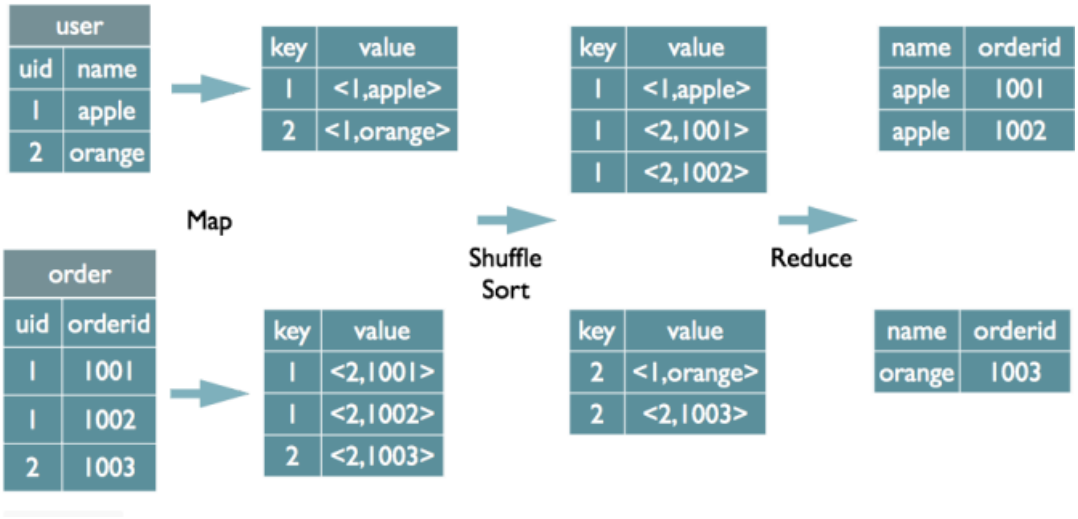
1、Common Join

```
select u.name o.orderid from order o join user u on o.uid = u.uid
```

Map阶段
读取源表的数据，Map输出的Key为Join on条件中的列，如果Join有多个关联键，则以关联键的组合作为Key
Value为之后所关心的列（select或者where中用到的）,同时value中会包含表的Tag

Shuffle阶段
根据Key的值进行hash，并将key/value按照hash值推送至不同的reduce中

Reduce阶段
根据Key的值完成Join操作，期间通过Tag来识别不同表中的数据

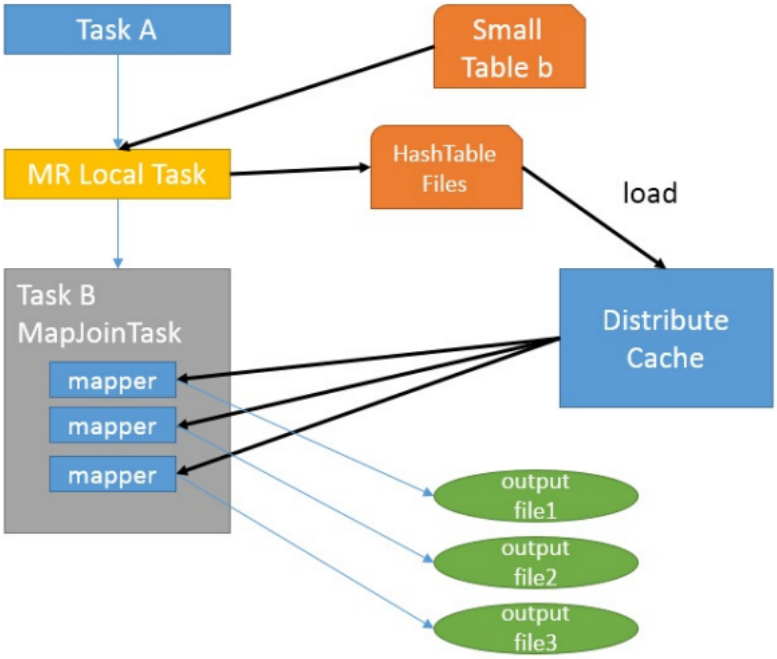


2、Map Join

小表Join大表的场景。在map端完成，节省了Shuffle阶段的开销

Mapjoin设置：hive.auto.convert.join=true
小表大小设置：hive.mapjoin.smalltable.filesize，默认为25M

TaskA是一个Local Task，负责扫描小表b中的数据，将其转化成一个HashTable的数据结构，并写入本地的文件中，文件之后加载到DistributeCache中
TaskB，是一个没有Reduce的MR，启动MapTasks扫描大表a，根据a的每一条记录去和DistributeCache中b表对应的HashTable关联，并直接输出结果



3、SMB Join

smb是sort merge bucket操作，也就是首先排序、继而合并、最后放到对应的bucket中去。

在进行两个表联合的时候，首先进行分桶，将table1中的小部分和table1中的小部分进行联合，减少无关项的扫描

Hive上传数据到HDFS，小文件问题

小文件太多的出现：
    源数据本身有很多小文件
    动态分区产生很多小文件
    reduce个数越多，小文件越多
    按分区插入数据产生很多小文件，文件个数=maptask个数*分区数

小文件太多的影响：
    从Hive角度看，会启动很很多map,一个map需要一个JVM去执行，任务的初始化浪费资源
    HDFS存储太多小文件，导致NN元数据特别多，占有其太多内存

1、通过调整参数合并

```
--调整每个Map最大输入大小：决定合并后的文件数量
set mapred.max.split.size=256000000
--一个节点上split的最小大小：决定多个DN上的文件是否需要合并
set mapred.min.split.size.per.node=1000000000
--一个交换机下split的最小大小：决定多个交换机上的文件是否需要合并
set mapred.min.split.size.per.rack=1000000000
--执行Map前，进行小文件合并
set hive.input.format=org.apache.hadoop.hive.sql.io.CombineHiveInputFormat

--在map-only job后合并文件,默认true
set hive.merge.mapfiles=true
--在map-reduce job后合并文件,默认false
set hive.merge.mapredfiles=true
--合并后每个文件的大小，默认256000000
set hive.merge.size.per.task=256000000
```

```
--平均文件大小，觉得是否执行合并操作的阈值，默认16000000
set hive.merge.smallfiles.avgsize=1000000000
```

2、针对分区插入数据的时候产生的大量小文件问题，使用DISTRIBUTE BY rand()将数据随机分配给Reduce，使每个Reduce处理的数据大体一致

```
--设置每个reducer处理的大小为5个G
set hive.exec.reducers.bytes.per.reducer=512000000

insert overwrite table test partition(dt)
select * from iteblog_tmp
DISTRIBUTE BY rand()
```

3、使用Sequencefile作为表存储格式

4、使用Hadoop的archhive归档

```
--用来控制归档是否可用
set hive.archive.enabled=true;
--通知Hive在创建归档时是否可以设置父目录
set hive.archive.har.parentdir.settable=true;
--控制需要归档文件的大小
set har.partfile.size=1099511627776

--使用以下命令进行归档
ALTER TABLE srcpart ARCHIVE PARTITION(ds='2008-04-08',hr='12');
--对已归档的分区恢复会原文件
ALTER TABLE srcpart UNARCHIVE PARTITION(ds='2008-04-08',hr='12');

--注意：归档的分区不能够INSERT OVERWRITE，必须先UNARCHIVE
```

Hive保存元数据的方式

- 内嵌模式：将元数据保存在本地内嵌的derby数据库中，内嵌的derby数据库每次只能访问一个数据文件
- 本地模式：将元数据保存在本地独立的数据库中（一般是mysql），支持多会话连接
- 远程模式：将元数据保存在远程独立的数据库中，避免每个客户都去安装数据库（例如mysql）

```
内嵌模式不需要额外启metastore服务，这个是默认的，配置简单，适用于实验。
本地模式不需要单独启metastore服务，用的是hive在同一个进程里的metastore服务
远程模式需要单独启metastore服务，然后每个客户端在配置文件里需要配置连接metastore服务
```

Hive开窗函数

分析函数：用于计算基于组的某种聚合值

开窗函数：指定分析函数工作的数据窗口大小

```
分析函数（sum(),max(),row_number()...） + 窗口自己（over函数）
```

```
1、SUM函数
sum()根据每一行的窗口返回各自行对应的值，有多少行记录就有多少个sum值

2、NTILE函数
```

NTILE(n)用于将分区数据按照顺序切分成n片，返回当前切片值
    如果切片不均匀，默认增加第一个切片的分布
    不支持ROWS BETWEEN

3、ROW_NUMBER函数
ROW_NUMBER()从1开始，按照顺序，生成分组内记录的序列

4、RANK和DENSE_RANK函数
RANK()生成数据项在分组中的排名，排名相等会在名次中留下空位
DENSE_RANK()生成数据项在分组中的排名，排名相等在名次中不会留下空位

5、CUME_DIST函数
CUME_DIST()返回小于等于当前值的行数/分组内总行数

6、PERCENT_RANK函数
PERCENT_RANK分组当前行的rank值-1/分组内总函数-1

7、LAG和LEAD函数
LAG(col,n,DEFAULT)用于统计窗口内往上第n行值
LEAD(col,n,DEFAULT)用于统计窗口内往下第n行值
col：列名　n：往上第n行 DEFALUT：往上第n行为NULL时，取默认值，不指定为NULL

8、FIRST_VALUE和LAST_VALUE函数
FIRST_VALUE()取分组内排序后，截止到当前行第一个值
LAST_VALUE()取分组内排序后，截止到当前行最后一个值