题目一： 分析一条 TPCDS SQL

通过以下命令执行获取SQL执行过程日志：

```
nohup ./spark-3.1.1-bin-hadoop2.7/bin/spark-submit --conf spark.sql.planChangeLog.level=WARN \
--class org.apache.spark.sql.execution.benchmark.TPCDSQueryBenchmark \
--jars spark-core_2.12-3.1.1-tests.jar,spark-catalyst_2.12-3.1.1-tests.jar spark-sql_2.12-3.1.1-tests.jar \
--data-location tpcds-data-1g --query-filter "q73" \
> running.log 2>&1 &
```

running.log

运行过程部分截图

```
, isnotnull(s_stor..., Format: Parquet, Location: InMemoryFileIndex[file:/Users/chenhao/github_code/spark-tpcds-datagen/tpcds-data-1g/store], PartitionFilters: [], PushedFilters: [In(s_county
[Williamson County,Franklin Parish,Bronx County,Orange County]), IsNotNull(s_store_..., ReadSchema: struct<s_store_sk:int,s_county:string>
                                                                                                                                  :                :     :             +- *(1) ColumnarToRow
        :             +- Project [hd_demo_sk#446]


                                                                                                                                  :                :     :                +- FileScan parquet [d_date_sk#329,d_year#335,d
om#338] Batched: true, DataFilters: [isnotnull(d_dom#338), (d_dom#338 >= 1), (d_dom#338 <= 2), d_year#335 IN (1999,2000,2001), isnotn..., Format: Parquet, Location: InMemoryFileIndex[file:/Us
s/chenhao/github_code/spark-tpcds-datagen/tpcds-data-1g/date_dim], PartitionFilters: [], PushedFilters: [IsNotNull(d_dom), GreaterThanOrEqual(d_dom,1), LessThanOrEqual(d_dom,2), In(d_year, [1
9,2000,2..., ReadSchema: struct<d_date_sk:int,d_year:int,d_dom:int>
        :                +- Filter ((((isnotnull(hd_vehicle_count#450) AND ((hd_buy_potential#448 = >10000) OR (hd_buy_potential#448 = unknown))) AND (hd_vehicle_count#450 > 0)) AND (CASE W
N (hd_vehicle_count#450 > 0) THEN (cast(hd_dep_count#449 as double) / cast(hd_vehicle_count#450 as double)) ELSE null END > 1.0)) AND isnotnull(hd_demo_sk#446))

                                                                                                                                  :                :     +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[
 int, true] as bigint)),false), [id=#1242]
        :                   +- FileScan parquet [hd_demo_sk#446,hd_buy_potential#448,hd_dep_count#449,hd_vehicle_count#450] Batched: true, DataFilters: [isnotnull(hd_vehicle_count#450), ((h
buy_potential#448 = >10000) OR (hd_buy_potential#448 = unk..., Format: Parquet, Location: InMemoryFileIndex[file:/Users/chenhao/github_code/spark-tpcds-datagen/tpcds-data-1g/household_dem...,
artitionFilters: [], PushedFilters: [IsNotNull(hd_vehicle_count), Or(EqualTo(hd_buy_potential,>10000),EqualTo(hd_buy_potential,unknow..., ReadSchema: struct<hd_demo_sk:int,hd_buy_potential:st
ng,hd_dep_count:int,hd_vehicle_count:int>                                                                                         :                :        +- *(2) Project [s_store_sk#666]
```

题目一： 分析一条 TPCDS SQL

■ ColumnPruning（列剪裁）

```
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch Union has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch OptimizeLimitZero has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch LocalRelation early has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch Pullup Correlated Expressions has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch Subquery has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch Replace Operators has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger: Batch Aggregate has no effect.
:1/11/13 21:30:21 WARN PlanChangeLogger:
=== Applying Rule org.apache.spark.sql.catalyst.optimizer.ColumnPruning ===
Aggregate [count(1) AS count#1816L]
ggregate [count(1) AS count#1816L]
 +-
elation[wp_web_page_sk#1759,wp_web_page_id#1760,wp_rec_start_date#1761,wp_rec_end_date#1762,wp_creation_date_sk#1763,wp_access_date_sk#1764,wp_autogen_flag#1765,wp_customer
_sk#1766,wp_url#1767,wp_type#1768,wp_char_count#1769,wp_link_count#1770,wp_image_count#1771,wp_max_ad_count#1772] parquet    +- Project

 -
elation[wp_web_page_sk#1759,wp_web_page_id#1760,wp_rec_start_date#1761,wp_rec_end_date#1762,wp_creation_date_sk#1763,wp_access_date_sk#1764,wp_autogen_flag#1765,wp_customer
_sk#1766,wp_url#1767,wp_type#1768,wp_char_count#1769,wp_link_count#1770,wp_image_count#1771,wp_max_ad_count#1772] parquet

:1/11/13 21:30:21 WARN PlanChangeLogger:
=== Result of Batch Operator Optimization before Inferring Filters ===
 Aggregate [count(1) AS count#1816L]
ggregate [count(1) AS count#1816L]
```

该优化规则，尝试从逻辑计划中去掉不需要的列，从而减少读取数据的量。
在优化查询时，不需要扫描表中的所有列值，对于这个SQL来说，只是做了个简单的count(1)，所以优化后的project里没有列。这个优化一方面大幅度减少了网络、内存数据量消耗，另一方面对于列存格式来说大大提高了扫描效率。

■ ReorderJoin（Join顺序优化）

```
21/11/13 21:30:23 WARN PlanChangeLogger:
=== Applying Rule org.apache.spark.sql.catalyst.optimizer.ReorderJoin ===
 OverwriteByExpression RelationV2[] noop-table, true, true
OverwriteByExpression RelationV2[] noop-table, true, true
 +- Sort [cnt#1828L DESC NULLS LAST], true
+- Sort [cnt#1828L DESC NULLS LAST], true
   +- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
+- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
   +- Filter ((ss_customer_sk#1157 = c_customer_sk#154) AND ((cnt#1828L >= cast(1 as bigint)) AND (cnt#1828L <= cast(5 as bigint))))
+- Filter ((ss_customer_sk#1157 = c_customer_sk#154) AND ((cnt#1828L >= cast(1 as bigint)) AND (cnt#1828L <= cast(5 as bigint))))
      +- Join Inner
+- Join Inner
         :- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
:- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
!     +- Filter (((((ss_sold_date_sk#1154 = d_date_sk#329) AND (ss_store_sk#1161 = s_store_sk#666)) AND (ss_hdemo_sk#1159 = hd_demo_sk#446)) AND (((d_dom#338 >= 1)
AND (d_dom#338 <= 2)) AND ((hd_buy_potential#448 = >10000) OR (hd_buy_potential#448 = unknown)))) AND (((hd_vehicle_count#450 > 0) AND (CASE WHEN (hd_vehicle_count#450 > 0)
THEN (cast(hd_dep_count#449 as double) / cast(hd_vehicle_count#450 as double)) ELSE cast(null as double) END > cast(1 as double))) AND (d_year#335 IN (1999,(1999 + 1),(1999
+ 2)) AND s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange County))))                :  +- Join Inner, ((((ss_hdemo_sk#1159 = hd_demo_sk#446) AND
((hd_buy_potential#448 = >10000) OR (hd_buy_potential#448 = unknown))) AND ((hd_vehicle_count#450 > 0) AND (CASE WHEN (hd_vehicle_count#450 > 0) THEN (cast(hd_dep_count#449
as double) / cast(hd_vehicle_count#450 as double)) ELSE cast(null as double) END > cast(1 as double)))
!        :     +- Join Inner
:        :- Join Inner, ((ss_store_sk#1161 = s_store_sk#666) AND s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange County))
:        :  :- Join Inner
:        :  :- Join Inner, ((((ss_sold_date_sk#1154 = d_date_sk#329) AND (d_dom#338 >= 1)) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))
:        :  :  :- Join Inner
:        :  :  :-
Relation[ss_sold_date_sk#1154,ss_sold_time_sk#1155,ss_item_sk#1156,ss_customer_sk#1157,ss_cdemo_sk#1158,ss_hdemo_sk#1159,ss_addr_sk#1160,ss_store_sk#1161,ss_promo_sk#1162,ss
```

该优化规则，会根据SQL执行成本，重新调整JOIN顺序和查询计划。
在这个SQL中，将原本过滤条件中的两组【(ss_sold_date_sk#1154 = d_date_sk#329)、(d_dom#338 >= 1) AND (d_dom#338 <= 2)、(d_year#335 IN (1999,(1999 + 1),(1999 + 2)】和【(ss_store_sk#1161 = s_store_sk#666)、s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange County)】重新调整至成两个JOIN中的条件：

:- Join Inner, ((ss_store_sk#1161 = s_store_sk#666) AND s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange
Count
y))
: Join Inner, ((((ss_sold_date_sk#1154 = d_date_sk#329) AND (d_dom#338 >= 1)) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))

题目一： 分析一条 TPCDS SQL

■ PushDownPredicates（谓词下推）

```
21/11/13 21:30:23 WARN PlanChangeLogger:
=== Applying Rule org.apache.spark.sql.catalyst.optimizer.PushDownPredicates ===
 OverwriteByExpression RelationV2[] noop-table, true, true
OverwriteByExpression RelationV2[] noop-table, true, true
 +- Sort [cnt#1828L DESC NULLS LAST], true
+- Sort [cnt#1828L DESC NULLS LAST], true
    +- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
+- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
!      +- Filter ((ss_customer_sk#1157 = c_customer_sk#154) AND ((cnt#1828L >= cast(1 as bigint)) AND (cnt#1828L <= cast(5 as bigint))))
+- Join Inner, (ss_customer_sk#1157 = c_customer_sk#154)
!         +- Join Inner
:- Filter ((cnt#1828L >= cast(1 as bigint)) AND (cnt#1828L <= cast(5 as bigint)))
!            :- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
:  +- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
!            :  +- Join Inner, ((((ss_hdemo_sk#1159 = hd_demo_sk#446) AND ((hd_buy_potential#448 = >10000) OR (hd_buy_potential#448 = unknown))) AND (hd_vehicle_count#450 >
0)) AND (CASE WHEN (hd_vehicle_count#450 > 0) THEN (cast(hd_dep_count#449 as double) / cast(hd_vehicle_count#450 as double)) ELSE cast(null as double) END > cast(1 as
double)))
:     +- Join Inner, (ss_hdemo_sk#1159 = hd_demo_sk#446)
!            :     :- Join Inner, ((ss_store_sk#1161 = s_store_sk#666) AND s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange County))
:        :- Join Inner, (ss_store_sk#1161 = s_store_sk#666)
!            :     :  :- Join Inner, ((((ss_sold_date_sk#1154 = d_date_sk#329) AND (d_dom#338 >= 1)) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))
```

```
Relation[ss_sold_date_sk#1154,ss_sold_time_sk#1155,ss_item_sk#1156,ss_customer_sk#1157,ss_cdemo_sk#1158,ss_hdemo_sk#1159,ss_addr_sk#1160,ss_store_sk#1161,ss_promo_sk#1162,ss
_ticket_number#1163,ss_quantity#1164,ss_wholesale_cost#1165,ss_list_price#1166,ss_sales_price#1167,ss_ext_discount_amt#1168,ss_ext_sales_price#1169,ss_ext_wholesale_cost#117
0,ss_ext_list_price#1171,ss_ext_tax#1172,ss_coupon_amt#1173,ss_net_paid#1174,ss_net_paid_inc_tax#1175,ss_net_profit#1176] parquet        :        :  :  :-
Relation[ss_sold_date_sk#1154,ss_sold_time_sk#1155,ss_item_sk#1156,ss_customer_sk#1157,ss_cdemo_sk#1158,ss_hdemo_sk#1159,ss_addr_sk#1160,ss_store_sk#1161,ss_promo_sk#1162,ss
_ticket_number#1163,ss_quantity#1164,ss_wholesale_cost#1165,ss_list_price#1166,ss_sales_price#1167,ss_ext_discount_amt#1168,ss_ext_sales_price#1169,ss_ext_wholesale_cost#117
0,ss_ext_list_price#1171,ss_ext_tax#1172,ss_coupon_amt#1173,ss_net_paid#1174,ss_net_paid_inc_tax#1175,ss_net_profit#1176] parquet
!           :     :  :  +-
Relation[d_date_sk#329,d_date_id#330,d_date#331,d_month_seq#332,d_week_seq#333,d_quarter_seq#334,d_year#335,d_dow#336,d_moy#337,d_dom#338,d_qoy#339,d_fy_year#340,d_fy_quarte
r_seq#341,d_fy_week_seq#342,d_day_name#343,d_quarter_name#344,d_holiday#345,d_weekend#346,d_following_holiday#347,d_first_dom#348,d_last_dom#349,d_same_day_ly#350,d_same_day
_lq#351,d_current_day#352,... 4 more fields] parquet                                          :        :  :  +- Filter
(((d_dom#338 >= 1) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))
!           :     :  +-
Relation[s_store_sk#666,s_store_id#667,s_rec_start_date#668,s_rec_end_date#669,s_closed_date_sk#670,s_store_name#671,s_number_employees#672,s_floor_space#673,s_hours#674,s_m
anager#675,s_market_id#676,s_geography_class#677,s_market_desc#678,s_market_manager#679,s_division_id#680,s_division_name#681,s_company_id#682,s_company_name#683,s_street_nu
mber#684,s_street_name#685,s_street_type#686,s_suite_number#687,s_city#688,s_county#689,... 5 more fields] parquet          :        :  :     +-
Relation[d_date_sk#329,d_date_id#330,d_date#331,d_month_seq#332,d_week_seq#333,d_quarter_seq#334,d_year#335,d_dow#336,d_moy#337,d_dom#338,d_qoy#339,d_fy_year#340,d_fy_quarte
r_seq#341,d_fy_week_seq#342,d_day_name#343,d_quarter_name#344,d_holiday#345,d_weekend#346,d_following_holiday#347,d_first_dom#348,d_last_dom#349,d_same_day_ly#350,d_same_day
_lq#351,d_current_day#352,... 4 more fields] parquet
!           :     +- Relation[hd_demo_sk#446,hd_income_band_sk#447,hd_buy_potential#448,hd_dep_count#449,hd_vehicle_count#450] parquet
:        :  +- Filter s_county#689 IN (Williamson County,Franklin Parish,Bronx County,Orange County)
!           +-
Relation[c_customer_sk#154,c_customer_id#155,c_current_cdemo_sk#156,c_current_hdemo_sk#157,c_current_addr_sk#158,c_first_shipto_date_sk#159,c_first_sales_date_sk#160,c_salut
ation#161,c_first_name#162,c_last_name#163,c_preferred_cust_flag#164,c_birth_day#165,c_birth_month#166,c_birth_year#167,c_birth_country#168,c_login#169,c_email_address#170,c
_last_review_date#171] parquet                                                                      :        :     +-
Relation[s_store_sk#666,s_store_id#667,s_rec_start_date#668,s_rec_end_date#669,s_closed_date_sk#670,s_store_name#671,s_number_employees#672,s_floor_space#673,s_hours#674,s_m
anager#675,s_market_id#676,s_geography_class#677,s_market_desc#678,s_market_manager#679,s_division_id#680,s_division_name#681,s_company_id#682,s_company_name#683,s_street_nu
mber#684,s_street_name#685,s_street_type#686,s_suite_number#687,s_city#688,s_county#689,... 5 more fields] parquet
!
:           +- Filter ((((hd_buy_potential#448 = >10000) OR (hd_buy_potential#448 = unknown)) AND (hd_vehicle_count#450 > 0)) AND (CASE WHEN (hd_vehicle_count#450 > 0) THEN
(cast(hd_dep_count#449 as double) / cast(hd_vehicle_count#450 as double)) ELSE cast(null as double) END > cast(1 as double)))
!
:              +- Relation[hd_demo_sk#446,hd_income_band_sk#447,hd_buy_potential#448,hd_dep_count#449,hd_vehicle_count#450] parquet
!
```

谓词下推，将过滤条件尽可能地下推到底层，最好是数据源。
如图所示，将原本在join中的过滤条件，下推至在读取相应parquet文件时进行过滤。如此系统在扫描表时就对数据进行了过滤，后续能够减少参加join的数据量大小，节省join算子所需的时间，提供关联效率。

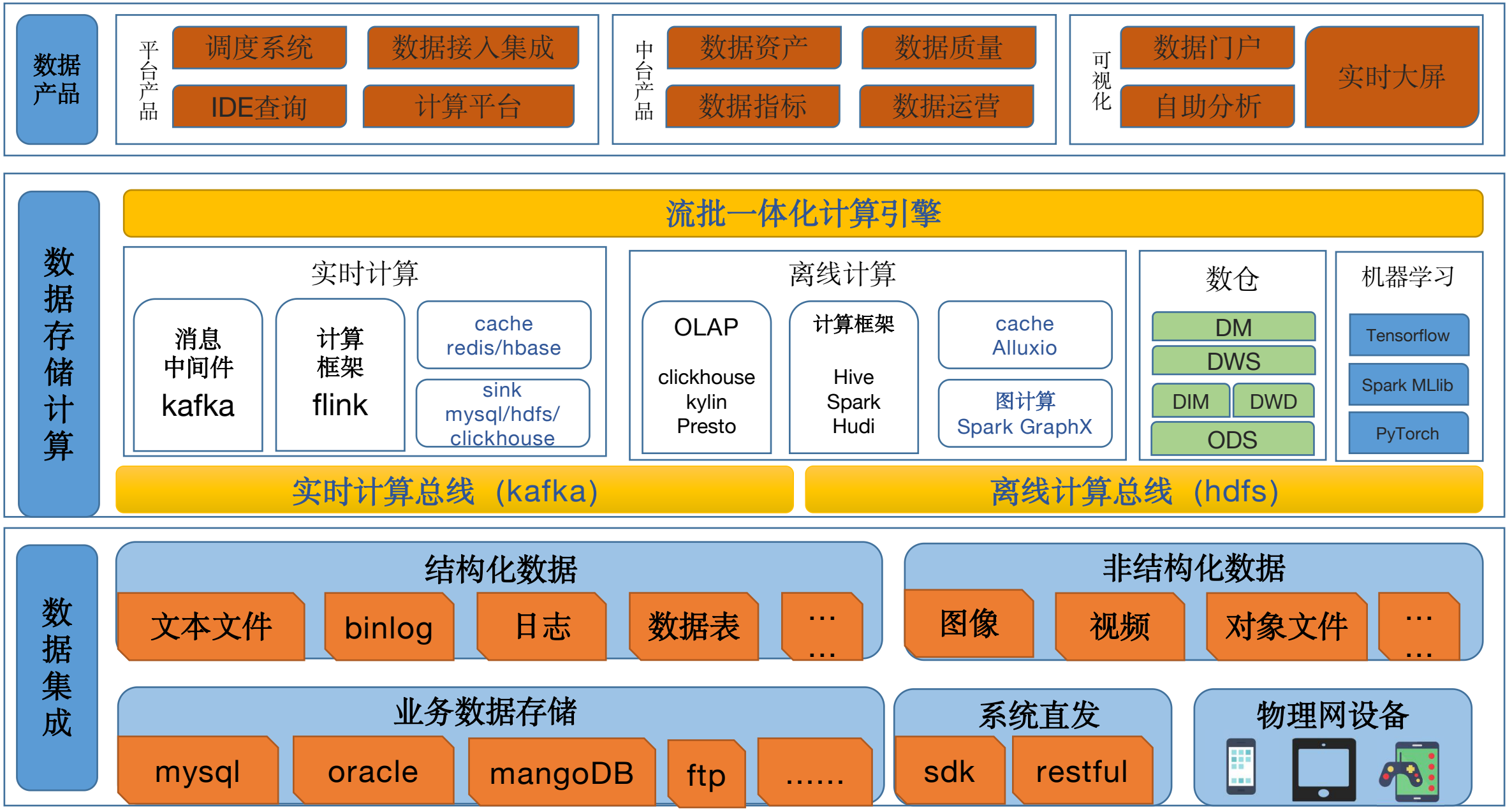
■ ConstantFolding（常量累加）

```
21/11/13 21:30:23 WARN PlanChangeLogger:
=== Applying Rule org.apache.spark.sql.catalyst.optimizer.ConstantFolding ===
 OverwriteByExpression RelationV2[] noop-table, true, true
OverwriteByExpression RelationV2[] noop-table, true, true
 +- Sort [cnt#1828L DESC NULLS LAST], true
+- Sort [cnt#1828L DESC NULLS LAST], true
    +- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
+- Project [c_last_name#163, c_first_name#162, c_salutation#161, c_preferred_cust_flag#164, ss_ticket_number#1163, cnt#1828L]
       +- Join Inner, (ss_customer_sk#1157 = c_customer_sk#154)
+- Join Inner, (ss_customer_sk#1157 = c_customer_sk#154)
!         :- Filter ((cnt#1828L >= cast(1 as bigint)) AND (cnt#1828L <= cast(5 as bigint)))
:- Filter ((cnt#1828L >= 1) AND (cnt#1828L <= 5))
          :  +- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
:  +- Aggregate [ss_ticket_number#1163, ss_customer_sk#1157], [ss_ticket_number#1163, ss_customer_sk#1157, count(1) AS cnt#1828L]
```

```
_ticket_number#1163,ss_quantity#1164,ss_wholesale_cost#1165,ss_list_price#1166,ss_sales_price#1167,ss_ext_discount_amt#1168,ss_ext_sales_price#1169,ss_ex
),ss_ext_list_price#1171,ss_ext_tax#1172,ss_coupon_amt#1173,ss_net_paid#1174,ss_net_paid_inc_tax#1175,ss_net_profit#1176] parquet        :
Relation[ss_sold_date_sk#1154,ss_sold_time_sk#1155,ss_item_sk#1156,ss_customer_sk#1157,ss_cdemo_sk#1158,ss_hdemo_sk#1159,ss_addr_sk#1160,ss_store_sk#1161
_ticket_number#1163,ss_quantity#1164,ss_wholesale_cost#1165,ss_list_price#1166,ss_sales_price#1167,ss_ext_discount_amt#1168,ss_ext_sales_price#1169,ss_ex
),ss_ext_list_price#1171,ss_ext_tax#1172,ss_coupon_amt#1173,ss_net_paid#1174,ss_net_paid_inc_tax#1175,ss_net_profit#1176] parquet
        :        :     :  +- Project [d_date_sk#329]
:          :     :     +- Project [d_date_sk#329]
!       :        :     :           +- Filter (((d_dom#338 >= 1) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))
:          :     :     +- Filter (((d_dom#338 >= 1) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,2000,2001))
        :        :           +-
```

常量累加，就是将有常量计算的地方预先处理好，减少后续不必要的重复处理过程。
该优化中，将过滤条件Filter (((d_dom#338 >= 1) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,(1999 + 1),(1999 + 2)))
直接处理成Filter (((d_dom#338 >= 1) AND (d_dom#338 <= 2)) AND d_year#335 IN (1999,2000,2001))
优化后的操作，不再需要每次处理时都执行(1999 + 1),(1999 + 2)的运算。

题目二：架构设计题

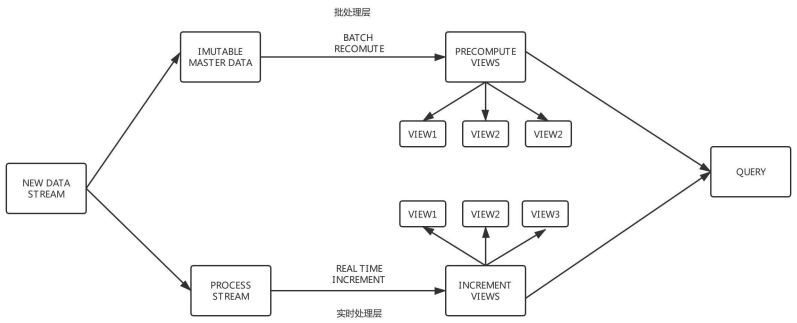

数据产品
平台产品
调度系统
数据接入集成
IDE查询
计算平台
中台产品
数据资产
数据质量
数据指标
数据运营
可视化
数据门户
自助分析
实时大屏
数据存储计算
流批一体化计算引擎
实时计算
消息中间件
kafka
计算框架
flink
cache
redis/hbase
sink
mysql/hdfs/
clickhouse
离线计算
OLAP
clickhouse
kylin
Presto
计算框架
Hive
Spark
Hudi
cache
Alluxio
图计算
Spark GraphX
数仓
DM
DWS
DIM
DWD
ODS
机器学习
Tensorflow
Spark MLlib
PyTorch
实时计算总线（kafka）
离线计算总线（hdfs）
数据集成
结构化数据
文本文件
binlog
日志
数据表
…
非结构化数据
图像
视频
对象文件
…
业务数据存储
mysql
oracle
mangoDB
ftp
……
系统直发
sdk
restful
物理网设备

题目二：Lambda 架构的优缺点

Lambda架构的概念，用在大数据架构中，是如何让实时流式处理与离线批处理更好的结合起来，以达成对大数据的实时处理。Lambda架构的目标是设计出一个能满足实时大数据系统关键特性的架构，包括有：高容错、低延时和可扩展等。Lambda架构整合离线计算和实时计算，融合不可变性（Immunability），读写分离和复杂性隔离等一系列架构原则，可集成Hadoop，Kafka，Storm，Spark，Hbase等各类大数据组件。

Lambda架构关键特性

- Robust and fault-tolerant（容错性和鲁棒性）：对大规模分布式系统来说，机器是不可靠的，可能会当机，但是系统需要是健壮、行为正确的，即使是遇到机器错误。除了机器错误，人更可能会犯错误。在软件开发中难免会有一些Bug，系统必须对有Bug的程序写入的错误数据有足够的适应能力，所以比机器容错性更加重要的容错性是人为操作容错性。对于大规模的分布式系统来说，人和机器的错误每天都可能会发生，如何应对人和机器的错误，让系统能够从错误中快速恢复尤其重要。
- Low latency reads and updates（低延时）：很多应用对于读和写操作的延时要求非常高，要求对更新和查询的响应是低延时的。
- Scalable（横向扩容）：当数据量/负载增大时，可扩展性的系统通过增加更多的机器资源来维持性能。也就是常说的系统需要线性可扩展，通常采用scale out（通过增加机器的个数）而不是scale up（通过增强机器的性能）。
- General（通用性）：系统需要能够适应广泛的应用，包括金融领域、社交网络、电子商务数据分析等。
- Extensible（可扩展）：需要增加新功能、新特性时，可扩展的系统能以最小的开发代价来增加新功能。
- Allows ad hoc queries（方便查询）：数据中蕴含有价值，需要能够方便、快速的查询出所需要的数据。
- Minimal maintenance（易于维护）：系统要想做到易于维护，其关键是控制其复杂性，越是复杂的系统越容易出错、越难维护。
- Debuggable（易调试）：当出问题时，系统需要有足够的信息来调试错误，找到问题的根源。其关键是能够追根溯源到每个数据生成点。



Lambda的三层架构

大数据系统的关键问题：如何实时地在任意大数据集上进行查询？

最简单的方法是，在全体数据集上在线运行查询函数得到结果。但如果数据量比较大，该方法的计算代价太大了，所以不现实。

Lambda架构通过分解的三层架构来解决该问题：Batch Layer，Speed Layer和Serving Layer。

题目二：Lambda 架构的优缺点

Batch Layer
在Lambda架构中，实现batch view的部分被称之为batch layer。它承担了两个职责：

- 存储Master Dataset，这是一个不变的持续增长的数据集
- 针对这个Master Dataset进行预运算

Batch Layer执行的是批量处理，例如Hadoop或者Spark支持的Map-Reduce方式。
利用Batch Layer进行预运算的作用实际上就是将大数据变小，从而有效地利用资源，改善实时查询的性能。但有一个前提，就是需要预先知道查询需要的数据，如此才能在Batch Layer中安排执行计划，定期对数据进行批量处理。此外，还要求这些预运算的统计数据是支持合并（merge）的。

Serving Layer
Batch Layer通过对master dataset执行查询获得了batch view，而Serving Layer就要负责对batch view进行操作，从而为最终的实时查询提供支撑。因此Serving Layer的职责包含：

- 对batch view的随机访问
- 更新batch view

Serving Layer应该是一个专用的分布式数据库，以支持对batch view的加载、随机读取以及更新，但不支持对batch view的随机写，因为随机写会为数据库引来许多复杂性。简单的特性才能使系统变得更健壮、可预测、易配置，也易于运维。

Speed Layer
speed layer与batch layer非常相似，它们之间最大的区别是前者只处理最近的数据，后者则要处理所有的数据。另一个区别是为了满足最小的延迟，speed layer并不会在同一时间读取所有的新数据，相反，它会在接收到新数据时，更新realtime view，而不会像batch layer那样重新运算整个view。speed layer是一种增量的计算，而非重新运算（recomputation）。
因而，Speed Layer的作用包括：

- 对更新到serving layer带来的高延迟的一种补充
- 快速、增量的算法
- 最终Batch Layer会覆盖speed layer

基于Lambda架构，一旦数据通过batch layer进入到serving layer，在realtime view中的相应结果就不再需要了。

题目二：Lambda 架构的优缺点

一个典型的Lambda架构如下：
数据从底层的数据源开始，经过各种各样的格式进入大数据平台，在大数据平台中经过Kafka、Flume等数据组件进行收集，然后分成两条线进行计算。一条线是进入流式计算平台（例如 Storm、Flink或者Spark Streaming），去计算实时的一些指标；另一条线进入批量数据处理离线计算平台（例如Mapreduce、Hive，Spark SQL），去计算T+1的相关业务指标，这些指标需要隔日才能看见。

Lambda架构其优点是稳定，对于实时计算部分的计算成本可控，批量处理可以用晚上的时间来整体批量计算，这样把实时计算和离线计算高峰分开，这种架构支撑了数据行业的早期发展，但是它也有一些致命缺点，并在大数据3.0时代越来越不适应数据分析业务的需求。缺点如下：

- 实时与批量计算结果不一致引起的数据口径问题：因为批量和实时计算走的是两个计算框架和计算程序，算出的结果往往不同，经常看到一个数字当天看是一个数据，第二天看昨天的数据反而发生了变化。

- 批量计算在计算窗口内无法完成：在IOT时代，数据量级越来越大，经常发现夜间只有4、5个小时的时间窗口，已经无法完成白天20多个小时累计的数据，保证早上上班前准时出数据已成为每个大数据团队头疼的问题。

- 开发和维护的复杂性问题：Lambda 架构需要在两个不同的 API（application programming interface，应用程序编程接口）中对同样的业务逻辑进行两次编程：一次为批量计算的ETL系统，一次为流式计算的Streaming系统。针对同一个业务问题产生了两个代码库，各有不同的漏洞。这种系统实际上非常难维护

- 服务器存储大：数据仓库的典型设计，会产生大量的中间结果表，造成数据急速膨胀，加大服务器存储压力。

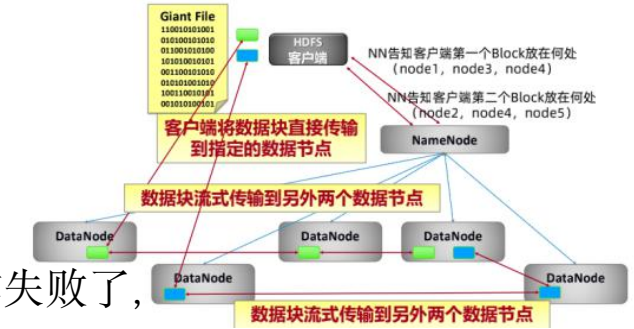
题目三：简答题

◆ HDFS的读写流程

HDFS中的block、packet、chunk

1. block：文件上传前需要分块，这个块就是block，一般为128MB。块大小影响寻址时间的快慢。它是HDFS上最大的一个单位。
2. packet：packet是第二大的单位，它是client端向DataNode，或DataNode的PipLine之间传数据的基本单位，默认64KB。
3. chunk：chunk是最小的单位，它是client向DataNode，或DataNode的PipLine之间进行数据校验的基本单位，默认512Byte。

HDFS写流程



① 客户端向NameNode发出写文件请求。

② 检查是否已存在文件、检查权限。若通过检查，直接先将操作写入EditLog，
并返回输出流对象。（WAL，write ahead log，先写Log，再写内存。
因为EditLog记录的是最新的HDFS客户端执行所有的写操作。如果后续真实写操作失败了，
由于在真实写操作之前，操作就被写入EditLog中了，故EditLog中仍会有记录，
不用担心后续client读不到相应的数据块，因为在第5步中DataNode收到块后会有一返回确认信息，若没写成功，
发送端没收到确认信息，会一直重试，直到成功）

③ client端按128MB的块切分文件。

④ client将NameNode返回的分配的可写的DataNode列表和Data数据一同发送给最近的第一个DataNode节点，此后client端和NameNode分配的多个DataNode构成pipeline管道，client端向输出流对象中写数据。client每向第一个DataNode写入一个packet，这个packet便会直接在pipeline里传给第二个、第三个...DataNode。

⑤ 每个DataNode写完一个块后，会返回确认信息。

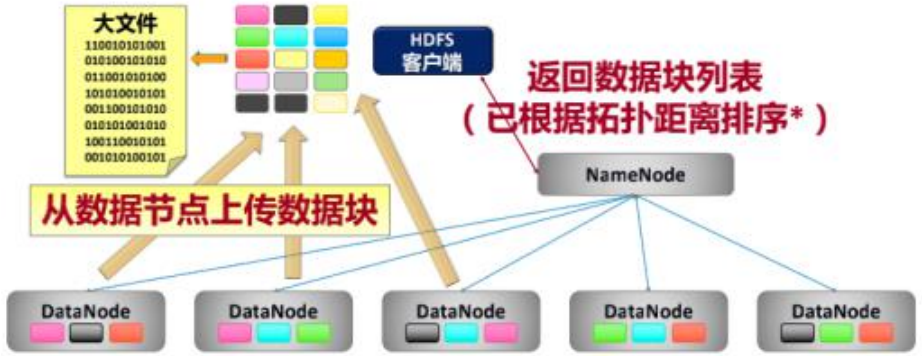
⑥ 写完数据，关闭输输出流。

⑦ 发送完成信号给NameNode。

题目三：简答题

HDFS读流程

① client访问NameNode，查询元数据信息，获得这个文件的数据块位置列表，返回输入流对象。
② 就近挑选一台datanode服务器，请求建立输入流 。
③ DataNode向输入流中中写数据，以packet为单位来校验。
④ 关闭输入流



HDFS读写过程保持数据完整性

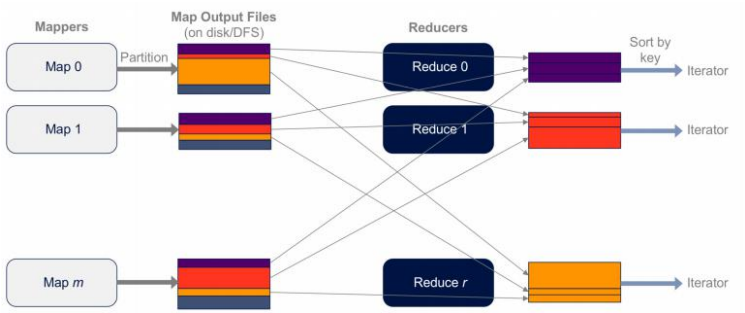
通过校验和保证完整性。因为每个chunk中都有一个校验位，一个个chunk构成packet，一个个packet最终形成block，故可在block上求校验和。

HDFS 的client端即实现了对 HDFS 文件内容的校验和 (checksum) 检查。当客户端创建一个新的HDFS文件时候，分块后会计算这个文件每个数据块的校验和，此校验和会以一个隐藏文件形式保存在同一个 HDFS 命名空间下。当client端从HDFS中读取文件内容后，它会检查分块时候计算出的校验和和读取到的文件块中校验和是否匹配，如果不匹配，客户端可以选择从其他 Datanode 获取该数据块的副本。

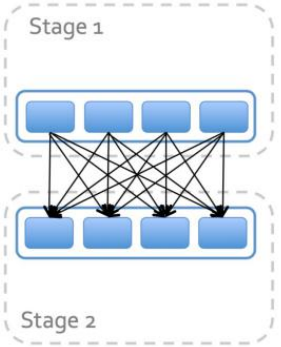
题目三：简答题

◆ Spark Shuffle 的工作原理

有些运算需要将各节点上的同一类数据汇集到某一节点进行计算，把这些分布在不同节点的数据按照一定的规则汇集到一起的过程称为 Shuffle。



- shuffle只发生在两个stage之间
- 在Partition内部重新排布数据



shuffle的过程

1. Map的输出文件写到本地磁盘
2. Reducer把Map的输出文件拉到本地
3. Reducer对输出结果进行排序

shuffle的读写过程

- Shuffle分为shuffle write阶段（map side）和shuffle read阶段（reduce side）
  - Write阶段（map side） 的任务个数是根据RDD的分区数决定的。
    - 假设从HDFS中读取数据，那么RDD分区个数由该数据集的block数决定，也就是一个split对应生成RDD的一个partition。
  - Read阶段（reduce side）的任务个数是通过配置spark.sql.shuffle.partitions决定的。

- Shuffle中间的数据交互
  - Write阶段（map side） 会将状态以及Shuffle文件的位置等信息封装到MapStatue对象中，然后发送给Driver。
  - Read阶段（reduce side）会从Driver拉取MapStatue，解析后开始执行reduce操作。

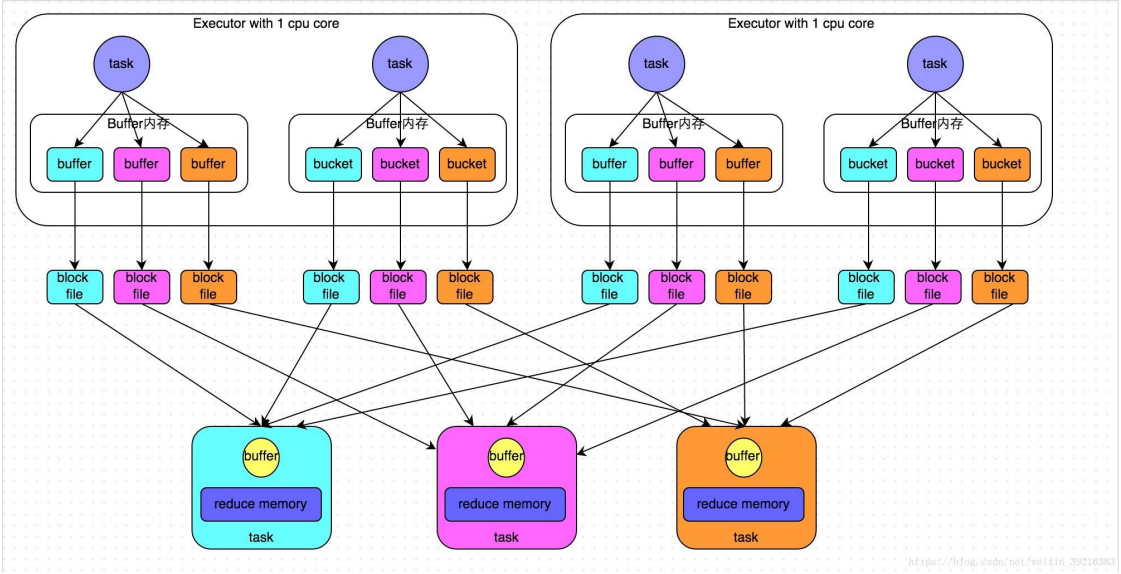
- Spark1.2前使用HashShuffle算法，1.2之后主要使用SortShuffle。

题目三：简答题

Hash Shuffle

Shuffle read阶段，从各个节点上通过网络拉取到reduce任务所在的节点，然后进行key的聚合或连接等操作。一般来说，拉取Shuffle中间结果的过程是一边拉取一边聚合的。每个shuffle read task都会有一个自己的buffer缓冲区，每次只能拉取与buffer缓冲区相同大小的数据，然后在内存中进行聚合。聚合完一批数据后，再拉取下一批数据，直到最后将所有数据到拉取完，得到最终的结果。

Shuffle write阶段，每个task根据记录的Key进行哈希取模操作（hash(key) % reduceNum），相同结果的记录会写到同一个磁盘文件中。会先将数据写入内存缓冲区，当内存缓冲填满之后，才会溢写（spill）到磁盘文件中。



Hash Shuffle存在两个问题

- 生成大量文件，占用文件描述符，同时引入 DiskObjectWriter 带来的 Writer Handler 的缓存也非常消耗内存；
- 如果在 Reduce Task 时需要合并操作的话，会把数据放在一个 HashMap 中进行合并，如果数据量较大，很容易引发 OOM。

Hash Shuffle引入 File Consolidation 机制

将spark.shuffle.consolidateFiles设为true，shuffle write阶段并不会为每个task创建reduceNum个文件，而是一个cpu core具有一个逻辑上shuffleFileGroup，每个Group会生成reduceNum个文件，这样大量减少了shuffle中间文件个数。
但同样，如果一个 Executor 上有 K 个 Core，还是会开 K*N 个 Writer Handler，所以这里仍然容易导致OOM。

题目三：简答题

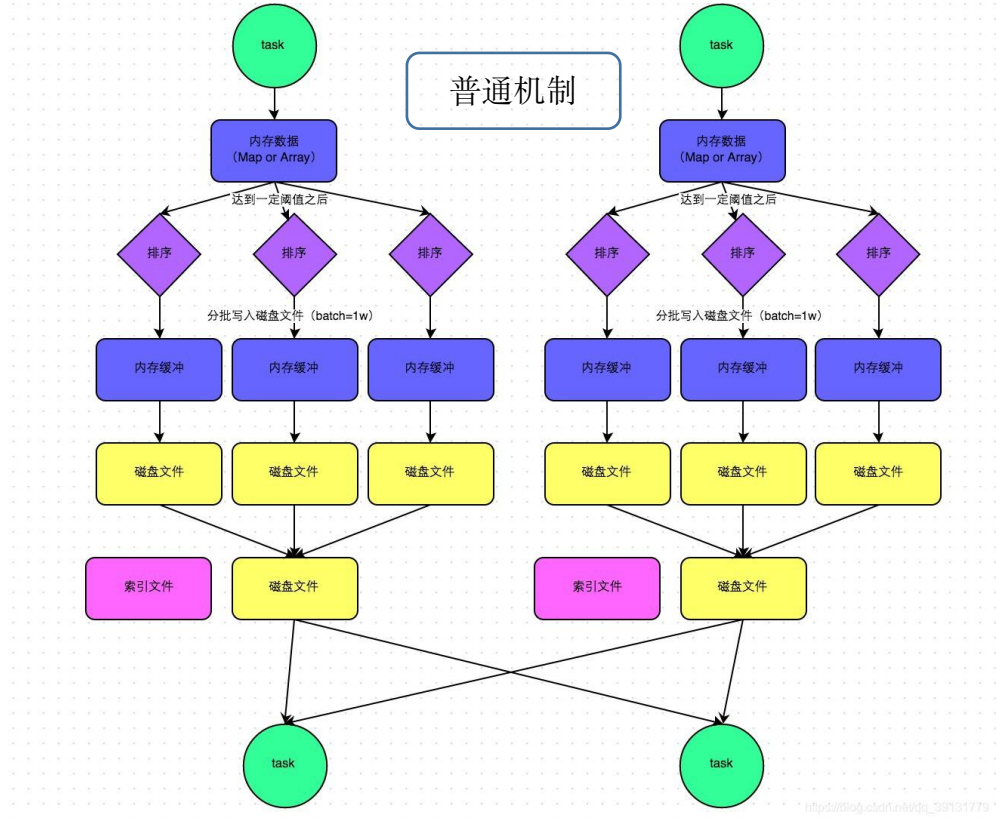
Sort Shuffle

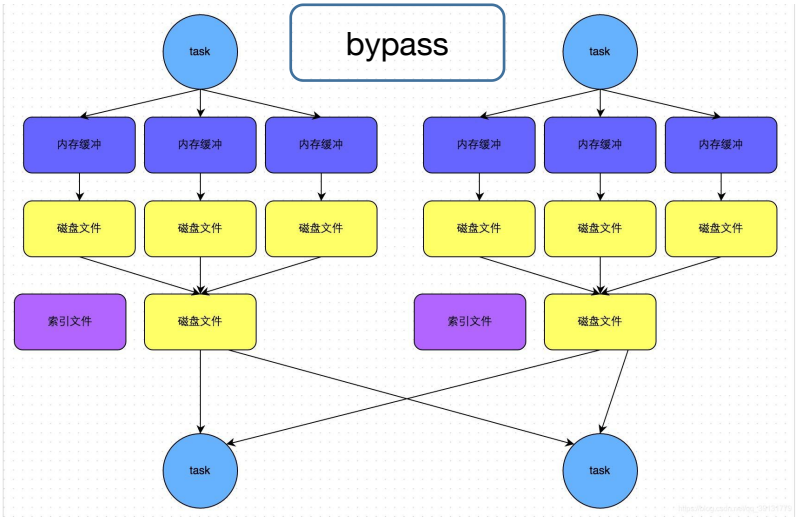
为解决Hash Shuffle的问题，Spark 参考了 MapReduce 中 Shuffle 的处理方式，引入基于排序的 Shuffle 写操作机制。

每个 Task 不会为后续的每个 Task 创建单独的文件，而是将所有对结果写入同一个文件。该文件中的记录首先是按照 Partition Id 排序，每个 Partition 内部再按照 Key 进行排序，Map Task 运行期间会顺序写每个 Partition 的数据，同时生成一个索引文件记录每个 Partition 的大小和偏移量。

在 Reduce 阶段，Reduce Task 拉取数据做 Combine 时不再是采用 HashMap，而是采用ExternalAppendOnlyMap，该数据结构在做 Combine 时，如果内存不足，会刷写磁盘，很大程度的保证了鲁棒性，避免大数据情况下的 OOM。

总体上看来 Sort Shuffle 解决了 Hash Shuffle 的所有弊端，但是因为需要其 Shuffle 过程需要对记录进行排序，所以在性能上有所损失。





Sort Shuffle运行机制又分成两种：

- 普通机制
- bypass机制

BypassSortShuffle的触发条件为：
1. shuffle map task数量小于spark.shuffle.sort.bypassMergeThreshold(默认200)
2. 不是聚合类的shuffle算子（比如reduceByKey）。

此时task会创建reduceNum个临时磁盘文件，并将数据按key进行hash取模，写入对应的磁盘文件。类似HashShuffle，写入磁盘文件时也是先写入内存缓冲，缓冲写满之后再溢写到磁盘文件的。最后，将所有临时磁盘文件都合并成一个磁盘文件，并创建一个单独的索引文件。

该Shuffle会生成大量中间文件，虽然最后都合并了。且不需要对数据进行排序。