

智能数据湖 AI DataLake

最佳实践

文档版本 01
发布日期 2026-10-20



版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：

<https://www.huawei.com/cn/psirt/vul-response-process>

如企业客户须获取漏洞信息，请参见如下网址：

<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

目 录

1 AI DataLake Spark 数据倾斜调优.....	1
1.1 如何判断发生数据倾斜.....	1
1.2 数据倾斜排查及解决方法.....	2
1.2.1 概述.....	2
1.2.2 Join.....	2
1.2.3 Group By.....	4
1.2.4 Count(Distinct).....	6
1.2.5 ROW_NUMBER (TopN)	7
1.2.6 动态分区.....	9
1.2.7 分区过少.....	10

1 AI DataLake Spark 数据倾斜调优

1.1 如何判断发生数据倾斜

数据倾斜多发生在Shuffle阶段。Mapper按InputSplit切分，一般相对均匀。数据倾斜指表中数据分布不均，Shuffle后分配到不同Task（运行在Executor上的工作进程）的数据量差异巨大。数据不均匀时，有的Task很快就计算完成，有的Task却需要运行很长时间。

作业运行表现为：作业执行进度长时间停留在99%，作业执行感觉被卡住；Spark UI上某个Stage的大部分Task已完成，仅剩少数Task长时间运行。

判断步骤

在History Server中通过WebUI判断数据倾斜，具体步骤如下：

1. 在Stages页签中选择运行时间较长的Stage。
2. 在Stage中查看Show Additional Metrics表格中的Duration行观察执行时间是否存在倾斜：比较Max列（执行时间最长的任务）与Median列（执行时间中位数）差距有多大。查看表格中其他指标，观察资源占用是否存在倾斜：比如内存（Peak Execution Memory）、磁盘（Input/Output Size）。
3. 根据Stage编号，可以在DAG图中定位到Stage对应的算子，进而定位导致数据倾斜的SQL代码片段。

使用示例

通过History Server或YARN/集群调度器入口找到对应作业的**Spark UI**链接。进入Spark UI，在**Stages**页签按Duration降序，快速锁定耗时最长的Stage。

例如某Stage的Task统计摘要为：

Duration: min 6s, 25th % 8s, median 13s, 75th % 18s, max 26m40s
Shuffle Read Size: min 2MB, median 15MB, max 12GB

表示该Stage下所有Task的最小运行时长6s，中位数13s，最大运行时长26min40s；Shuffle Read同样存在极端长尾。

在**Tasks**表中按Duration降序，可看到少数几个运行时间显著偏长的Task。切换到**Event Timeline**视图，可直观看到大部分Task很快结束，而个别Task拖成一条很长的“长尾”横条。

对比这些长尾Task与正常Task的Shuffle Read Size / Records：若长尾Task的读取量是平均值的数十倍甚至更高，即可确认为数据倾斜（而非算子本身耗时）。单击该Task查看其stderr/stdout或对应Executor日志，结合Stage DAG中的Shuffle Read上游算子，定位到导致倾斜的key（如 new_uri_path_structure、cookie_x5check_userid、cookie_userid），从而定位到数据倾斜的SQL片段。

1.2 数据倾斜排查及解决方法

1.2.1 概述

引起数据倾斜的主要原因有如下几类，按出现频率排序如下：

- Join
- Group By
- Count (Distinct)
- ROW_NUMBER (TopN)
- 动态分区
- 分区过少

1.2.2 Join

针对Join端产生的数据倾斜，会存在多种不同的情况，例如大表和小表Join、大表和中表Join、Join热点值长尾。

大表 Join 小表

- **数据倾斜示例**

如下示例中t1是一张大表，t2、t3是小表。

```
SELECT t1.ip
      ,t1.is_anon
      ,t1.user_id
      ,t1.user_agent
      ,t1.referer
      ,t2.ssl_ciphers
      ,t3.shop_province_name
      ,t3.shop_city_name
FROM   viewtable t1
LEFT OUTER JOIN other_viewtable t2
      ON t1.header_eagleeye_traceid = t2.eagleeye_traceid
LEFT OUTER JOIN (
      SELECT shop_id
            ,city_name AS shop_city_name
            ,province_name AS shop_province_name
      FROM   tenanttable
      WHERE  ds = (SELECT max(ds) FROM tenanttable)
      AND    is_valid = 1
    ) t3
      ON t1.shopid = t3.shop_id
```

- **解决方案**

使用Hint（查询优化提示指令）语法，广播小表，如下所示。

```
SELECT /*+ BROADCAST(t2, t3) */
      t1.ip
      ,t1.is_anon
      ,t1.user_id
```

```
,t1.user_agent
,t1.referer
,t2.ssl_ciphers
,t3.shop_province_name
,t3.shop_city_name
FROM viewtable t1
LEFT OUTER JOIN other_viewtable t2
ON t1.header_eagleeye_traceid = t2.eagleeye_traceid
LEFT OUTER JOIN (
SELECT shop_id
,city_name AS shop_city_name
,province_name AS shop_province_name
FROM tenanttable
WHERE ds = (SELECT max(ds) FROM tenanttable)
AND is_valid = 1
) t3
ON t1.shopid = t3.shop_id
```

- **注意事项**
 - Spark支持自动触发广播，阈值配置参数为 spark.sql.autoBroadcastJoinThreshold=10485760（默认值为10MB）。手动 Hint则不受阈值参数限制，需要注意广播引起的Driver/Executor内存膨胀问题。
 - 引用小表或子查询时，需要引用别名。

Join 热点值长尾

- **数据倾斜示例**

在下面这个表中，eleme_uid中存在很多热点数据，容易发生数据倾斜。

```
SELECT
  eleme_uid,
  ...
FROM (
  SELECT
    eleme_uid,
    ...
  FROM <viewtable>
) t1
LEFT JOIN (
  SELECT
    eleme_uid,
    ...
  FROM <customertable>
) t2
ON t1.eleme_uid = t2.eleme_uid;
```

- **解决方案**

可以通过如下三种方法解决。

序号	方案	说明
方案一	手动切分热点值	将热点值分析出来后，从主表中过滤出热点值记录先进行广播（Broadcast Join），剩余非热点记录走Sort Merge Join，最后UNION ALL。
方案二	AQE Skew Join / SkewJoin Hint	AQE自动检测倾斜key拆分；也可手动用 /*+ SKEW */ Hint。
方案三	倍数表取模相等Join	利用倍数表将热点值放大N倍，然后Join时使用用户ID和number两个关联键。

- 手动切分热点值。

将热点值分析出来后，从主表中过滤出热点值记录，先进行广播，再将剩余非热点值记录进行Sort Merge Join，最后合并两部分的Join结果。具体可以参考如下代码示例：

```
-- 热点值走Broadcast Join
SELECT /*+ BROADCAST(t2) */
      eleme_uid, ...
FROM (SELECT eleme_uid, ... FROM viewtable WHERE eleme_uid = <skewed_value>) t1
LEFT JOIN (SELECT eleme_uid, ... FROM customertable WHERE eleme_uid = <skewed_value>) t2
      ON t1.eleme_uid = t2.eleme_uid
UNION ALL

-- 非热点值走Sort Merge Join
SELECT eleme_uid, ...
FROM (SELECT eleme_uid, ... FROM viewtable WHERE eleme_uid != <skewed_value>) t3
LEFT JOIN (SELECT eleme_uid, ... FROM customertable WHERE eleme_uid != <skewed_value>) t4
      ON t3.eleme_uid = t4.eleme_uid
```

- AQE Skew Join / SkewJoin Hint。

优先推荐开启AQE Skew Join，配置参数如下：

```
spark.sql.adaptive.enabled=true
spark.sql.adaptive.skewJoin.enabled=true
```

Spark也支持手动Hint（查询优化提示指令）指定倾斜值，使用示例如下：

```
SELECT /*+ SKEW('a', 'c0', (1, '2'), (3, '4')) */ *
FROM T0 a JOIN T1 b
      ON a.c0 = b.c0 AND a.c1 = b.c1 AND a.c2 = b.c2;
```

支持的Join类型：

- Inner Join可以对Join两侧表中的任意一侧进行Hint。
- Left Join、Semi Join和Anti Join只可以Hint左侧表。
- Right Join只可以Hint右侧表。
- Full Join不支持Skew Join Hint。

建议只对一定会出现数据倾斜的Join添加Hint，因为Hint会运行一个Aggregate，存在一定代价。Join key类型必须一致（两侧字段类型相同），否则Hint不生效；可通过CAST在子查询中统一类型。

- 倍数表取模相等Join

该方案和前两个方案的逻辑不同，不是分而治之的思路，而是利用一个倍数表，其值只有一列：int列，比如可以从1到N（具体可根据倾斜程度确定），利用这个倍数表可以将用户行为表放大N倍，然后Join时使用用户ID和number两个关联键。这样原先只按照用户ID分发导致的数据倾斜就会由于加入了number关联条件而减少为原先的1/N。但是这样做也会导致数据膨胀N倍。

```
SELECT eleme_uid, ...
FROM (SELECT eleme_uid, ... FROM viewtable) t1
LEFT JOIN (SELECT /*+ BROADCAST(multipletable) */
          eleme_uid, number, ...
          FROM customertable JOIN multipletable) t2
      ON t1.eleme_uid = t2.eleme_uid
      AND mod(t1.value_col, 10) + 1 = t2.number
```

1.2.3 Group By

一个带Group By的伪代码示例如下。

```
SELECT shop_id
      ,sum(is_open) AS 营业天数
```

```
FROM table_xxx_di
WHERE dt BETWEEN '${bizdate_365}' AND '${bizdate}'
GROUP BY shop_id;
```

当发生数据倾斜时，可以通过如下两种方案解决：

序号	方案	说明
方案一	添加随机数	把引起长尾的key进行拆分。
方案二	创建滚存表	降本提效。

方案一：添加随机数。

对SQL进行改写，添加随机数，把引起长尾的key进行拆分是解决Group By长尾的一种比较好的方法。

```
SELECT Key, Count(*) AS Cnt FROM TableName GROUP BY Key;
```

不考虑Combiner，M节点会Shuffle到R节点上，然后R节点再做Count操作，对应的执行计划是M->R。

假定已经找到了引起长尾的key，对长尾的key再做一次工作再分配，就变成：

```
-- 假设长尾的Key已经找到是KEY001
SELECT a.Key
      ,SUM(a.Cnt) AS Cnt
FROM (
  SELECT Key
        ,COUNT(*) AS Cnt
  FROM   TableName
  GROUP BY Key
        ,CASE WHEN Key = 'KEY001' THEN pmod(hash(rand()), 50)
        ELSE 0
  END
) a
GROUP BY a.Key;
```

改完之后的执行计划变成了 M->R->R，虽然执行步骤变长了，但是长尾的key经过了 2 个步骤的处理，整体的时间消耗可能反而有所减少。与不加随机数的直接聚合相比，资源消耗与耗时效果基本持平；但实际场景中引发长尾的key不止一个，再考虑寻找长尾key和SQL改写的投入成本，直接对长尾key添加随机数（不做两阶段改写）的投入成本会更低一些。

方案二：创建滚存表。

降本提效，核心诉求是取过去一年的商户数据，对于线上任务而言，每次都要读取T-1至T-365的所有分区其实是对资源的很大浪费，创建滚存表可以减少分区的读取但是又不影响过去一年的取数，示例如下。

首次初始化365天的商户营业数据（Group By汇总），标记数据更新日期，记为表a；后续线上任务切换为T-2日表a关联table_xxx_di表再Group By，这样每天读取的数据从365减少到了2个，主键shopid的重复性极大降低，对资源的消耗也会减少。

```
-- 创建滚存表
CREATE TABLE IF NOT EXISTS m_xxx_365_df (
  shop_id STRING COMMENT '商户ID',
  last_update_ds STRING COMMENT '最近更新日期',
  `365d_open_days` BIGINT COMMENT '365天营业天数'
)
PARTITIONED BY (ds STRING COMMENT '日期分区');
```

```
-- 假定 365d 是 2021.5.1-2022.5.1，先完成一次初始化
INSERT OVERWRITE TABLE m_xxx_365_df PARTITION(ds = '20220501')
SELECT shop_id
      ,max(ds) AS last_update_ds
      ,sum(is_open) AS `365d_open_days`
FROM table_xxx_di
WHERE ds BETWEEN '20210501' AND '20220501'
GROUP BY shop_id;

-- 那么之后线上任务要执行的是
INSERT OVERWRITE TABLE m_xxx_365_df PARTITION(ds = '${bizdate}')
SELECT aa.shop_id
      ,aa.last_update_ds
      ,`365d_open_days` - COALESCE(is_open, 0) AS `365d_open_days` -- 消除营业天数的无限滚存
FROM (
  SELECT shop_id
        ,max(last_update_ds) AS last_update_ds
        ,sum(`365d_open_days`) AS `365d_open_days`
  FROM (
    SELECT shop_id
          ,ds AS last_update_ds
          ,sum(is_open) AS `365d_open_days`
    FROM table_xxx_di
    WHERE ds = '${bizdate}'
    GROUP BY shop_id
    UNION ALL
    SELECT shop_id
          ,last_update_ds
          ,`365d_open_days`
    FROM m_xxx_365_df
    WHERE ds = '${bizdate_2}' AND last_update_ds >= '${bizdate_365}'
    GROUP BY shop_id
  )
  GROUP BY shop_id
) AS aa
LEFT JOIN (
  SELECT shop_id, is_open
  FROM table_xxx_di
  WHERE ds = '${bizdate_366}'
) AS bb
ON aa.shop_id = bb.shop_id;
```

1.2.4 Count(Distinct)

假如一个表数据分布如下。

ds (分区)	cnt (记录数)
20220416	73025514
20220415	2292806
20220417	2319160

使用下面的语句就容易发生数据倾斜：

```
SELECT
  ds,
  COUNT(DISTINCT shop_id) AS cnt
FROM
  demo_data0
GROUP BY
  ds;
```

解决方案如下：

序号	方案	说明
方案一	通用两阶段聚合	在partition字段值拼接随机数。
方案二	类似两阶段聚合	先通过Group By两分组字段(ds+shop_id)再使用Count(Distinct)。

方案一：通用两阶段聚合。

若shop_id字段数据分布均匀，可先按ds、shop_id两字段分组，再对ds做Count(Distinct)；若shop_id字段数据不均匀（存在热点值），简单的两阶段聚合仍会倾斜，需在分区字段值中拼接随机数打散热点，具体见以下方式1、方式2。

```
-- 方式1：拼接随机数
CONCAT(ROUND(rand(),1)*10,'_', ds) AS rand_ds
SELECT split_part(rand_ds, '_', 2) AS ds
      ,COUNT(*) AS id_cnt
FROM (
  SELECT rand_ds, shop_id
  FROM demo_data0
  GROUP BY rand_ds, shop_id
)
GROUP BY split_part(rand_ds, '_', 2);

-- 方式2：新增随机数字段
round(rand()*10) AS randint10
SELECT ds
      ,COUNT(*) AS id_cnt
FROM (
  SELECT ds, randint10, shop_id
  FROM demo_data0
  GROUP BY ds, randint10, shop_id
)
GROUP BY ds;
```

方案二：类似两阶段聚合。

如果GroupBy与Distinct的字段数据都均匀，则可以采用如下方式优化，先Group By两分组字段（ds和shop_id）再使用Count(Distinct)命令。

```
SELECT ds
      ,COUNT(*) AS cnt
FROM (
  SELECT ds, shop_id
  FROM demo_data0
  GROUP BY ds, shop_id
)
GROUP BY ds;
```

1.2.5 ROW_NUMBER（TopN）

Top10的示例如下。

```
SELECT main_id, type
FROM (
  SELECT main_id
        ,type
        ,ROW_NUMBER() OVER(PARTITION BY main_id ORDER BY type DESC) AS rn
  FROM data_demo2
) A
WHERE A.rn <= 10;
```

当发生数据倾斜时，可以通过以下几种方式解决：

序号	方案	说明
方案一	SQL写法的两阶段聚合。	增加随机列或拼接随机数，将其作为分区（ Partition ）中一参数。
方案二	UDAF写法的两阶段聚合。	通过UDAF实现最小堆（ TopN ）的方式进行调优。

方案一：SQL写法的两阶段聚合。

为使Map阶段中Partition各分组数据尽可能均匀，增加随机列，将其作为Partition中一参数。

```
-- 1. 随机数基于源表行分布
SELECT main_id, type
FROM (
  SELECT main_id, type
    ,ROW_NUMBER() OVER(PARTITION BY main_id ORDER BY type DESC) AS rn
  FROM (
    SELECT main_id, type
    FROM (
      SELECT main_id
        ,type
        ,ROW_NUMBER() OVER(PARTITION BY main_id, src_pt ORDER BY type DESC) AS rn
      FROM (
        SELECT main_id, type
          ,CAST(ceil(110 * rand()) % 11 AS INT) AS src_pt
        FROM data_demo2
      )
    ) B
    WHERE B.rn <= 10
  )
) A
WHERE A.rn <= 10;
-- 2. 随机数直接自定义
SELECT main_id, type
FROM (
  SELECT main_id, type
    ,ROW_NUMBER() OVER(PARTITION BY main_id ORDER BY type DESC) AS rn
  FROM (
    SELECT main_id, type
    FROM (
      SELECT main_id, type
        ,ROW_NUMBER() OVER(PARTITION BY main_id, src_pt ORDER BY type DESC) AS rn
      FROM (
        SELECT main_id, type
          ,CAST(ceil(10 * rand()) AS INT) AS src_pt
        FROM data_demo2
      )
    ) B
    WHERE B.rn <= 10
  )
) A
WHERE A.rn <= 10;
```

方案二：UDAF写法的两阶段聚合。

Spark推荐用窗口函数 + **approx_count_distinct**或**collect_set**限制，或直接借助groupBy().agg(collect_list()) + UDF取 TopN。

```
# 用Spark内置算子组合实现（更地道，避免UDAF）
from pyspark.sql import functions as F
```

```
from pyspark.sql.window import Window
# Step1: 增加随机列
df1 = df.withColumn("src_pt", F.ceil(F.rand()*10))
# Step2: 在打散后的分区取TopN
w1 = Window.partitionBy("main_id", "src_pt").orderBy(F.col("type").desc())
df2 = df1.withColumn("rn", F.row_number().over(w1)).filter("rn <= 10")
# Step3: 合并后再次取TopN
w2 = Window.partitionBy("main_id").orderBy(F.col("type").desc())
result = df2.withColumn("rn2", F.row_number().over(w2)).filter("rn2 <= 10").drop("rn", "rn2", "src_pt")
```

1.2.6 动态分区

动态分区是指在往分区表里插入数据时，可以在分区中指定一个分区列名，但不给出具体值，而是在SELECT子句中的对应列来提供分区值的一种语法。在SQL运行之前不知道会产生哪些分区，只有在SQL语句运行结束后，才能根据分区列产生的值确定会产生哪些分区。

根据动态分区插数据时Map阶段就存在的数据倾斜问题，可通过查找到存在记录数较多的分区裁剪后单独插入的方式解决，具体步骤如下。

1. 使用如下命令示例查询记录数较多的特定分区。

```
SELECT ds, hh, COUNT(*) AS cnt
FROM   dwd_alsc_ent_shop_info_hi
GROUP BY ds, hh
ORDER BY cnt DESC;
```

部分分区如下：

ds	hh	cnt
20200928	17	1052800
20191017	17	1041234
20210928	17	1034332
20190328	17	1000321
20210504	1	19
20191003	20	18
20200522	1	18
20220504	1	18

2. 使用如下命令示例过滤上述分区插入后，再单独插入大记录数分区数据。

```
-- 小分区批量插入
INSERT OVERWRITE TABLE data_demo3 PARTITION(ds, hh)
SELECT *
FROM   dwd_alsc_ent_shop_info_hi
WHERE  concat(ds, hh) NOT IN ('2020092817','2019101717','2021092817','2019032817');

-- 大分区单独插入（可配合 repartition 控制并发）
INSERT OVERWRITE TABLE data_demo3 PARTITION(ds, hh)
SELECT *
FROM   dwd_alsc_ent_shop_info_hi
WHERE  concat(ds, hh) IN ('2020092817','2019101717','2021092817','2019032817');

-- 验证
SELECT ds, hh, COUNT(*) AS cnt
FROM   dwd_alsc_ent_shop_info_hi
GROUP BY ds, hh
ORDER BY cnt DESC;
```

1.2.7 分区过少

非热点key导致的数据倾斜场景，当作业分区设置过少时，每个task处理数据量较大，从而产生长尾作业，可以考虑调大分区数。

- 开启自适应查询执行（Adaptive Query Execution，简称AQE）场景（默认场景）：可以调大`spark.sql.adaptive.coalescePartitions.minPartitionNum`，同时设置`spark.sql.adaptive.coalescePartitions.initialPartitionNum`为可用的core数。
- 关闭AQE场景：可以调大`spark.sql.shuffle.partitions`。