《Hive 数据仓库应用教程》

模块八 Hive 优化(yōu huà)

—— 讲义原稿 ——

课程内容：

任务 8-1 Hive 参数优化

任务 8-2 HQL 优化

任务 8-3 本章小结

任务 8-4 本章习题

【幻灯片 1】模块八 Hive 优化 — 课程导入

同学们好！欢迎来到我们 Hive 数据仓库课程的第八章。

今天我们要学习一个非常重要的话题——Hive 优化(yōu huà)。

大家想一想，我们之前学了很多 Hive 的操作，比如建表、查询、分区(fēn qū)等等。
但是在实际工作中，数据量可能非常大，几百万、几千万甚至上亿条数据。
如果我们不做优化，一个查询可能要跑很长时间，几分钟甚至几个小时。

所以，今天我们就来学习怎么让 Hive 跑得更快、效率更高。

打个比方：优化就像我们开车上高速，不优化就像走乡间小路，同样的目的地(mù dì dì)，时间差别很大。

本章分为两大部分：
第一部分：Hive 参数优化 —— 就是通过调整(tiáo zhěng)Hive 的配置参数来提高性能。
第二部分：HQL 优化 —— 就是通过优化我们写的 SQL 语句来提高性能。

【幻灯片 2】本章目录

我们先来看一下这一章的整体内容安排：

1. 任务 8-1　Hive 参数优化

→ 学习怎么通过设置参数让 Hive 运行更快

2. 任务 8-2　HQL 优化

→ 学习怎么写出更高效的 HQL 语句

3. 任务 8-3　本章小结

→ 对本章内容做一个总结回顾(huí gù)

4. 任务 8-4　本章习题

→ 课后思考和练习

好，我们先从第一个任务开始。

【幻灯片 3】任务 8-1 Hive 参数优化

现在我们进入第一个任务：Hive 参数优化。

什么是参数优化呢？

简单来说，Hive 有很多可以调整的设置项，就像手机的设置一样。
我们可以根据不同的情况，调整这些设置，让 Hive 运行得更好。

比如说，你的手机可以调亮度(liàng dù)、调音量。
Hive 也一样，可以调 Map 的个数、Reduce 的个数等等。

接下来我们就来详细学习这些参数怎么调。

【幻灯片 4】8.1.1 任务描述(rèn wù miáo shù)

在大数据技术生态系统(shēng tài xì tǒng)中，Hive 参数调优在实际运行中经常会遇到。

为什么需要参数优化呢？原因很简单：

在企业(qǐ yè)里，数据量是不断增长的。

今天可能是 100 万条数据，明天就变成 1000 万条。

如果我们的 Hive 配置不变，数据多了以后，查询速度就会越来越慢。

所以，我们需要根据实际情况来调整 Hive 的参数。

★ Hive 拥有很多优化特性(tè xìng)，如何利用好这些特性，是参数调优的关键(guān jiàn)。

举个简单的例子：

如果你要搬家，搬的东西很少，自己开一辆小车就够了；

但如果东西很多，你就需要叫一辆大卡车。

Hive 参数优化就是帮你选择合适的方式来完成任务。

【幻灯片 5-6】8.1.2 任务准备知识

在开始实验之前，我们需要做一些准备工作。

为了不影响(yǐng xiǎng)前面章节的数据，我们要新建一个数据库，叫做 unit8db。

本章所有的实验操作，都在这个数据库里进行。

重要提示：

- 本地文件资源存储(cún chǔ)在 /usr/datas8 目录中
- HDFS 文件系统中的资源存储在 /tmp 目录中

例题 1：创建数据库 unit8db

操作过程很简单，就三步：

第一步(step01)：创建数据库

实验操作: 创建数据库 unit8db

```
hive> CREATE DATABASE IF NOT EXISTS unit8db;
```

第二步(step02)：查看数据库是否创建成功

实验操作: 查看数据库

```
hive> SHOW DATABASES;
```

第三步(step03)：使用(切换到)数据库 unit8db

实验操作: 使用数据库 unit8db

```
hive> USE unit8db;
```

大家看图 8-1 的截图(jié tú)，可以看到：

- 输入 SHOW DATABASES 命令后，在列表里可以看到 unit8db
- 输入 USE unit8db 命令后，提示符(tí shì fú)变成了 hive (unit8db)>
- 这说明我们已经成功切换到 unit8db 数据库了

★ 注意：每次打开 Hive，都要先 USE unit8db; 才能操作本章的表！

【幻灯片 7】8.1.3 任务实施 — Hive 参数优化概览(gài lǎn)

Hive 参数优化主要有以下五种方式：

一、本地模式(běn dì mó shì)

二、配置 MapReduce 压缩(yā suō)

三、配置 Map 个数

四、配置 Reduce 个数

五、合并(hé bìng)小文件

我们先来学习第一种：本地模式。

一、本地模式

什么是本地模式？让我用一个例子来解释(jiě shì)：

假设(jiǎ shè)你要复印(fù yìn)一张纸。

正常情况下，你去复印店(默认模式) —— 需要出门、排队、等待。

但如果你家里就有打印机(本地模式) —— 直接在家里就能完成，快多了！

Hive 的本地模式也是一样的道理：

当数据量很小的时候，不需要把任务提交到 Hadoop 集群(jí qún)上去跑。

直接在本机上运行就可以了，这样省去了任务提交和调度(diào dù)的时间。

【幻灯片 8-9】例题 1：本地模式 vs 默认模式

例题 1：使用本地模式和默认模式分别查询表 bigtable

第一步(step01)：创建表 bigtable 并加载数据

实验操作: 创建表 bigtable

```
hive> CREATE TABLE bigtable
  (id bigint, t bigint, uid string, keyword string,
   url_rank int, click_num int, click_url string)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

实验操作: 加载数据到 bigtable

```
hive> LOAD DATA LOCAL INPATH '/usr/datas8/bigtable001.csv'
  INTO TABLE bigtable;
```

大家看图 8-2 和图 8-3，表 bigtable 中共有 320 条数据。

图中显示了前 15 条数据的样子，有 id、时间、用户名、关键词等字段。

第二步(step02)：使用默认模式查询

先看一下当前的本地模式设置：

实验操作: 查看本地模式设置

```
hive> SET hive.exec.mode.local.auto;
```

然后执行 count 查询：

实验操作: 默认模式下执行查询

```
hive> SELECT count(*) FROM bigtable;
```

大家看图 8-4 和图 8-5 的截图:

用默认模式查询，时间是 31.658 秒(miǎo)!

这么小的数据就跑了 30 多秒，是不是很慢?

【幻灯片 10-12】例题 1 续：本地模式对比结果

第三步(step03)：开启本地模式，再次查询

实验操作: 开启本地模式

```
hive> SET hive.exec.mode.local.auto=true;
```

实验操作: 本地模式下执行查询

```
hive> SELECT count(*) FROM bigtable;
```

大家看图 8-6 和图 8-7 的截图：

用本地模式查询，时间只有 **13.028 秒**！

对比一下：

- 默认模式：31.658 秒
- 本地模式：13.028 秒
- 快了将近(jiāng jìn)一半多！

★ 注意看图 8-6 中的提示：Job running in-process (Local Hadoop)

这说明任务是在本地运行的，没有提交到集群上去。

本地模式小结：

从 Hive 0.7 版本开始，引入了本地模式。

对于小数据量的任务，Hive 客户端不需要到 YARN 上申请 Map 和 Reduce 任务。

只在本地执行，可以大大加快执行速度。

通常可以把分钟级别(jí bié)的任务降低到秒级。

★ 记住：数据量小的时候，开启本地模式可以显著(xiǎn zhù)提高速度！

【幻灯片 13-14】二、配置 MapReduce 压缩

二、配置 MapReduce 压缩(yā suō)

什么是压缩？就像我们用 WinRAR 压缩文件一样。

文件变小了，传输(chuán shū)起来就更快，存储空间也更小。

MapReduce 中有三个地方可以用压缩：

1. Map 之前（输入端采用压缩）

2. Map 之后（Mapper 输出采用压缩）

3. Reduce 之后（Reducer 输出采用压缩）

压缩的原则(yuán zé)：

1) 运算密集型(mì jí xíng)的 Job —— 少用压缩（因为压缩也需要 CPU 计算）

2) IO 密集型的 Job —— 多用压缩（因为压缩后数据变小，读写更快）

压缩性能对比（表 8-3）：

同样是 8.3GB 的原始文件：

- Gzip：压缩速度 17.5MB/s，解压速度 58MB/s
- bzip2：压缩速度 2.4MB/s，解压速度 9.5MB/s（最慢！）
- LZO：压缩速度 49.3MB/s，解压速度 74.6MB/s（最快！）

打个比方：LZO 像高铁(gāo tiě)，Gzip 像普通火车，bzip2 像绿皮火车。

这里有个重要概念(gài niàn)叫"可切片(kě qiē piàn)"：
如果一个压缩文件可以切片，多个 Map 任务可以同时处理这个文件的不同部分。
bzip2 和 LZO 支持切片，Gzip 和 Snappy 不支持。

【幻灯片 15-17】压缩参数配置与选择策略(cè lüè)

压缩算法怎么选?

需要考虑三个因素(yīn sù)：压缩/解压速度、压缩率、是否可以切片

大家看图 8-8（MapReduce 工作原理图），图中画了一条流程(liú chéng)：

输入端采用压缩 → Map → Mapper 输出采用压缩 → Reduce → Reducer 输出采用压缩

Map 之前:

数据量小 → 选 Snappy 或 LZO（追求速度）

数据量大 → 选 LZO 或 bzip2（需要切片功能）

Map 之后:

主要考虑速度 → 选 Snappy 或 LZO

Reduce 之后:

数据量小 → 选 Snappy 或 LZO

数据量大 → 选 LZO 或 bzip2

★ 企业中最常用的是 Snappy 和 LZO，因为速度快!

【幻灯片 18-22】三、配置 Map 和 Reduce 个数

三、配置 Map 和 Reduce 个数

Map 任务的个数由什么决定?

主要取决于(qǔ jué yú)三个因素：

1. 输入文件的总个数

2. 输入文件的大小

3. 集群设置的文件块(kuài)大小（默认 128MB）

★ 注意：Map 数并不是越多越好!

情况一：Map 太多

100 个小文件会启动 100 个 Map，启动时间比处理时间还长!

就像开 100 辆出租车去拉 100 个乘客(chéng kè)，每辆车只坐一个人，太浪费(làng fèi)了。

情况二：Map 太少

一个 127MB 的文件有几千万条记录(jì lù)，只用一个 Map 处理太慢。

就像只开一辆公交车去拉 1000 个人，太挤(jǐ)了。

例题 2：使用增加 Map 数的方式给表 emp 调优

step01：创建表 emp 并加载数据

实验操作: 创建表 emp

```
hive> CREATE TABLE IF NOT EXISTS emp(
  empno int, ename string, job string, mgr int,
  hiredate string, sal double, comm double, deptno int)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

实验操作: 加载数据到 emp

```
hive> LOAD DATA LOCAL INPATH '/usr/datas8/emp.txt'
  INTO TABLE emp;
```

step02：先用默认设置查询

实验操作: 默认设置查询 emp

```
hive> SELECT count(*) FROM emp;
```

看图 8-10，默认情况下只有 1 个 mapper。

step03：设置最大切片值为 100 字节，再查询

实验操作: 设置最大切片值并查询

```
hive> SET mapreduce.input.fileinputformat.split.maxsize=100;
hive> SELECT count(*) FROM emp;
```

看图 8-11，现在有 7 个 mapper 了！maxsize 越小，切片越多，Map 越多。

★ 总结：通过调整 maxsize 参数，可以控制 Map 的数量。

【幻灯片 23-27】小文件合并与 Reduce 数配置

例题 3：小文件进行合并

小文件问题是 Hadoop 中一个很常见的问题。太多小文件会导致(dǎo zhì)很多 Map 启动，浪费资源。

1) Map 执行前合并小文件：

实验操作: 设置 Map 前合并小文件

```
hive> SET hive.input.format=
  org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

2) Map-Reduce 任务结束时合并小文件：

实验操作: 设置任务结束时合并

```
-- map-only 任务结束时合并小文件（默认 true）
hive> SET hive.merge.mapfiles=true;
-- map-reduce 任务结束时合并小文件（默认 false）
hive> SET hive.merge.mapredfiles=true;
-- 合并文件的大小（默认 256M）
hive> SET hive.merge.size.per.task=268435456;
-- 平均文件大小小于此值时启动合并（默认 16M）
hive> SET hive.merge.smallfiles.avgsize=16777216;
```

例题 4：合理设置 Reduce 数

Reduce 个数也需要合理配置，不要太多也不要太少。

方法一：通过数据量参数自动计算

实验操作: 方法一：自动计算 Reduce 数

```
-- 每个 Reduce 处理的数据量（默认 256MB）
hive.exec.reducers.bytes.per.reducer=256000000
-- 每个任务最大 Reduce 数（默认 1009）
hive.exec.reducers.max=1009
-- 计算公式：N = min(参数 2, 总输入数据量/参数 1)
```

方法二：直接设置 Reduce 个数

实验操作: 方法二：手动设置 Reduce 数

```
hive> SET mapreduce.job.reduces = 15;
```

★ 注意：Reduce 个数太多会启动开销(kāi xiāo)大，还会产生很多小文件！

【幻灯片 28-31】四、合并小文件（详细）

四、合并小文件

为什么小文件问题这么严重(yán zhòng)？三个原因：

原因一：影响 NameNode 性能

每个文件大约占用 150 字节内存(nèi cún)。文件达到千万级时，NameNode 内存超过 3GB。

原因二：影响 HDFS 读写

每次读写都要先查询元数据(yuán shù jù)。文件太多，查询就慢。

原因三：影响 Hive 任务执行

每个小文件启动一个 MapTask，大量 MapTask 启动时间远大于处理时间，甚至导致 OOM。

例题 5：合并小文件的三种方式

方式一：输入合并

实验操作: 输入合并设置

```
hive> SET hive.input.format=
  org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

```
hive> SET mapred.max.split.size=256000000;
hive> SET mapred.min.split.size.per.node=100000000;
hive> SET mapred.min.split.size.per.rack=100000000;
```

方式二：输出合并

实验操作: 输出合并设置

```
hive> SET hive.merge.mapredfiles=true;
hive> SET hive.merge.mapfiles=true;
hive> SET hive.merge.smallfiles.avgsize=16000000;
hive> SET hive.merge.size.per.task=256000000;
```

方式三：JVM 重用(chóng yòng)

一个 Container 中运行多个 MapTask，减少 Container 开启和注销(zhù xiāo)时间。

实验操作: JVM 重用设置

```
hive> SET mapred.job.jvm.num.tasks;
```

【幻灯片 32-34】任务 8-2 HQL 优化

现在我们进入第二个大的部分：HQL 优化。

前面我们学的是通过调参数来优化 Hive。

现在我们要学的是通过优化 SQL 语句本身来提高性能。

Hive 面临的一个最大挑战(tiǎo zhàn)是：

与传统关系数据库相比，Hive 的查询速度通常慢得多。

打个比方：同样是查数据，MySQL 可能几毫秒(háo miǎo)就出结果了。

但 Hive 可能需要几秒甚至几分钟。所以，写好 HQL 语句非常重要！

HQL 优化主要有四种方式：

一、Group By 优化

二、Join 优化

三、Count(Distinct) 优化

四、行/列过滤(guò lǜ)优化

【幻灯片 35-39】一、Group By 优化

一、Group By 优化

首先要了解一个重要概念：数据倾斜(shù jù qīng xié)。

什么是数据倾斜？举个例子：

一个班级(bān jí)有 100 个学生，分成两组做实验。

如果第一组有 90 个人，第二组只有 10 个人，这就是倾斜。

第一组太忙(máng)了，第二组太闲(xián)了。

在 Hive 中也一样：

默认情况下，map 阶段同一 key 的数据会分发给同一个 reduce。

如果某个 key 的数据特别多，那个 reduce 就会特别慢。

例题 6：Group By 优化案例

假设表 test 有 10 条数据：

张三,10 张三,20 张三,30 张三,10 张三,20

张三,30 张三,20 张三,30 李四,30 李四,40

这里张三有 8 条，李四只有 2 条，数据明显不均匀(bù jūn yún)！

常用参数：

实验操作: Group By 优化参数

```
-- 是否在 Map 端进行聚合（默认 True）
SET hive.map.aggr=true;
-- Map 端聚合的条目数
SET hive.groupby.mapaggr.checkinterval=100000;
-- 数据倾斜时进行负载均衡（默认 false）
SET hive.groupby.skewindata=true;
```

负载均衡(fù zài jūn héng)策略怎么工作?

开启负载均衡后，会生成两个 MR Job：

第一个 MR Job：

Map 端随机(suí jī)把数据分发给 Reduce 端。注意是随机分发，不是按 key 分发！

这样每个 Reduce 收到的数据量就比较均匀了。

第二个 MR Job：

根据第一个 Job 的预处理结果，按照 Group By Key 再次分发数据完成分组。

大家看图 8-12（未开启负载均衡）：所有张三的数据都发给了 reduce1，明显不均衡！

再看图 8-13（已开启负载均衡）：两轮处理后，两个 reduce 的负载(fù zài)比较均衡。

★ 记住：当 Group By 遇到数据倾斜时，设置 hive.groupby.skewindata=true！

【幻灯片 40-45】二、Join 优化

二、Join 优化

Join 相关的优化是 Hive 优化中最重要的部分！大部分性能问题都和 Join 有关。

Join 优化分两类：

1. 大表 join 小表 —— 用 MapJoin 解决（相对简单）

2. 大表 join 大表 —— 比较复杂

什么是 MapJoin?

普通 Join：数据在 Reduce 端汇合(huì hé)，需要 Shuffle，很慢。

MapJoin：把小表加载到内存，在 Map 端就完成 Join，不需要 Reduce，很快！

例题 7：Join 优化案例

step01：设置参数

实验操作: 设置 MapJoin 参数

```
-- 设置自动选择 MapJoin（默认 true）
hive> SET hive.auto.convert.join=true;
-- 设置小表阈值（默认 25M 以下是小表）
hive> SET hive.mapjoin.smalltable.filesize=25000000;
```

step02：创建小表 smalltable 和结果表 jointable

实验操作: 创建小表 smalltable

```
hive> CREATE TABLE smalltable
  (id bigint, t bigint, uid string, keyword string,
   url_rank int, click_num int, click_url string)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
hive> LOAD DATA LOCAL INPATH '/usr/datas8/smalltable001.csv'
  INTO TABLE smalltable;
```

实验操作: 创建结果表 jointable

```
hive> CREATE TABLE jointable
  (id bigint, t bigint, uid string, keyword string,
   url_rank int, click_num int, click_url string)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

step03：小表 join 大表

实验操作: 小表 join 大表

```
hive> INSERT OVERWRITE TABLE jointable
  SELECT b.id, b.t, b.uid, b.keyword,
    b.url_rank, b.click_num, b.click_url
  FROM smalltable s JOIN bigtable b ON b.id = s.id;
```

看图 8-17 和图 8-18：耗时 189.204 秒。

step04：大表 join 小表

实验操作: 大表 join 小表

```
hive> INSERT OVERWRITE TABLE jointable
  SELECT b.id, b.t, b.uid, b.keyword,
    b.url_rank, b.click_num, b.click_url
  FROM bigtable b JOIN smalltable s ON s.id = b.id;
```

看图 8-19 和图 8-20：耗时 173.713 秒。

★ 新版本 Hive 已自动优化 Join 顺序，但了解原理仍然很重要!

【幻灯片 46-48】三、Count(Distinct) 优化

三、Count(Distinct) 优化

Count Distinct 用来统计(tǒng jì)不重复(chóng fù)值的个数。

问题：Count Distinct 操作只用一个 Reduce，数据量大时非常慢！

解决方法：用先 Group By 再 Count 的方式来替代(tì dài)。

例题 8：Count(Distinct)优化案例

step02：设置 reduce 个数为 5

实验操作: 设置 Reduce 数

```
hive> SET mapreduce.job.reduces = 5;
```

step03：使用 Count Distinct 查询

实验操作: Count Distinct 方式

```
hive> SELECT COUNT(DISTINCT id) FROM bigtable;
```

看图 8-21 和图 8-22：耗时 159.662 秒，只用了 1 个 Reduce。

即使设置了 5 个 Reduce，实际只用了 1 个！

step04：使用先 Group By 再 Count 查询

实验操作: 先 Group By 再 Count 方式

```
hive> SELECT COUNT(id) FROM
```

```
  (SELECT id FROM bigtable GROUP BY id) a;
```

看图 8-23 和图 8-24：第一个 Job 用了 5 个 Reduce（并行处理），负载更均衡。

★ 数据量大时，先 Group By 再 Count 比直接 Count Distinct 更好！

【幻灯片 49-52】四、行/列过滤优化

四、行/列过滤(guò lǜ)优化

1. 列处理：

在 SELECT 中，只查询需要的列。如果有分区，尽量使用分区过滤。

★ 少用 SELECT *，只选需要的列！

2. 行处理：

使用外关联时，如果把过滤条件写在 WHERE 后面，会先全表关联再过滤——慢！

应该先子查询过滤，再关联——快！

例题 9：行/列过滤优化案例

方式一：先关联表，再用 WHERE 过滤（不推荐）

实验操作: 先关联再过滤（慢）

```
hive> SELECT o.id
  FROM bigtable b JOIN bigtable o ON o.id = b.id
  WHERE o.id <= 10;
```

看图 8-25 和图 8-26：耗时 177.196 秒。

方式二：先子查询过滤，再关联表（推荐！）

实验操作: 先过滤再关联（快）

```
hive> SELECT b.id FROM bigtable b JOIN
  (SELECT id FROM bigtable WHERE id <= 10) o
  ON b.id = o.id;
```

看图 8-27 和图 8-28：耗时 182.115 秒。

数据量非常大时，先过滤再关联的优势(yōu shì)就很明显了。

★ 原则：尽早过滤数据，减少后续(hòu xù)操作的数据量！

【幻灯片 53-54】任务 8-3 本章小结

好，同学们，我们来总结一下今天学的内容。

第一部分：Hive 参数优化（5 种方式）

1. 本地模式 —— 小数据量时开启，在本机运行，省去提交任务的开销(kāi xiāo)
2. MapReduce 压缩 —— IO 密集时使用，推荐 Snappy 和 LZO
3. 配置 Map 个数 —— 根据数据量和复杂度(fù zá dù)调整
4. 配置 Reduce 个数 —— 合理设置，不要太多也不要太少
5. 合并小文件 —— 减少 MapTask 数量，提高效率

第二部分：HQL 优化（4 种方式）

1. Group By 优化 —— 开启负载均衡，解决数据倾斜
2. Join 优化 —— 使用 MapJoin，大表 join 小表
3. Count(Distinct)优化 —— 用先 Group By 再 Count 替代
4. 行/列过滤优化 —— 尽早过滤，少用 SELECT *

★ 总结一句话：优化的核心(hé xīn)思想就是——减少数据量、减少任务数、减少 IO！

【幻灯片 55-56】任务 8-4 本章习题

最后我们来看一下课后习题：

问题 1：Hive 生态系统中参数优化有哪几种方式?

答案要点：五种方式——本地模式、配置 MapReduce 压缩、配置 Map 个数、配置 Reduce 个数、合并小文件。

问题 2：Hive 生态系统中 HQL 优化有哪几种方式?

答案要点：四种方式——Group By 优化、Join 优化、Count(Distinct)优化、行/列过滤优化。

问题 3：结合实际，说明 Hive 优化需要注意哪些方面?

这道题是开放性的，同学们可以从以下几个角度(jiǎo dù)思考：

- 了解数据的特点(tè diǎn)（大小、分布(fēn bù)是否均匀等)
- 根据任务类型选择合适的优化方式
- 注意小文件问题
- 养成(yǎng chéng)良好的 SQL 编写习惯

好，今天的课就到这里了。同学们辛苦(xīn kǔ)了！

如果有什么问题，随时来找老师讨论(tǎo lùn)。再见！