MS SQL Server 中大数据量表的查询优化

Query optimizing method with large datum volume condition in MS SQL Server

尹永顺　（中国科学院研究生院软件学院 100039）

摘要：在 MS SQL Server 中如何处理记录条数 2000 万以上且每日增加 20 万条的数据表，相信是很多开发人员面临的难题。本文以实际案例描述了此问题的解决方法及其存在的问题。在 SQL Server 2000 中，可以通过分区视图的定义来支持大数据量表的水平拆分和查询时的数据合并，且查询引擎提供的优化机制，使得 SQL Server 在大数据量条件下的查询性能得到了明显改进。最后，指出了本解决方法所带来的问题及其适用范围。

1　问题提出

基于二维关系表的数据库管理系统（DBMS）是数据处理的核心，承担着是信息系统中业务数据的存储、管理的功能。目前，比较典型的数据库产品包括 Access，MS SQL Server，Oracle，DB2 等。本人最近参与开发的网管系统，就采用了微软的 MS SQL Server 2000 数据库系统。

应用系统现场运行 4 个月后，开始出现复杂条件查询取不回来结果的情况。经过检查发现，故障是由于复杂条件的 SQL 语句查询超时，导致应用程序得不到数据引起的。继续检查 SQL 语句为单表查询，但是该表记录数将近 2000 万，且每天增加 20 万。很明显，问题是由于查询数据表中记录过多引起的。

这是一个很典型的数据库问题，即采用何种方法才能实现大数据量（2000 万以上）表的查询优化？

2　解决方法

大数据量表的查询属于数据库优化的一个方面，可以按照尽量减少 I/O 数量的原则进行。具体来说，就是采用水平分区的方法，将一个表分割成结构相同的多个表，每个表都只包括大表中的一部分数据行。这样，查询时就可以只查找包含查询结果数据行的一两个小表，而不用查找整个大表，从而使得 I/O 数量大幅降低，查询时响应速度得到很大提升。

在 MS SQL Server 2000 中，可以通过分区视图来实现表的水平分区，进而解决大数据量表的查询超时问题。下文将以本系统的数据表 PM_HOUR_SECTOR_HISTORY 为例进行介绍。主要的实现方法包括：

（1）以月份为单位将数据表拆分为多张小表；

（2）定义分区视图合并小表中的数据，应用程序通过视图访问所需的数据；

（3）在小表中定义查询引擎所需的主键和 check 约束；

（4）调整应用程序中对数据的访问方式；

接下来，将详细介绍具体的优化处理内容。

2.1　原数据表结构

原数据表 pm_hour_sector_history，用来存放所有小区的性能指标的历史数据，共保存有 6 个月的数据，每天新增加 20 万条，半年数据量约 3600 万。表结构定义如下：

```
CREATE TABLE [dbo].[pm_hour_sector_history] (
[sector_id] [varchar] (20) NULL ,
[record_date] [varchar] (50) NOT NULL ,
[start_of_interval] [varchar] (50) NOT NULL ,
[end_of_interval] [varchar] (50) NOT NULL ,
[city_name] [varchar] (50) NULL ,
[city_id] [char] (10) NULL ,
[omc_name] [char] (10) NULL ,
[bts_name] [varchar] (50) NULL ,
[omc_id] [smallint] NOT NULL ,
[mm_id] [smallint] NOT NULL ,
[subj_id_1] [smallint] NOT NULL ,
[subj_id_2] [smallint] NOT NULL ,
[f001] [float] NULL ,
[f002] [float] NULL ,
[f003] [float] NULL , ……,
[f199] [float] NULL ,
[f200] [float] NULL
```

) ON [PRIMARY]

由于篇幅原因,中间省略了部分字段。

该数据表主键定义如下:[record_date], [start_of_interval], [end_of_interval], [omc_id], [mm_id], [subj_id_1], [subj_id_2]。

2.2 数据表拆分

对表进行水平分区,首先要确定数据拆分的原则。一般来说,可以采用按时间、区域、或其他业务分类标示来进行,应当使数据尽可能均匀分布到多张小表中,且查询引用的表尽可能少。否则,用于在查询时按逻辑合并表的 UNION 操作就会过多,从而会影响性能。

本文中数据按时间增加,可以采用按照时间进行分拆的方法。因此,考虑按月将 PM_HOUR_SECTOR_HISTORY 分拆成小表,每月一张,每个小表的数据结构和原表相同。拆分后的小表如下所示:

```
pm_hour_sector_200401
pm_hour_sector_200402
pm_hour_sector_200403
……
```

2.3 查询数据合并

表分区减少了查询时 I/O 的数量,但是增加了查询时的表选择的负担。SQL Server 通过使用分区视图将数据合并,通过单一视图面对应用程序,这样在查询数据时可以不必手动引用相应的基础表。分区视图的定义如下:

```
CREATE view pm_hour_sector_history_view as
select * from pm_hour_sector_200401
union all
select * from pm_hour_sector_200402
union all
select * from pm_hour_sector_200403
union all
select * from pm_hour_sector_200404
……
```

2.4 查询优化的条件

分区视图使得开发人员不需要直接引用分拆后的基础表,但是最为关键的是 SQL Server 查询引擎针对分区视图做的优化。即针对该视图执行的任何查询都被优化成只搜索查询结果所涉及的基础表,其余表自动忽略。这才是分区视图中最为核心的内容。

分区视图创建后,分区视图的优化能否发挥作用还有着一些限制条件(具体内容请参考 MS SQLServer 联机帮助),主要限制是通过每张表必须建立 Check 约束,限制该表所能存放的数据范围,且彼此的范围不能重叠。这样,SQL Server 才能知道数据的分布状况,从而做出相应的优化选择。

具体本数据表,需要作如下调整:

(1) 每张小表增加 time_id 字段,numeric 类型,存放时间信息,格式 yyyymmddhhmm

(2) 每张小表建立主键,包括如下字段:time_id, omc_id, mm_id, subj_id_1, subj_id_2。(time_id 需要设置为第一个字段)

(3) 每张小表建立约束(check),在 time_id 字段建立约束,限制表中数据的范围,如 pm_hour_sector_200401 表上建立如下约束:time_id between 200401010000 and 200401312359,限制表中的只能是 2004 年 1 月份的数据。其他小表照此处理。

(4) 每张小表建立非聚簇索引(index),包括如下字段:time_id, city_id, omc_id, mm_id, subj_id_1, subj_id_2, sector_id。

在上面调整中:

① 增加 time_id 字段不是分区视图优化所必需的,而是由于原数据表中时间分别存放在 record_date(mm/dd/yyyy 格式)和 start_of_interval(hh:mm 格式)字段中,使用不方便,且字段为 varchar 类型,作为主键效率也不高,所以做的优化调整。

② 增加非聚簇索引不是分区视图优化所必需的,而是针对该表的常用查询语句所作的优化。

至此,分区视图已经创建完毕,接下来需要对应用程序进行必要的调整。

2.5 分区视图的使用

分区视图创建时,如果能够保持视图名称、结构和原来数据表的完全一致,应用程序就可以直接使用,不用做任何修改。满足某些特定条件,分区视图还可以更新数据。

由于本次优化时没有使用原来数据表的名称,且表主键进行了调整,因此应用程序中查询语句的需要按如下方法进行修改:

(1) 原来查询语句的数据表更换为新创建的分区视图;

(2) 原来的时间范围条件务必使用为新创建的 time_id。

下面以一条实际使用的查询语句的修改为例进行说明:

① 原来的 sql 语句(请注意黑体为需要修改的部分):select replace(record_date +´´+ start_of_interval,´-´,´/´) as 开始时间, end_of_interval as 结束时间, City_ID as 城市, OMC_Id as OMC_ID, OMC_Name as OMC 名称, mm_id as CBSC_ID, subj_id_1 as BTS_ID , f050 as ´业务信道负载率´ ,f036 as ´业务信道拥塞率´

from pm_hour_sector_history

where 1=1 and city_id in (´nanjing´) and omc_id in (´301´) and mm_id in ('3011','3012','3013') and 1=1 and city_id in (´nanjing´,´wuxi´,´changzhou´,´zhenjiang´,´suzhou´,´nantong´)

and cast(record_date +´´+ start_of_interval as datetime) between cast('2004-01-12´+´´+ '01:00:00´ as datetime) and cast('2004-1-14´+´´+ '12:00:00´ as datetime)

and subj_id_1 in (1´,10´,107´,11´,115´,121´,13´,17´,19´,20´,201´)

and sector_id in ('3011-10-1´,'3011-10-2´,'3011-10-3´,'3011-107-1´,'3011-107-2´,'3011-107-3´)

order by record_date, start_of_interval, omc_id, mm_id, subj_id_1

② 修改后的 sql 语句(请注意黑体为修改后的部分):

select replace(record_date +´´+ start_of_interval,´-´,´/´) as 开始时间, end_of_interval as 结束时间, City_ID as 城市, OMC_Id as OMC_ID, OMC_Name as OMC 名称, mm_id as CBSC_ID, subj_id_1 as BTS_ID ,f050 as ´业务信道负载率´ ,f036 as ´业务信道拥塞率´

from pm_hour_sector_history_view

where 1=1 and city_id in (´nanjing´) and omc_id in (´301´) and mm_id in ('3011','3012','3013') and 1=1 and city_id in (´nanjing´,´wuxi´,´changzhou´,´zhenjiang´,´suzhou´,´nantong´)

and time_id between 200401120100 and 200401141200

and subj_id_1 in (1´,10´,107´,11´,115´,121´,13´,17´,19´,20´,201´)

and sector_id in ('3011-10-1´,'3011-10-2´,'3011-10-3´,'3011-107-1´,'3011-107-2´,'3011-107-3´)

order by record_date, start_of_interval, omc_id, mm_id, subj_id_1

2.6 查询语句执行分析

在 SQL Server 的查询分析器中分别执行以上两条查询语句,同时显示其查询计划。两者的查询计划有明显的差异:

前者对整个大数据表进行主键扫描,而后者只对查询结果所在的 pm_hour_sector_200401 表进行了非聚簇索引扫描。

大数据表中存放有 4 个月的数据,而在 pm_hour_sector_200401 表仅存放有一个月的数据,且索引项目已经覆盖了查询条件中所有条件,因此可以预计其执行速度会有明显提升。从实际的执行结果来看,前者执行时间约 20 秒,而后者仅为 2 秒。

上述情况表明在 MS SQL Server 2000 中的分区视图,不仅实现了数据表的水平分区,而且通过视图合并拆分后的小表,这样 SQL 查询尤为简单。系统内嵌的优化机制,使得系统大数据量条件下的查询性能得到了明显改进。

3 存在问题和适用范围

分区视图虽然有着很强的功能,但是它的使用并不是没有代价的。有如下问题需要开发人员仔细考虑:

(1) 按照时间存储数据的小表谁来创建,手工提前建好还是程序自动创建?

(2) 小数据表增加以后,分布视图谁来负责更新?

(3) 小数据表和分区视图变动后,应用程序中用户的访问权限是否受到影响?

(4) 数据以何种方式存储到数据小表中,是使用可更新的分区视图还是单独编写程序进行数据的存储?

由于分区视图使用的复杂性,在决定哪些数据表需要进行水平拆分时,需要慎重考虑。一般而言,如果数据表中的数据大于 2000 万,建议考虑使用本解决方法。如果不想采用本方法,可以参考本方法中(1)、(4)建立非聚簇索引的方式进行优化。

4 结束语

微软的 SQL Server 数据库在很多开发人员眼里,只是二流的产品,开发小型系统还可以,处理海量数据则根本不行。其实,SQL Server 从 7.0 版本开始,性能已经大幅提升,有其到 2000 版本后,更是增强了许多功能,如 Anlysis Service 服务、数据复制、XML 集成、索引视图、分布式分区视图等等。据我所知,在实际业务系统中也有使用 SQL Server 管理 800G 以上数据的案例。 (下转第 64 页)

(上接第 81 页)

使用 SQL Server 来管理海量数据,一方面受硬件环境的限制,另一方面和数据库系统的配置、应用程序对数据库的使用方式有着很大的关系。稍不注意,就会使数据库成为系统的瓶颈。而数据库性能的优化,往往可以收到意想不到的效果,将系统的处理能力提高几倍。在本文中介绍了大数据量表的查询优化内容,只是数据库优化的一个方面。完整的数据库优化涉及到内存使用、提高磁盘 I/O、创建和管理索引、数据分区、查询优化等诸多方面,内容繁杂而重要,希望能与各位专家、同行探讨,也希望本文能起到抛砖引玉之功效。

参考文献

1 Microsoft Corporation 著,郭东青等译,数据库创建、数据仓库与优化,清华大学出版社,2001.8。

2 Microsoft Corporation, MS SQL Server2000 联机丛书。