面向城轨线网的海量数据查询优化方法①

赵 驰[1], 刘建委[1], 饶里强[2], 刘 琼[2]

[1](广州市地下铁道总公司 建设事业总部, 广州 510380)
[2](华南理工大学 软件学院, 广州 510006)

摘 要: 城轨线网数据中心汇集多条线路数据, 单表记录量达数十亿条, 当前系统数据查询响应时间过长、效率低下. 提出利用数据库集群及中间件优化系统架构突破单库存储与处理瓶颈, 多节点并行处理提升查询速度. 按线路水平切分数据等方法, 保证 JOIN 操作的局部性, 满足新线路扩展需求; 利用表分区、索引、物化视图、SQL 语句优化等技术优化单机查询. 其中, 针对集群数据透明访问系统架构, 设计专用数据库访问中间件, 解决查询解析、路由及结果合成等关键问题. 以广州城轨线路数据为例进行实验, 结果表明通过本文方法各类查询响应时间至少降低 90%.

关键词: 查询优化; 中间件; 海量数据; 城轨线网

Methods of Massive Data Query Optimization for Urban Rail Transit Network

ZHAO Chi[1], LIU Jian-Wei[1], RAO Li-Qiang[2], LIU Qiong[2]

[1](Construction Division, Guangzhou Metro, Guangzhou 510380, China)
[2](School of Software Engineering, South China University of Technology, Guangzhou 510006, China)

Abstract: The center of urban rail network needs to collect the data of all the urban rail lines and the size of table records will reach billions. The data query on urban rail network will need too much time and the system efficient is very low. We propose the program to optimize the system architecture by database cluster and middleware, which improves query efficiency because of more powerful storage and parallel processing capacity than that of a single database. An sharding method that divide the data horizontally by lines to avoid the expensive table joins crossing databases and the new rail line can be easily extended just by adding more database nodes. Serveral technologies, such as table partitioning, index, materialized view and SQL etc. are used also to optimize the reaction time when standalone inquiring. A special light-weight database access middleware used in the system architecture is designed to solved some key problems such as SQL parsing, route inquiring and result data merging etc.. The experiments are carrieded out on data from Guangzhou Metro as verifying the scheme of this paper. The results show that the reaction time of all types of queries is reduced 90% at least.

Key words: query optimization; middleware; massive data; urban rail transit network

近年来, 城市轨道交通系统(简称城轨)发展迅速, 不断有新的线路建成, 线线之间纵横交错, 单条线路出现问题可能波及或蔓延其他线路. 各城市轨道交通系统发展初期仅有少数几条线路, 各条线路独立运营, 线路生产数据独立存储. 随着城轨建设规模扩大和发挥运力, 迫切需要建设线网级的数据中心集中监视、指挥管理和协调各线路. 线路数据库单表年记录量可达上亿条, 查询响应时间将超过十分钟, 按此, 汇集十几条线路的数据线网数据中心, 单表记录量将达数十亿条, 现实数据的查询响应时间很难满足需求.

数据查询效率优化历来是业界的热点问题. David Taniar 等人[1]提出利用提示(Hint)优化嵌套查询提高效率的方法, 但是, 以固定方式查询动态变化的数据很难保证最优. 仰燕兰等[2]针对车辆定位监控系统大表引起的访问效率问题, 通过分流方式将原表划分为多个子表, 并采用并行处理改善数据访问效率, 但是分

① 收稿时间:2015-03-27;收到修改稿时间:2015-06-03

表方法对应用不透明. 刘晓娜和杜永文[3]利用物化视图[4-5]对移动数据库进行查询优化, 他们的技术依赖于数据刷新的及时性, 仅适合用于周期性查询. 杨悦[6]针对卫星测控数据以时间区间检索, 结果按时间排序, 提出基于索引组织表的存储和查询方法, 表数据按主键有序存储在索引上, 以减少磁盘 I/O, 但数据更新开销较大. 杨玉娟[7]为提高上海地铁六号线 AFC 数据库性能, 采用操作系统参数调整和 SQL 语句复用等优化技术虽提高了系统性能, 但并未作针对性优化.

查询优化在实际应用中受到较多关注, 但对于城轨数据库建设, 尤其是新近发展的城轨线网数据中心建设, 或者未受到足够重视, 或者还是个新话题. 目前线路级数据库多采用系统参数调整、SQL 语句复用、存储过程等方法, 优化效果十分有限, 无法满足城轨线网数据中心实时查询海量数据的需求.

1 城轨线网数据及查询的特点分析

1.1 数据特点

城轨线网数据中心需要整合多条线路多个专业的生产运营数据, 特点如下:

①时间跨度长, 数据量庞大. 历史数据需要保存较长时间, 以利用其对业务及系统运行状况进行宏观统计分析, 为城轨决策和规划提供数据支撑. 随着时间的迁移, 数据不断累积, 需处理的数据量将非常庞大. 查询优化应具有可持续性, 查询响应时间不致随时间迁移而发生显著变化.

②层次性. 为便于集中监控和协调指挥, 各线路控制中心需整合各个车站(车辆段、停车场等)的数据, 线网数据中心需整合各线路控制中心的数据, 因此, 城轨线网数据具有层次性, 可以按所属线路、车站等进行归类划分. 考虑到未来轨道交通的建设性发展, 查询优化应能满足新线路建成时的扩展需求.

③内容的多样性. 线网数据中心要实现对各线路设备、环境、供电、行车和客流等信息的监管, 存储的数据种类多样, 主要包括: 设备状态、环境温湿度、事件报告及报警、能耗、行车、客流和操作人员的操作信息等. 数据内容的多样性导致数据查询的多样性.

1.2 查询特点

线网数据中心的重要功能是对海量的线网数据进行统计分析, 为城轨的运营管理决策提供支持, 因此查询的种类以报表查询为主, 如广州地铁, 已知的报表数共 50 多个. 根据查询涉及操作种类及其复杂程度, 城轨线网的主要查询模式可归为以下 3 类:

①浏览查询. 语句简单, 以 SPJ 查询为主, 通常输入时间区段、线路、车站等参数进行条件查询, 并多用模糊查询, 如“ev.asset_name LIKE '%PSCADA%”等. 不对表数据进行优化组织, 将导致全表扫描操作, 对于记录量上亿甚至数十亿的表来说, 开销巨大.

②简单统计查询. 用于统计符合条件记录的数目, 如统计各线路控制中心/车站/系统报警数量分布、各线路重要设备故障次数等. 该方法有一定的周期性、包含 COUNT(*)操作, 但统计操作涉及的目标记录非常多, 根据查询条件建立索引也很难起到优化作用, 若不作针对性优化, 也将导致全表扫描.

③复杂统计查询. 用于统计各线路车站供电情况、各线路温湿度情况等. 这类查询也有一定的周期性, 与简单统计查询相比, 语句的结构更为复杂, 子查询嵌套层数较多, 一般包含了多个表连接操作.

2 优化策略与方法

查询优化的目的是尽可能减少每个查询的响应时间, 尽量减轻数据库的操作负担, 提高系统效率. 提高查询速度主要有两种途径: 尽可能减少需要数据库处理的数据规模; 或用空间换取时间.

本文提出“分而治之、以空间换时间”的优化策略. 前者指利用数据库集群、表分区等技术将大表划分为若干个粒度更小、更易于处理的区段, 避免对大量无关数据进行处理, 同时利用并行能力提高处理效率; 后者则是利用索引、物化视图等结构的冗余存储及预处理能力缩短查询响应时间.

2.1 数据分库并开发中间件优化架构

城轨线网数据量庞大, 且城轨线路数量仍将不断增加, 单个数据库的存储和处理能力十分有限, 当系统负载过高、处理能力不足时系统性能将急剧下降, 而高性能存储和服务器价格昂贵, 单纯靠升级硬件提升存储和处理能力, 不仅开销巨大, 可扩展性也得不到保障. 增加数据库节点, 将数据分布存储, 并利用多个节点并行处理查询, 查询效率有望大幅提高.

为了使客户端能够正确地访问分库存储的目标数据, 尤其是查询使客户端能够透明地、像访问一个数据库一样地访问分布于多个库节点中的数据, 本文设计实现了一个能够接收查询请求、进行路由查询及查

询结果整合的数据库访问中间件, 客户端的查询请求先经过中间件接收处理, 并将其路由到相关数据库节点执行, 中间件再收集整合各数据库节点的执行结果并反馈客户端. 分布式优化查询系统架构如图 1 所示.

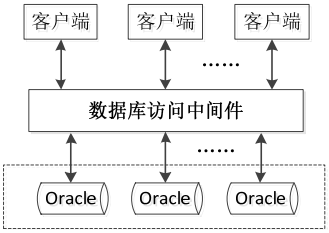

图 1 分布式优化查询系统架构

2.2 大表分区

当查询条件为模糊表达式或者涉及目标记录较多的情况下, 不得不对数据进行扫描, 此时, 分区组织表数据可避免全表扫描, 减小需处理的数据规模. 查询时在查询条件中指定分区的列值或范围, 以列取值将表中记录划分为若干个物理上相互独立的分区, 就仅读取相关分区的数据, 在分库的基础上进一步减小所处理数据的规模.

分区技术可用于优化各类查询, 对于记录量较大的表进行分区. 根据城轨线网数据、查询以及管理维护特点, 采取组合范围-列表分区方案, 即按月份(若数据量特别大, 则按天或小时)进行范围分区, 并按线路划分列表子分区. 该分区方案有以下优点:

①按月份分区, 符合查询需求且便于管理维护. 查询条件中按月设置时间区间, 将查询范围缩小到月份. 特别是需定期清理越来越多的表中数据时, 通常采用低效的 DELETE 操作删除历史记录, 而按月分区可使用更快速的 TRUNCATE 操作清除某分区数据.

②按线路建立子分区, 可进一步缩小数据范围. 若查询只关注某条线路时, 只需处理月份分区中对应线路的子分区数据. 按月份和线路实现组合分区, 将避免查询响应时间因历史数据的累积、线网规模的扩大而显著增加, 改善了系统效率和可扩展性.

2.3 索引技术

索引有助于快速访问数据库中特定行, 从而避免对数据扫描. 对于浏览查询, 简单统计查询涉及的目标记录较多, 通过统计符合条件的记录数量建立索引很难起到优化作用. 而在非空列上建立的索引和记录存在一一对应的关系, 可通过扫描索引实现对记录的统计, 这类查询只涉及几个关键列, 不需记录更多具体信息, 一个表往往有几十甚至上百个列, 若对查询的主数据表涉及的所有列(不仅仅是查询条件中出现的列)建立联合索引, 扫描索引所需的磁盘 I/O 次数远少于扫描记录的次数, 将大幅减少查询响应时间.

2.4 物化视图预处理

物化视图通过刷新维护与基表数据保持同步, 频繁刷新将影响系统性能, 故物化视图不适合处理实时性要求高的查询, 但能够优化有固定执行周期的查询.

本文利用物化视图优化复杂统计查询, 因为这类查询具有一定的周期性、子查询嵌套层次深、处理复杂, 以空间换时间, 提前计算部分子查询结果并保存于物化视图中, 藉此, 查询时只需进行简单处理, 缩短了查询响应时间. 例如, 对于城轨各线路日供电报表, 日净耗电量的计算方式是当天 23 时与前一天 23 时的累计耗电量之差, 得到某天的累计耗电量数据需执行一个表连接, 因而该查询至少需两次表连接. 对于这种情况, 利用物化视图预先计算并存储日累计耗电量数据, 查询时便可减少两次表连接操作.

为了避免数据累积导致查询效率的下降, 进一步对物化视图进行分区, 根据数据量按年分区即可. 物化视图刷新方式又分为增量刷新和完全刷新, 为减少刷新开销, 采用增量刷新, 但要建立日志以记录表的变更情况. 计算日累计耗电量的物化视图脚本如表 1.

表 1 广州城轨某线路日累计耗电量物化视图脚本

```
CREATE MATERIALIZED VIEW mv_power_day
PARTITION BY RANGE( day )
INTERVAL( NUMTOYMINTERVAL( 1,'YEAR' ) )
(PARTITION p_2013 VALUES LESS THAN ( TO_DATE ( '2014-01-01', 'yyyy-mm-dd' ) ) )
START WITH TRUNC( SYSDATE )+23.5/24
NEXT TRUNC(SYSDATE)+47.5/24
REFRESH FAST ON DEMAND WITH ROWID
AS SELECT r.entity_key entity_key , r.line_key , TRUNC (lg.createtime)day, lg.value total_value, r.rowid r_rowid,lg.rowid lg_rowid
FROM rpt_power_total_report r , datalog_dp_log_report lg
WHERE r.entity_key = lg.entity_key
AND r.line_key = lg.line_key
AND EXTRACT( HOUR FROM CAST (lg.createtime AS TIMESTAMP ) ) = 23
```

2.5 优化 SQL 语句

许多查询性能问题源自 SQL 语句执行效率的低下, 可通过优化将其转换为语义上等价而效率更高的形式

[8-9], 部分 SQL 语句优化方法如下:

①避免在谓词中对索引列或分区列使用函数、运算或模糊表达式, 以便发挥索引或分区消除特性;

②使用 UNION 改写 OR, 因为 WHERE 子句中使用 OR 则不能利用索引, 只能执行扫描;

③使用 EXISTS 替换 IN, 因为 IN 子句将执行全表扫描, 并对子查询结果执行排序和合并;

④使用 WHERE 子句替换 HAVING 子句, 因为后者在聚集计算后检查各分组是否满足条件, 而前者在分组之前过滤数据, 可减少参加聚集计算的数据量.

例如, 线路级的报表查询中, 在限定时间区间时, 普遍使用类似“TRUNC(createdatetime) >= {起始日期} AND TRUNC(createdatetime) <= {截至日期}”的写法, 在 createdatetime 列使用函数将无法利用该列上建立的索引, 且该列往往是分区列, 还将导致无法利用分区消除特性, 可优化为“createdatetime >= {起始日期} AND createdatetime < {截止日期}+1”.

3 数据库访问中间件

不少大型 IT 公司为解决海量数据处理问题, 开发出各具特色的数据库中间件, 如 Atlas、TDDL 和 Cobar 等[10], 但这些中间件的实现与具体业务需求有较强的相关性, 复杂度高、公开资料少且源码开放程度有限, 另外存在尚未得到解决的技术问题, 如跨库 JOIN 操作. 本文根据城轨线网的业务特点, 设计并实现一个轻量级的数据库访问中间件.

3.1 总体结构

城轨线网中间件主要包括查询解析、路由查询、数据库访问、结果合成和通信等模块, 总体结构如图 2 所示, 客户端使用 TCP 协议与通信模块交互, 数据库访问模块负责与后端数据库的交互.

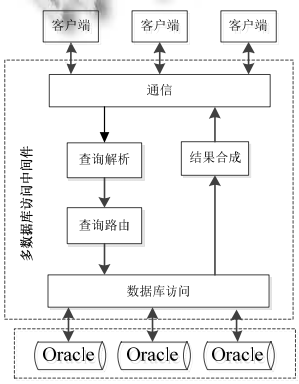

图 2　中间件总体结构

引入中间件后的查询请求处理流程如下:

①信模块对客户端用户身份进行验证;

②验证通过, 通信模块接收并解析查询请求报文;

③查询解析模块对 SQL 语句进行解析;

④查询路由模块确定目标数据库集并分发查询;

⑤数据库访问模块与后端数据库交互, 执行查询;

⑥结果合成模块对各路执行结果进行合成;

⑦通信模块将结果封装成报文后发送给客户端.

限于篇幅, 下面重点介绍数据切分、查询解析、查询路由和查询结果合成模块的设计与实现.

3.2 数据切分策略

数据切分分为垂直切分和水平切分, 前者扩展性差, 由于新线路不断投入运营, 持续增长的数据量可能单台数据库的存储空间不堪重负, 而商用存储系统扩容成本较高. 为使各台数据库的负载均衡, 采用水平切分, 使用哈希算法将数据均匀地映射到各台数据库, 但跨库 JOIN 操作代价太大, 需要在程序中分解数据访问逻辑、消除 JOIN 操作, 这将增加代码实现的复杂度和开发人员的工作量.

由城轨线网数据的层次性可知, 线网数据是由各条线路的数据组成, 如果一条线路的数据全部置于单个数据库, 可保证 JOIN 操作的局部性, 各节点完成单库 JOIN 操作后再由中间件合并操作结果即可, 避免跨库 JOIN. 故本文按线路对数据进行水平切分, 每条线路的数据限存储于一台数据库, 但根据实际容量, 每台数据库可能存储多条线路的数据.

3.3 查询解析

查询解析要的关键是解决 SQL 语句的词法和语法分析, 从中提取出路由查询和结果合成所需的信息. SQL 语句中出现的标记可分为数字、标点符号、运算符号、字符串、SQL 关键字和标识符等 6 类, 解析时只需从头到尾扫描一次, 边扫描边分析.

为将查询路由到相关的后端数据库, 需解析出查询条件中目标线路范围的信息, 路由查询模块将根据该信息及线路数据分布计算出目标数据库集; 为对各数据库返回的执行结果进行合并, 需解析出查询语句中的聚集函数、分组属性、分组过滤条件和排序属性等信息, 语法分析算法流程如图 3 所示.

3.4 路由查询

路由查询是确定目标数据库并将查询分发到后端数据库. 根据数据库与线路的映射关系以及从查询语

句中解析出的目标线路范围, 可计算出查询的目标数据库集. 使用线程池分发调度多路查询, 各路查询都以工作任务的形式提交给线程池并行执行, 减少了等待时间和频繁创建、销毁线程的开销.

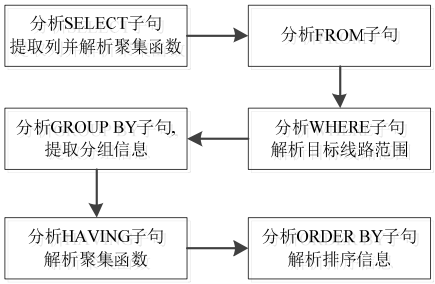

图 3 查询解析算法流程

当存在多个目标数据库时, 分发查询前可能还要对 SQL 语句进行改写. 例如, 如果 SELECT 子句中的列包含 AVG 函数且不按线路分组, 则需添加对应的 COUNT 和 SUM 列, 结果合成时利用这两列计算平均值; 由于针对全局查询进行分组过滤操作, 如果有 HAVING 子句, 则需从 SQL 语句中删除, 将过滤操作推迟到合成多路数据库查询结果时进行, 若聚集函数没有出现在 SELECT 子句中, 则需添加对应列. 路由查询算法流程如图 4 所示.

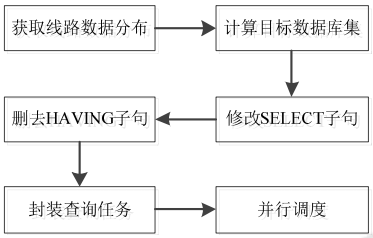

图 4 路由查询算法流程

3.5 查询结果合成

当存在多个目标数据库时, 需对各线路执行结果进行合成. 若 SQL 语句中, SELECT 子句的列不含聚集函数且没有 HAVING、ORDER BY 子句, 则只需将各执行结果汇总, 否则, 需根据分组属性将结果分组, 并根据聚集函数种类(如 AVG、COUNT、MIN 和 MAX 等)对每个分组进行相应计算, 然后根据过滤条件过滤聚集计算结果, 按排序属性及方式(升序或降序)排序.

由于数据按线路分库, 若检测到分组属性中包含 LINE_KEY(线路编号), 则不需再进行分组和聚集计算; 若 LINE_KEY 为第一个排序属性, 只需按该列排序, 因为各路执行结果已经部分有序. 查询结果合成算法流程如图 5 所示.

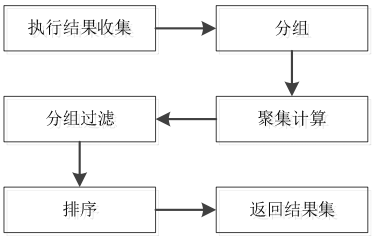

图 5 查询结果合成算法流程

3.6 中间件特点

文中根据城轨线网的业务特点, 设计实现了专门的数据库访问中间件, 特点如下:

①可避免跨库 JOIN. 针对按业务垂直分库不利于扩展、一般的水平分库方法存在跨库 JOIN 的问题, 提出按线路水平分库的方法, 保证 JOIN 操作的局部性, 不需要在应用中修改数据访问逻辑、消除 JOIN, 减轻了开发人员的负担.

②根据数据切分特点进行针对性优化, 减少了查询结果合成时进行聚集计算和排序等处理的开销.

③可扩展性好, 由于采取按线路分库, 可通过增加节点满足新线路建成时的扩展需求, 不再受限于单台数据库的容量与速度瓶颈.

④节约成本. 集群节点对硬件的要求不高, 不再需要昂贵的高性能服务器和存储设备, 后端的 Oracle 也可替换为 MySQL 等成本较低的数据库.

4 实验

为验证文中方法的优化效果, 从真实城轨线网的三大类查询中各取一个实例进行实验. 实验基于 Oracle 11g 数据库, 采用广州地铁六号线和广佛线四个月的运营数据作为线网实验数据, 使用中间件时, 两条线路的数据分别存储于一台数据库中. 测试所用服务器包括两台数据库服务器和一台中间件服务器, 服务器 CPU 为 Intel Core i5-4570. 实验结果如表 2.

表 2 优化前后查询响应时间比较

序	查询类型	实例	最大表记录行数	优化前/s	优化后/s
1	浏览	各线路车站 ISCS PSCADA 操作记录	93771203	360	13.1
2	简单统计	各线路车站报警数量分布月报表	3310678	36.0	0.733
3	复杂统计	各线路供电日报表	9071705	26.6	0.428

由表 2 可看出, 由于采用了分库分区的方法, 可持续性和可扩展性得到明显改善, 查询响应时间不会随着数据的累计和线网规模的扩大而发生明显变化, 如, 查询 1、2 和 3 的响应时间分别减少了 96.4%、98.0% 和 98.4%, 查询响应时间已能够较好地满足城轨线网数据中心的业务查询需求.

5 结语

各大城市城轨线网数据中心建设在即, 城轨数据量的攀升明显达到海量级别, 数据查询速度直接影响数据中心的可用性. 面对城轨线路数据库查询技术落后很难满足线网数据中心海量数据查询的实际需求, 提出利用数据库集群及中间件技术对系统架构进行优化, 并协同利用表分区、索引、物化视图和 SQL 语句等多种优化技术进行查询优化. 结论如下:

①针对城轨线网的业务特点, 提出了一套有效的优化方法, 各类查询的响应时间的减少了 90%以上;

②提出了一种优化的数据切分方法, 避免了一般的数据切分方法存在的跨库 JOIN 问题, 且易于扩展;

③根据城轨业务特点设计专门的数据库访问中间件, 有效解决查询解析、路由和结果合成等关键问题.

实验表明, 本文优化方案明显减少城轨线网数据查询响应时间, 成果具有较好的可推广性.

参考文献

1 Taniar D, Khaw HY, Tjioe HC, et al. The use of Hints in SQL-Nested query optimization. Information sciences, 2007, 177(12): 2493–2521.

2 仰燕兰,叶桦,费树岷.车辆定位监控系统数据库的设计与优化.东南大学学报(自然科学版),2010,40(S1):44–47.

3 刘晓娜,杜永文.物化视图在移动数据库查询优化中的应用.通化师范学院学报,2013,34(1):36–37.

4 张银玲,武彤.常用 OLAP 查询优化方法性能分析.计算机技术与发展,2014,24(1):39–42.

5 武彤,赵雪,赵洵.动态更新实物化视图以提高 OLAP 查询效率.计算机科学,2012,39(B06):315–317.

6 杨悦.基于海量卫星测控数据存储与查询方法.科学技术与工程,2013,13(25):7352–7356.

7 杨玉娟.地铁 AFC 系统数据库设计.维护和优化.铁路计算机应用,2011,20(3):56–58.

8 李展涛,曹英忠.基于 Oracle 数据库的 SOL 语句优化.微型机与应用,2011,30(21):11–13.

9 郭珉.ORACLE 数据库 SQL 优化原则.计算机系统应用,2010,19(4):170–173,165.

10 子柳.淘宝技术这十年.北京:电子工业出版社,2013.