76 / 686

**Chapter 7**

**性能调优建议**

**7.1**

**7.1.1**

**含有大几何对象的小表**

**问题描述**

当前版本的PostgreSQL（包括8.0版本）在TOAST表方面，查询优化不够好。TOAST表是一种用于存储字段超大的表，正常情况下字段超大（比如长文本，图像或者带有许多顶点的复杂几何对象）的值不能存储在物理页上，参考 the PostgreSQL Documentation for TOAST 获取更多信息。

如果你有一个包含很大几何类型但记录条数不多的表（就像一个表包含了欧洲所有国家的高清边界图），那么就会出现这种问题。虽然这个表很小，但是却使用了大量TOAST空间。在我们的样例中，该表有80行数据，使用了3个数据页，但TOAST表使用了8225页。现在考虑一个问题你使用几何对象运算符&&来搜索匹配少了记录的bounding box，查询优化器发现表只有3页和80条记录，它会估算出在这样一个小表上面用全表扫描要比使用索引快得多，于是查询优化器决定忽略GIST索引。通常情况下，这种估算是正确的。但在我们的案例中，运算符&&必须要从磁盘中读取每一个几何对象来对比bounding box，因此它要读取所有的TOAST页。

确认你的PostgreSQL是否有这样的bug，使用命令“EXPLAIN ANALYZE “来看你的SQL执行计划。更多信息和技术细节，你可以参考PostgresSQL的性能邮件列表：http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php

**7.1.2**

**解决办法**

PostgreSQL 开发人员正尝试解决这个问题，解决办法是通过查询时候估算TOAST表因素的影响。现在有两种解决办法：

第一种办法是强制让查询分析器使用索引。也就是在查询前，先做如下设置 SET enable\_seqscan TO off。这样可以从根本上强制查询优化器任何时候都避免全表扫描。这样查询就会使用GIST索引。但这个设置标志会影响到所有的连接，导致查询优化器会对其他查询做出错误的估计，因此查询完成后，你需要SET enable\_seqscan TO on，即关闭全表扫描。

第二种解决办法是让查询优化器认为全表扫描和使用索引一样快。这可以通过创建额外的存储bounding box的列来实现。在我们的样例中，创建该列命令如下：

SELECT AddGeometryColumn(‘myschema’,’mytable’,’bbox’,’4326’,’GEOMETRY’,’2’);

UPDATE mytable SET bbox = ST\_Envelope(ST\_Force2D(the\_geom));

现在使用&&操作符来操作bbox列，而不是geom\_column中的列，如下：

SELECT geom\_column

FROM mytable

WHERE bbox && ST\_SetSRID(‘BOX3D(0 0,1 1)’::box3d,4326);

77 / 686

当然，如果你修改了或添加表记录，你必须要保持bbox这一列要实时更新。最方便的方法是使用触发器，但你也可以每次修改后运行上面的UPDATE语句来更新bbox列。

**7.2**

**聚簇geometry列的索引**

对于那些大多数情况都是读数据的表，并且大多数的查询都使用单一的索引，PostgreSQL提供了CLUSTER命令。

该命令会将所有的记录重新进行物理排序以保持和索引的排序一致，并得到两个性能优势：首先，对于索引范围查询，可以大幅度减少查询的数据量。第二，如果你的工作主要集中于在索引上间隔比较小的记录上（就是列的值会均匀分布），你的查询会有更高的缓存命中率，因为数据记录会放在更少的数据页上。（关于这一点请查看PostgreSQL手册，浏览关于CLUSTER命令的信息）

然而当前PostgreSQL不允许在PostGIS 的GIST索引上面做CLUSTER命令，因为GIST索引只是简单地忽略了NULL值，所以你会得到如下错误：

lwgeom=# CLUSTER my\_geom\_index ON my\_table;

ERROR: cannot cluster when index access method does not handle null values

HINT: You may be able to work around this by marking column "the\_geom" NOT NULL.

正如HINT信息告诉你的那样，你可以通过将该表的这个字段设置为NOT NULL来解决这个问题：

lwgeom=# ALTER TABLE my\_table ALTER COLUMN the\_geom SET not null;

ALTER TABLE

当然，该命令对于你的几何类型列中含有NULL值的不起作用。除此之外，你必须使用上面的方法来添加约束，使用CHECK方式如

"ALTER TABLE blubb ADD CHECK (geometry is not null);" 不起作用。

**7.3**

**避免维度转换**

有时候，你表中的数据是3D或4D对象数据，但必须要使用OpenGIS兼容的只支持2D输出的函数ST\_AsText() 或

ST\_AsBinary() 。这两个函数内部都是调用函数ST\_Force2D()实现的，该函数会对大几何对象有比较大的资源开销。为了避免这种开销，先一次性或永久丢弃掉额外的维度是较好的方案：

UPDATE mytable SET the\_geom = ST\_Force2D(the\_geom);

VACUUM FULL ANALYZE mytable;

注意：如果你使用函数AddGeometryColumn()添加了几何对象列，那么这种是通过约束方式实现的。为了避开这一点，你需要删除约束。注意需要更新系统表geometry\_columns，然后再重新创建约束。

对于大表，更明智的做法是每次通过WHERE子句限定主键或其他唯一约束的更新范围的方式逐步更新，并且在执行在执行每个UPDATE语句之前对该表进行“VACUUM“操作。这会大大减少临时存储空间的需要。除此之外，如果你有混合维度的几何对象，在UPDATE语句中加上限制"WHERE dimension(the\_geom)>2" 跳过哪些geometry对象已经是2D的记录

**7.4**

**参数配置调优**

这些调优建议是根据Kevin Neufeld的2007年参加FOSS4G会议提出的关于PostGIS的一些建议。

根据你的PostGIS使用场景（例如统计数据和复杂分析这种场景vs频繁更新数据和面临很多用户的场景），这些建议能为你的查询提供更快的查询性能。

获取更多建议（或者更好的格式），原始报告在 [http://2007.foss4g.org/presentations/view.php?abstract\_id=117.](http://2007.foss4g.org/presentations/view.php?abstract_id=117)

78 / 686

**7.4.1**

**启动配置**

该配置在配置文件postgresql.conf:

[checkpoint\_segments](http://www.postgresql.org/docs/current/static/runtime-config-wal.html#GUC-CHECKPOINT-SEGMENTS)

默认值：·切换WAL日志时候，最大的log段文件数（每个段文件默认是16MB），默认值是3

· 优化建议：设置改值最少为10，对于写频繁或者数据库负载更大的情况最少30。更多关于该参数讨论的参考文章

[Greg Smith: Checkpoint and Background writer](http://www.westnet.com/~gsmith/content/postgresql/chkp-bgw-83.htm)

·最好把xlog放在其他单独的盘上（相比较存储数据的盘）

译者注：PostgreSQL 9.5版本开始，这个参数已经取消了，由参数：max\_wal\_size和checkpoint\_timeout参数控制

[constraint\_exclusion](http://www.postgresql.org/docs/current/static/runtime-config-query.html#GUC-CONSTRAINT-EXCLUSION)

· 默认值：off（PostgreSQL 8.4之前版本该值为off，PostgreSQL 8.4+版本该值为partition）

· 该参数通常用于表分区。如果你对PostgreSQL版本低于8.4版本，设置改值为on来保证查询会使用查询优化器。对于PostgreSQL 8.4及之后的版本设置改值为“partition“是更理想的，因为它会强制分析带有约束限制的表，如果这些表是继承表，否则不会有性能损失（译者注：PostgreSQL的分区表是通过继承方式进行的）

[shared\_buffers](http://www.postgresql.org/docs/current/static/runtime-config-resource.html#GUC-SHARED-BUFFERS)

· 默认值是：32MB

· 建议值：将该值设置为内存的1/3到3/4之间

**7.4.2**

**运行时参数**

[work\_mem](http://www.postgresql.org/docs/current/static/runtime-config-resource.html#GUC-WORK-MEM) (用于排序操作和复杂查询)

· 默认值: 1MB

· 建议：

对于有很多内存，数据库很大，并且复杂查询很多的情形，调大该值

· 对于内存较小，并且有很多在线用户的情况，调小该值

· 如果你的内存很大，开发者很少，可以设置改值如下

SET work\_mem TO 1200000;

[maintenance\_work\_mem](http://www.postgresql.org/docs/current/static/runtime-config-resource.html#GUC-MAINTENANCE-WORK-MEM) (用于VACUUM,和创建索引操作等等)

· 默认值为: 16MB

· 默认值为16MB通常太低了，交换内存时候会占用I/O，锁定表对象

· 生产库上内存很大的时候建议32MB到256MB，但依赖于当前使用的用户数。如果你的内存很大，开发者很少，可以设置改值如下：

SET maintainence\_work\_mem TO 1200000