POSTGRESQL：性能提升之并行创建索引

作者： Hans-Jürgen Schönig译者: 陈雁飞 陈雁飞 崔鹏

### 作者简介

**Hans-Jürgen Schönig**：从20世纪90年代开始使用PostgreSQL，担任CYBERTEC公司的CEO和技术主管（[www.cybertec-postgresql.com](http://www.cybertec-postgresql.com)）， CYBERTEC是该领域的市场领导者之一，自2000年以来已为全球无数客户提供服务。

### 译者简介

**陈雁飞**：开源PostgreSQL爱好者，一直从事PostgreSQL数据库运维工作

**崔鹏：**任职于海能达通信股份有限公司，致力于PostgreSQL数据库在专网通信领域的应用与推广

PostgreSQL新版本即将发布，因此是时候看看PostgreSQL11提供的最重要新功能之一：并行创建索引（注：原博客发表于2018年）。多年以来，各种商业数据库供应商已经提供该功能，因此很高兴PostgreSQL成为该精英俱乐部的一员，它提供了多核索引创建能力，这将极大的改善大型数据库部署的可用性。

## PostgreSQL中创建大表

从11版本开始，PostgreSQL支持经典的“[存储过程](https://www.postgresql.org/docs/11/sql-createprocedure.html)”。一个过程的优点是可以运行多个事务，如果要生成大量随机数据，这是一个最理想的选择。当你使用generate\_series生成一百万行时，PostgreSQL必须将这些数据保存在内存中，因此使用多个事务来生成亿万个随机行对于减少内存非常有用。下面是如何使用：

|  |
| --- |
| CREATE TABLE t\_demo (data numeric);    CREATE OR REPLACE PROCEDURE insert\_data(buckets integer)  LANGUAGE plpgsql  AS $$     DECLARE        i int;     BEGIN        i := 0;        WHILE i < buckets        LOOP           INSERT INTO t\_demo SELECT random()              FROM generate\_series(1, 1000000);           i := i + 1;           RAISE NOTICE 'inserted % buckets', i;           COMMIT;        END LOOP;        RETURN;     END;  $$;    CALL insert\_data(500); |

上面部分代码加载了5亿个随机数值，这足以说明在PostgreSQL11中对CRETAE INDEX的改进。在上面的例子中，5亿行记录大约是21GB的数据：

|  |
| --- |
| test=# \d+   List of relations   Schema | Name   | Type  | Owner | Size  | Description  --------+--------+-------+-------+-------+-------------   public | t\_demo | table | hs    | 21 GB |  (1 row) |

我选择numeric的原因是因为在所有的数字数据类型中，numeric类型的代价是最大的。在numeric类型上创建索引比其他数据类型上索引代价要大的多，比如int4或者int8。目的是通过在昂贵的字段上建立一个大索引可以节省多少CPU时间。

**创建索引：仅使用1个CPU核**

在PostgreSQL11中默认是支持并行索引创建。可以在postgresql.conf文件中设置max\_parallel\_maintenance\_workers参数来选择。

|  |
| --- |
| test=# SHOW max\_parallel\_maintenance\_workers;   max\_parallel\_maintenance\_workers  ----------------------------------   2  (1 row) |

默认值告诉PostgreSQL，如果表足够大，可以启动两个工作程序来帮助创建索引。为了比较“传统”的索引创建和新场景之间的差异，这里将设置参数max\_parallel\_maintenance\_workers 为0。这将确保不会选择多核创建索引：

|  |
| --- |
| test=# SET max\_parallel\_maintenance\_workers TO 0;  SET |

结果是索引创建过程将持续很长时间。运行CREATE INDEX语句时，将看到很多I/O和CPU。更糟糕的是，将所有的内存参数都设置为默认值，这就意味着创建索引的时候仅能使用4MB内存，对于给定的表大小，这没有任何意义。

下面是在“Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz”上的测试结果

|  |
| --- |
| test=# CREATE INDEX idx1 ON t\_demo (data);  CREATE INDEX  Time: 1031650.658 ms (17:11.651) |

17分钟，还不错。记住，现在讨论的是5亿条让人厌烦的数据。

### 使用多核工作

在2核上运行相同类型的索引创建：

|  |
| --- |
| test=# SET max\_parallel\_maintenance\_workers TO 2;  SET    test=# CREATE INDEX idx2 ON t\_demo (data);  CREATE INDEX  Time: 660672.867 ms (11:00.673) |

哇，将时间减少到了11分钟。当然，这个运算不完全是线性的，因为我们必须记住需要合并这些部分结果和所有其他结果在一起。但是，有一个误导：如果将max\_parallel\_maintenance\_workers设置为2，而我们看到的是2个核，对吗？如果将该值设置为4呢？在我的情况下，机器中物理内核的数量是4，因此设置任何更高的值都没有意义。将看到PostgreSQL仍然使用两个核心工作。

我们应该怎样改变上面的结果？答案就在下面的列表语句中：ALTER TABLE .. SET .. 语句允许我们抛开这个约束并使用更多的工作进程：

|  |
| --- |
| test=# ALTER TABLE t\_demo SET (parallel\_workers = 4);  ALTER TABLE    test=# SET max\_parallel\_maintenance\_workers TO 4;  SET |

在上面的例子中，max\_parallel\_workers和表级参数均设置为4。我们将看到PostgreSQL将利用5个进程工作。为什么会什么这样的情况？你将看到一个主进程和4个辅助进程来创建索引。这个例子可能不是很明显，但是当你思考它的时候是有意义的。

当然，我们不能期望通过无限制设置工作进程个数来线性增加性能。在这个阶段，我们的（单个）SSD将遇到性能瓶颈，不会看到两倍的性能提升：

|  |
| --- |
| test=# CREATE INDEX idx3 ON t\_demo (data);  CREATE INDEX  Time: 534775.040 ms (08:54.775) |

每个进程都在同时做着同样的事情，因此我们将看到IO曲线的巨大波动。这自然会让整个过程变慢而不是线性的。尽管如此，我们最终还是通过简单地向数据库系统增加工作核的方式，将索引创建时间从17分钟提升到9分钟。

### 索引创建提供更多内存

影响索引创建的不仅仅是CPU核心数，内存也同样非常重要。maintenance\_work\_mem 默认值设置在一个非常小的数值（64MB），这极大地限制了数据量，原本这些数据可以在内存中进行排序。因此下面的逻辑步骤是增加这个值并在一个大值的基础上再次创建索引：

|  |
| --- |
| test=# SET maintenance\_work\_mem TO '4 GB';  SET |

在这里我将这个值设置为4GB。我的机器有32GB内存，但是我们必须要考虑到机器不仅仅只是进行一个索引的创建操作，并且在实际环境采用4 GB x 5核的数值是一个非常有挑战的数值。

在创建索引时，我们将看到在索引创建的第一个阶段有更多的并行操作，这正是我们所期望看到的。同时，还可以清晰的看到在末尾的时候CPU使用率相当低，此时PostgreSQL正在等待磁盘来完成它的工作。系统的配置是默认值，没有对磁盘操作进行优化，因此这将是一个优化点。

最后，我们将仍然看到一个完美地性能提升

|  |
| --- |
| test=# CREATE INDEX idx4 ON t\_demo (data);  CREATE INDEX  Time: 448498.535 ms (07:28.499) |

7分28秒。这个结果已经非常不错。但是让我们看看能否做得更好。到目前为止，我们看到的是检查点和I/O已经开始成为一个限制因素。因此，我们将尝试通过告诉PostgreSQL使用更大的检查点距离来改进这一点。在本例中，我们将改变postgresql.conf中下面的参数值：

|  |
| --- |
| checkpoint\_timeout = 120min  max\_wal\_size = 50GB  min\_wal\_size = 80MB |

这些参数可以简单地通过重新加载激活配置文件生效：

|  |
| --- |
| test=# SELECT pg\_reload\_conf();   pg\_reload\_conf  ----------------   t  (1 row) |

让我们使用更大的检查点距离下创建一个新索引。

在构建索引的同时查看进程表，将看到PostgreSQL花费大量时间在写WAL日志到磁盘上。如果我们坚持使用单一的固态硬盘，就没有什么优化空间了。然而，如果我们打出下一张王牌，会发生什么？额外增加硬件。如果我们在一个磁盘上创建了所有临时数据，将WAL发送到主磁盘上，并在第三个SSD上创建索引，会怎么样？这样我们就可以很好地分配所需的I/O量，看看会发生什么。

### 利用PostgreSQL中表空间来加速索引创建

如前所述，使用表空间并增加更多硬件可能是一个好主意。我很清楚，在现代云环境中这可能是不可能的。然而，在我的测试服务器上，我仍然可以得到一些奢侈的硬件：一批真正的固态磁盘驱动器。

所以，我们将尝试创建两个表空间来存储数据。除此之外，我将在PostgreSQL中利用这两个表空间来完成排序和存储新索引数据：

|  |
| --- |
| test=# CREATE TABLESPACE indexspace LOCATION '/ssd1/tabspace1';  CREATE TABLESPACE    test=# CREATE TABLESPACE sortspace LOCATION '/ssd2/tabspace2';  CREATE TABLESPACE |

设置PostgreSQL中临时数据的存放空间

|  |
| --- |
| test=# SET temp\_tablespaces TO sortspace;  SET |

下面开始创建索引：

|  |
| --- |
| test=# CREATE INDEX idx6 ON t\_demo (data) TABLESPACE indexspace;  CREATE INDEX  Time: 408508.976 ms (06:48.509) |

在索引创建过程中看到的是，吞吐量的峰值比以前更高，因为多个SSD可以同时工作。磁盘吞吐量不再是500MB/s的峰值，而是达到了900MB/s。整体速度也有所提升。已经在7分钟以下了，这个结果非常不错。

如果你在盒子中添加了更多的硬件，那么可以考虑同时使用所有磁盘创建一个文件系统。我没有时间测试这个优化点，但是我认为它可能与我在第一次测试中能想到的结果相似，甚至更好。

|  |
| --- |
| PostgreSQL, CREATE INDEX on many CPUs |

在PostgreSQL中使用多核创建索引。CREATE INDEX可以使用不止一个CPU。

**提示**：请注意数据类型的重要性。如果我们使用正常的整数值进行相同的测试，则可以在3分51秒内完成索引的创建。也就是说：数据类型非常重要。

在这篇博客中，你已经看到对索引创建的改进。但是，请记住，新索引并不总是有益的。无意义的索引甚至可能会减慢速度。要弄清楚哪些索引可能不需要，请阅读Laurenz Albe的[博客](https://www.cybertec-postgresql.com/en/get-rid-of-your-unused-indexes/)，文中介绍了怎样解决此类问题。

### 原文地址

<https://www.cybertec-postgresql.com/en/postgresql-parallel-create-index-for-better-performance/>