**理解Postgres中的缓存--深度指南**

译者：LittleWuCoding 校对：朱君鹏

最初发布于：2016年5月16日

最后更新于：2016年8月3日

**作者简介**

**Madusudanan.B.N，**software engineer based out of Chennai (India).

**译者简介**

**LittleWuCoding，**

**朱君鹏，**华东师范大学博士研究生，个人兴趣主要集中在：新型硬件（GPU、RDMA、FPGA）等在数据库中的应用，架构设计与并行计算。

**缓存**

缓存可以被认为是调整数据库系统性能的重要方面。虽然这篇文章主要关注postgres，但它可以很容易地与其他数据库系统进行比较和理解。

**什么是缓存，为什么我们需要缓存**

不同的计算机组件以不同的速度运行。我们人类在理解计算机所能达到的尺度上的数字方面非常差。

从下表我们就可以有一个清晰的认识。这些数字是按人的感知估计的。

|  |  |  |
| --- | --- | --- |
| **Access type** | **Actual time** | **Approximated time** |
| 1 CPU cycle | 0.3 ns | 1 s |
| Level 1 cache access | 0.9 ns | 3 s |
| Level 2 cache access | 2.8 ns | 9 s |
| Level 3 cache access | 12.9 ns | 43 s |
| Main memory access | 120 ns | 6 min |
| Solid-state disk I/O | 50-150 μs | 2-6 days |
| Rotational disk I/O | 1-10 ms | 1-12 months |
| Internet: SF to NYC | 40 ms | 4 years |
| Internet: SF to UK | 81 ms | 8 years |
| Internet: SF to Australia | 183 ms | 19 years |
| OS virtualization reboot | 4 s | 423 years |
| SCSI command time-out | 30 s | 3000 years |
| Hardware virtualization reboot | 40 s | 4000 years |
| Physical system reboot | 5 m | 32 millenia |

在数据库系统中，我们主要关注磁盘I / O。

与较新的固态硬盘相比，磁盘的随机I / O较差。

大多数OLTP工作负载都是随机I / O，因此从磁盘获取可能非常慢。

为了解决这个问题，postgres将数据缓存在RAM中，这可以大大提高性能。即使在SSD的情况下，RAM也要快得多。

缓存的这种一般概念对于几乎所有数据库系统都是通用的。

**了解术语**

在我们继续深入之前，有必要了解某些术语。因此我推荐阅读PostgreSQL 的物理存储（http://rachbelaid.com/introduction-to-postgres-physical-storage）。完成后，inter db（http://www.interdb.jp/pg/pgsql01.html）是另一个更深入的数据库。特别是关于堆元组的部分。官方文档也可以使用，但是有点难以理解（http://www.postgres.cn/docs/9.5/storage.html）。

无论内容如何，​​postgres都有一个称为页面的存储抽象（大小为8KB）。下图给出了一个粗略的想法。这个抽象是我们将在本文的其余部分处理的内容。

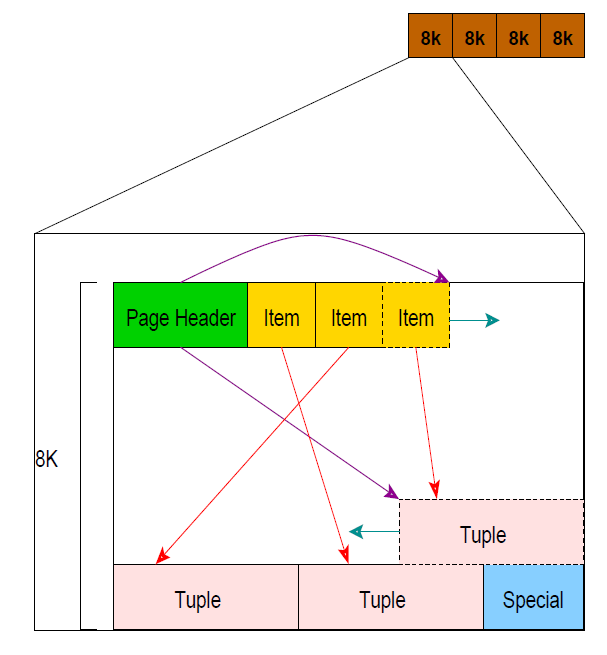


图1 postgres页面存储抽象

**什么被缓存了?**

Postgres缓存以下内容。

***Table data（表数据）*** 这是表的实际内容。

***Indexes（索引）***  所以也存储在8K块中，它们与表数据存储在同一位置，请参阅本文后续存储区域部分。

***Query execution plans（查询执行计划）***当您查看查询执行计划时，会有一个称为计划阶段的阶段，它基本上选择适合查询的最佳计划。 Postgres也可以缓存计划，这是基于每个会话，一旦会话结束，缓存的计划就会被丢弃。这对于优化/分析来说可能很棘手，但通常不太重要，除非您执行的查询非常复杂和/或存在大量重复查询。

是的，这里Postgres也把这个查询计划缓存了下来。这个查询计划一般是在优化或者analyze时会用到。当然，除非我么遇到那种特别复杂或者重复的查询，否则我们一般不用太关注它。文档很详细地解释了这些。我们可以查询pg\_prepared\_statements来查看缓存的内容。请注意，它不是跨会话可用的，并且只对当前会话可见。

我们将在本文中进一步探讨如何缓存表数据和索引的详细信息。

**存储区域**

Postgres有几个配置参数，理解它们的意义非常重要。

对于缓存，最重要的配置是shared\_buffers。

在postgres源代码的内部，这被称为NBuffers，并且所有共享数据都位于内存中。

shared\_buffers只是一个8KB块的数组。每个页面都有自己的元数据，以区分自己，如上所述。在postgres从磁盘中检出数据之前，它首先查找shared\_buffers中的页面，如果有命中，则它从那里返回数据，从而避免磁盘I / O。

**LRU/时钟扫描缓存算法**

将数据放入高速缓存并从中逐出的机制由时钟扫描算法控制。

它是为处理OLTP工作负载而构建的，因此几乎所有流量都在内存中处理。

让我们详细谈谈每个操作。

**缓冲区的分配**

Postgres是一个基于进程的系统，即每个连接都有自己的本机操作系统进程，它是从postgres根进程（以前称为postmaster）生成的。

当进程请求LRU高速缓存中的页面时（无论何时通过典型的SQL查询访问该页面都会执行此操作），它会请求缓冲区分配。

如果块已经在缓存中，它将被固定然后返回。固定过程是增加下面讨论的使用次数的一种方法。当使用计数为零时，称一个页面被取消固定。

只有当页面没有可用的缓冲区/插槽时，才会进行缓冲区换出操作。

**缓存区的换出**

决定哪些页面应该从内存中逐出并写入磁盘是一个典型的计算机科学问题。

普通LRU（最近最少使用）算法在现实中不能很好地工作，因为它没有前一次运行的记忆。

Postgres会跟踪页面使用情况计数，因此如果页面使用计数为零，则会从内存中换出并写入磁盘。当页面变脏时，它也会写入磁盘（见下文）。

因此，作为一个折中和替代，我们追踪并记录每个page的"usage count"，在有需要时，将那些"usage count"为0的page换出并写回到disk中。后面也会提到，脏页面也会被写回disk。

无论细节如何，缓存算法本身几乎不需要调整，并且比人们通常想象的要聪明得多。

**脏页面和缓存失效**

到目前为止，我们一直在讨论选择查询，DML查询会发生什么？

很简单，它们被写入相同的页面。如果存在于内存中，则将它们写入它，否则它们将从磁盘中获取然后写入它。

这就是脏页面的概念所在，即页面已被修改并且尚未写入磁盘。

在我们开始之前，还有一些更多的功课/学习要做，特别是关于WAL和检查点。[WAL](http://www.interdb.jp/pg/pgsql09.html) 和 [checkpoints](http://www.postgres.cn/docs/9.5/sql-checkpoint.html)。

WAL是一个重做日志，基本上可以跟踪系统发生的任何事情。这是通过将所有更改分别记录到WAL日志来完成的。 Checkpointer是一个将所谓的脏页定期写入磁盘并由时间设置控制的过程。它这样做，因为当数据库崩溃时，它不需要从头开始重放所有内容。

这是最常见的从内存中逐出的页面方式，LRU驱逐几乎从未在典型情况下发生过。

**从explain analyze命令理解缓存**

Explain是理解幕后发生的事情的绝佳方式。它甚至可以判断来自磁盘的数据块有多少，以及来自shared\_buffers（即内存）的数据块有多少。

下面的查询计划给出了一个例子，

performance\_test=# explain (analyze,buffers) select \* from users order by userid limit 10;

Limit (cost=0.42..1.93 rows=10 width=219) (actual time=32.099..81.529 rows=10 loops=1)

Buffers: shared read=13

-> Index Scan using users\_userid\_idx on users (cost=0.42..150979.46 rows=1000000 width=219) (actual time=32.096..81.513 rows=10 loops=1)

Buffers: shared read=13

Planning time: 0.153 ms

Execution time: 81.575 ms

(6 rows)

共享读取，意味着它来自磁盘并且没有缓存。如果再次运行查询，并且缓存配置正确（我们将在下面讨论它），它将显示为共享命中。

performance\_test=# explain (analyze,buffers) select \* from users order by userid limit 10;

Limit (cost=0.42..1.93 rows=10 width=219) (actual time=0.030..0.052 rows=10 loops=1)

Buffers: shared hit=13

-> Index Scan using users\_userid\_idx on users (cost=0.42..150979.46 rows=1000000 width=219) (actual time=0.028..0.044 rows=10 loops=1)

Buffers: shared hit=13

Planning time: 0.117 ms

Execution time: 0.085 ms

(6 rows)

通过这种方式可以非常方便地了解从查询角度缓存多少而不是了解OS / Postgres的内部。

**顺序扫描的情况**

顺序扫描是像这样的缓存的问题区域，即没有索引和postgres必须从磁盘获取所有数据。

由于单个seq扫描可以擦除缓存中的所有数据，因此处理方式不同。

它使用一系列总长度为256 K.B的缓冲区，而不是使用普通的LRU /时钟扫描算法。以下计划显示了如何处理。

performance\_test=# explain (analyze,buffers) select count(\*) from users;

Aggregate (cost=48214.95..48214.96 rows=1 width=0) (actual time=3874.445..3874.445 rows=1 loops=1)

Buffers: shared read=35715

-> Seq Scan on users (cost=0.00..45714.96 rows=999996 width=0) (actual time=6.024..3526.606 rows=1000000 loops=1)

Buffers: shared read=35715

Planning time: 0.114 ms

Execution time: 3874.509 ms

再次执行上面的查询。

performance\_test=# explain (analyze,buffers) select count(\*) from users;

Aggregate (cost=48214.95..48214.96 rows=1 width=0) (actual time=426.385..426.385 rows=1 loops=1)

Buffers: shared hit=32 read=35683

-> Seq Scan on users (cost=0.00..45714.96 rows=999996 width=0) (actual time=0.036..285.363 rows=1000000 loops=1)

Buffers: shared hit=32 read=35683

Planning time: 0.048 ms

Execution time: 426.431 ms

我们可以看到正好有32个块已经移入内存，即32 \* 8 = 256 KB。这在src / backend / storage / buffer / README中有解释。

**内存流和操作系统缓存**

Postgres作为跨平台数据库，在很大程度上依赖于操作系统来进行缓存。

shared\_buffers实际上是重复操作系统所做的事情。下面给出了数据如何流经postgres的典型图片。

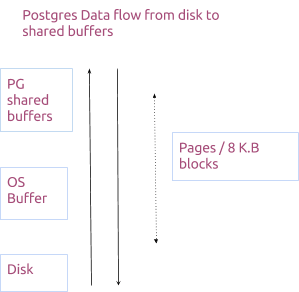


图2 postgres中的数据流

这首先是令人困惑的，因为缓存由OS和postgres管理，但是有理由这样做。

谈论操作系统缓存需要自己的另一个帖子，但网上有很多可以利用的资源。

请记住，操作系统缓存数据的原因与我们上面看到的相同，即为什么我们需要缓存？

我们可以将I / O分类为两种类型，即读取和写入。更简单的说，数据从磁盘流向内存，用于读取和从内存流向磁盘以进行写入。

**读操作**

对于读取，当您考虑上面的流程图时，数据从磁盘流向OS缓存，然后流向shared\_buffers。我们已经讨论过如何将页面固定到shared\_buffers上，直到它们变脏/取消固定为止。

有时，OS缓存和shared\_buffers都可以保存相同的页面。这可能导致空间浪费，但请记住OS缓存使用简单的LRU而不是数据库优化的时钟扫描。一旦页面在shared\_buffers上受到攻击，读取就永远不会到达操作系统缓存，如果有任何重复，则可以轻松删除它们。

实际上，没有多少页面堆叠在内存区域上。

这是建议仔细调整shared\_buffers大小的原因之一。使用硬性和快速的规则，例如给予它大部分内存或者给它太少的内存会损害性能。

  我们将在下面讨论更多优化问题。

**写操作**

写入从内存到磁盘的流。这就是脏页面的概念所在。

 一旦页面被标记为脏，它就会被刷新到OS缓存，然后写入磁盘。这是操作系统可以根据传入流量更自由地调度I / O的地方。

  如上所述，如果OS高速缓存大小较小，则它不能重新排序写入并优化I / O。这对于繁重的工作量来说尤为重要。因此，OS缓存大小也很重要。

**初始配置**

与许多数据库系统一样，没有银弹配置可以正常工作。 PostgreSQL附带了一个基本配置，调整了广泛的兼容性而不是性能。

  数据库管理员/开发人员有责任根据应用程序/工作负载调整配置。然而，postgres的人们有很好的文档说明从哪里开始。

一旦设置了默认/启动配置。我们可以进行负载/性能测试，看看它是如何阻碍的。

请记住，初始配置也针对可用性而非性能进行了调整，最好始终进行实验并获得更适合所考虑工作负载的配置。

**随时优化**

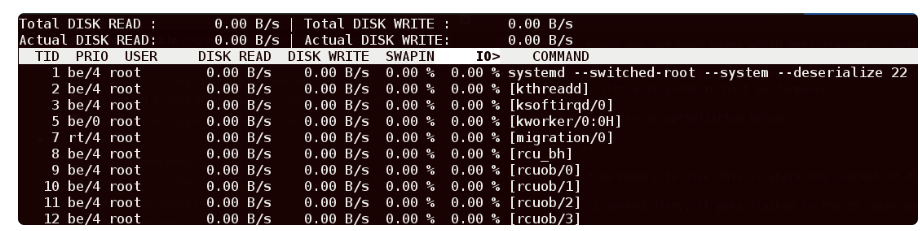
如果您无法测量某些东西，则无法对其进行优化

使用postgres，有两种方法可以衡量。

**操作系统层面**

虽然没有普遍的共识，哪个平台postgres效果最好，但我假设你在linux系列操作系统中使用了一些东西。但这个想法有点类似。

首先，有一个名为Io top的工具可以测量磁盘I / O。与top类似，在测量磁盘I / O时可以派上用场。只需运行命令iotop来测量写入/读取。



这可以提供有关postgres在负载下如何表现的有用见解，即命中磁盘的数量以及可以根据生成的负载到达RAM的数量。

**直接从postgres监控**

最好直接从postgres监控某些东西，而不是通过操作系统路径。

通常情况下，如果我们认为postgres本身存在问题，我们会进行操作系统级监控，但这种情况很少发生。

用postgres，我们可以使用几种工具来测量内存方面的性能。

* Explain

默认是SQL explain。比任何其他数据库系统提供的信息更多，但有点难以理解。需要练习习惯。不要错过几个有用的标志，特别是我们之前看到的缓冲区。

查看以下链接以深入了解说明。

* + http://postgresguide.com/performance/explain.html
  + https://github.com/AlexTatiyants/pev
* **查询日志**

查询日志是了解系统内部情况的另一种方法。

我们可以使用log\_min\_duration\_statement参数仅记录跨越特定持续时间的查询或以其他方式调用慢查询日志，而不是记录所有内容。

* **auto\_explain插件**

这是另一件很酷的事情，你可以自动记录执行计划和慢速查询。对于调试非常有用，无需手动运行解释。

* **pg\_stat\_statements插件**

上述方法很好，但缺乏统一的视图。

这是在postgres内部构建的模块，但默认情况下禁用。  
我们可以通过创建扩展pg\_stat\_statements来实现这一点。

一旦启用此功能，在运行了大量查询后，我们就可以触发如下所示的查询。

SELECT

(total\_time / 1000 / 60)::decimal as total\_minutes,

(total\_time/calls)::decimal as average\_time,

query

FROM pg\_stat\_statements

ORDER BY 1 DESC

LIMIT 100;

提供有关查询花费的时间和平均值的大量详细信息。

这种方法的缺点是需要一定的性能，因此通常不建议在生产系统中使用。

* **pg\_buffer\_cache和PG fincore**

如果你想深入一点，那么有两个模块可以直接挖掘到shared\_buffers和OS缓存本身。

需要注意的一点是，explain（analyze，buffers）仅显示来自shared\_buffers的数据，而不显示来自OS缓存的数据。

* + PG buffer cache

这有助于我们实时查看共享缓冲区中的数据。从shared\_buffers收集信息并将其放在pg\_buffercache中以供查看。

示例查询如下所示，其中列出了前100个表以及缓存的页数。

SELECT c.relname,count(\*) AS buffers

FROM pg\_class c INNER JOIN pg\_buffercache b

ON b.relfilenode=c.relfilenode INNER JOIN pg\_database d

ON (b.reldatabase=d.oid AND

d.datname=current\_database())

GROUP BY c.relname ORDER BY 2 DESC LIMIT 100;

* + PG fincore

这是一个外部模块，提供有关操作系统如何缓存页面的信息。这是非常底层的操作，也非常强大。

* + **pg\_prewarm插件**

这是一个内置模块，可以将数据实际加载到shared\_buffers / OS缓存中或两者中。如果您认为内存预热是问题，那么这对于调试非常有用。

还有几个，但我已经列出了最流行和易于使用的方法来理解postgres缓存以及一般情况。有了这些工具，由于内存问题而导致数据库运行缓慢，再也没有借口了。

原文链接：

https://madusudanan.com/blog/understanding-postgres-caching-in-depth/