Oracle 数据库服务器 IO 高的分析方案（理论讲解+案例分享）

作者: 初见博客 分类: oracle调优 发布时间: 2022-09-20 17:06

本文原题《ORACLE数据库服务器IO高的分析方案和案例探讨》

目录：

作者：黄远邦（小y），就职于北京中亦安图科技股份有限公司，任数据库团队技术总监。长期活跃于国内多家银行总行生产数据中心,提供Oracle第三方服务，解决过无数疑难问题，在业内具有极佳的口碑。

前言

远邦在为数据中心提供Oracle第三方服务的过程中接触过很多系统/存储管理员，发现很多SA对Oracle数据库缺乏足够的了解，导致在处

理综合问题时容易各说各话，因此有了写一个系列文章的想法，本意是尽可能用大白话为大家普及一些常见问题所需的理论，辅以几个实际的案例分析，希望对大家以后的工作有所帮助。

言归正传，在部署了ORACLE数据库的服务器上，我们大家或多或少的遇到过下列情况：

1. 业务系统运行缓慢，作为系统管理员需要检查包括IO在内的系统资源，这时系统管理员、存储管理员可能得到DBA（数据库管理员）的反馈说，IO的响应时间很慢，达到了30毫秒以上，要求解决。但存储管理员检查又不存在热点盘的情况，系统的IO量就是很大，除了使用更多的RAID组来重新分布数据、更换为更高端的存储外，似乎没有太好的办法；

2. 我们可能通过iostat和sar -d命令观察到磁盘的busy很高、每秒的IOPS很高、每秒的IO读写量很大、HBA卡的流量很高等危险的现象；

3. IO响应时间长，到底是导致业务慢的原因还是结果？

4. IOPS很高、IO读写量很大，到底是原因还是结果？

5. 除了硬件的扩容或升级，难道没有别的解决方法么？

6. 如何识别ORACLE服务器上的IO来源，如何判断这些IO是否是有效IO,怎么消除无效IO？

7. 作为系统管理员和存储管理员需要掌握哪些数据库简单技能才不会在出现IO问题时处于被动的局面？

8. ORACLE DBA评判IO是否有性能问题的标准是什么？

9. ORACLE数据库的IO有什么特点？哪些IO是比较关键，是必须保障性能的？

我们将通过理论和实际案例穿插介绍的方式为大家进行讲解和分享，希望对大家有所启发。

本文是系列的第一篇。需要说明的是，由于篇幅有限，会暂时省略掉部分在过程中实际发生但与本主题不是那么密切的内容，如UNDO、checkpoint等内容。

同时考虑到AIX专家俱乐部可能更多的是系统/存储管理员，因此会有部分科普的内容，水平较好的ORACLE DBA可以自行跳到案例探讨环节。对于没有真正做过DBA的同学来说，ORACLE可能稍微有点难，但只要静下心来花点时间去主动了解了，那就不难了。

慧眼识珠——服务器磁盘这么繁忙，到底是谁干的？
出一个问题时段的AWR报告，将awr报告对应的html文件下载到PC终端
用IE等浏览器打开，查找”SQL ordered by Reads”部分，如下图所示：

SQL ordered by Reads

- %Total - Physical Reads as a percentage of Total Disk Reads
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Disk Reads: 17,068,968
- Captured SQL account for 99.8% of Total

Physical Reads	Executions	Reads per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
17,032,430	8	2,129,053.75	99.79	3,492.79	9.77	90.14	9q96pxjr5b742	SQL*Plus	select
0	3	0.00	0.00	0.00	0.00	0.00	062savj8zgzut		UPDAT
0	1	0.00	0.00	0.00	0.00	0.00	08qpghs7y4q2f		UPDAT
0	1	0.00	0.00	0.00	0.00	0.00	0cn2wm9d7zq8d		SELEC
0	9	0.00	0.00	0.00	0.00	0.00	0k8522rmdzg4k		select
0	11	0.00	0.00	0.01	0.00	0.00	0kkhhb2w93cx0		update
0	10	0.00	0.00	0.00	0.00	0.00	0ws7ahf1d78qa		select
0	1	0.00	0.00	0.00	0.00	0.00	12xttzc893c2g		insert i
0	1	0.00	0.00	0.00	0.00	0.00	15rbgh4d2ku4u		insert i
0	1	0.00	0.00	0.00	573.95	0.00	18c2yb5aj919t		SELEC

可以看到：

1) 排名第一的SQL语句，占了整个数据库服务器IO的99.79%

2) 一共执行了8次，每次执行发生的IO是2，129,053个BLOCK，一个BLOCK是8K，即每次执行该SQL，将发生2，129,053*8K=16.24G

知识点：

BLOCK是ORACLE数据文件的最小分配单元，类似LV的PP,数据就存储在BLOCK中，一个BLOCK可以存储几十到几百条不等的用户表的数据。

是不是很简单？

当然，大家也可以用操作系统的命令看到进程级的IO分布情况。

例如Linux环境下通过pidstat -d 可以监控哪个进程IO消耗较高。

当然，采用操作系统方式查看进程级别的IO分布的方式的缺点是很显然的，更可怕的是，这表明你依然在把数据库当黑盒子来看待，进程具体在做什么？为什么IO那么高?

我们需要继续往前一步。

```
# pidstat -d
Linux 2.6.32-642.el6.x86_64 (ntos)      02/13/2017      _x86_64_      (2 CPU)

05:16:08 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
05:16:08 PM         1     60.58      0.81      0.01  init
05:16:08 PM       360      0.00      1.13      0.00  jbd2/dm-0-8
05:16:08 PM       457      7.59      0.00      0.00  udevd
05:16:08 PM       990      0.13      0.00      0.00  udevd
05:16:08 PM      1111      0.00      0.02      0.00  jbd2/dm-3-8
05:16:08 PM      1113      0.00      0.05      0.00  jbd2/dm-5-8
05:16:08 PM      1115      0.00     60.30      0.00  jbd2/dm-4-8
05:16:08 PM      1117      0.00      0.66      0.00  jbd2/dm-2-8
05:16:08 PM      1131      0.00      0.02      0.00  jbd2/dm-10-8
05:16:08 PM      1393      0.00      0.04      0.00  auditd
05:16:08 PM      1425      0.02      0.00      0.00  portreserve
05:16:08 PM      1435      0.15      0.05      0.00  rsyslogd
05:16:08 PM      1469      0.00      0.00      0.00  irqbalance
05:16:08 PM      1487      0.22      0.00      0.00  rpcbind
```


谨记于心——ORACLE DBA判断IO有性能问题的标准
知识点：
一般来说，如果单个IO的响应时间在20毫秒以内，是可以接受的，较好的性能应该在10个毫秒以下，越低越好。超过20毫秒的单个IO响应时间，则可认为性能不佳，需要做调优。需要说明的是，对于IO次数只有个位数的文件，IO超过20毫秒，也是可以接受的，因为在存储层面不容易被cache。

通过OS和数据库AWR报告两个方式均可以判断IO是否有问题，建议以OS方式为准。

1. 操作系统方式

sar –d 2 10的输出中，avwait和avserv两列之和即为IO的响应时间（AIX环境），单位为毫秒。LINUX环境下有区别，IO的响应时间为AVWAIT列。

```
/tmp#sar -d 1 120

15:01:22     device    %busy    avque    r+w/s    Kbs/s   avwait   avserv

15:01:32     hdisk0        0      0.0        0        0      0.0      0.0
             hdisk1        0      0.0        0        0      0.0      0.0
             hdisk3        0      0.0        0        0      0.0      0.0
             hdisk2        0      0.0        0        0      0.0      0.0
             hdisk4       16      0.0        6      140      0.0   4564.4
             hdisk6        0      0.0        0        0      0.0      0.0
             hdisk5        0      0.0        0        0      0.0      0.0
             hdisk7        0      0.0        0        0      0.0      0.0
               cd0         0      0.0        0        0      0.0      0.0
      hdiskpower0         0      0.0        0        0      0.0      0.0
      hdiskpower1        16      0.0       11      140      0.0      0.0

15:01:45     hdisk0        0      0.0        0        0      0.0      0.0
             hdisk1        0      0.0        0        0      0.0      0.0
             hdisk3        0      0.0        0        0      0.0      0.0
             hdisk2        0      0.0        0        0      0.0      0.0
             hdisk4       23      0.0       13      163      0.0   2029.8
             hdisk6        0      0.0        0        0      0.0      0.0
```


可以看到：

hdisk4上单个IO的响应时间达到4000多毫秒和2000多毫秒，远远大于20毫秒，IO性能到了无法忍受的地步，需要尽快分析是否存储存储cache被关闭，硬盘是否出现故障、链路是否出现问题等情况。

2. 数据库AWR报告方式

下图的Av Rd(MS)表示单次读的毫秒数，即为单个IO的响应时间。可以看到，在0.01毫秒，远远低于20毫秒，IO性能非常的好！（能达到整个性能，往往是在文件系统缓存中被缓存了）

File IO Stats

- ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
SYSAUX		0	0			278	0	0	0.00
SYSTEM		0	0			10	0	3	0.00
		33,736	61	0.01	125.89	0	0	0	0.00
		34,200	61	0.01	125.68	0	0	0	0.00
		34,016	61	0.01	125.01	0	0	0	0.00
		34,096	61	0.01	125.27	0	0	0	0.00
UNDOTBS1		0	0			12	0	0	0.00

下图的Av Rd(MS)表示单次IO读的毫秒数，即为单个IO的响应时间。可以看到，大部分数据文件的IO响应时间超过40毫秒，远远大于20毫秒，IO性能不理想，在对存储进行扩容或者升级前，应该先好好分析IO是否是无效IO,是否可以消除无效IO！通过SQL优化消除无效IO，可以有效保护存储等硬件的投资，满足未来多年的业务发展，而不是盲目扩容。

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
		33,149	1	47.76	12.26	870	0	762	14.54
		32,017	1	43.15	11.07	1,874	1	748	16.62
		32,257	1	44.71	12.79	1,074	0	713	13.17
		33,058	1	38.27	9.86	884	0	802	15.61
		33,328	1	53.85	14.97	806	0	956	18.23
		33,577	1	51.23	14.12	1,836	1	937	21.64
		34,055	1	53.73	11.71	1,972	1	1,095	22.66

带刀侍卫——处理IO问题必须掌握的一个ORACLE工具

上述的AWR报告是怎么获得的呢？什么是AWR报告呢?容许我啰嗦一下：

很多同学可能听过AWR报告，收集AWR报告的步骤是固定的，很简单，步骤如下：

#su – oracle

$sqlplus “/ as sysdba”

SQL>exec dbms_workload_repository.create_snapshot();

SQL>@?/rdbms/admin/awrrpt.sql

依次输入

1）Html

2）回车

3）输入想要抓取的时间范围所对应的开始snap_id

4）输入想要抓取的时间范围所对应的结束snap_id

5）输入想要保存为报告的名称（自己选个名字即可）

6）刷屏出报告中……

通过AWR报告这个数据库内置的工具，可以清楚的了解到某个时段，数据库中到底执行了哪些SQL，产生了多少IO，消耗了多少CPU。

也许有人会说，AWR报告我已经会出了，但看懂AWR报告才是关键。

是的，你需要更深入地了解一些数据库知识，不妨耐心往下看。

说难不难——用几句话来说清ORACLE数据库
ORACLE数据库，简单来说，主要就是对外提供数据存储服务，同时可以通过内置的丰富的SQL/PLSQL接口，对外提供数据检索、比对、关联等计算服务。

具体一点来说，ORACLE数据库通过服务器上的一组进程（后台进程与前台进程）和内存结构，对存储上的数据进行读写和计算。

知识点：

ORACLE实例和ORACLE数据库的区别是什么？

做SA的很多同学一直不太清楚什么是ORACLE实例，其实很简单：

进程和内存结构加起来就称之为ORACLE实例即instance。因为进程和内存是随着数据库/OS重启而消失的，因此oracle实例中不恒久的存储数据。

我们的用户数据，最终是存放在磁盘中的，具体来说，是存在在磁盘中的数据文件中。数据文件、控制文件、在线日志文件，这些物理上存在的文件组成了我们传统所说的ORACLE数据库。

知识点：

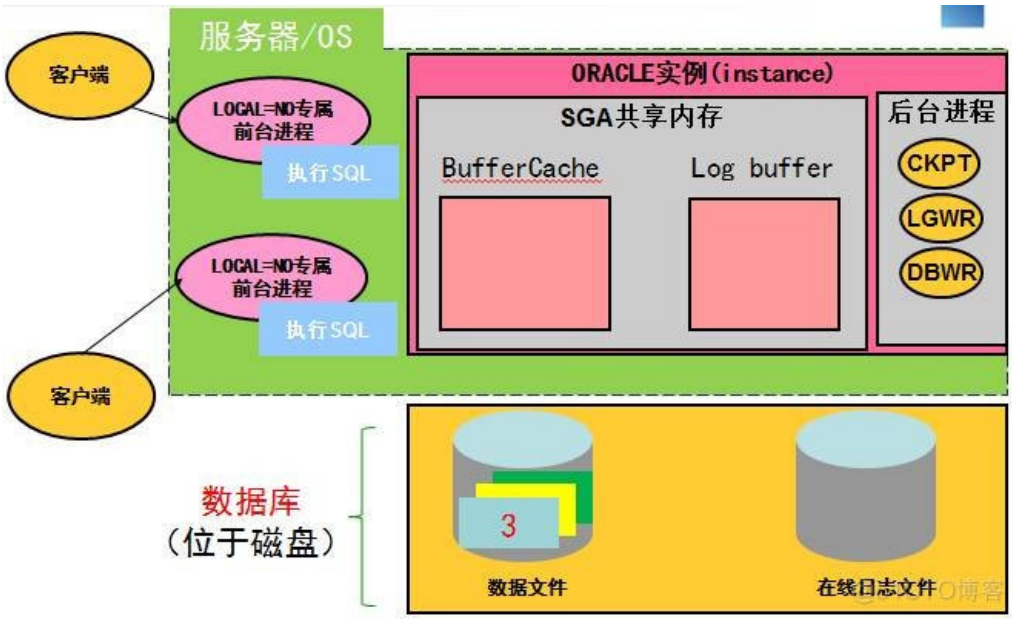
ORACLE是一种典型的C/S架构么？

答：是的。

我们知道，客户端或者应用程序是通过SQL语句和数据库交互的。

当客户端或应用程序想连接到数据库执行SQL语句来完成查询和增删改数据时，ORACLE服务器上默认地，将为每一个客户端对应地创建一个专有的服务进程（在操作系统上将看到LOCAL=NO的进程）来单独的为这个客户端服务，这个专有的服务进程帮助这个客户端执行相关SQL，当执行完SQL时，这个进程也不能做别的，就只能等着客户端发起的下一条SQL。这就好比，我们去一个高级的餐厅，有一个专属的服务员为我们服务。

我们接下来用一幅图来总结上述的内容，即ORACLE的简单架构：



可以看到：

每个客户端要执行SQL，只需要通过网络将SQL传到对应的服务进程，由服务进程帮忙执行即可。多个客户端则对应多个服务进程（见上图中的LOCAL=NO专属前台进程），这些服务进程我们也称之为前台进程。

ORACLE包含内存结构和后台进程

1. SGA共享内存，又可细分为

1) Buffer cache,用来缓存最近访问的数据，避免出现IO。Buffer cache中的数据可能比磁盘中的数据要新，例如读进内存后再修改为新的数据。我们称只要的数据为脏数据，脏块。

2) Log buffer，用一组记录来表示对数据库的修改过程，我们称之为改变向量。

3) 其他如shared pool/large pool/java pool/stream pool等，不在此介绍

2. ORACLE后台进程，又可细分为

1) DBWR进程，由于ORACLE定期会像word那样暂存一下最近所做的修改（我们称之为检查点checkpoint触发DBWR写脏数据），就是将buffer cache中的脏数据写回磁盘中的数据文件。这个IO属于随机写。

2) LGWR进程，在提交commit命令发出时，将log buffer中的修改记录以同步IO的形式写到磁盘中的在线日志文件后返回，commit才能完成，此时虽然buffer cache内存中的数据比磁盘中的数据文件中的数据要新，但是因为已经确保有一份修改过程写到了磁盘的在线日志文件，这个时候即使数据库掉电，也可以通过重新执行在线日志文件的修改记录，来保证数据不出现丢失。这在任何关系型数据库中常见的“日志先行”策略。由于lgwr进程采用追加写的方式把改变向量写到在线日志文件后面，因此LGWR的IO属于连续写。

3) 其他以ora_开头的oracle后台进程，如pmon/smon/ckpt,不在此介绍

知识点：

LGWR进程的IO是否支持写到文件系统缓存就返回？

不支持，LGWR进程的IO是透写的。如果只写到文件系统缓存（如果数据文件存放在文件系统缓存）就返回，则一旦系统crash,文件系统缓存来不及刷到磁盘，则会出现用户commit后已经提示修改成功，但由于log buffer/buffer cache中的改变最终没有落盘而出现数据丢失的情况。

LGWR进程的IO是异步IO么？

不是，因为要确保数据不丢失，lgwr必须等IO返回才会接着处理下一个IO写请求。

ORACLE架构中最大的瓶颈在哪里？

在ORACLE 12C之前，Lgw后台进程只有1个,由于所有进程在commit前都需要通知lgwr进程帮忙把之前在log buffer中生成的修改过程记录(改变向量)写到磁盘中。当大量进程要同时请lgwr进程帮忙写时，就出现排队的情况。在高并发的联机交易OLTP系统中，单进程的lgwr进程有可能成为一个大瓶颈，特别是在无法保证在线日志IO写性能的情况下，很容易出现排队等lgwr进程的情况。这其实也是很容易引发问题的一个点，是ORACLE一个相对脆弱的地方。

知识点：

为什么ORACLE那么吃内存？服务器一半左右的内存都被ORACLE吃掉了…

因为IO相比内存要慢非常多，因此很多关系型数据库为了更好的性能，大量采用内存换IO的策略,ORACLE也不例外，具体来说，ORACLE利用SGA中的buffer cache来缓存最近访问的数据，从而避免再次访问时需要发生IO。Buffer cache通常会占到SGA大小的80%，即buffer cache占到服务器内存的50%*0.8=40%左右。

案例分享（一）每秒800M IO流量
1. 问题描述：

客户反映，数据库服务器的IO量非常大，达到每秒800M，几乎将HBA的带宽打满,交易出现缓慢的情况。

2. 分析过程：

收集当前时段的Oracle AWR报告，找到Load Profile部分，如下图所示

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	30.4	0.6	0.09	0.02
DB CPU(s):	2.8	0.1	0.01	0.00
Redo size:	118,802.2	2,312.3		
Logical reads:	215,924.9	4,202.7		
Block changes:	866.1	16.9		
Physical reads:	104,207.9	2,028.3		
Physical writes:	320.0	6.2		
User calls:	1,764.5	34.3		
Parses:	105.4	2.1		
Hard parses:	78.4	1.5		
W/A MB processed:	6.3	0.1		
Logons:	0.2	0.0		
Executes:	346.6	6.8		
Rollbacks:	2.1	0.0		

可以看到：

每秒的物理读Physical Reads达到104，207个BLOCK，每个BLOCK的大小是8K

即每秒的IO量达到104207*8K=814M！

IO的流量太大了，几乎打满了HBA所支持的吞吐，影响到整体的性能是必然的！

根据上述知识点，我们知道，ORACLE采用内存换IO的策略，避免出现过多IO。

那么我们不妨猜一下，是否是因为buffer cache太小导致IO无法缓存呢？

进一步检查AWR报告中的”Cache Sizes”部分

如下图所示，buffer cache大小仅为128M！而shared pool大小为15,232M.

而该服务器配置了60G的内存！

我们提到，buffer cache可以设置到服务器内存的40%，即24G。

Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	128M	128M	Std Block Size:	8K
Shared Pool Size:	15,232M	15,232M	Log Buffer:	72,708K

通过查找AWR报告中的”Buffer Pool Advisory”，如下图所示，可以看到：

当buffer cache从128M设置到256M时，IO即物理读的个数将从3600万下降到1900万，就下降了一倍！

Buffer Pool Advisory

- Only rows with estimated physical reads >0 are displayed
- ordered by Block Size, Buffers For Estimate

P	Size for Est (M)	Size Factor	Buffers (thousands)	Est Phys Read Factor	Estimated Phys Reads (thousands)	Est Phys Read Time	Est %DBtime for Rds
D	128	1.00	16	1.00	36,395,517	1	2285410.00
D	256	2.00	32	0.52	19,035,327	1	940171.00

3. 原因

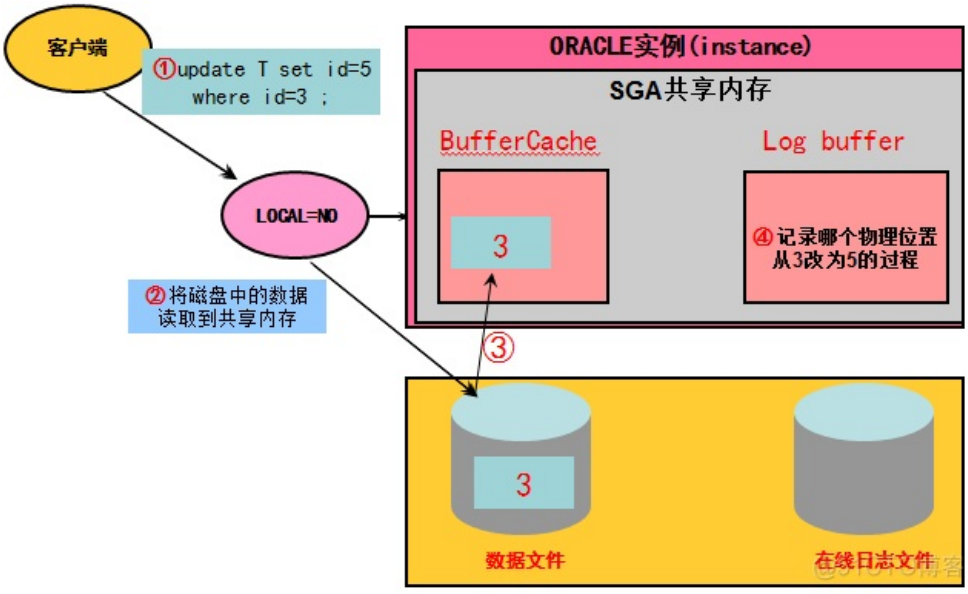
难道是配置失误？不是的。实际上，客户通过一个memory_target参数设置了为ORACLE数据库分配总计40G的内存。这是一种常见的做法，即交给ORACLE来动态分配在SGA和PGA之间，在SGA内部各个组件的内存大小。由于内存动态调整算法的不完善，导致过多的内存分给了PGA而SGA不够,又由于应用程序绑定变量的使用不够理想，导致shared pool不断的膨胀，一步一步地，将buffer cache压缩到了只剩128M.

4. 解决方法

为buffer cache设定一个基准值20G后，IO高的问题得到解决。

我们首先接触了第一个IO高的案例，接下来，我们通过一个例子来进一步的学习ORACLE的工作过程，以及更多的了解ORACLE的IO特点。

活得明白——一个例子说明ORACLE的工作过程



当客户端或者应用程序发起一条UPDATE语句时，到底经历了哪些事情和哪些IO？

简化后的过程如下：

客户端连接到数据库后，数据库服务器上将创建一个LOCAL=NO的进程来专门为这个客户端服务

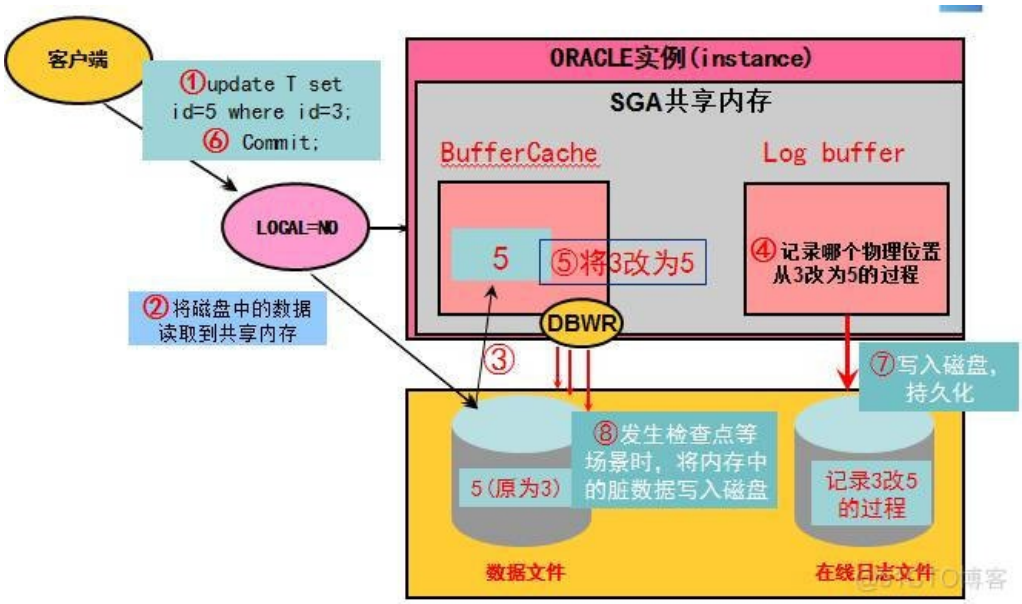
1. 客户端发起update T set id=5 where id=3的SQL语句，其中表T的大小为1G，表上不存在任何索引

2. 服务器上的服务进程（LOCAL=NO），将判断表T的数据BLOCK在内存中的buffer cache是否存在，如果不存在，则由服务进程发起IO，从磁盘先后将1G的数据读到内存中。具体来说，是先读取16个BLOCK即128K（一个BLOCK可以存储几十到几百条不等的记录），然后逐个判断这些BLOCK中是否存在id=3的数据。这个IO属于随机读，由LOCAL=NO前台进程来发起IO。

3. 最后在内存中同时存在一个BLOCK，BLOCK中存在id=3这条满足条件的记录

4. 接下来，在前台进程将id=3修改为id=5之前，需要先在log buffer中生成哪个BLOCK哪个位置从3修改为5的过程的对应记录（改变向量）

5.将内存buffer cache中的id=3修改为id=5，见下图中中的步骤5



6. 客户端发起commit

7. 因为一旦commit,则表示数据持久化，即表示将不会随着数据库crash/os 重启而丢失，因此，此时lgwr进程需要将log buffer中id=3修改为id=5的记录写到磁盘中的在线日志文件。这个IO只能是同步IO（非异步IO），等IO确认写到磁盘后，步骤6的commit完成，返回客户端的结果为修改成功。此时虽然buffer cache内存中的数据(id=5)比磁盘中的数据文件中的数据(id=3)要新，但是因为已经确保有一份修改过程(id=3修改为id=5)写到了磁盘的在线日志文件，这个时候即可数据库掉电，也可以通过重演在线日志文件的修改记录(id=3修改为id=5)，来保证数据最终被修改为5（宕机前的状态），即保证了数据不会出现丢失。

8. 当发生checkpoint等场景时，DBWR进程将内存buffer cache中的脏数据(内存比磁盘数据要新)写到磁盘中的数据文件，这个过程是异步的。

知识点：

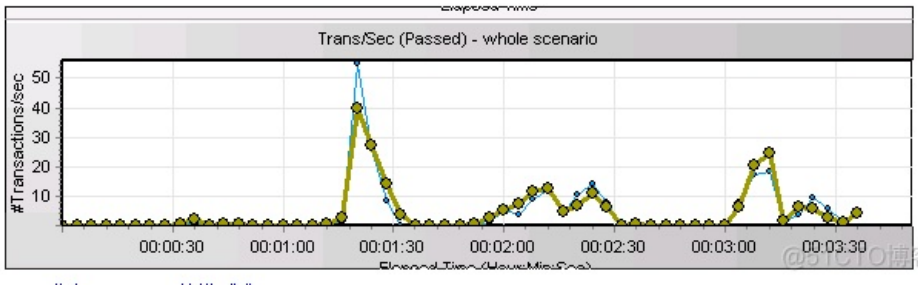
ORACLE为什么不把buffer cache中的脏数据也实时地在commit提交时刷到磁盘中？

Lgw进程将log buffer中的修改记录（改变向量）写到磁盘中的在线日志文件，是采用追加写到在线日志文件后面，因此LGWR的IO属于连续写。

而内存中的数据刷到磁盘中，属于随机写(每个客户端每次可能改不同物理位置的记录)，随机写的性能显然不如连续写的性能好，因此ORACLE允许脏数据的存在，而不是实时地在commit提交时将脏数据刷到磁盘中，采用异步的方式往下写即可，因为lgwr的连续写已经保证了数据不会出现丢失。

案例分享（二）压测TPS上不去
1.问题描述：

客户新上的一个关键业务系统，在做上线前的压力测试时，应用的并发无法达到上线前的并发指标和响应时间指标要求。压测时TPS的曲线很不稳定，如下所示：



2. 分析过程：

从上述知识点可以知道：

ORACLE中LGWR进程只有一个，由于所有进程在commit前都需要通知lgwr进程帮忙把之前在log buffer中生成的修改过程记录(改变向量)写到磁盘中。

当大量进程要同时请lgwr进程帮忙写时，就出现排队的情况。

在高并发的联机交易OLTP系统中，单进程的lgwr进程有可能成为一个大瓶颈，特别是在无法在线日志IO写性能出现问题的情况下。

因此，我们需要检查lgwr进程的状态。

通过gv$session观察RAC两个节点lgwr进程写日志的情况，结果如下图所示：

```
SQL> select event,state,seq#,seconds_in_wait,p1,p2 from gv$session where program like '%LGWR%';

EVENT                                    STATE                 SEQ# SECONDS_IN_WAIT         P1         P2
---------------------------------------- ------------------- ---------- --------------- ---------- ----------
log file parallel write                  WAITING                35693              13          1          2
rdbms ipc message                        WAITING                50909               2        300          0

SQL> /

EVENT                                    STATE                 SEQ# SECONDS_IN_WAIT         P1         P2
---------------------------------------- ------------------- ---------- --------------- ---------- ----------
log file parallel write                  WAITING                35693              19          1          2
rdbms ipc message                        WAITING                50913               2        300          0

SQL> /

EVENT                                    STATE                 SEQ# SECONDS_IN_WAIT         P1         P2
---------------------------------------- ------------------- ---------- --------------- ---------- ----------
log file parallel write                  WAITING                35693              21          1          2
```


可以看到：

RAC(数据库集群)两个节点中，只有1个节点出现log file parallel write的等待，该等待表示lgwr进程正在对磁盘的在线日志文件进行写操作。

在state是waiting的情况下，节点1 log file parallel等待的seq#都是35693，但是seconds_in_wait达到了21秒。简单来说，就是lgwr进程写一个IO需要21秒！

这意味着，压测时所有并发进程必须要发生等待，等lgwr进程完成这个的IO，才可以继续通知LGWR进程帮忙刷log buffer的改变向量，因此从压测的TPS曲线来看，就是不稳定，出现了大幅衰减。

至此，我们可以肯定，IO子系统有问题。需要重点排查IO路径下的光纤线、SAN交换机、存储的报错和性能情况。

考虑到客户那边管存储的团队/部门可能不承认数据库的IO慢的证据，同时为了让对方增加排查力度，远邦让客户发出以下命令，查看多路径软件的IO情况，结果如下图所示：

```
[root@XXXX log]# dlnkmgr view -path
 iLU              ChaPort Status     Type IO-Count    IO-Errors    CCNum HDevName
 0006             1A      Online     Own      726321           5     0 sddlmac
 0005             1A      Online     Own      724749           7     0 sddlmaa
 0008             1A      Online     Own         134           0     0 sddlmae
 0007             1A      Online     Own      719857           5     0 sddlmab
 0010             1A      Online     Own         130           0     0 sddlmai
 0009             1A      Online     Own         153           0     0 sddlmad
 0011             1A      Online     Own     1795741         461     0 sddlmaf
 0015             1A      Online     Own         593           1     0 sddlmaj
 0014             1A      Online     Own     1224304         467     0 sddlmal
 0012             1A      Online     Own     1383731         453     0 sddlmah
 0013             1A      Online     Own     1228823         505     0 sddlmag
 0016             1A      Online     Own         647           0     0 sddlmak
```

节点1上出现明显的IO ERROR，并且在持续增加！

继续检查节点2，发现节点2上没有任何IO ERROR！

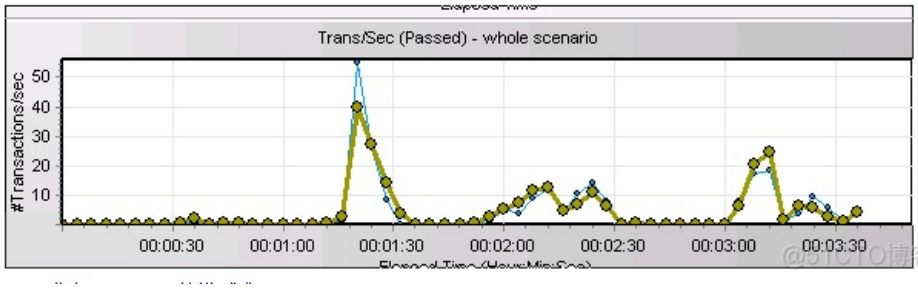
这个与gv$session仅有一个进程在等log file parallel write写完是完全吻合的。

3. 原因

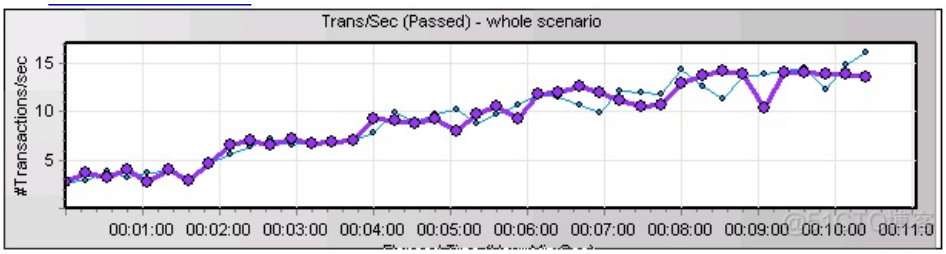
在铁的证据面前，客户的存储团队没有再挣扎，而是开始认认真真逐个在排查，最终在更换了光纤线后问题得到圆满解决。以下是更换光纤线后再次压测的等待事件！

4. 问题得到解决

压测的TPS曲线从原来的波浪形

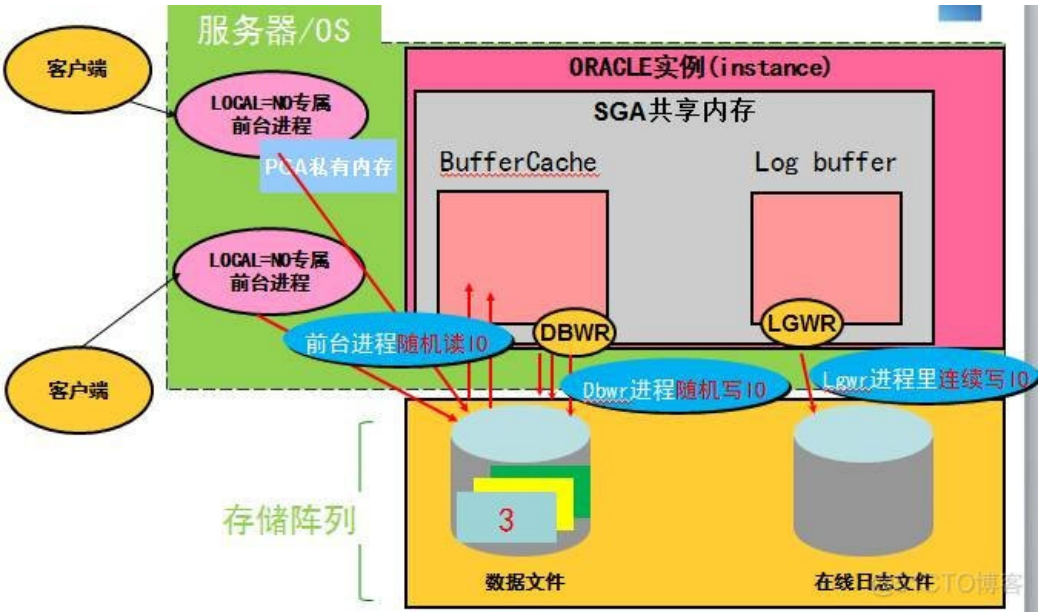


变成了如下的良好曲线



牢记于心——一幅图来总结ORACLE的IO特点

下图显示了数据库关键IO的特点



知识点：

1. 读数据由前台进程各自来完成，IO的特点是随机读，将数据读取到内存中再进行操作

2. DBWR进程的IO特点是随机写，DBWR进程支持多个进程同时往下刷数据，ORACLE为了避免在某一个时间点大量往下写脏块而导致磁盘压力过大，会采用异步地、慢慢地方式来刷脏块，减少IO压力，但当发出checkpoint和archive log current/all命令时，将激活大量DBWR进程全力往下刷脏块，可能对IO造成较大压力而影响整体性能，影响LGWR进程的写性能和响应时间。

3. Lgwr进程是连续写，有条件的情况下尽量为Redo分配单独的RAID组，物理上和其他文件分开。

案例分享（三）并发大时性能下降严重
1.问题描述：

收到客户的邮件，原来客户在进行X86和小机的性能对比测试时出现以下问题：

一个高端pc server，内存32G，cpu是32核

pc server上测试8个语句并行，小机下降不明显，但在此pc server上却下降明显。

执行时间由1分多，降到7分钟。小机上始终是2分钟。

附件是AWR报告，请分析原因。

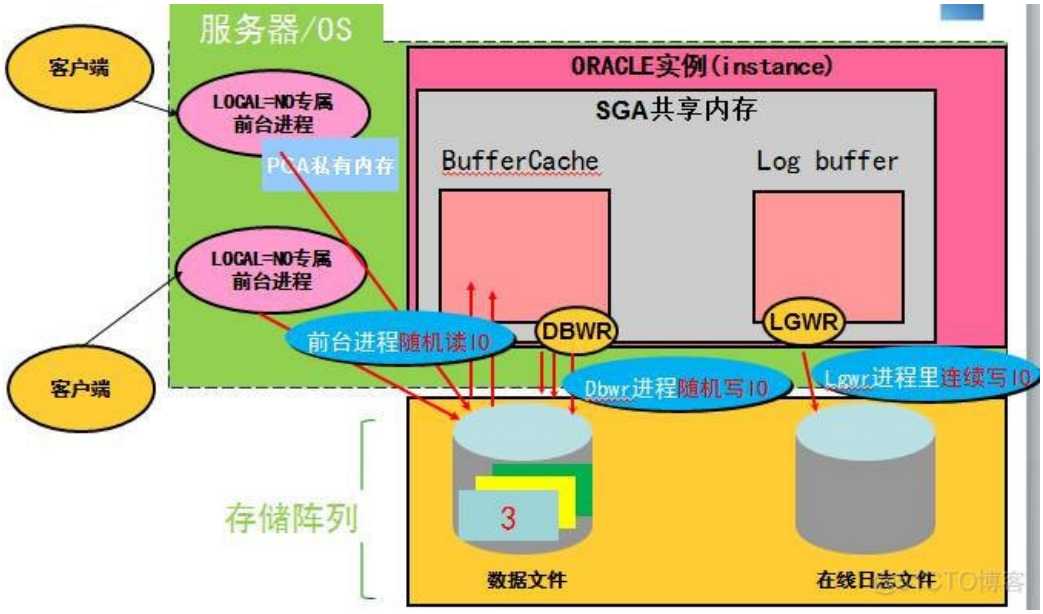
2. 分析过程：

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	6.3	233.0	3.93	16.49
DB CPU(s):	0.6	23.0	0.39	1.62
Redo size:	2,475.8	91,824.5		
Logical reads:	30,695.3	1,138,476.7		
Block changes:	5.4	201.9		
Physical reads:	30,680.6	1,137,931.2		

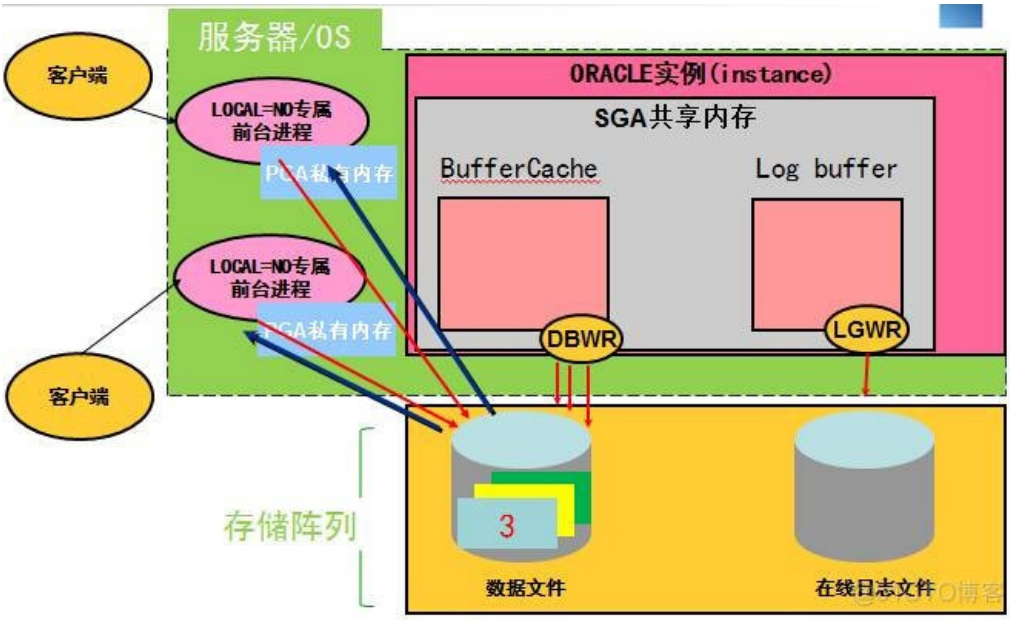
8个并行测试的情况下，逻辑读=物理读， 逻辑读表示操作内存中的BLOCK的个数，通过物理读（IO）读进内存后必然发生逻辑读。 这说明ORACLE根本没能把数据缓存到共享内存buffer cache，以便供其他进程复用。 从AWR报告中的等待事件可以看到，排在第一位的是direct path read,这是一个IO事件 即上面说的，绕开BUFFER CACHE，直接读到PGA私有内存（非共享内存）中时,单次IO的时间达到了42毫秒，太大，说明磁盘IO竞争严重，性能不佳。 但这个是原因还是结果呢？我们不妨往下看。

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
direct path read	75,500	3,155	42	90.25	User I/O
DB CPU		344		9.85	
Disk file operations I/O	45	1	21	0.03	User I/O
control file sequential read	153	0	0	0.00	System I/O
log file sync	7	0	4	0.00	Commit

如下图所示，不再是由一个进程将数据读取到buffer cache,其他人直接复用内存中的数据，从而降低了对IO的请求次数，降低了磁盘的繁忙程度。



而是每个进程各自读取到自己的私有内存PGA中，每个进程执行同一条SQL都需要各自读取各自的，显然大量进程同时反复读取同一片数据，势必造成磁盘的繁忙和IO的性能下降。这就是从AWR报告中得到的分析结论。如下图所示。



那么为什么不是一个人读取到共享内存，其他人坐享其成就好了呢？

这是11G的新特性引起的。11g下当优化器判断需要较多物理IO的时候，那么就绕开BUFFER CACHE，直接读到PGA私有内存中。

这个特性的初衷是：当ORACLE默认一次读16个BLOCK时，由于所需的部分BLOCK已经在buffer cache里了，不连续了，因此这16个BLOCK，很可能需要拆分为多次IO，导致原来的一次多块读变为了多次单块读。

但现实中，当并行个数多的时候，由于该特性，这个时候很容易把磁盘搞得很忙，IO性能下降严重，也就出现了执行时间从1分钟升至7分钟的情况。

而原来10G下的机制是：

一个会话读到内存中，其他会话坐享其成，直接读内存中的数据就可以，因此读磁盘的个数会小一些。

3.问题得到解决：

使用下列方法临时禁止该特性后，再次测试，问题得到解决。 alter system set event= ‘10949 trace name context forever, level 1’ scope=spfile; –重启数据库 Shutdown immediate startup alter system register; 因此不难看出，IO慢实际上是结果，而不是原因，原因在于对同一片数据反复读取，出现太多IO了。

怎么破——什么是无效IO以及解决方法

我们不放来回顾“用一个例子说明ORACLE的工作过程”这个章节，不难发现，其实那就是一个活生生无效IO的例子。

客户端发起update T set id=5 where id=3的SQL语句

其中，表T的大小为1G，表上不存在任何索引

在执行过程中，前台进程总计读取了磁盘中的1G的数据，经由SAN交换机传输到ORACLE共享内存中，再进行过滤，最后只有1条数据满足，最终从id=3被更新为id=5。

试想一下，如果这个表的大小不是1个G，而是100个G，也不是一个客户端发起对该表的更新，而是多个会话同时更新不同的记录，那么整个系统的IO将会异常繁忙。

我们从字典中找一个“喜”字，如果是挨页翻，挨页对，那么势必会多做很多无用功，最简单的方式就是从偏旁部首或者拼音来检索，就可以快速的找到“喜”字。
同样是找一个“喜”字，前者多翻了很多字典，即产生了很多无效IO。后者则很高效。

因此，无效IO，说到底，是SQL语句缺少一个定位数据的高效方式，导致读取了很多数据，但是不满足又被丢弃了，导致了很多无效的IO。如果，我们对表T的id字段创建索引，那么将可以快速精确定位到id=3的数据，只需要读取几个BLOCK，整体的IO量可以控制在40K以内，不是之前的1个G。

应用程序的SQL语句不够高效，是无效IO的主要原因。

SQL语句的优化不是索引那么简单，索引只是众多单表访问路径的一种，SQL优化还涉及到表连接方式优化、表连接顺序优化、SQL改写等手段，后续将会陆续介绍这些优化手段。

知识点：

我们需要有这么一个意识：

磁盘100% busy,IO响应时间很长，这很可能是因为某些不够高效的SQL语句，产生了很多无效的IO，或者导致IOPS超过了整个磁盘（阵

列）所能提供的IO能力，或者是占用了无效的IO带宽导致了IO的拥堵。

磁盘繁忙，IO响应时间长，可能已经是结果，而不是导致业务慢的真正原因。

通过优化高IO的SQL，消除无效IO，将IO控制在合理范围内，提升整体IO性能。

案例分享（四）不了解业务逻辑的情况下实现每秒IO 359M到每秒1M的优化
1. 问题描述：

系统的IO的IO量很大，经常性地，IO的吞吐量达到每秒300M以上。

通过AWR报告，找到问题集中在一条SQL上。

问题来了，我们不是做开发的，这个业务系统的业务逻辑我们也不清楚，我们可以优化么？

答案是可以的，

实际上我们完全在可以不懂业务逻辑的情况下完成绝大部分情况的优化，我们只要获得SQL执行的过程和明细即可快速完成优化，以下这条最占IO的SQL的优化我们在1分钟内就完成了优化。

为了说明我们不需要了解业务逻辑也可以完成优化，你会发现从头到尾没有看到过任何的SQL语句 ^_^

2. 优化过程：

获取执行计划和执行明细(哪些步骤消耗多少时间，花费多少IO)

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1		19	00:00:39.04	1832K	1750K
1	SORT ORDER BY		1	420	19	00:00:39.04	1832K	1750K
* 2	FILTER		1		19	00:00:00.25	1832K	1750K
3	NESTED LOOPS OUTER		1	420	19	00:00:00.25	1832K	1750K
4	VIEW		1	1340	19	00:00:00.25	1832K	1750K
5	UNION-ALL		1		19	00:00:00.25	1832K	1750K
6	NESTED LOOPS		1	13	13	00:00:00.23	80961	0
7	NESTED LOOPS		1	13	13	00:00:00.23	80948	0
* 8	TABLE ACCESS FULL		1	13	13	00:00:00.23	80907	0
* 9	INDEX UNIQUE SCAN		13	1	13	00:00:00.01	41	0
10	TABLE ACCESS BY INDEX ROWID		13	1	13	00:00:00.01	13	0
11	NESTED LOOPS		1	1327	6	00:00:36.78	1751K	1750K
12	NESTED LOOPS		1	1327	6	00:00:36.78	1751K	1750K
13	PARTITION LIST ALL		1	1327	6	00:00:36.78	1751K	1750K
* 14	TABLE ACCESS FULL	A表	10	1327	6	00:00:38.50	1751K	1750K
* 15	INDEX UNIQUE SCAN		6	1	6	00:00:00.01	20	0
16	TABLE ACCESS BY INDEX ROWID		6	1	6	00:00:00.01	6	0
17	TABLE ACCESS BY INDEX ROWID		19	1	18	00:00:00.01	22	0
* 18	INDEX UNIQUE SCAN		19	1	18	00:00:00.01	4	0

可以看到：

上述SQL的执行时间是39秒，其中id=14的步骤，占了38秒，必须优化掉该瓶颈步骤。

Id=14的步骤，reads为1750K个BLOCK，即读了1750K*8K=13G,单次执行13G,38秒读完，即每秒的IO达到359M，但是最后只返回了6条记录，该步骤对A表进行全表扫描，显然，读取13G，应用了过滤条件后，最后只返回了6条记录！很多数据在读取到内存后基本都被丢弃了。缺少定位数据的高效方式，而索引是最适合定位少量数据的。

3. 优化方式

上图id=14的谓词部分，即红色加框部分，可以看到对A表扫描了13G，主要的过滤条件是c_captialmode和c_state这两个字段把大部分数据全滤掉了，因此创建复合索引即可，命令如下：

Create index idx_1 on A(c_captialmode,c_state) tablespace &tbs online;

4. 优化效果

优化后，每次执行，IO从13G下降至0 ；

优化后，执行时间从50秒下降至50毫秒

优化后，整个系统的IO从每秒359M下降到1M以下。

知识点：

在不了解业务逻辑的情况下，也可以快速实现对最消耗IO的SQL语句的快速优化。

精通数据库知识的DBA往往比不懂数据库原理的开发和程序员更懂SQL优化。

————————————

由于篇幅的原因，本次分享到此。

在遇到ORACLE数据库IO量高的情况时，至少有一个意识，我们的IO是无效的么？
————————————————-
Oracle 数据库服务器 IO 高的分析方案（理论讲解+案例分享）
https://blog.51cto.com/lhrbest/3298362

Oracle 数据库服务器 IO 高

发表回复

您的电子邮箱地址不会被公开。 必填项已用*标注

昵称 *
邮箱 *
网站

☐ 在此浏览器中保存我的显示名称、邮箱地址和网站地址，以便下次评论时使用。

发表评论

更多阅读

- 11g 设置memory_target参数
- Oracle 数据库服务器 IO 高的分析方案（理论讲解+案例分享）
- oracle truncate table recover(oracle 如何拯救误操作truncate的表)
- Fy_Recover_Data ———— 用于数据恢复的PLSQL包

- oracle ora错误代码大全
- ORACLE里锁有以下几种模式,v$locked_object,locked_mode
- docker安装部署
- ASM的文件管理深入解析（内含开源的ASM文件挖掘研究版程序）
- docker 基础操作(实战一) 怎样删除容器、镜像
- RAID技术详解:1、RAID 类型介绍
- mysql order by 中文排序
- 物化视图————为提升性能
- SSD固态硬盘技术扫盲
- Oracle登录PDB的几种操作

标签云

ASM docker docker安装部署 docker容器删除 fail_timeout https IOPS IOPS计算 mysql nginx ORA-00600 oracle temp oracle中的sys登录用户的验证方式 oracle压力测试 Oracle备份 Oracle 权限设置 oracle的ASM ora错误 ora错误合集 order by pivot函数 profiler脚本 RAID Raid-7 RAID 类型介绍 redo log session_cached_cursors sga SSL证书 sysbench工具 tomcat tomcat log unpivot函数 _optimizer_cost_based_transformation 中文排序 临时表空间 临时表空间不释放的原因 可以减少sql解析 工具软件 归档日志 文件同步 机械硬盘 磁盘寻址 计算磁盘性能 高原反应

×

赞赏支持

扫描二维码，输入您要赞赏的金额

赞赏不用多，心意到了即可

搜索…

初见博客