ORACLE 数据库优化技术

徐宏文　张勇（广东大亚湾核电站电脑中心　518124）

摘要：从数据库管理员的角度，结合几年来在ORACLE数据库系统上所做的实际工作，探讨DBA如何优化系统性能，其中包括优化数据库设计、优化应用、优化内存、优化I/O，优化资源冲突以及性能优化后对应用系统的改善和提高所起到的作用。

关键词：Oracle　数据库　优化技术

1　优化应用

对应用程序的优化主要体现在对程序中用到的SQL语句的优化和对数据库对象的优化。

1.1　优化SQL语句

在编写程序中，为达到同样目的的数据库操作，可以编写出各不相同的SQL语句，而其执行的效率也是千差万别的。据统计相当多数的性能问题是由于应用程序中使用了不合适的SQL语句而造成系统的响应速度减慢。因此提高SQL语句编写的质量和标准化对应用的优化能起很大的作用。

创建索引是提高检索效率最有效的方法之一，正确地建立索引会极大的提高查询速度。

下面通过一个诊断和优化的实例来说明利用可用的索引优化SQL语句对整个系统性能提高所带来的益处和产生的效益。我公司计算机重点应用项目MAXIMO系统目前正处在试运行阶段，用户在使用中发现数据查询和修改的速度很正常，但是在增加一条新记录时数据库响应很慢。当时我们根据用户提供的情况做了相同的操作，结果花了近40秒左右，显然这一现象是不正常的，于是检查了相关的SQL语句，以及所用到的表和索引的存储参数、表记录的数量、记录是否存在行迁移等因素，也没有发现明显的问题。然后请应用开发人员一道，对其程序中编写的SQL语句逐条跟踪，结果非常艰难地发现问题出在一个不起眼的地方。按照程序设计思想，用户追加插入一条新记录，程序在确认前先会做一个查询，该查询会使用索引，但是真正建的索引中却没有该索引字段，问题找到解决起来就很快了，根据此查询要求重建索引，保证查询语句使用可用的索引。当重新做以上的追加记录操作时，结果立即就出来了。问题的顺利解决使用户打消了疑虑，同时也为MAXIMO系统的向前推进起到了一定作用。

另外SQL语句书写应标准化（Identical SQL），这样有助于相同的SQL语句再次被执行时，能用到以前的分析结果，避免了重复分析过程，节省了系统开销，并提高响应效率。

1.2　优化数据库对象

数据库对象包括表、聚集、索引、表空间等。他们建立时存储参数的确定对于以后数据实际所占空间是紧密相关的，因此在设计时就要预测到将来的情况，估算出初始空间，扩展空间以及最终的数据所需的空间。确认是使用系统缺省参数，还是特殊情况重新设置。否则，不适当的存储参数将导致空间动态分配并无限制扩张，表的分区增多，链式记录行的出现，所有这些都会对性能带来严重的影响。

通过对表中数据记录增长的定期检查和预测，合理地调整初始分区INITNAL和扩展分区NEXT的值，限制表空间无限制动态扩展，降低因I／O增加而造成的性能下降。

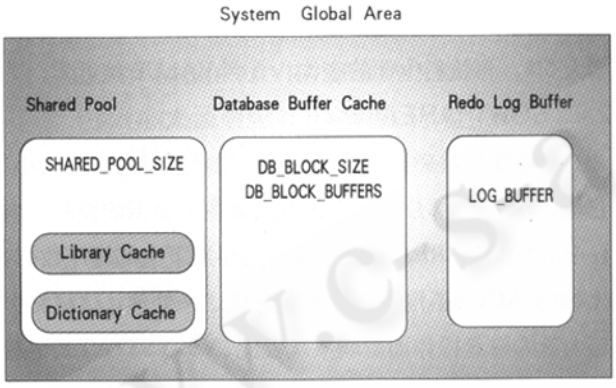
根据表中数据记录的实际情况，合理对PCTFREE和PCTUSED存储参数进行调整。一般来说，对一个很少修改的静态表，PCTFREE设为0，而PCTUSED设为99，能节约存储空间。当对于一个需常常修改记录的表来说，PCTFREE设为15，PCTUSED设为60更加合适些。但这也不能一概而论，理想的情况是在程序运行稳定一段时间后，通过ANALYZE TABLE命令来收集表的统计量，得到记录的平均长度，然后根据定义的块长就能估算出PCTUSED的值。

同样用ANALYZE TABLE命令收集到的统计值，还可以查出有多少链式记录行，当链式记录行很多时，采用

的唯一修复方案就是备份并导出表，删除再导入表。

2 优化内存

ORACLE的信息存储在内存和磁盘上，由于访问内存比访问磁盘快得多，所以ORACLE希望把尽可能多的数据存放在内存中。但机器的内存是有限的，所以系统给ORACLE分配的内存空间也是有限的。因此，如何配置ORACLE系统参数，对系统性能的优劣起很大的作用。ORACLE的内存空间结构如下图:



在内存调整之前，我们先要检测到系统性能缺陷的原因，有多种方法可以获得系统性能指标：OEM、UTLBSTAT/UTLESTAT、直接查系统动态视图等，前两种方法获取系统统计数据比较直观，下面以直接查系统动态视图的方法，讨论如何调整其中最重要的几个参数。

2.1 DB_BLOCK_BUFFERS（高速缓冲区块数）

此参数是定义内存中高速缓存的数据库块缓冲区的数目。用户使用的所有数据都要通过DB_BLOCK_BUFFERS数据高速缓存，高速缓存越大，ORACLE可装入内存的数据就越多，磁盘的I/O就越少，系统的性能越好。

通过计算高速缓冲区命中率，我们可以知道DB_BLOCK_BUFFERS是否足够。

```
SELECT NAME,VALUE
FROM V$SYSSTAT
  WHERE NAME IN ('db block gets','consistent gets','physical reads')
```

查询结果:

NAME	VALUE
db block gets	24372082
consistent gets	2.493E+09
physical reads	50606953

Hit Ratio = 97.9%

计算高速缓冲区命中率的公式:

Hit Ratio = 1 - (physical reads / (db block gets + consistent gets))

对于通常的环境，要求此值大于80%；对于UNIX上应用RAW DEVICE的系统，要求此值大于90%。当命中率低于标准值时，需要增加DB_BLOCK_BUFFERS，使保证命中率为80%以上，增加的数目可由下面步骤确定:

· 设置db_block_lru_extended_statistics=1000;

· 重启动数据库;

· 经过一段时间的运行后查询:

```
SELECT 250*TRUNC(ROWNUM/250)+1||' to '||250*(TRUNC(ROWNUM/250)+1)
  "Interval", SUM(count) "Buffer Cache Hits"
  FROM V$RECENT_BUCKET
  GROUP BY TRUNC(ROWNUM/250);
```

查询结果:

Interval	Buffer Cache Hits
1 to 250	16080
251 to 500	10950
501 to 750	710
751 to 1000	140

根据查询的结果，增加前500个DB_BLOCK_BUFFERS后，高速缓冲区命中率有较大提高，再增加时对命中率提高较小，我们可以把DB_BLOCK_BUFFERS参数增加500。

DB_BLOCK_BUFFERS增加到一定的数值后，再加大此值，对系统性能的提高微乎其微，但却浪费了有限的内存资源。在系统内存资源有限，同时高速缓冲区命中率很高的情况下，我们在保证命中率为80%以上的情况下，可以考虑减少一定数量的DB_BLOCK_BUFFERS，减少缓冲区的数目可由下面步骤确定:

· 设置db_block_lru_statistics=true;

· 重启动数据库;

· 经过一段时间的运行后查询:

```
SELECT 250*TRUNC(ROWNUM/250)+1||' to '||250*(TRUNC(ROWNUM/250)+1)
  "Interval", SUM(count) "Buffer Cache Hits"
```

```
FROM V$CURRENT_BUCKET
WHERE ROWNUM > 0
GROUP BY TRUNC(ROWNUM/250);
```

查询结果:

Interval	Buffer Cache Hits
1 to 250	4900
251 to 500	2120
501 to 750	2865
751 to 1000	171

根据查询的结果，减少250个DB_BLOCK_BUFFERS后对缓冲区命中率的影响较小，再减少250后，大大减低缓冲区命中率了，据此我们可以把DB_BLOCK_BUFFERS参数减小250。

2.2 SHARED_POOL_SIZE（共享缓冲区大小）

此参数是定义内存中LIBRARY CACHE（存放共享SQL和PL/SQL）和DATA DICTIONARY CACHE（存放数据字典对象信息）的字节数量。LIBRARY CACHE和DATA DICTIONARY CACHE的大小只能通过SHARED_POOL_SIZE间接调整。

检测LIBRARY CACHE:

```
SELECT SUM（RELOADS）/SUM（PINS）*100
  FROM V$LIBRARYCACHE;
```

查询结果:

```
SUM(RELOADS)/SUM(PINS)*100
--------------------------------------------
                          .59195703
```

此值应小于1，否则要加大SHARED_POOL_SIZE。

检测DATA DICTIONARY CACHE:

```
SELECT SUM(GETS) "Gets",SUM(GETMISSES)
"Getmisses",
SUM(GETMISSES)/SUM(GETS)*100 "Rate"
FROM V$ROWCACHE;
```

查询结果:

Gets	Getmisses	Rate
13502532	765065	5.6660854

此Rate应小于15，否则要加大SHARED_POOL_SIZE。

2.3 LOG_BUFFER （重做日志缓冲区大小）

此参数是用来定义内存中重做日志缓冲区的大小，虽然它相对SGA较小，但是当此值设置太小时，LGWR进程会频繁将LOG BUFFER中的数据写入磁盘，增加I/O的次数，影响系统性能。

查V$SYSSTAT表:

```
SELECT NAME,VALUE FROM V$SYSSTAT
  WHERE NAME='redo log space requests';
```

查询结果:

NAME	VALUE
redo log space requests	261

VALUE值应接近于零，否则每次将LOG_BUFFERS增大5%，再执行上面的查询，直到VALUE接近零;

2.4 SORT_AREA_SIZE（排序区大小）

此参数是用来指定排序的每个用户进程内存的大小，它是PGA的一部分。对于含有大量索引数据的排序，内存中的SORT AREA可能不能完全存放下全部排序数据，此时ORACLE不得不在磁盘上完成这一处理，这样将引起大量的磁盘操作而影响系统性能。V$SYSSTAT表记载有系统的排序信息，通过查看此动态视图可确定是否有大量的排序是在磁盘上完成的，从而决定修改此参数: 此参数可以动态调整，ALTER SESSION改变当前会话，ALTER SYSTEM DEFERRED为数据库的所有后继连接更改此参数。

```
SELECT NAME,COUNT(*) FROM V$SYSSTAT
  WHERE NAME LIKE 'sort%'
  GROUP BY NAME;
```

查询结果:

NAME	COUNT(*)
sorts (disk)	24549
sorts (memory)	611780
sorts (rows)	294742953

如果有相当多的排序在磁盘上进行，应增大SORT_AREA_SIZE的数值。

3 优化数据库磁盘I/O

数据库中大部分操作因为存储或检索数据而需要读/写磁盘，故磁盘I/O的次数多少直接影响到数据库的性能，I/O次数多了就会使系统性能降低，而实际I/O的次数又与数据文件的分配方式密切相关。以下是建立和优化数据文件的基本准则，适用于在数据库设计和性能优化方面的

考虑。

·为表和索引分别建立不同的数据库表空间，分别单独存放。

·将表空间和索引空间尽量存放在不同的磁盘上。

·将数据库的redo日志文件和数据库的回滚段表空间尽可能在不同磁盘上存放。

·将频繁访问的表，索引所在表空间，放在单独的磁盘上。

·将ORACLE的可执行文件和数据库文件分别放在单独的磁盘上。

系统管理员可以通过查询V $ FILESTAT系统动态视图来监测数据文件的I/O情况。

```
SELECT NAME,PHYRDS,PHYWRTS
    FROM v$DATAFILE,v$FILESTAT
   WHERE v$DATAFILE.FILE# = v$FILES-
TAT.FILE#;
```

查询结果：

NAME	PHYRDS	PHYWRTS
/usr/data/data1/oradata/ora8/system01.dbf	63746	24125
/usr/data/data3/app11data.dbf	4732613	43937
/usr/data/data3/app12data.dbf	4226310	27211
/usr/data/data3/app13data.dbf	449712	17564
/usr/data/data3/app14data.dbf	568420	96563
/usr/data/data3/app11indx.dbf	57118	10752
/usr/data/data3/app12indx.dbf	53128	9107

从上面的查询结果可以看出，我们在设计数据库表空间时，已将数据和索引分开在不同的表空间上，分别叫app_data和app_index。同时又将一个表空间分成多个数据文件，如将app_data表空间分为app11data.dbf-app14data.dbf四个文件分别放在不同磁盘上保存，物理读（PHYRDS）和物理写（PHYWRTS）之和就是该数据文件目前总的I／O数，可以看出这样做得目的就在于尽可能的产生并发I／O操作，来降低整个表空间的存取次数，提高系统效率。

4 优化回滚段

一个事务处理在其所有的修改结果存入磁盘前，回滚段中保存了恢复该事务所需的全部信息。因此在下列情况下需要用到回滚段。

·执行的事务产生一个ROLLBACK操作

·读一致性数据库信息

·数据库恢复时

良好的回滚段配置对一个优化的ORACLE数据库来说是很重要的。一个回滚段由多个回滚区组成，它保存着需要回滚的事务处理的信息，回滚段的大小取决于数据库正常运行时的活动状态，一般来说，回滚段的设置基于下列原则：

·一个回滚段的每个回滚区分配相同的值。

·回滚段的初始分区值（INITIAL）可设为2K，4K，8K等等，便于空间回收再用。

·每个下一分区值（NEXT）应与初始分区值设为一致。

·最小分区数（MINEXTENTS）设为20，以保证扩展时不影响使用其他分区的事务。

确定回滚段的数目。根据经验值，对于OLTP环境，每十个用户需建一个回滚段，而对于批作业，每一个并发作业要使用一个回滚段。ORACLE是根据自己的规则来决定使用哪个回滚段，即总是先分配具有最少活动事务数的回滚段供一个新事务使用。用户也可以通过命令来强制一个事务使用某一指定的回滚段。这对于一个确定的执行长查询的大事务是有用的，它可将大事务用到的回滚段的OPTIMAL值设成足够大来避免ORACLE出现"Snapshot too old"（快照太旧）这样一个常见的错误，保证事务得以完成。

另外在建立回滚段时要注意的一点是，要将回滚段建成为PUBLIC的，或者是在数据库初始文件上写上所有回滚段的列表，否则，回滚段虽然建了，但是用不到。

5 优化重做日志文件

重做日志文件（redo log），也是数据库的重要组成部分。它们维护数据更新记录，当系统出错时可用来恢复数据库。特别是当数据库系统运行在归档方式下时，可通过日志文件来保证数据库恢复到最近的状态。

为了使重做日志文件能正常运作，互相切换，至少要建二个相同大小的重做日志文件组。基于安全考虑，一个重做日志文件组内再建二个成员，把他们放在不同的磁盘上保存。

当一个重做日志文件组写满时，或强制执行ALTER SYSTEM LOG SWITCH命令时，重做日志文件组进行切换，如果重做日志文件组的切换过于频繁，会造成系统的性能下降。我们可以通过查看ALERT日志文件中的信息，来确定重做日志文件的切换频率。

一般来说，在大多数的情况下文件的切换在30分钟

左右产生一次是较好的，如果切换频率过快，几分钟甚至更短时间就切换一次，显然会造成I/O次数大大增加，从而影响整个系统性能。出现这种现象时，就需要重建有更大容量日志文件的重做日志文件组。

同样我们在对MAXIMO系统做优化时发现，它的重做文件平均每5分钟左右就要切换一次，而它的重做日志文件的大小感觉已经很大了，不应该是重做日志文件尺寸不够大的原因，于是我们考虑到在初始参数文件INIT中还有一个LOG_CHECKPOINT_INTERVAL的参数可用来控制重做日志组的切换，检查MAXIMO服务器数据库参数，果然该系统中此参数定的太小，使得重做文件未用完而是按照该参数指定的值就进行了切换，为了配合工作，我们将这个参数设定为大于重做文件尺寸的大小。再检查ALERT文件，发现情况大大改观，同时性能也有好转。

6 优化锁冲突

由于ORACLE对一事务处理发生死锁没有自动检测定时退出的功能，故当一事务执行时产生死锁时，需要数据库管理员能及时检测出死锁的事务，执行的SQL语句以及获得执行该死锁事务的用户名并将死锁用户强制退出。同时协助开发人员找出死锁的原因和产生死锁的地方。

公司财务系统FIS在刚投产运行期间，发现输入账号和密码后，自己的客户端一直没响应，不能进入到FIS系统主屏幕菜单。我们将应用系统执行程序移植到本地运行，经测试后排除了应用程序本身以及应用程序服务器的问题，然后在客户端用SQL * PLUS工具编写SQL语句模拟FIS系统登录过程，发现很快就可以连接到数据库中，检测后确认数据库系统运行也是正常的。接着我们对应用程序本身找原因，在跟踪检查中发现用户发出登录请求后，已经系统响应并实际连接到数据库，但在查询自己的授权表时被其他请求锁住，用户需要的资源一直得不到释放，所以反映在客户端上，主屏幕调不出来，系统没响应。我们在找到原因后，请程序开发人员检查并修改系统登录过程的源程序，从而彻底解决了用户整天抱怨财务系统无法正常进入使用的问题。

下面就是我们通过SQL语句的方式来检测死锁原因，并解除死锁现象的。

```
SELECT oracle_username,os_user_name,session_id,locked_mode
  FROM v$locked_object;
```

如果查询有结果显示，表示有数据锁存在。例如，某刻我们查询到下列结果

```
ORACLE_USERNAME  OS_USER_NAME  SESSION_ID  LOCKED_MODE
-----------------------------------------------------
APP1             P200001       15          3
```

表明一个对话ID为15，叫APP1的用户执行的DML产生了一个行记录级的排斥锁。

再进一步，我们可以查询到该锁所对应的表。

```
SELECT owner,object,type
  FROM v$access
    WHERE sid = '?';    (?是从上面查询中得到的SESSION_ID)
```

查询结果:

```
OWNER    OBJECT    TYPE
-----------------------------------
SCOTT    TEST      TABLE
```

表明数据锁产生在SCOTT.TEST的表上。

通过更进一步的查询可知该锁的SQL语句:

```
SELECT user_name, sid, sql_text
  FROM v$open_cursor
  WHERE sid = '?';
```

锁冲突解决方法:

大多数锁冲突产生来自于应用程序，要解除死锁引起的竞争，就需要释放被占用的资源，可以考虑由以下几种方法来解决

· 让锁的占有者自己作确认（COMMIT）或回滚（ROLLBACK）操作。

· 通过命令强行删除引起锁的对话（SESSION）。

```
ALTER SESSION KILL SESSION 'SID,SERIAL#';
```

· 强行删除锁的占有者的UNIX进程，但这不是一个好方法，特别是在多线程的环境下

以上我们仅从几个方面探讨了如何对ORACLE数据库进行性能优化，并根据实践总结出的一些优化的技术和经验，同时也解决了一部分实际遇到的问题。但我们认为数据库的性能优化是一个综合、复杂而又长期的工作，除需要不断加深对数据库系统的知识理解之外，还要熟悉Clint/Server开发环境以及具备相当的网络知识，这样才能把这项工作做深做好。■

参考文献

1 ORACLE8 TUNING

2 ORACLE 技术通讯