等待事件系列（1）--User I/O 类型

1.1 BLOG 文档结构图

1.2 前言部分

1.2.1 导读和注意事项

各位技术爱好者，看完本文后，你可以掌握如下的技能，也可以学到一些其它你所不知道的知识，~O(∩_∩)O~：

① 等待事件系列（1）--User I/O 类型

Tips：

① 本文在 ITpub（http://blog.itpub.net/26736162）、博客园（http://www.cnblogs.com/lhrbest）和微信公众号（xiaomaimiaolhr）有同步更新

② 文章中用到的所有代码，相关软件，相关资料请前往小麦苗的云盘下载（http://blog.itpub.net/26736162/viewspace-1624453/）

③ 若文章代码格式有错乱，推荐使用搜狗、360 或 QQ 浏览器，也可以下载 pdf 格式的文档来查看，pdf 文档下载地址：http://blog.itpub.net/26736162/viewspace-1624453/，另外 itpub 格式显示有问题，可以去博客园地址阅读

④ 本篇 BLOG 中命令的输出部分需要特别关注的地方我都用灰色背景和粉红色字体来表示，比如下边的例子中，thread 1 的最大归档日志号为 33，thread 2 的最大归档日志号为 43 是需要特别关注的地方；而命令一般使用黄色背景和红色字体标注；对代码或代码输出部分的注释一般采用蓝色字体表示。

```
List of Archived Logs in backup set 11
Thrd Seq     Low SCN    Low Time            Next SCN   Next Time
---- ------- ---------- ------------------- ---------- ---------
1    32      1621589    2015-05-29 11:09:52 1625242    2015-05-29 11:15:48
1    33      1625242    2015-05-29 11:15:48 1625293    2015-05-29 11:15:58
2    42      1613951    2015-05-29 10:41:18 1625245    2015-05-29 11:15:49
2    43      1625245    2015-05-29 11:15:49 1625253    2015-05-29 11:15:53
```

```
[ZHLHRDB1:root]:/>lsvg -o
T_XDESK_APP1_vg
rootvg
[ZHLHRDB1:root]:/>
00:27:22 SQL> alter tablespace idxtbs read write;


====》2097152*512/1024/1024/1024=1G
```

本文如有错误或不完善的地方请大家多多指正，ITPUB 留言或 QQ 皆可，您的批评指正是我写作的最大动力。

1.3 等待事件的源起

谈到等待事件，必然会提到一种流行的诊断方法论 OWI，即 Oracle Wait Interface.

等待事件的概念大概是从 Oracle 7.0.12 中引入的，刚引入的时候大约有 100 多个等待事件，在 Oracle 8.0 中这个数目增大到了大约 150 个，在 Oracle 8i 中大约有 220 个事件，在 Oracle 9i 中大约有 400 多个等待事件，在 Oracle 10gR2 中，大约有 800 多个等待事件，在 11gR2 中约有 1000 多个等待事件。随着等待事件的逐步完善，也能够反映出对于问题的诊断粒度越来越细化。

虽然不同版本和组件安装可能会有不同数目的等待事件，但是这些等待事件都可以通过查 V$EVENT_NAME 视图获得：

10.2.0.5 版本：

```
select count(1) from v$event_name;
```

COUNT(1)
916

11g：

```
select count(1) from v$event_name;
```

COUNT(1)
1152

1.4 分类

ORACLE 的等待事件，主要可以分为两类，即空闲(IDLE)等待事件和非空闲(NON-IDLE)等待事件。

1). 空闲等待事件指 ORACLE 正等待某种工作，在诊断和优化数据库的时候，不用过多注意这部分事件。

2). 非空闲等待事件专门针对 ORACLE 的活动，指数据库任务或应用运行过程中发生的等待，这些等待事件是在调整数据库的时候需要关注与研究的。

下面来看一下 ORACLE 中等待事件的主要分类及各类等待事件的个数：

```
SELECT a.INST_ID, A.EVENT, COUNT(1)
  FROM gv$session a
 where a.username is not null
   and a.STATUS = 'ACTIVE'
  AND A.WAIT_CLASS<>'Idle'
GROUP BY a.INST_ID,A.EVENT;
```

```
SELECT wait_class#,
       wait_class_id,
       wait_class,
```

```
       COUNT(*) AS "count"
FROM   v$event_name
GROUP  BY wait_class#,
          wait_class_id,
          wait_class
ORDER  BY wait_class#;
```

WAIT_CLASS#	WAIT_CLASS_ID	WAIT_CLASS	count
0	1893977003	Other	622
1	4217450380	Application	14
2	3290255840	Configuration	23
3	4166625743	Administrative	46
4	3875070507	Concurrency	25
5	3386400367	Commit	1
6	2723168908	Idle	63
7	2000153315	Network	28
8	1740759767	User I/O	20
9	4108307767	System I/O	24
10	2396326234	Scheduler	2
11	3871361733	Cluster	48

11g：

	WAIT_CLASS#	WAIT_CLASS_ID	WAIT_CLASS	count
1	0	1893977003	Other	745
2	1	4217450380	Application	17
3	2	3290255840	Configuration	24
4	3	4166625743	Administrative	55
5	4	3875070507	Concurrency	33
6	5	3386400367	Commit	2
7	6	2723168908	Idle	95
8	7	2000153315	Network	35
9	8	1740759767	User I/O	48
10	9	4108307767	System I/O	31
11	10	2396326234	Scheduler	8
12	11	3871361733	Cluster	50
13	12	644977587	Queueing	9

常见的空闲事件有:

- dispatcher timer
- lock element cleanup
- Null event
- parallel query dequeue wait
- parallel query idle wait - Slaves
- pipe get
- PL/SQL lock timer
- pmon timer- pmon
- rdbms ipc message
- slave wait
- smon timer
- SQL*Net break/reset to client
- SQL*Net message from client
- SQL*Net message to client
- SQL*Net more data to client
- virtual circuit status
- client message

一些常见的非空闲等待事件有:

- db file scattered read
- db file sequential read
- buffer busy waits
- free buffer waits
- enqueue
- latch free
- log file parallel write
- log file sync

This appendix contains the following topics:

■ Classes of Wait Events

■ Descriptions of Common Wait Event Parameters

■ Descriptions of Wait Events

Information about wait events is displayed in three dynamic performance views:

■ V$SESSION_WAIT displays the events for which sessions have just completed waiting or are currently waiting.

■ V$SYSTEM_EVENT displays the total number of times all the sessions have waited for the events in that view.

■ V$SESSION_EVENT is similar to V$SYSTEM_EVENT, but displays all waits for each session.

Many of these wait events are tied to the internal implementation of Oracle and therefore are subject to change or deletion without notice. Application developers should be aware of this and write their codeto tolerate missing or extra wait events.

The following SQL statement displays an alphabetical list of all Oracle wait events and the wait class to which they belong:

```
SQL> SELECT name, wait_class FROM V$EVENT_NAME ORDER BY name;
```

1.4.1 Classes of Wait Events

Every wait event belongs to a class of wait event. The following list describes each of the wait classes.

Administrative

Waits resulting from DBA commands that cause users to wait (for example, an index rebuild)

Application

Waits resulting from user application code (for example, lock waits caused by row level locking or explicit lock commands)

Cluster

Waits related to Real Application Clusters resources (for example, global cache resources such as 'gc cr block busy')

Commit

This wait class only comprises one wait event - wait for redo log write confirmation after a commit (that is, 'log file sync')

Concurrency

Waits for internal database resources (for example, latches)

Configuration

Waits caused by inadequate configuration of database or instance resources (for example, undersized log file sizes, shared pool size)

Idle

Waits that signify the session is inactive, waiting for work (for example, 'SQL*Net message from client')

Network

Waits related to network messaging (for example, 'SQL*Net moredata to dblink')

Other

Waits which should not typically occur on a system (for example, 'wait for EMON to spawn')

Queue

Contains events that signify delays in obtaining additional data in a pipelined environment. The time spent on these wait events indicates inefficiency or other problems in the pipeline. It affects features such as Oracle Streams, parallel queries, or DBMS_PIPEPL/SQL packages.

Scheduler

Resource Manager related waits (for example, 'resmgr: become active')

System I/O

Waits for background process I/O (for example, DBWR wait for 'db file parallel write')

User I/O

Waits for user I/O (for example 'db file sequential read')

1.4.2 Descriptions of Common Wait Event Parameters

Oracle® Database Reference 11g Release 2 (11.2) E40402-09
Descriptions of Common Wait Event Parameters

This section provides descriptions of some of the more common wait event parameters.

block#

This is the block number of the block for which Oracle needs to wait. The block number is relative to the start of the file. Tofind the object to which this block belongs, issue the following SQL statement:

```
select segment_name, segment_type, owner, tablespace_name
from dba_extents
where file_id = file#
and block#
between block_id and block_id + blocks - 1;
```

blocks

The number of blocks that is being either read from or written to the file. The block size is dependent on the file type:

- Database files have a block size of DB_BLOCK_SIZE
- Logfiles and control files have a block size that is equivalent to the physical block size of the platform

break?

If the value for this parameter equals 0, a reset was sent to the client. A nonzero value indicates that a break was sent to the client.

class

The class of the block describes how the contents of the block are used. For example, class 1 represents data block, and class 4 represents segment header.

dba

The initials "dba" represents the data block address, which consists of a file number and a block number.

driver id

The address of the disconnect function of the driver that is currently being used.

file#

The following query returns the name of the database file:

```
select *
from v$datafile
where file# = file#;
```

id1

The first identifier (id1) of the enqueue or global lock takes its value from P2 or P2RAW. The meaning of the identifier depends on the name (P1).

id2

The second identifier (id2) of the enqueue or global lock takes its value from P3 or P3RAW. The meaning of the identifier depends on the name (P1).

le

The relative index number into V$GC_ELEMENT.

mode

The mode is usually stored in the low order bytes of P1 or P1RAW and indicates the mode of the enqueue or global lock request.This

parameter has one of the following values:

Table C-1 Lock Mode Values

Mode Value	Description
1	Null mode
2	Sub-Share
3	Sub-Exclusive
4	Share
5	Share/Sub-Exclusive
6	Exclusive

Use the following SQL statement to retrieve the name of the lock and the mode of the lock request:

```
select chr(bitand(p1,-16777216)/16777215)||
chr(bitand(p1, 16711680)/65535) "Lock",
bitand(p1, 65535) "Mode"
from v$session_wait
where event = 'DFS enqueue lock acquisition';
```

name and type

The name or "type" of the enqueue or globallock can be determined by looking at the two high order bytes of P1 or P1RAW. The name is always two characters. Use the following SQL statement to retrieve the lock name.

```
select chr(bitand(p1,-16777216)/16777215)||
chr(bitand(p1,16711680)/65535) "Lock"
from v$session_wait
where event = 'DFS enqueue lock acquisition';
```

namespace

The name of the object namespace as it is displayed in V$DB_OBJECT_CACHE view.

requests

The number of I/Os that are "requested." This differs from the number of blocks in that one request could potentially contain multiple blocks.

session#

The number of the inactive session. Use the following SQL statement to find more information about the session:

```
select *
from v$session
where sid = session#;
```

waited

This is the total amount of time the sessionhas waited for this session to terminate.

1.5 重要等待事件

一些常见的重要的等待事件：

1)数据文件 I/O 相关的等待事件：

- db file sequential read
- db file scattered read
- db file parallel read
- direct path read
- direct path write

2)控制文件 I/O 相关的等待事件：

- control file parallel write
- control file sequential read
- control file single write

3)重做日志文件 I/O 相关的等待事件：

- log file parallel write
- log file sync
- log file sequential read
- log file single write

- switch logfile command
- log file switch completion
- log file switch (clearing log file)
- log file switch (checkpoint incomplete)
- log switch/archive
- log file switch (archiving needed)

4)高速缓存区 I/O 相关的等待事件：

- db file parallel write
- db file single write
- write complete waits
- free buffer waits

1.6 User I/O 类型

```
SELECT * FROM V$EVENT_NAME A WHERE A.WAIT_CLASS = 'User I/O';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAM
1	6	1179235204	Parameter File I/O	blkno	#blks
2	11	166678035	Disk file operations I/O	FileOperation	fileno
3	12	2215689832	Disk file I/O Calibration	count	
4	13	13102552	Disk file Mirror Read	fileno	blkno
5	14	2577606720	Disk file Mirror/Media Repair Write	fileno	blkno
6	15	2093619153	direct path sync	File number	Flags
7	17	4155572307	Datapump dump file I/O	count	intr
8	18	1791724291	dbms_file_transfer I/O	count	intr
9	19	432473858	DG Broker configuration file I/O	count	intr
10	20	2326919048	Data file init write	count	intr
11	21	3992632846	Log file init write	count	intr
12	86	1959037329	Shared IO Pool IO Completion		

11g User I/O 大约有 84 个。

1.6.1 db file sequential read（数据文件顺序读）

db file sequential read 这个等待事件在实际生产库非常常见，是个与 User I/O 相关的等待事件，通常显示与单个数据块相关的读取操作，在大多数情况下，读取一个索引块或者通过索引读取一个数据块时，都会记录这个等待。当 Oracle 需要每次 I/O 只读取单个数据块这样的操作时，会产生这个等待事件。最常见的情况有索引的访问（除 IFFS 外的方式），回滚操作，以 ROWID 的方式访问表中的数据，重建控制文件，对文件头做 DUMP 等。如果 db file scattered read 事件是伴随 Multi Block I/O 发生的等待事件，那 db file sequential read 事件就是伴随 Single Block I/O 发生的等待事件。每次发生 Single Block I/O 时，就会发生一次 db file sequential read 事件的等待。Single Block I/O 可以发生在从文件读取一个块的所有工作上，一般在索引扫描、通过 ROWID 的表扫描、读取控制文件和文件头时发生。

在 V$SESSION_WAIT 这个视图里面，这个等待事件有三个参数 P1、P2、P3，其中 P1 代表 Oracle 要读取的文件的绝对文件号即 File#，P2 代表 Oracle 从这个文件中开始读取的起始数据块的 BLOCK 号即 Block#，P3 代表 Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量即 Blocks，通常这个值为 1，表明是单个 BLOCK 被读取，如果这个值大于 1，则是读取了多个 BLOCK，这种多 BLOCK 读取常常出现在早期的 Oracle 版本中从临时段中读取数据的时候。

这个等待事件有三个参数：

File#： 要读取的数据块所在数据文件的文件号。

Block#： 要读取的起始数据块号。

Blocks：要读取的数据块数目（这里应该等于 1）。

```
SELECT *
FROM   v$event_name
WHERE  NAME = 'db file sequential read';
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
117	2652584166	db file sequential read	file#	block#	blocks	1740759767	8	User I/O

db file sequential read 等待使性能出现问题，这些性能问题大多数发生在低效的索引扫描、行迁移、行链接引发附加的 I/O 过程中。

1、应用程序层

低效的 sql 语句或低效的索引扫描经常被使用时，因不必要的物理 I/O 增加，可能增加 db file sequential read 等待。使用选择性较差的索引是发生 db file sequential read 等待的主要原因。

2、oracle 内存层

如果高速缓冲区过小，就会反复发生物理 I/O，因此可能增加 db file sequential read 等待，这时同时发生 free buffer waits 等待的概率较高。如果大量发生 free buffer waits 等待，应该考虑扩展高速缓存区的大小。始终要考虑利用多重缓冲池，有效使用高速缓存区。利用多重缓冲池减少 db file sequential read 等到的原理，与减少 db file scattered read 等待的原理相同。

3、OS/裸设备层

如果 sql 优化或高速缓存区优化、重建表也不能解决问题，就应该怀疑 I/O 系统本身的性能。将 db file sequential read 事件的等待次数和等待时间比较后，如果平均等待时间长，缓慢的 I/O 系统成为原因的可能性高。之前也讨论过，I/O 系统上的性能问题在多钟情况下均会发生，因此需要充分调查各种因素。

利用 v$filestat 视图，可分别获得各数据文件关于 Multi Block I/O 和 Single Block I/O 的活动信息。

```
SELECT F.FILE#,
       F.NAME,
       S.PHYRDS,
       S.PHYBLKRD,
       S.READTIM, --所有的读取工作信息
       S.SINGLEBLKRDS,
       S.SINGLEBLKRDTIM, --SINGLE BLOCK I/O
       (S.PHYBLKRD - S.SINGLEBLKRDS) AS MULTIBLKRD, --MULTI BLOCK I/O 次数
       (S.READTIM - S.SINGLEBLKRDTIM) AS MULTIBLKRDTIM, --MULTI BLOCK I/O 时间
```

```
     ROUND(S.SINGLEBLKRDTIM /
          DECODE(S.SINGLEBLKRDS, 0, 1, S.SINGLEBLKRDS),
          3) AS SINGLEBLK_AVGTIM, --SINGLE BLOCK I/O 平均等待时间(CS)
     ROUND((S.READTIM - S.SINGLEBLKRDTIM) /
          NULLIF((S.PHYBLKRD - S.SINGLEBLKRDS), 0),
          3) AS MULTIBLK_AVGTIM --MULTI BLOCK I/O 平均等待时间(CS)
 FROM V$FILESTAT S, V$DATAFILE F
WHERE S.FILE# = F.FILE#;
```

如果特点文件上平均执行时间表现的过高，则应该通过提高该文件所在的 I/O 系统的性能，以此改善性能。没有关于 Multi Block I/O 的最合理的平均等待时间值，但一般应该维持 10 微妙左右的平均等待时间。

在 Oracle 10g 中，这个等待事件被归入 User I/O 一类：

这一事件通常显示与单个数据块相关的读取操作(如索引读取)。如果这个等待事件比较显著，可能表示在多表连接中，表的连接顺序存在问题，可能没有正确的使用驱动表；或者可能索引的使用存在问题，不加选择地进行索引，并非索引总是最好的选择。

还有一种特殊的情况是，全表扫描过程还会产生单块读的情况有，读 UNDO 块。可以参考最后的老熊文章的例子 http://blog.itpub.net/26736162/viewspace-2123513/。对于这种情况的解决办法是加索引，或等大事务执行完成后再执行 SQL。

这里的 sequential 也并非指的是 Oracle 按顺序的方式来访问数据，和 db file scattered read 一样，它指的是读取的数据块在内存中是以连续的方式存放的。

在大多数的情况下读取一个索引数据的 BLOCK 或者通过索引读取数据的一个 BLOCK 的时候都会去要读取相应的数据文件头的 BLOCK。在早期的版本中会从磁盘中的排序段读取多个 BLOCK 到高速缓存区的连续的缓存中。

在大多数情况下，通过索引可以更为快速地获取记录，所以对于一个编码规范、调整良好的数据库，这个等待事件很大通常是正常的。有时候这个等待过高和存储分布不连续、连续数据块中部分被缓存有关，特别对于 DML 频繁的数据表，数据以及存储空间的不连续可能导致过量的单块读，定期的数据整理和空间回收有时候是必须的。

但是在很多情况下，使用索引并不是最佳的选择，比如读取较大表中大量的数据，全表扫描可能会明显快于索引扫描，所以在开发中就应该注意，对于这样的查询应该避免使用索引扫描。

图 9-16 简要说明了单块读取的操作方式。

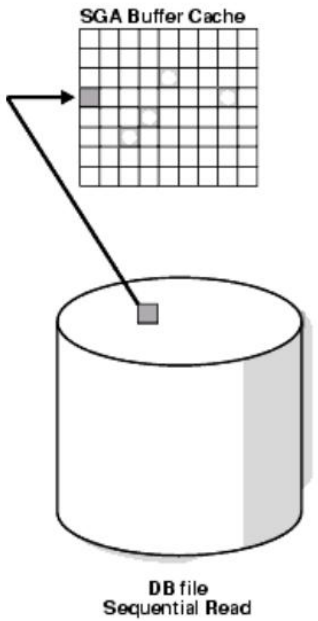


图 9-16　单块读取的操作

如果这个等待事件在整个等待时间中占主要的部分，可以采用以下的几种方法来调整数据库。

方法一：从 AWR 的报告中的"SQL ordered by Reads"部分或者从 V$SQL 视图中找出读取物理磁盘 I/O 最多的几个 SQL 语句，优化这些 SQL 语句以减少对 I/O 的读取需求。

如果有 Index Range scans，但是却使用了不该用的索引，就会导致访问更多的 BLOCK，这个时候应该强迫使用一个可选择的索引，使访问同样的数据尽可能的少的访问索引块，减少物理 I/O 的读取；如果索引的碎片比较多，那么每个 BLOCK 存储的索引数据就比较少，这样需要访问的 BLOCK 就多，这个时候一般来说最好把索引 rebuild，减少索引的碎片；如果被使用的索引存在一个很大的 Clustering Factor，那么对于每个索引 BLOCK 获取相应的记录的时候就要访问更多表的 BLOCK，这

个时候可以使用特殊的索引列排序来重建表的所有记录，这样可以大大的减少 Clustering Factor，例如：一个表有 A,B,C,D,E 五个列，索引建立在 A,C 上，这样可以使用如下语句来重建表：

```
CREATE TABLE TABLE_NAME AS SELECT * FROM old ORDER BY A,C;
```

此外，还可以通过使用分区索引来减少索引 BLOCK 和表 BLOCK 的读取。

方法二：如果不存在有问题的执行计划导致读取过多的物理 I/O 的特殊 SQL 语句，那么可能存在以下的情况：

数据文件所在的磁盘存在大量的活动，导致其 I/O 性能很差。这种情况下可以通过查看 AWR 报告中的"File I/O Statistics"部分或者 V$FILESTAT 视图找出热点的磁盘，然后将在这些磁盘上的数据文件移动到那些使用了条带集、RAID 等能实现 I/O 负载均衡的磁盘上去。

使用如下的查询语句可以得到各个数据文件的 I/O 分布：

```
SELECT d.name NAME,
       f.phyrds,
       f.phyblkrd,
       f.phywrts,
       f.phyblkwrt,
       f.readtim,
       f.writetim
FROM   v$filestat f,
       v$datafile d
WHERE  f.file# = d.file#
ORDER  BY f.phyrds  DESC,
          f.phywrts DESC;
```

从 Oracle9.2.0 开始，我们可以从 V$SEGMENT_STATISTICS 视图中找出物理读取最多的索引段或者是表段，通过查看这些数据，可以清楚详细的看到这些段是否可以使用重建或者分区的方法来减少所使用的 I/O。如果 Statpack 设置的 level 为 7 就会在报告中产生"Segment Statistics"的信息。

```
SELECT statistic_name,
       COUNT(1)
FROM   v$segment_statistics T
GROUP  BY T.STATISTIC_NAME;
```

STATISTIC_NAME	COUNT(1)
logical reads	5799
segment scans	5799
gc buffer busy	5799
physical writes	5799
physical reads direct	5799
space allocated	5799
gc cr blocks received	5799
gc current blocks received	5799
ITL waits	5799
physical reads	5799
db block changes	5799
row lock waits	5799
space used	5799
buffer busy waits	5799
physical writes direct	5799

从上面的查询可以看到相应的统计名称，使用下面的查询语句就能得到读取物理 I/O 最多的段：

```
SELECT object_name,
       object_type,
       statistic_name,
       VALUE
FROM   v$segment_statistics
WHERE  statistic_name = 'physical reads'
ORDER  BY VALUE DESC;
```

方法三：如果不存在有问题的执行计划导致读取过多的物理 I/O 的特殊 SQL 语句，磁盘的 I/O 也分布的很均匀，这种时候我们可以考虑增大的高速缓存区。对于 Oracle8i 来说增大初始化参数 DB_BLOCK_BUFFERS，让 Statpack 中的 Buffer Cache 的命中率达到一个满意值；对于 Oracle9i 来说则可以使用 Buffer Cache Advisory 工具来调整 Buffer Cache；对于热点的段可以使用多缓冲池，将热点的索引和表放入到 KEEP Buffer Pool 中去，尽量让其在缓冲中被读取，减少 I/O。

1.6.1.1 例子

老熊的一篇文章：常识之外，全表扫描为何产生大量 db file sequential read 单块读(常识之外：全表扫描为何产生大量 db file sequential read 单块读？http://blog.itpub.net/26736162/viewspace-2123513/)：，介绍了，**全表扫描过程还会产生单块读的情况有，读 UNDO 块。**

1.6.2 db file scattered read（数据文件离散读）

在 V$SESSION_WAIT 这个视图里面，这个等待事件有三个参数 P1、P2、P3，其中 P1 代表 Oracle 要读取的文件的绝对文件号，P2 代表 Oracle 从这个文件中开始读取的 BLOCK 号，P3 代表 Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量。

```
SELECT *
FROM   v$event_name
WHERE  NAME IN ('db file sequential read', 'db file scattered read');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
117	2652584166	db file sequential read	file#	block#	blocks	1740759767	8	User I/O
118	506183215	db file scattered read	file#	block#	blocks	1740759767	8	User I/O

从 V$EVENT_NAME 视图可以看到，该等待事件有3个参数：

File#： 要读取的数据块所在数据文件的文件号。

Block#： 要读取的起始数据块号。

Blocks：需要读取的数据块数目。

这样就可以找到那个对象：

```
SELECT EVENT, P1, P2, P3, ROW_WAIT_OBJ#
  FROM GV$SESSION
 WHERE EVENT = 'db file scattered read';
SELECT OBJECT_NAME, OBJECT_TYPE
  FROM DBA_OBJECTS
 WHERE OBJECT_ID = ROW_WAIT_OBJ#;
```

起始数据块号加上数据块的数量，这意味着 Oracle session 正在等待多块连续读操作的完成。

这个等待事件在实际生产库中经常可以看到，这是一个用户操作引起的等待事件，当用户发出每次 I/O 需要读取多个数据块这样的 SQL 操作时或者说当 Oracle 从磁盘上读取多个 BLOCK 到不连续的高速缓存区的缓存中，会产生这个等待事件，这个事件表明用户进程正在读数据到 Buffer Cache 中，等待直到物理 I/O 调用返回。最常见的两种情况是全表扫描（FTS： Full Table Scan）和索引快速全扫描（IFFS： index fast full scan）。为保障性能，尽量一次读取多个块，这称为 Multi Block I/O。每次执行 Multi Block I/O，都会等待物理 I/O 结束，此时等待 db file scattered read 事件。根据经验，通常大量的 db file scattered read 等待可能意味着应用问题或者索引缺失。Oracle 一次能够读取的最多的 BLOCK 数量是由初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT 来决定。

这个名称中的 scattered(发散)，可能会导致很多人认为它是以 scattered 的方式来读取数据块的，其实恰恰相反，当发生这种等待事件时，SQL 的操作都是顺序地读取数据块的，比如 FTS 或者 IFFS 方式（如果忽略需要读取的数据块已经存在内存中的情况）。这里的 scattered 指的是读取的数据块在内存中的存放方式，他

们被读取到内存中后，是以分散的方式存在在内存中，而不是连续的。

在生产环境之中，db file scattered read 这个等待事件可能更为常见。DB File Scattered Read 发出离散读，将存储上连续的数据块离散的读入到多个不连续的内存位置。Scattered Read 通常是多块读，在 Full Table Scan 或 Fast Full Scan 等访问方式下使用。Scattered Read 代表 Full Scan，当执行 Full Scan 读取数据到 Buffer Cache 时，通常连续的数据在内存中的存储位置并不连续，所以这个等待被命名为 Scattered Read（离散读）。每次多块读读取的数据块数量受初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT 限制。

Oracle 按照 db_file_multiblock_read_count（以下简称 MBRC）参数值进行 Multi Block I/O。这个值每个 OS 都有最大的界定，可以通过如下方法确认最大值。

```
SQL> show parameter db_file_multiblock_read_count

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_multiblock_read_count        integer     128
SQL> alter system set db_file_multiblock_read_count=100000;    --试图变更为超大值

系统已更改。

SQL> show parameter db_file_multiblock_read_count;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_multiblock_read_count        integer     4096          --确认 4096 是一次可以读取的最多块数
```

oracle 在执行 FTS 时也进行 Single Block I/O。这时即便是 FTS 也会发生 db file sequential read 等待。FTS 上使用 Single Block I/O 或读取比 MBRC 值小的块数的情况如下：

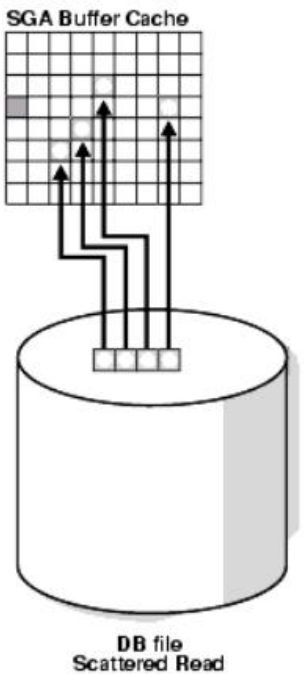
（1）达到区的界线时：如一个区有 9 个块，一次 Multi Block I/O 读取 8 个块，则一次以 Multi Block I/O 读取之后的剩余一个块通过 Single Block I/O

读取，如果剩下的块有两个，就会执行 Multi Block I/O，而且只读取两个块。

（2）扫描过程中读取被缓存的块时：如读取 8 个块时，其中第三个块被缓存，oracle 将前两个块通过 Multi Block I/O 读取，对于第三个块执行一次 Logical I/O，剩下的 5 个块通过 Multi Block I/O 读取。这种情况经常发生时，因引发多次的 I/O，可能成为 FTS 速度下降的原因。

（3）存在行链接时：在执行 FTS 的过程中，如果发现了行链接，oracle 为了读取剩下的行引起的附加 I/O，此时执行 Single Block I/O。

图 9-17 简要说明了 Scattered Read 的数据读取方式。



9-17　Scattered Read 的数据读取

完成对等待事件的分类之后，Oracle 10g 的 ADDM 可以很容易地通过故障树分析定位到问题所在，帮助用户快速发现数据库的瓶颈及瓶颈的根源，这就是 Oracle

的 ADDM 专家系统的设计思想。通过图 9-18 可以直观而清晰地看到这个等待模型和 ADDM 结合实现的 Oracle 专家诊断系统。

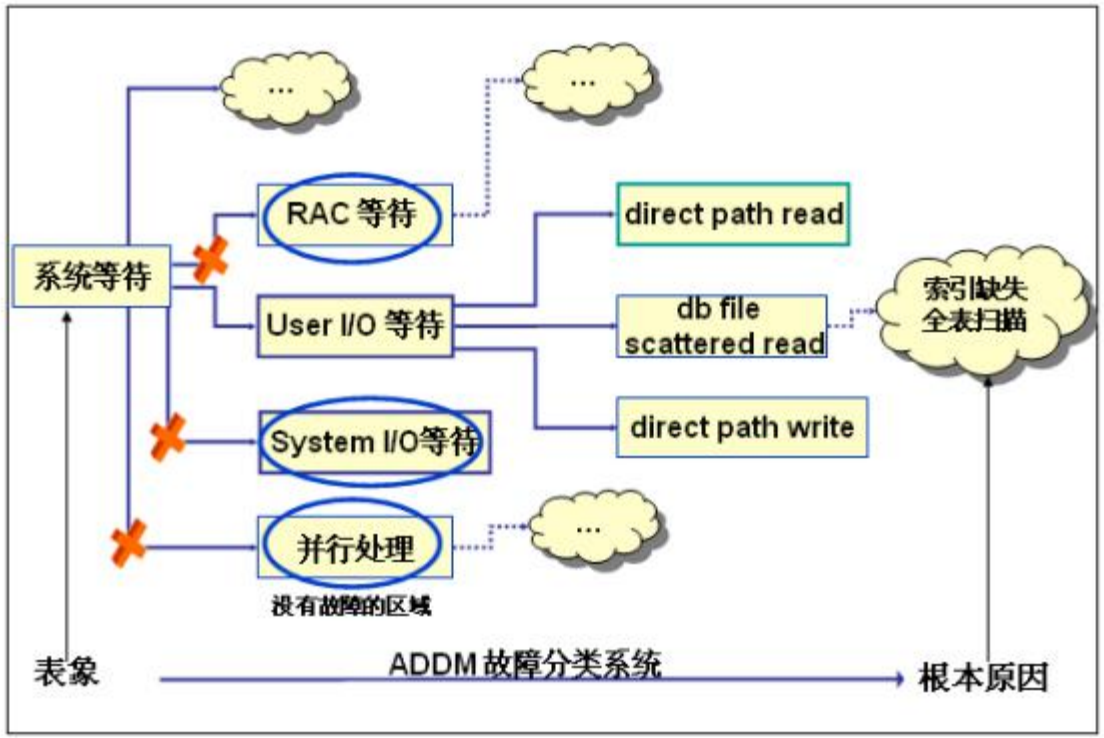


图 9-18　Oracle 专家诊断系统

这种情况通常显示与全表扫描相关的等待。当数据库进行全表扫时，基于性能的考虑，数据会分散(scattered)读入 Buffer Cache。如果这个等待事件比较显著，可能说明对于某些全表扫描的表，没有创建索引或者没有创建合适的索引，我们可能需要检查这些数据表已确定是否进行了正确的设置。然而这个等待事件不一定意味着性能低下，在某些条件下 Oracle 会主动使用全表扫描来替换索引扫描以提高性能，这和访问的数据量有关，在 CBO 下 Oracle 会进行更为智能的选择，在 RBO 下 Oracle 更倾向于使用索引。因为全表扫描被置于 LRU(Least Recently Used，最近最少适用)列表的冷端(cold end)，对于频繁访问的较小的数据表，可以选择把他们 Cache 到内存中，以避免反复读取。

在实际环境的诊断过程中，可以通过 v$session_wait 视图发现 session 的等待，再结合其他视图找到存在问题的 SQL 等根本原因，从而从根本上解决问题。在 11g 也可以直接通过 v$session 视图来查询等待事件。当这个等待事件比较显著时，也可结合 v$session_longops 动态性能视图来进行诊断，该视图记录了长时间（运行时间超过 6 秒的）运行的事务。

db file scattered read 事件与 db file sequential read 事件相同，是 oracle 中最经常发生的等待事件。因为从数据文件读取块时只能执行 Multi Block I/O 或 Single Block I/O。

1、应用程序层

需要筛选出主要发生 db file scattered read 等待的 sql 语句。如果不必要的执行 FTS 或 Index Full San，修改 sql 语句或创建更合理的索引就可以解决。大量读取数据时多数情况下 FTS 性能更好。不是盲目的创建索引，而是要考虑相应的 sql 语句后，判断 FTS 有利，还是 Index Full San 有利。

2、oracle 内存层

如果高速缓存区过小，就会反复需要物理 I/O，相应的 db file scattered read 等待也会增加。这时 free buffer waits 等待事件一同出现的几率较高。FTS 引起的 db file scattered read 等待的严重性不仅在于需要 I/O，而且在于降低高速缓存区的效率，进而影响会话工作。从这种角度出发，处理 FTS 的有效方法之一就是使用多重缓冲池。读取一次后不再使用的数据，有必要保存到高速缓存区从而导致影响其他用户的工作吗？多重缓冲池虽然是有效管理高速缓存区的强有力的方法，但是遗憾的是没有被广泛使用。多重缓冲池从三个方面改善高速缓存区的性能。第一，将经常访问的对象保存与内存，进而将物理 I/O 最小化。第二，临时性数据所占用的内存被快速的重新使用，进而将内存的浪费最小化。第三，因为每个缓冲池各使用不同的 cache buffers lru chain 锁存器，所以有减少锁存器争用的效果。指定 DEFAULT 将适用默认的缓冲池。这个选项适用于没有分配给 KEEP 缓冲池和 RECYCLE 缓冲池的其它数据库对象。通常将经常访问的对象放入 KEEP 缓冲池中，指定 KEEP

将把数据块放入 KEEP 缓冲池中。维护一个适当尺寸的 KEEP 缓冲池可以使 Oracle 在内存中保留数据库对象而避免 I/O 操作。通常将偶尔访问的大表放入 RECYCLE 缓冲池中，指定 RECYCLE 将把数据块放入 RECYCLE 缓冲池中。一个适当尺寸的 RECYCLE 缓冲池可以减少默认缓冲池为 RECYCLE 缓冲池的数据库对象的数量，以避免它们占用不必要的缓冲空间。

有效使用 FTS 的另一种方法是将 db_file_multiblock_read_count 参数值提高。这个参数决定执行 Multi Block I/O 时一次读取的块数。因此这个值高，FTS 速度相应也会提升，而且 db file scattered read 等待也会相应减少。将这个值在全系统级上设定得高，并不太妥当。最好是利用 alter session set ...命令，只在执行 sql 语句期间提升这个值。因为这个值如果升高，有关 FTS 的费用会算的较低，可能会导致 sql 执行计划的变更。

较大的块也是提高 FTS 性能的方法。较大的块在如下两个方面改善 FTS 的性能。第一，增加一个块所包含的行数，这样相同大小的表时使用更少的块数，相应的 Multi Block I/O 次数也会减少。第二，块的大小较大，则发生行链接或行迁移的概率会降低，附加的 I/O 也随之降低。大部分 OLTP 系统上一般只是用标准块大小（8K）。但是经常扫描大量数据的 OLAP 上使用更大的块能改善性能。

3、oracle 段层

需要检查，通过合理执行 partition 能否减少 FTS 范围。例如为获得 100 万个数据中 10 万个数据而执行 FTS 时，将 10 万个数据相应的范围利用 partition 分开，则可以将 FTS 的范围缩小至 1/10。

4、OS/裸设备层

如果利用 sql 的优化或高速缓存区的优化也不能解决问题，就应该怀疑 I/O 系统本身的性能。将 db file scattered read 事件的等待次数和等待时间比较后，如果平均等待时间长，缓慢的 I/O 系统成为原因的可能性高。之前也讨论过，I/O 系统上的性能问题在多钟情况下均会发生，因此需要充分调查各种因素。

利用 v$filestat 视图，可分别获得各数据文件关于 Multi Block I/O 和 Single Block I/O 的活动信息。

```
select f.file#,
       f.name,
       s.phyrds,
       s.phyblkrd,
       s.readtim, --所有的读取工作信息
       s.singleblkrds,
       s.singleblkrdtim, --Single Block I/O
       (s.phyblkrd - s.singleblkrds) as multiblkrd, --Multi Block I/O 次数
       (s.readtim - s.singleblkrdtim) as multiblkrdtim, --Multi Block I/O 时间
       round(s.singleblkrdtim /
             decode(s.singleblkrds, 0, 1, s.singleblkrds),
             3) as singleblk_avgtim, --Single Block I/O 平均等待时间（cs）
       round((s.readtim - s.singleblkrdtim) /
             nullif((s.phyblkrd - s.singleblkrds), 0),
             3) as multiblk_avgtim --Multi Block I/O 平均等待时间（cs）
  from v$filestat s, v$datafile f
 where s.file# = f.file#;
```

如果特点文件上平均执行时间表现的过高，则应该通过提高该文件所在的 I/O 系统的性能，以此改善性能。没有关于 Multi Block I/O 的最合理的平均等待时间值，但一般应该维持 10 微妙左右的平均等待时间。

如果这个等待事件在整个等待时间中占了比较大的比重，可以如下的几种方法来调整 Oracle 数据库：

方法一：找出执行全表扫描（FTS： Full Table Scan）和索引快速全扫描（IFFS： index fast full scan）扫描的 SQL 语句，判断这些扫描是否是必要的，是否导致了比较差的执行计划，如果是，则需要调整这些 SQL 语句，可以结合 v$session_longops 动态性能视图来进行诊断，该视图中记录了长时间（运行时间超过 6 秒的）运行的事物，可能很多是全表扫描操作。

从 Oracle9i 开始提供了一个视图 V$SQL_PLAN 用于记录当前系统 Library Cache 中 SQL 语句的执行计划，可以通过这个视图找到存在问题的 SQL 语句，即可以很快的帮助找到那些全表扫描或者 Fast Full Index 扫描的 SQL 语句，这个视图会自动忽略掉关于数据字典的 SQL 语句。

查找全表扫描的 SQL 语句可以使用如下语句：

通过 V$SQL_PLAN 和 V$SQLTEXT 联合，获得全表扫描的 SQL 语句

```
SELECT sql_text
FROM   v$sqltext  t,
       v$sql_plan p
WHERE  t.hash_value = p.hash_value
AND    p.operation = 'TABLE ACCESS'
AND    p.options = 'FULL'
ORDER  BY p.hash_value,
          t.piece;
```

获得全表扫描的对象

```
SELECT DISTINCT object_name,
                object_owner
FROM   v$sql_plan p
WHERE  p.operation = 'TABLE ACCESS'
AND    p.options = 'FULL'
AND    object_owner = 'SYS';
```

查找 Fast Full Index 扫描的 SQL 语句可以使用如下语句：

```
SELECT sql_text
FROM   v$sqltext  t,
       v$sql_plan p
WHERE  t.hash_value = p.hash_value
AND    p.operation = 'INDEX'
AND    p.options = 'FULL SCAN'
ORDER  BY p.hash_value, t.piece;
```

获得全索引扫描的对象

```
SELECT DISTINCT object_name,
                object_owner
FROM   v$sql_plan p
WHERE  p.operation = 'INDEX'
AND    p.options = 'FULL SCAN'
AND    object_owner = 'SYS';
```

如果是 Oracle8i 的数据库，可以从 V$SESSION_EVENT 视图中找到关于这个等待事件的进程 sid，然后根据 sid 来跟踪相应的会话的 SQL。

```
select sid,event from v$session_event where event='db file sequential read'
```

或者可以查看物理读取最多的 SQL 语句的执行计划，看是否里面包含了全表扫描和 Fast Full Index 扫描。通过如下语句来查找物理读取最多的 SQL 语句：

```
select sql_text from (
select * from v$sqlarea
order by disk_reads)
where rownum<=10;
```

方法二：有时候在执行计划很好情况下也会出现多 BLOCK 扫描的情况，这时可以通过调整 Oracle 数据库的多 BLOCK 的 I/O，设置一个合理的 Oracle 初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT，尽量使得满足以下的公式：

DB_BLOCK_SIZE x DB_FILE_MULTIBLOCK_READ_COUNT = max_io_size of system

DB_FILE_MULTIBLOCK_READ_COUNT 是指在全表扫描中一次能够读取的最多的 BLOCK 数量，这个值受操作系统每次能够读写最大的 I/O 限制，如果设置的值按照上面的公式计算超过了操作系统每次的最大读写能力，则会默认为 max_io_size/db_block_size。例如 DB_FILE_MULTIBLOCK_READ_COUNT 设置为 32，DB_BLOCK_SIZE 为 8K，这样每次全表扫描的时候能读取 256K 的表数据，从而大大的提高了整体查询的性能。设置这个参数也不是越大越好的，设置这个参数之前应该要先了解应用的类型，如果是 OLTP 类型的应用，一般来说全表扫描较少，这个时候设定比较大的 DB_FILE_MULTIBLOCK_READ_COUNT 反而会降低 Oracle 数据库的性

能，因此 CBO 在某些情况下会因为多 BLOCK 读取导致 COST 比较低从而错误的选用全表扫描。

方法三： 通过对表和索引使用分区、将缓存区的 LRU 末端的全表扫描和 IFFS 扫描的的 BLOCK 放入到 KEEP 缓存池中等方法调整这个等待事件。

1.6.3 db file parallel read

```
SELECT *
FROM   v$event_name
WHERE  NAME IN ('db file parallel read');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
152	834992820	db file parallel read	files	blocks	requests	1740759767	8	User I/O

在 V$SESSION_WAIT 这个视图里面，这个等待事件有三个参数 P1、P2、P3，其中 P1 为 files 代表有多少个文件被读取所请求，P2 为 blocks 代表总共有多少个 BLOCK 被请求，P3 为 requests 代表总共有多少次 I/O 请求。

db file parallel read 物理读等待事件涉及到的数据块均是不连续的，同时可以跨越 extent，这点不像 db file scattered read。

这是一个很容易引起误导的等待事件，实际上这个等待事件和并行操作（比如并行查询，并行 DML）没有关系。这个事件发生在数据库恢复的时候，当有一些数据块需要恢复的时候，Oracle 会以并行的方式把他们从数据文件中读入到内存中进行恢复操作。当 Oracle 从多个数据文件中并行的物理读取多个 BLOCK 到内存的不连续缓冲中(可能是高速缓存区或者是 PGA)的时候可能就会出现这个等待事件。这种并行读取一般出现在恢复操作中或者是从缓冲中预取数据达到最优化(而不是多次从单个 BLOCK 中读取，buffer prefetch 以优化多个单块读)。这个事件表明会话正在并行执行多个读取的需求。注意：在 11g 之前，这个等待事件发生在数据文件的恢复过

程中，但 11g 中新增了 prefetch 的特性，所以也可能导致这个等待事件的产生。

如果在等待时间中这个等待事件占的比重比较大，可以按照处理 db file sequential read 等待事件的方法来处理这个事件。

若是由于 prefetch 引起的性能问题，我们可以通过添加隐含参数来解决该问题。可以参考 blog：http://blog.itpub.net/26736162/viewspace-2123473

```
set pagesize 9999
set line 9999
col NAME format a40
col KSPPDESC format a50
col KSPPSTVL format a20
SELECT a.INDX,
      a.KSPPINM NAME,
      a.KSPPDESC,
      b.KSPPSTVL
FROM   x$ksppi  a,
      x$ksppcv b
WHERE  a.INDX = b.INDX
and lower(a.KSPPINM) IN ('_db_block_prefetch_quota','_db_block_prefetch_limit','_db_file_noncontig_mblock_read_count');
ALTER SYSTEM SET  "_db_block_prefetch_quota"=0 SCOPE=SPFILE SID='*';
ALTER SYSTEM SET  "_db_block_prefetch_limit"=0 SCOPE=SPFILE SID='*';
ALTER SYSTEM SET  "_db_file_noncontig_mblock_read_count"=0 SCOPE=SPFILE SID='*';
```

```
SYS@oraESKDB1> set pagesize 9999
SYS@oraESKDB1> set line 9999
SYS@oraESKDB1> col NAME format a40
SYS@oraESKDB1> col KSPPDESC format a50
SYS@oraESKDB1> col KSPPSTVL format a20
SYS@oraESKDB1> SELECT a.INDX,
  2         a.KSPPINM NAME,
  3         a.KSPPDESC,
  4         b.KSPPSTVL
  5  FROM   x$ksppi  a,
  6         x$ksppcv b
  7  WHERE  a.INDX = b.INDX
  8  and lower(a.KSPPINM) IN ('_db_block_prefetch_quota','_db_block_prefetch_limit','_db_file_noncontig_mblock_read_count');

     INDX NAME                                 KSPPDESC                                           KSPPSTVL
--------- ---------------------------------------- -------------------------------------------------- --------------------
      881 _db_block_prefetch_quota                 Prefetch quota as a percent of cache size          10
      883 _db_block_prefetch_limit                 Prefetch limit in blocks                           0
     1156 _db_file_noncontig_mblock_read_count     number of noncontiguous db blocks to be prefetched 11

SYS@oraESKDB1> ALTER SYSTEM SET  "_db_file_noncontig_mblock_read_count"=0 SCOPE=SPFILE SID='*';

System altered.

SYS@oraESKDB1> ALTER SYSTEM SET  "_db_block_prefetch_quota"=0 SCOPE=SPFILE SID='*';
```

```
System altered.

SYS@oraESKDB1> ALTER SYSTEM SET  "_db_block_prefetch_limit"=0 SCOPE=SPFILE SID='*';

System altered.

SYS@oraESKDB1> ALTER SYSTEM SET  "_db_file_noncontig_mblock_read_count"=0 SCOPE=SPFILE SID='*';
```

1.6.4 db file single write

这个等待事件通常只发生在一种情况下，就是 Oracle 更新数据文件头信息时（比如发生 Checkpoint）。

当这个等待事件很明显时，需要考虑是不是数据库中的数据文件数量太大，导致 Oracle 需要花较长的时间来做所有文件头的更新操作（checkpoint）。

```
SELECT *
FROM   v$event_name
WHERE  NAME IN ('db file single write');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
149	1307477558	db file single write	file#	block#	blocks	1740759767	8	User I/O

这个等待事件有三个参数：

- **file#:** 需要更新的数据块所在的数据文件的文件号。查询文件号的 SQL 语句是：SELECT * FROM v$datafile WHERE file# = <file#>;
- **block#：**需要更新的数据块号，如果 BLOCK 号不是 1，则可以通过如下查询查出 Oracle 正在写入的对象是什么：

```
SELECT  segment_name ,  segment_type ,
owner , tablespace_name
FROM  sys.dba_extents
WHERE  file_id = <file#>
AND  <block#>
BETWEEN block_id AND block_id + blocks -1 ;
```

- **blocks :**需要更新的数据块数目（通常来说应该等于 1），或 Oracle 写入 file＃的数据文件中从 BLOCK#开始写入的 BLOCK 的数量。头一般来说都是 BLOCK1，

操作系统指定的文件头是 BLOCK0，如果 BLOCK 号大于 1，则表明 Oracle 正在写入的是一个对象而不是文件头。

1.6.5 direct path read（直接路径读、DPR）

直接路径读等待事件的 3 个参数分别是：file#（指绝对文件号）、first block#和 block 数量。

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME = 'direct path read';
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
197	3926164927	direct path read	file number	first dba	block cnt	1740759767	8	User I/O

这个等待事件有三个参数：

file number：等待 I/O 读取请求的文件的绝对文件号

first dba：等待 I/O 读取请求的第一个 BLOCK 号

block cnt：以 first block 为起点，总共有多少个连续的 BLOCK 被请求读取

由参数 P1 与 P2 推得访问的数据对象：

```
select s.segment_name, s.partition_name
  from dba_extents s
 where  between s.block_id and (s.block_id + s.blocks -1) and s.file_id =
```

直接路径读（direct path read）通常发生在 Oracle **直接读取数据到 PGA 时**，这个读取不需要经过 SGA。这类读取通常在以下情况被使用：

① **大量的磁盘排序 IO 操作** 在排序操作(order by, group by, union, distinct, rollup,合并连接)时，由于 PGA 中的 SORT_AREA_SIZE 空间不足，无法在 PGA 中完成排序，需要利用 temp 表空间进行排序，当从临时表空间中读取排序结果时，会产生 direct path read，**从 10g 开始表现为 direct path read temp**

等待事件。

② **大量的 Hash Join 操作**，利用 temp 表空间保存 hash 区。使用 HASH 连接的 SQL 语句，将不适合位于内存中的散列分区刷新到临时表空间中。为了查明匹配 SQL 谓词的行，临时表空间中的散列分区被读回到内存中（目的是为了查明匹配 SQL 谓词的行），ORALCE 会话在 direct path read 等待事件上等待。

③ **SQL 语句的并行查询，并行查询从属进程** 使用并行扫描的 SQL 语句也会影响系统范围的 direct path read 等待事件。在并行执行过程中，direct path read 等待事件与从属查询有关，而与父查询无关，运行父查询的会话基本上会在 PX Deq:Execute Reply 上等待，从属查询会产生 direct path read 等待事件。

④ **预读操作**

⑤ **串行全表扫描**(Serial Table Scan)，**大表的全表扫描**，在 Oracle11g 中，全表扫描的算法有新的变化，根据表的大小、高速缓存的大小等信息，决定是否绕过 SGA 直接从磁盘读取数据。而10g 则是全部通过高速缓存读取数据，称为 table scan(large)。11g 认为大表全表时使用直接路径读，可能比10g 中的数据文件散列读(db file scattered reads)速度更快，使用的 latch 也更少。

最常见的是第一种情况。在 DSS 系统中，存在大量的 Direct path read 是很正常的，但是在 OLTP 系统中，通常显著的直接路径读都意味着系统应用存在问题，从而导致大量的磁盘排序读取操作。

db file sequential read、db file scattered read、direct path read 是常见的集中数据读方式，下图简要描述了这 3 种方式的读取示意。

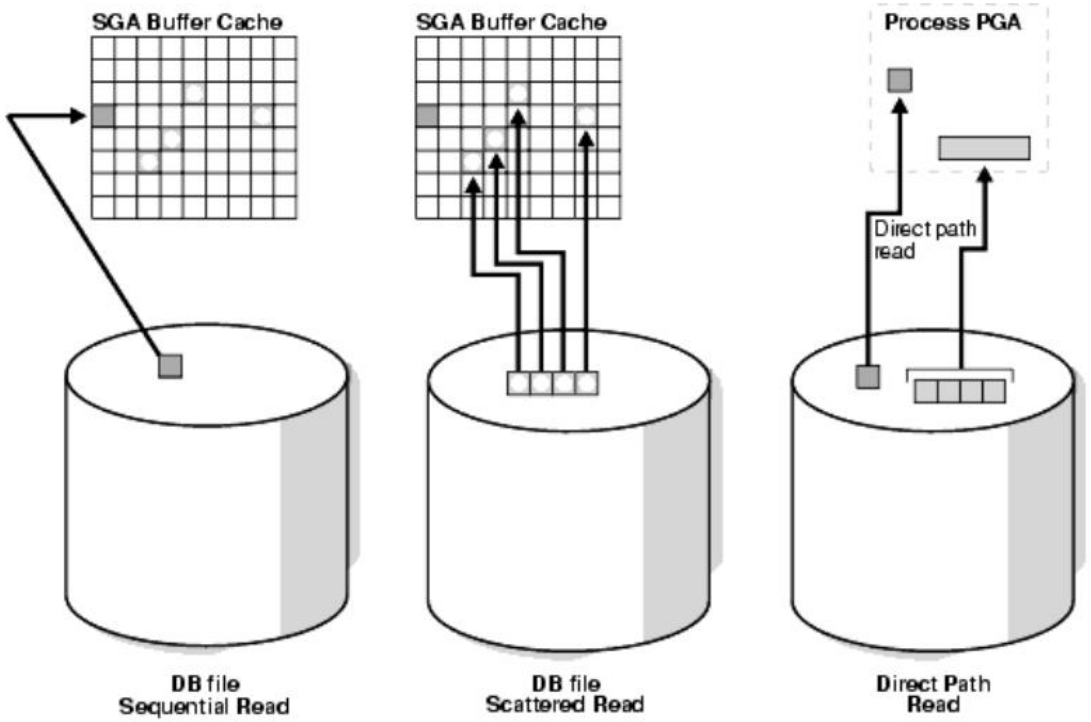


图 9-19　3 种读取方式

大量的 direct path read 等待时间最可能是一个应用程序问题。

direct path read 事件由 SQL 语句驱动，这些 SQL 语句执行来自临时的或常规的表空间的直接读取操作。当输入的内容大于 PGA 中的工作区域时，带有需要排序的函数的 SQL 语句将排序结果写入到临时表空间中，临时表空间中的排序顺序串随后被合并，用于提供最终的结果。读取排序结果时，Oracle 会话在 direct path read 等待事件上等待。

对于这一写入等待，我们应该找到 I/O 操作最为频繁的数据文件（如果有过多的排序操作，很有可能就是临时文件），分散负载，加快其写入操作。

DB_FILE_DIRECT_IO_COUNT 初始化参数可能影响 direct path read 的性能。

直接读取可能按照同步或异步的方式执行，取决于平台和初始化参数 disk_asynch_io 参数的值。使用异步 I/O 时，系统范围的等待的事件的统计可能不准确，会造成误导作用。

该事件一般不可能显示为主要的瓶颈，但它实际上也许是就是祸首。由于 ORACLE 统计等待时间的方式会造成统计的时间量不准确(如：从属查询产生的时间无法进行统计)，所以对该事件不应该使用 v$session_event 视图中的 total_wait 或 time_waited 进行估计，应该使用 v$sesstat 视图中的直接读取操作次数(physical reads direct)进行判断：

```
select a.NAME,
      b.SID,
      b.VALUE,
      round((sysdate - c.LOGON_TIME) * 24) hours_connected
  from v$statname a, v$sesstat b, v$session c
 where b.SID = c.SID
   and a.STATISTIC# = b.STATISTIC#
   and b.VALUE > 0
   and a.NAME = 'physical reads direct'
 order by b.VALUE;
```

由 direct path read 事件产生的原因，我们需要判断该事件正在读取什么段(如：散列段、排序段、一般性的数据文件)，由此可判断产生该事件的原因是什么，可使用以下语句进行查询：

```
SELECT a.event,
      a.sid,
      c.sql_hash_value hash_vale,
      decode(d.ktssosegt,
            1,
            'SORT',
            2,
            'HASH',
            3,
            'DATA',
            4,
            'INDEX',
            5,
            'LOB_DATA',
            6,
            'LOB_INDEX',
            NULL) AS segment_type,
      b.tablespace_name,
      b.file_name
```

```
  FROM v$session_wait a, dba_data_files b, v$session c, x$ktsso d
 WHERE c.saddr = d.ktssoses(+)
   AND c.serial# = d.ktssosno(+)
   AND d.inst_id(+) = userenv('instance')
   AND a.sid = c.sid
   AND a.p1 = b.file_id
   AND a.event = 'direct path read'
UNION ALL
SELECT a.event,
       a.sid,
       d.sql_hash_value hash_value,
       decode(e.ktssosegt,
              1,
              'SORT',
              2,
              'HASH',
              3,
              'DATA',
              4,
              'INDEX',
              5,
              'LOB_DATA',
              6,
              'LOB_INDEX',
              NULL) AS segment_type,
       b.tablespace_name,
       b.file_name
  FROM v$session_wait a,
       dba_temp_files b,
       v$parameter    c,
       v$session      d,
       x$ktsso        e
 WHERE d.saddr = e.ktssoses(+)
   AND d.serial# = e.ktssosno(+)
   AND e.inst_id(+) = userenv('instance')
   AND a.sid = d.sid
   AND b.file_id = a.p1 - c.VALUE
   AND c.NAME = 'db_files'
   AND a.event = 'direct path read';
```

注：如果是从临时文件中读取排序段的会话，则表明 SORT_AREA_SIZE 或 PGA_AGGREGATE_TARGET 的设置是不是偏小。如果是从临时文件中读取 HASH 段的会话，则表明 HASH_AREA_SIZE 或 PAG_AGGREGATE_TARGET 的设置是不是偏小。

当 direct path read 等待事件是由于并行查询造成的(读取的是一般的数据文件而非临时文件)，父 SQL 语句的 HASHVALUE 与子 SQL 语句的 HASHVALUE 不同，

可以通过以下 SQL 查询产生子 SQL 语句的父 SQL 语句：

```
SELECT DECODE(A.QCSERIAL#, NULL, 'PARENT', 'CHILD') STMT_LEVEL,
      A.SID,
      A.SERIAL#,
      B.USERNAME,
      B.OSUSER,
      B.SQL_HASH_VALUE,
      B.SQL_ADDRESS,
      A.DEGREE,
      A.REQ_DEGREE
  FROM V$PX_SESSION A, V$SESSION B
 WHERE A.SID = B.SID
 ORDER BY A.QCSID, STMT_LEVEL DESC;
```

尽量减少 I/O 请求的次数，初始化参数 db_file_direct_io_count 用来设置直接读出和写入操作设置最大的 IO 缓冲区大小，因此能影响 direct path read 的性能，通过设置初始化参数 DB_FILE_DIRECT_IO_COUNT，使得满足 DB_BLOCK_SIZE x DB_FILE_DIRECT_IO_COUNT = max_io_size of system，在 Oracle8i 中默认这个值为 64 个 BLOCK；在 Oracle9i 中可以设置隐含参数_DB_FILE_DIRECT_IO_COUNT，参数的值也变成了 BYTES 而不是 BLOCK 数量了，默认值也变成了 1M。

- 使用 10046 第 8 层跟踪直接读取操作的 ORACLE 会话，其中 P3 参数表明块读取的数量。
- 也可使用 strace，truss 追踪直接读取或直接写入操作的 UNIX 进程，从生成的 TRACE 文件可获得相应的直接 IO 大小。
- 在第 1 层使用追踪事件 10357，启动执行直接 IO 操作的会话的调试信息。

注：

>> 1. 如果是 Temp 文件，则表示该会话正在读取它先前用 direct path write 操作所创建的临时段，查明使用的是什么类型的临时段，有助于了解会话所做的事情。

```
SELECT DISTINCT decode(t.ktssosegt,
                    1,'SORT',
                    2,'HASH',
                    3,'DATA',
                    4,'INDEX',
```

```
                    5,'LOB_DATA',
                    6,'LOB_INDEX',
                    'UNDEFINED')
  FROM sys.x$ktsso t
 WHERE t.inst_id = userenv('instance') AND
       t.kssoses = <当前 session 地址> AND
       t.ktssosno =
```

>> 2．如果是数据文件，则可能是并行查询从属操作在工作，通过 P1 值确定数据文件的名称：

```
select s.NAME from v$datafile s where s.FILE# =
union all
select a.name
from v$tempfile a, v$parameter b
where b.NAME = 'db_files'
     and a.FILE# + b.VALUE =
```

1.6.5.1 串行全表扫描(Serial Table Scan)--Oracle 11g 全表扫描以 Direct Path Read 方式执行

在 Oracle 11g 之前，全表扫描使用 db file scattered read 的方式，将表中的数据块离散的读到 Buffer Cache 之后，供用户访问和使用，但是如果全表访问的表非常大，则有可能占用大量的 Buffer Cache 内存，这会导致 Buffer Cache 中其他数据被老化和挤出内存，而且在这一系列的读取操作中，Oracle 引擎需要去判断每一个数据块是否已经存在于内存中，然后还要去请求内存空间，不断使用 Cache Buffer Chain 和 Cache Buffer Lru Chain 两个 Latch 进行判断，在某种程度上会加剧 Latch 竞争，如果全表访问的数据只是偶尔个别的访问，则占据大量 Buffer Cache 就显得过于昂贵，在 Oracle Database 11g 中，一种被称为串行全表扫描(Serial Table Scan)的技术被引入，该特性根据数据块的设置和统计信息等，自动决定是采用 Direct Path Read 绕过 SGA，还是采用常规方式读取，因为这种自动选择，这一特性又被称为**自适应直接读(Adaptive Direct Read)**.这种方式的好处是可以降低 Buffer Cache 的竞争，但是每次都要发生物理读，若是有多个会话同时去扫描同一张大表，这样会增大 IO，也有可能导致系统的问题，而且在读取之前可能需要触发检查点，避免读到旧的映像。

在 Oracle Database 11g 中有一个新特性，全表扫描可以通过直接路径读的方式来执行（Direct Path Read），这是一个合理的变化，如果全表扫描的大量数据读取是偶发性的，则直接路径读可以避免大量数据对于 Buffer Cache 的冲击。

当然对于小表来说，Oracle 允许通过 Buffer Cache 来进行全表扫描，因为这可能更快，也对性能影响不大。小表受到隐含参数：_small_table_threshold 影响。如果表大于 5 倍的小表限制，则自动会使用 DPR 替代 FTS。可以设置初始化参数： _serial_direct_read 来禁用串行直接路径读。

当然，Oracle 通过一个内部的限制，来决定执行 DPR 的阈值。

可以通过设置 10949 事件屏蔽这个特性，返回到 Oracle 11g 之前的模式上：

```
altersession setevents '10949 trace name context forever, level 1';
```

还有一个参数 _very_large_object_threshold 用于设定（MB 单位）使用 DPR 方式的上限，这个参数需要结合 10949 事件共同发挥作用。10949 事件设置任何一个级别都将禁用 DPR 的方式执行 FTS，但是仅限于小于 5 倍 BUFFER Cache 的数据表，同时，如果一个表的大小大于 0.8 倍的 _very_large_object_threshold 设置，也会执行 DPR。

这些限定的目标在于：

对于大表的全表扫描，必须通过 Direct Path Read 方式执行，以减少对于 Buffer Cache 的冲击和性能影响。但是我们可以通过参数调整来决定执行 DPR 的上限和下限。

Oracle 通过隐含参数_small_table_threshold 来界定大表小表的临界，Oracle 认为对于大表执行直接路径读取的意义比较大，对于小表通过将其缓存可能受益更大。_small_table_threshold 的单位为 block。默认为 db cache size 的 2%大小，在实例启动过程中动态决定。11GR2 之前，表的大小要是 _small_table_threshold 参数值的 5 倍才会采取直接路径读取方式，11GR2 后只需要满足_small_table_threshold 定义的大小就会采取直接路径读取。

以下的 AWR 信息是典型的 DPR 症状：

Top SQL with Top Events

SQL ID	Planhash	Sampled # of Executions	% Activity	Event	% Event	Top Row Source	% RwSrc	SQL Text
bg0v160a87xaa	1574385589	68	7.85	direct path read	7.15	TABLE ACCESS - FULL	7.15	SELECT BRANDNAME, Id, sum(NUM)...
9d7g86gt5039a	1574385589	67	7.35	direct path read	7.25	TABLE ACCESS - FULL	7.25	SELECT BRANDNAME, Id, sum(NUM)...
1k3j36p8ttg16	1574385589	62	7.25	direct path read	7.05	TABLE ACCESS - FULL	7.05	SELECT BRANDNAME, Id, sum(NUM)...
0r1jzpztu7wf5	1574385589	66	6.85	direct path read	6.65	TABLE ACCESS - FULL	6.65	SELECT BRANDNAME, Id, sum(NUM)...
3xb979ygsbrwp	3948063362	66	6.85	direct path read	6.65	TABLE ACCESS - FULL	6.65	SELECT BRANDNAME, Id, sum(NUM)...

Back to Top SQL
Back to Top

Top SQL with Top Row Sources

SQL ID	PlanHash	Sampled # of Executions	% Activity	Row Source	% RwSrc	Top Event	% Event	SQL Text
bg0v160a87xaa	1574385589	68	7.85	TABLE ACCESS - FULL	7.85	direct path read	7.15	SELECT BRANDNAME, Id, sum(NUM)...
9d7g86gt5039a	1574385589	67	7.35	TABLE ACCESS - FULL	7.25	direct path read	7.25	SELECT BRANDNAME, Id, sum(NUM)...
1k3j36p8ttg16	1574385589	62	7.25	TABLE ACCESS - FULL	7.15	direct path read	7.05	SELECT BRANDNAME, Id, sum(NUM)...
0r1jzpztu7wf5	1574385589	66	6.85	TABLE ACCESS - FULL	6.85	direct path read	6.65	SELECT BRANDNAME, Id, sum(NUM)...
3xb979ygsbrwp	3948063362	66	6.85	TABLE ACCESS - FULL	6.85	direct path read	6.65	SELECT BRANDNAME, Id, sum(NUM)...

在 11g 中，全表扫描可能使用 direct path read 方式，绕过 buffer cache，这样的全表扫描就是物理读了。 在 10g 中，都是通过 gc buffer 来读的，所以不存在 direct path read 的问题。

一个隐含参数_serial_direct_read 可以禁用串行直接路径读，11g 默认值为 auto：

禁用 direct path read： _serial_direct_read = false

启用 direct path read： _serial_direct_read = true

alter system set "_serial_direct_read"=never scope=both sid='*'; 可以显着减少 direct path read

调整数据库参数 alter system setevent='10949 trace name context forever, level 1'来关闭“direct path read”（直接路径读）特性,使 SQL 语句可以从缓存中查询数据，达到降低 I/O 读取量，使全表扫描的数据从缓存中读取，加快 SQL 语句运行速度的目的。

一、 Oracle 11g 新特性触发 Direct Path Read 等待事件案例

数据库环境：

Oracle 11.2.0.3 单实例，操作系统是 Windows Server 2008 R2。

故障现象：

数据库访问缓慢，I/O 使用率达到 100%

故障分析：

1. DB Time 高，数据库压力大。将近 60 分钟的采样时间内 DB Time 高达 6983.24。

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	18143	20-Aug-15 13:00:59	141	1.9
End Snap:	18145	20-Aug-15 14:00:27	242	1.9
Elapsed:		59.47 (mins)		
DB Time:		6,983.24 (mins)		

2. 物理读（Physical read）和逻辑读(Logical read)的数量级相同。看来这么大的物理读就是 I/O 达到 100%的原因。

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	117.4	119.5	2.50	0.86
DB CPU(s):	2.5	2.6	0.05	0.02
Redo size:	18,920.3	19,254.6		
Logical reads:	142,621.1	145,140.7		
Block changes:	80.3	81.7		
Physical reads:	141,364.2	143,861.5		
Physical writes:	1,441.3	1,466.8		
User calls:	137.1	139.6		
Parses:	6.8	6.9		
Hard parses:	3.3	3.4		
W/A MB processed:	15.6	15.9		
Logons:	0.1	0.1		
Executes:	47.0	47.8		
Rollbacks:	0.1	0.1		
Transactions:	1.0			

3. SGA 区的 Buffer Nowait 100%，看起来和大量的物理读有些矛盾。

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	99.88	In-memory Sort %:	100.00
Library Hit %:	82.58	Soft Parse %:	51.66
Execute to Parse %:	85.50	Latch Hit %:	99.94
Parse CPU to Parse Elapsd %:	78.42	% Non-Parse CPU:	99.32

4. 前台等待事件排在第一位的是直接路径读 direct path read，占据整个 DB Time 的 85.97%。直接路径读的特点是不经过 SGA 的缓冲区，直接从存储获取数

据。

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
direct path read	3,624,418	360,211	99	85.97	User I/O
direct path read temp	165,553	28,926	175	6.90	User I/O
direct path write temp	75,922	14,517	191	3.46	User I/O
DB CPU		8,979		2.14	
Disk file operations I/O	444	3,034	6834	0.72	User I/O

5. 从 TOP SQL 上可以看到最耗时的 sql 语句都是在等待 I/O，并且这些 I/O 来自同一张大表 CX_BAS_CUS_CON_SUMUP。系统产生的逻辑读、物理读、直接路径读都来自于这张大表。问题找到了！通过执行计划看到了访问这张大表的 sql 执行计划是全表扫，该表大小为 488M。

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 9.2% of Total DB Time (s): 418,994
- Captured PL/SQL account for 0.0% of Total DB Time (s): 418,994

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
2,905.87	97	29.96	0.69	2.57	97.24	7bmcz3k7gskdz		select cuscall0_.CONTACT_ID as...
1,721.92	5	344.38	0.41	2.70	96.68	0fq1rmh5u6fp2		select * from (select cuscall...
1,598.99	2	799.49	0.38	1.32	98.19	3ajbwnu0xtk6j		select * from (select cuscall...
1,592.97	2	796.49	0.38	1.31	97.86	7pwqr1qf3pq4r		select * from (select cuscall...
1,568.68	2	784.34	0.37	1.34	98.05	6hdh1cah0s5d3		select * from (select cuscall...
1,559.57	2	779.78	0.37	1.27	98.49	1uh2v0quzwfyr		select * from (select cuscall...
1,472.13	2	736.06	0.35	1.55	97.85	97gnz91t0dat8		select * from (select cuscall...
1,437.29	2	718.65	0.34	1.51	97.75	bw69vdsww7hjq		select * from (select cuscall...
1,397.02	2	698.51	0.33	1.62	96.42	d80w04shcupbg		select * from (select cuscall...
1,365.47	2	682.73	0.33	1.79	97.42	3kfdm9sgr4wwt		select * from (select cuscall...

```
Plan hash value: 2063375816

-------------------------------------------------------------------------------
---------------------------------------
| Id  | Operation                       | Name                       | Rows  | B
ytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
---------------------------------------
|   0 | SELECT STATEMENT                |                            |       |
     |       |   159K(100)|          |
|   1 |  COUNT STOPKEY                  |                            |       |
     |       |            |          |
|   2 |   VIEW                          |                            | 45360 |
 343M|       |   159K  (1)| 00:31:53 |
|   3 |    SORT ORDER BY STOPKEY        |                            | 45360 |
  29M|   547M|   159K  (1)| 00:31:53 |
|   4 |     FILTER                      |                            |       |
     |       |            |          |
|   5 |      HASH JOIN                  |                            |   770K|
 504M|   166M| 47885   (1)| 00:09:35 |
|   6 |       TABLE ACCESS FULL         | CX_BAS_VOICE_SEQ           |   809K|
 157M|       |  6638   (1)| 00:01:20 |
|   7 |       TABLE ACCESS FULL         | CX_BAS_CUS_CON_SUMUP       |   770K|
 354M|       | 14939   (1)| 00:03:00 |
|   8 |      TABLE ACCESS BY INDEX ROWID| CX_BAS_VOICE_SEQ           |     1 |
  42 |       |     5   (0)| 00:00:01 |
|   9 |       INDEX RANGE SCAN          | IDX_CXBASVOICESEQ_CALLER   |     2 |
     |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Segments by Logical Reads

- Total Logical Reads: 508,863,207
- Captured Segments account for 100.2% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Logical Reads	%Total
CSRBANK	CC_DATA	CX_BAS_CUS_CON_SUMUP		TABLE	487,919,408	95.88
CSRBANK	USERS	CX_BAS_VOICE_SEQ		TABLE	13,413,744	2.64
CSRBANK	CSR_BANK	CX_BAS_SETREADY_LOG		TABLE	3,648,784	0.72
CSRBANK	USERS	CX_BAS_SETBUSY_LOG		TABLE	1,252,960	0.25
CSRBANK	CC_DATA	CX_BAS_MOBILENOTRACK		TABLE	987,424	0.19

Back to Segment Statistics
Back to Top

Segments by Physical Reads

- Total Physical Reads: 504,378,447
- Captured Segments account for 99.4% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Physical Reads	%Total
CSRBANK	CC_DATA	CX_BAS_CUS_CON_SUMUP		TABLE	487,882,408	96.73
CSRBANK	USERS	CX_BAS_VOICE_SEQ		TABLE	13,352,860	2.65
SYS	SYSTEM	AUD$		TABLE	81,450	0.02
CSRBANK	USERS	CX_BAS_CONTACT_HIST		TABLE	56,939	0.01
CSRBANK	CC_DATA	PK_CX_BAS_CUS_CON_SUMUP_ID		INDEX	6,419	0.00

Segments by Direct Physical Reads

- Total Direct Physical Reads: 504,367,486
- Captured Segments account for 99.4% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Direct Reads	%Total
CSRBANK	CC_DATA	CX_BAS_CUS_CON_SUMUP		TABLE	487,932,210	96.74
CSRBANK	USERS	CX_BAS_VOICE_SEQ		TABLE	13,352,618	2.65
SYS	SYSTEM	AUD$		TABLE	81,450	0.02
CSRBANK	USERS	CX_BAS_CONTACT_HIST		TABLE	56,914	0.01
SYS	SYSAUX	SYS_LOB0000006331C00004$$		LOB	50	0.00

最终结论：

在 Oracle 11g 中有一个新特性，为了保护已经缓存在 buffer cache 的数据，当出现全表扫的查询时会判断该表的大小。如果该表过大，则使用直接路径读(Direct Path Read) 来获取数据。避免大量冷数据对 Buffer Cache 的冲击。此次问题的原因就是因为这个新特性。大量的并发查询 CX_BAS_CUS_CON_SUMUP，并且执行计划都是采用了全表扫，满足了 11g 的这个新特性，通过直接路径读的方式绕过 SGA 从存储上获取数据。由于没有 SGA 的缓存，每一次查询都需要从存储读取产生了大量的物理读，最终导致 I/O 100%。由于处理速度慢，CPU 又产生了大量的等待队列，所以 DB Time 也非常高。

新特性中如何判断全表扫的大小呢？

下面看一个隐含参数：_small_table_threshold

该参数默认为 Buffer Cache 的 2%，如果表大于 5 倍_small_table_threshold 就触发该特性。

可以通过设置 10949 事件屏蔽这个特性

```
alter session set events '10949 trace name context forever, level 1';
```

解决方案：

应用团队确认了该表的数据，删除了大量的历史数据，使得全表扫后远低于_small_table_threshold x 5 后的数值。再次执行该 sql 语句就可以缓存在 buffer cache 中了，物理读和 I/O 负载全部恢复到合理的范围。

由于不让修改应用程序，我们无法优化该 SQL。所以该问题没有从根本上解决。当数据量增大到阈值，问题会卷土重来。

1.6.6 direct path write (直接路径写、DRW)

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME = 'direct path write';
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
200	885859547	direct path write	file number	first dba	block cnt	1740759767	8	User I/O

这个等待事件有三个参数：

file number：要写入的绝对文件号 file number，可发现所进行的操作性质(如：排序/并行操作)

first dba：起始块号 first dba

block cnt：块数 block cnt，可发现直接写入 IO 的大小

由参数 P1 与 P2 推得访问的数据对象：

```
select s.segment_name, s.partition_name
  from dba_extents s
 where  between s.block_id and (s.block_id + s.blocks -1) and s.file_id =
```

这个等待事件和 direct path read 正好相反，**是会话将一些数据从 PGA 中直接写入到磁盘文件(数据文件或临时文件)上，而不经过 SGA**。这类读取通常在以下情况被使用：

1. **直接路径加载**（使用 append 方式加载数据、CREATE TABLE AS SELECT）
2. **并行 DML 操作**
3. **磁盘排序使用临时表空间排序**（内存不足）

最常见的直接路径写，多数因为磁盘排序导致。对于这一写入等待，应该找到 I/O 操作最为频繁的数据文件（如果有过多的排序操作，很有可能就是临时文件），分散负载，加快其写入操作。

如果系统存在过多的磁盘排序，会导致临时表空间操作频繁，对于这种情况，可以考虑为不同用户分配不同的临时表空间，使用多个临时文件，写入不同磁盘或者裸设备，从而降低竞争提高性能。

direct path write 是允许一个会话让一个 I/O 写请求入队列的同时处理操作系统的 I/O。如果会话想确认明显的写是否已经完成就会出现这个等待事件。因为会话需要空的缓存和空的槽位(等待之前的 I/O 释放)，或者是会话需要确认所有的写操作都已经完成。如果没有使用异步 I/O，I/O 请求会被阻塞直到之前的 I/O 请求完成后，但是此时不会出现 I/O 等待，会话稍后重新恢复并加速 I/O 请求的完成，此时就会出现 direct path write 等待。因此，对于这个等待事件容易产生两方面的误解：一是认为等待的总的数量不能反映出 I/O 请求的数量，二是消耗在这个等待事件上的总的时间不能反映出实际的等待时间。这类型的写请求主要是用于直接装载数据的操作(create table as select)、并行的 DML 操作、不在内存中排序的 I/O 以及写入没有 cache 的 LOB 段操作。

关于该等待事件，以下的几点需要注意：

1. 从 PGA 写入数据文件，一个会话可以发布多个写入请求和连续的处理。
2. 直接写入可以按同步或异步方式执行，取决于平台和 DISK_ASYNC_IO 参数的值。
3. 通常用于在数据加载（APPEND 提示、CTAS－CREATE TABLE AS SELECT）、并行 DML 操作时写入到临时段。
4. 在使用异步 IO 时，direct path write 事件产生的等待时间不准确，所以通过 v$sesstat 视图来获得直接写入次数来评估该事件的影响情况：

```
SELECT A.NAME,
       B.SID,
       B.VALUE,
       ROUND((SYSDATE - C.LOGON_TIME) * 24) HOURS_CONNECTED
  FROM V$STATNAME A, V$SESSTAT B, V$SESSION C
 WHERE A.STATISTIC# = B.STATISTIC#
   AND B.SID = C.SID
   AND B.VALUE > 0
   AND A.NAME = 'PHYSICAL WRITES DIRECT';
```

这个等待事件的等待时间是指等待 BLOCK 直到明显的 I/O 请求完成的时间。通常来说，如果不是存在特殊的 JOB,一般是不会出现这个等待事件，如果在等待事件中这个等待事件占的比重比较大，可以从如下几个方面来调整：

如果是等待的文件是临时表空间的文件，那么需要查看是否存在大量不合理的磁盘排序，优化相应的存在问题的 SQL 语句。如果是 Oracle9i 可以考虑使用自动 SQL 执行内存管理，Oracle8i 的话可以手工的调整各种排序区。

确认异步 I/O 是否配置正确，异步 I/O 不会减少这个等待事件的等待时间但是却可以减少会话所消耗的时间。

检查是否存在 I/O 消耗很严重的 SQL 语句，如果存在，尝试优化 SQL 语句减少 I/O 的消耗。

最后确认一下是否达到了磁盘的 I/O 极限，如果是，则需要考虑更换更好的硬件设备。

大量的 direct path read 等待事件最可能是一个应用程序的问题。

注：

>> 1．如果是 Temp 文件，则表示该会话正在写入临时表空间，查明使用临时段的类型，有助于了解会话所做的事情。

```
SELECT DISTINCT decode(t.ktssosegt,
                      1,'SORT',
                      2,'HASH',
                      3,'DATA',
                      4,'INDEX',
                      5,'LOB_DATA',
                      6,'LOB_INDEX',
                      'UNDEFINED')
  FROM sys.x$ktsso t
 WHERE t.inst_id = userenv('instance') AND
       t.kssoses = <当前 session 地址> AND
       t.ktssosno =
```

>> 2．如果是数据文件，则可能正在执行一项直接路径加载操作，通过 P1 值确定数据文件的名称：

```
select s.NAME from v$datafile s where s.FILE# =
union all
select a.name
from v$tempfile a, v$parameter b
where b.NAME = 'db_files'
     and a.FILE# + b.VALUE =
```

1.6.6.1 direct path read temp、direct path write temp

为了排序工作在临时区域读写时，等待 direct path read temp、direct path write temp 事件。oracle 9i 为止是通过 direct path read、direct path write 等待观察的。在 Oracle 10g/11g 中，为了区分特定的对于临时文件的直接读写操作，Oracle 对 direct path read/write 进行了分离，将这类操作分列出来：

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('direct path read temp', 'direct path write temp');
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	198	861319509	direct path read temp	file number	first dba	block cnt	1740759767	8	User I/O
2	200	38438084	direct path write temp	file number	first dba	block cnt	1740759767	8	User I/O

可以看到，现在的 direct path read/write temp 就是单指对于临时文件的直接读写操作。排序段上的 direct path I/O 是在需要排序的数据比排序所分配的 PGA 内存区大时发生的。因此在排序工作时若大量发生 direct path read temp、direct path write temp 等待，就可以通过追加分配内存区域而避免等待。

1、应用程序层

检查需要排序的 sql 语句是否已经最优化。不必要的排序操作会导致 CPU 浪费、PGA 区域浪费、磁盘 I/O 浪费。从 UNION 和 UNION ALL 的性能差异上可以得知，只靠减少不必要的排序操作，也能解决许多问题。

2、oracle 内存层

在进程上分配的工作区大小内一次性实现的排序称为 One pass sort。与此相反的情况称为 Multi pass sort。发生 Multi pass sort 时，排序工作过程中将排序结果读写到排序段（sort segment）区域，因此发生 direct path read temp、direct path write temp 等待。如果该等待大量发生，就可以适当提高 pga_aggregate_target 值，以此消除问题。

oracle 在调优指南上推荐如下设定 pga_aggregate_target 值。

OLTP：pga_aggregate_target=(total_mem * 80%) * 20%

OLAP：pga_aggregate_target=(total_mem * 80%) * 50%

上述的意思是，假设 OS 本身使用 20%左右的内存，OLTP 系统上使用剩余内存的 20%左右，OLAP 系统因为排序工作较多，所以使用剩余内存的 50%左右。

结合 Oracle 10g 的一些特性，来进一步研究一下直接路径读/写与临时文件。

首先在一个 session 中执行一个能够引发磁盘排序的查询：

```
tq@CCDB> select sid from v$mystat where rownum <2;
       SID
----------
      1066
tq@CCDB> select a.table_name,b.object_name,b.object_type
  2  from t1 a,t2 b
  3  where a.table_name = b.object_name
  4  order by b.object_name,b.object_type;
```

在另外 sessoin 查询相应等待事件：

```
tq@CCDB> select event,p1text,p1,p2text,p2,p3text,p3
  2  from v$session_wait_history
  3  where sid = 1066;
EVENT                     P1TEXT             P1 P2TEXT            P2 P3TEXT            P3
------------------------- ------------- ------- ------------ ------- ------------ -------
direct path read temp     file number       201 first dba     313512 block cnt         31
direct path read temp     file number       201 first dba     313481 block cnt         31
direct path read temp     file number       201 first dba     386887 block cnt         31
direct path read temp     file number       201 first dba     317736 block cnt         31
direct path read temp     file number       201 first dba     317193 block cnt         31
direct path read temp     file number       201 first dba     316646 block cnt         31
direct path read temp     file number       201 first dba     316134 block cnt         31
direct path read temp     file number       201 first dba     315622 block cnt         31
direct path read temp     file number       201 first dba     315079 block cnt         31
direct path read temp     file number       201 first dba     314567 block cnt         31
10 rows selected.
```

从以上输出可以看到最近 10 次等待，direct path read temp 就是这个查询引起的磁盘排序。注意这里的 file number 为 201。而实际上，通过 v$tempfile 来查询，临时文件的文件号仅为 1：

```
tq@CCDB> select file#,name from v$tempfile;
     FILE# NAME
---------- ----------------------------------------
         1 /oracle/oradata/ccdb/ccdb/temp01.dbf
```

如果通过 10046 事件跟踪，也可以获得类似的结果：

```
WAIT #3: nam='direct path write temp' ela= 1 file number=201 first dba=437862 block cnt=31 obj#=112141 tim=1270780
330976998
WAIT #3: nam='direct path write temp' ela= 1 file number=201 first dba=437416 block cnt=31 obj#=112141 tim=1270780
330977070
WAIT #3: nam='direct path read temp' ela= 7 file number=201 first dba=438471 block cnt=31 obj#=112141 tim=12707803
30982214
WAIT #3: nam='direct path read temp' ela= 4 file number=201 first dba=438502 block cnt=31 obj#=112141 tim=12707803
30983765
WAIT #3: nam='direct path read temp' ela= 8 file number=201 first dba=387015 block cnt=31 obj#=112141 tim=12707803
30993872
```

在 Oracle 文档中，file#被定义为绝对文件号（The Absolute File Number）。这里的原因何在呢？研究这个问题要先研究一下 v$tempseg_usage 这个视图，可以从这个视图出发动手研究一下这个对象究竟来自何方。

查询 dba_objects 视图，发现 v$tempseg_usage 原来是一个同义词。

```
sys@CCDB> select object_type from dba_objects where object_name = 'V$TEMPSEG_USAGE';
OBJECT_TYPE
-------------------
SYNONYM
```

再追本溯源原来 v$tempseg_usage 是 v_$sort_usage 的同义词，也就是和 v$sort_usage 同源。从 Oracle 9i 开始，Oracle 将 v$sort_usage 视图从文档中移除了，因为这个名称有所歧义，容易使人误解仅记录排序内容，所以 v$tempseg_usage 视图被引入，用于记录临时段的使用情况：

```
sys@CCDB> select * from dba_synonyms where synonym_name = 'V$TEMPSEG_USAGE';
OWNER      SYNONYM_NAME         TABLE_OWNER     TABLE_NAME      DB_LINK
---------- -------------------- --------------- --------------- ----------
PUBLIC     V$TEMPSEG_USAGE      SYS             V_$SORT_USAGE
```

如果再进一步，可以看到这个视图的构建语句：

```
sys@CCDB> select view_definition from v$fixed_view_definition
  2  where view_name = 'GV$SORT_USAGE';
VIEW_DEFINITION
--------------------------------------------------------------------------------
select x$ktsso.inst_id, username, username, ktssoses, ktssosno, prev_sql_addr, p
rev_hash_value, prev_sql_id, ktssotsn, decode(ktssocnt, 0, 'PERMANENT', 1, 'TEMP
ORARY'), decode(ktssosegt, 1, 'SORT', 2, 'HASH', 3, 'DATA', 4, 'INDEX', 5, 'LOB_
DATA', 6, 'LOB_INDEX' , 'UNDEFINED'), ktssofno, ktssobno, ktssoexts, ktssoblks,
ktssorfno from x$ktsso, v$session where ktssoses = v$session.saddr and ktssosno
= v$session.serial#
```

格式化一下，v$sort_usage 的创建语句如下：

```
SELECT   x$ktsso.inst_id,username,username,ktssoses,ktssosno,
         prev_sql_addr,prev_hash_value,prev_sql_id,ktssotsn,
         DECODE (ktssocnt,
                 0,'PERMANENT',
                 1,'TEMPORARY'),
         DECODE (ktssosegt,
                 1, 'SORT',
                 2, 'HASH',
                 3, 'DATA',
                 4, 'INDEX',
                 5, 'LOB_DATA',
                 6, 'LOB_INDEX',
                 'UNDEFINED'),
         ktssofno,ktssobno,
         ktssoexts,ktssoblks,ktssorfno
  FROM   x$ktsso, v$session
WHERE   ktssoses = v$session.saddr AND ktssosno = v$session.serial#;
```

注意到在 Oracle 文档中对 v$sort_usage 字段 SEGFILE#的定义为：

SEGFILE#   NUMBER   File number of initial extent

在视图中，这个字段来自 x$ktsso.ktssofno，也就是说这个字段实际上代表的是绝对文件号。那么这个绝对文件号如何与临时文件关联呢？能否与 v$tempfile 中的 file#字段关联呢？

再来看一下 v$tempfile 的来源，v$tempfile 由如下语句创建：

```
sys@CCDB> select view_definition from v$fixed_view_definition
  2  where view_name = 'GV$TEMPFILE';
VIEW_DEFINITION
--------------------------------------------------------------------------------
select tf.inst_id, tf.tfnum, to_number(tf.tfcrc_scn), to_date(tf.tfcrc_tim,'MM/D
D/RR HH24:MI:SS','NLS_CALENDAR=Gregorian'), tf.tftsn, tf.tfrfn, decode(bitand(tf
.tfsta, 2),0,'OFFLINE',2,'ONLINE','UNKNOWN'), decode(bitand(tf.tfsta, 12), 0,'DI
```

```
SABLED',4, 'READ ONLY', 12, 'READ WRITE',
                         'UNKNOWN'), fh.fhtmpfsz*tf.tfbsz, fh.fhtmpfsz,  tf.tf
csz*tf.tfbsz,tf.tfbsz, fn.fnnam  from x$kcctf tf, x$kccfn fn, x$kcvfhtmp fh  whe
re fn.fnfno=tf.tfnum and fn.fnfno=fh.htmpxfil and tf.tffnh=fn.fnnum  and tf.tfdu
p!=0 and bitand(tf.tfsta, 32) <> 32  and fn.fntyp=7 and fn.fnnam is not null
```

格式化 v$tempfile 如下：

```
SELECT   tf.inst_id,tf.tfnum,TO_NUMBER (tf.tfcrc_scn),
         TO_DATE (tf.tfcrc_tim,'MM/DD/RR HH24:MI:SS','NLS_CALENDAR=Gregorian'),
         tf.tftsn,tf.tfrfn,
         DECODE (BITAND (tf.tfsta, 2), 0, 'OFFLINE', 2, 'ONLINE', 'UNKNOWN'),
         DECODE (BITAND (tf.tfsta, 12),
                 0, 'DISABLED',
                 4, 'READ ONLY',
                 12, 'READ WRITE',
                 'UNKNOWN'),
         fh.fhtmpfsz * tf.tfbsz,fh.fhtmpfsz,tf.tfcsz * tf.tfbsz,tf.tfbsz,fn.fnnam
  FROM   x$kcctf tf, x$kccfn fn, x$kcvfhtmp fh
WHERE       fn.fnfno = tf.tfnum
         AND fn.fnfno = fh.htmpxfil
         AND tf.tffnh = fn.fnnum
         AND tf.tfdup != 0
         AND BITAND (tf.tfsta, 32) <> 32
         AND fn.fntyp = 7
         AND fn.fnnam IS NOT NULL;
```

考察 x$kcctf 底层表，注意到 TFAFN（Temp File Absolute File Number）在这里存在：

```
sys@CCDB> desc x$kcctf
Name             Null?    Type
---------------- -------- ----------------
ADDR                      RAW(8)
INDX                      NUMBER
INST_ID                   NUMBER
TFNUM                     NUMBER
TFAFN                     NUMBER
TFCSZ                     NUMBER
TFBSZ                     NUMBER
TFSTA                     NUMBER
TFCRC_SCN                 VARCHAR2(16)
TFCRC_TIM                 VARCHAR2(20)
TFFNH                     NUMBER
TFFNT                     NUMBER
TFDUP                     NUMBER
TFTSN                     NUMBER
TFTSI                     NUMBER
TFRFN                     NUMBER
TFPFT                     NUMBER
TFMSZ                     NUMBER
TFNSZ                     NUMBER
```

而这个字段在构建 v$tempfile 时并未出现，所以不能通过 v$sort_usage 和 v$tempfile 直接关联绝对文件号。可以简单构建一个排序段使用，然后来继续研究一下：

```
sys@CCDB> select username,segtype,segfile#,segblk#,extents,segrfno# from v$sort_usage;
USERNAME     SEGTYPE     SEGFILE#    SEGBLK#    EXTENTS   SEGRFNO#
------------ --------- ---------- ---------- ---------- ----------
SYS          LOB_DATA         201     340361          1          1
```

看到这里的 SEGFILE#=201，而在 v$tempfile 是找不到这个信息的：

```
sys@CCDB> select file#,rfile#,ts#,status,blocks from v$tempfile;
     FILE#     RFILE#        TS# STATUS      BLOCKS
---------- ---------- ---------- ------- ----------
         1          1          3 ONLINE      443520
```

但是可以从 x$kcctf 中获得这些信息，v$tempfile.file#实际上来自 x$kcctf.tfnum，是临时文件的文件号；而绝对文件号是 x$kcctf.tfafn，这个字段才可以与 v$sort_usage.segfile#关联：

```
sys@CCDB> select indx,tfnum,tfafn,tfcsz from x$kcctf;
      INDX      TFNUM      TFAFN      TFCSZ
---------- ---------- ---------- ----------
         0          1        201       2560
```

再进一步可以知道，实际上，为了分离临时文件号和数据文件号，Oracle 对临时文件的编号以 db_files 为起点，所以临时文件的绝对文件号应该等于 db_files+file#。

db_files 参数的缺省值为 200：

```
sys@CCDB> show parameter db_files
NAME          TYPE           VALUE
------------- -------------- -----------
db_files      integer        200
sys@CCDB> select file#,name from v$tempfile;
     FILE# NAME
---------- ----------------------------------------------
         1 /oracle/oradata/ccdb/ccdb/temp01.dbf
```

所以在 Oracle 文档中 v$tempfile.file#被定义为 The absolute file number 是不确切的。

1.6.7 read by other session

WAITEVENT: "read by other session" Reference Note (文档 ID 732891.1)

当多个进程访问同一个数据块，而此数据块不在内存中，这时会有一个进程将它从磁盘读到内存时，其它读取此数据块进程的状态就是 read by other session；因为 Oracle 内存不允许多个进程同时读到同一个数据块到内存，其它进程只能等待。

当我们查询一条数据时，Oracle 第一次会将数据从磁盘读入 buffer cache。如果有两个或者多个 session 请求相同的信息，那么第一个 session 会将这个信息读入 buffer cache，其他的 session 就会出现等待。

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('read by other session');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
97	3056446529	read by other session	file#	block#	class#	1740759767	8	User I/O

- P1 = **file#** Absolute File# (AFN) This is the file number of the data file that contains the block that the waiting session wants.
- P2 = **block#** This is the block number in the above file# that the waiting session wants access to. See Note:181306.1 to determine the tablespace, filename and object for this file#,block# pair.
- P3 = **class#** Block class
  This is the class of block being waited on. In particular:
  class 1 indicates a "data block", which could be table or index
  class 4 indicates a "segment header"
  class >=15 indicate undo blocks

我们可以根据 P1 和 P2 参数值获取到等待的对象名称和类型：

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, OWNER, TABLESPACE_NAME
```

```
FROM DBA_EXTENTS
WHERE FILE_ID = FILE#
AND BLOCK#
BETWEEN BLOCK_ID AND BLOCK_ID + BLOCKS - 1;
```

另外，其实我们也可以根据 v$session 的 ROW_WAIT_OBJ#列获取到等待的对象的名称，SQL 如下：

```
SELECT A.ROW_WAIT_OBJ#,
      B.OBJECT_NAME,
      A.SQL_ID,
      A.SID,
      A.BLOCKING_SESSION,
      A.EVENT,
      A.P1TEXT,
      A.P1,
      A.P2TEXT,
      A.P2,
      A.P3TEXT,
      A.P3,
      A.WAIT_CLASS
  FROM V$SESSION A, DBA_OBJECTS B
 WHERE A.ROW_WAIT_OBJ# = B.OBJECT_ID
   AND A.EVENT='read by other session';
```

其实 read by other session 是在 Oracle 10g （10.1.0.2 and later） 新引入的一个等待事件，在 10g 以前版本，等待为 buffer busy waits，10g 以后做的细分，所以才有了 read by other session。

Oracle 官方解释如下：

This event occurs when a session requests a buffer that is currently being read into the buffer cache by another session. Prior to release 10.1, waits for this event were grouped with the other reasons for waiting for buffers under the 'buffer busy wait' event。

此等待事件从侧面也说明了数据库存在读的竞争，所以该等待事件经常会和 db file sequential read 和 db file scattered read 同时出现。

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
db file sequential read	1,192,656	3,052	3	54.58	User I/O
read by other session	901,964	2,285	3	40.87	User I/O
DB CPU		258		4.62	
log file sync	421	5	12	0.09	Commit
db file scattered read	903	5	5	0.09	User I/O

下面是在 Metalink 上的解释：

Solution：

This wait event occurs when we are trying to access a buffer in the buffer cache but we find that the buffer is currently being read from disk by another user so we need to wait for that to complete before we can access it. In previous versions, this wait was classified under the "buffer busy waits" event. However, in Oracle 10.1 and higher, the wait time is now broken out into the "read by other session" wait event.

Excessive waits for this event are typically due to several processes repeatedly reading the same blocks, e.g. many sessions scanning the same index or performing full table scans on the same table. Tuning this issue is a matter of finding and eliminating this contention。

一般来说出现这种等待事件是因为多个进程重复的读取相同的 blocks，比如一些 session 扫描相同的 index 或者在相同的 block 上执行 full table scan。解决这个等待事件最好是找到并优化相关的 SQL 语句

1． 如果系统中有这种等待事件，我们可以通过以下 SQL 查询 v$session_wait 得到详细信息

```
SELECT p1 "file#", p2 "block#", p3 "class#"
FROM v$session_wait
WHERE event = 'read by other session';
```

2． 如果上述查询出的结果是热块造成的，运行如下 SQL，查询出具体对象信息，其实这部分可以直接从 AWR 的 Segments by Buffer Busy Waits 看出来。

```
SELECT RELATIVE_FNO, OWNER, SEGMENT_NAME, SEGMENT_TYPE
FROM DBA_EXTENTS
WHERE FILE_ID = &FILE
AND &BLOCK BETWEEN BLOCK_ID AND BLOCK_ID + BLOCKS - 1;
```

3． 可以通过下面的 SQL 脚本查询到具体的 SQL 语句

```
SELECT HASH_VALUE, SQL_TEXT
  FROM V$SQLTEXT
 WHERE (HASH_VALUE, ADDRESS) IN
      (SELECT A.HASH_VALUE, A.ADDRESS
        FROM V$SQLTEXT A,
             (SELECT DISTINCT A.OWNER, A.SEGMENT_NAME, A.SEGMENT_TYPE
               FROM DBA_EXTENTS A,
                    (SELECT DBARFIL, DBABLK
                      FROM (SELECT DBARFIL, DBABLK
                              FROM X$BH
                             ORDER BY TCH DESC)
                     WHERE ROWNUM < 11) B
              WHERE A.RELATIVE_FNO = B.DBARFIL
                AND A.BLOCK_ID <= B.DBABLK
                AND A.BLOCKS > B.DBABLK) B
       WHERE A.SQL_TEXT LIKE '%' || B.SEGMENT_NAME || '%'
         AND B.SEGMENT_TYPE = 'TABLE')
 ORDER BY HASH_VALUE, ADDRESS, PIECE;
```

4.查看对应 SQL 的执行计划是否最优，必要时可以通过 DBMS_SQLTUNE 包迚行优化，通过 SQL_PROFILE 文件稳固执行计划

5.查看表和索引的统计信息是否陈旧，必要时收集统计信息

1.6.7.1 read by other session 等待事件模拟

```
CREATE TABLE TB_RBOS_20160829_LHR AS SELECT * FROM DBA_OBJECTS;
INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
```

```
INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
COMMIT;

SELECT DISTINCT SID FROM V$MYSTAT;
DECLARE
   I       NUMBER := 0;
   V_STRING VARCHAR2(50) := 'alter system flush buffer_cache';
BEGIN
   LOOP
      SELECT COUNT(*) INTO I FROM TB_RBOS_20160829_LHR;
      EXECUTE IMMEDIATE V_STRING;
   END LOOP;
END;
/


SELECT A.SID,A.BLOCKING_SESSION,A.EVENT,A.P1TEXT,A.P1,A.P2TEXT,A.P2,A.P3TEXT,A.P3,A.WAIT_CLASS
  FROM V$SESSION  A
 WHERE SID IN (190, 5, 68);
```

首先，建表 TB_RBOS_20160829_LHR：

```
SYS@lhrdb> SELECT * FROM V$VERSION;

BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
PL/SQL Release 11.2.0.4.0 - Production
CORE    11.2.0.4.0      Production
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
NLSRTL Version 11.2.0.4.0 - Production

SYS@lhrdb> CREATE TABLE TB_RBOS_20160829_LHR AS SELECT * FROM DBA_OBJECTS;
Table created.
SYS@lhrdb> INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
```

```
87145 rows created.
SYS@lhrdb> INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
174290 rows created.
SYS@lhrdb> INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
348580 rows created.
SYS@lhrdb> INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
697160 rows created.
SYS@lhrdb> INSERT INTO TB_RBOS_20160829_LHR SELECT * FROM TB_RBOS_20160829_LHR;
1394320 rows created.
SYS@lhrdb> COMMIT;
Commit complete.
```

我们开 3 个 session 分别清空 buffer 同时对表 TB_RBOS_20160829_LHR 做统计操作：

session 1:

```
SYS@lhrdb> SELECT DISTINCT SID FROM V$MYSTAT;
      SID
----------
      190

SYS@lhrdb> DECLARE
  2     I       NUMBER := 0;
  3     V_STRING VARCHAR2(50) := 'alter system flush buffer_cache';
  4  BEGIN
  5     LOOP
  6        SELECT COUNT(*) INTO I FROM TB_RBOS_20160829_LHR;
  7        EXECUTE IMMEDIATE V_STRING;
  8     END LOOP;
  9  END;
 10  /
```

session 2:

```
SYS@lhrdb> SELECT DISTINCT SID FROM V$MYSTAT;
      SID
----------
        5

SYS@lhrdb> DECLARE
  2     I       NUMBER := 0;
  3     V_STRING VARCHAR2(50) := 'alter system flush buffer_cache';
  4  BEGIN
  5     LOOP
  6        SELECT COUNT(*) INTO I FROM TB_RBOS_20160829_LHR;
  7        EXECUTE IMMEDIATE V_STRING;
  8     END LOOP;
  9  END;
 10  /
```

session 3:

```
SYS@lhrdb> SELECT DISTINCT SID FROM V$MYSTAT;
       SID
----------
        68

SYS@lhrdb> DECLARE
  2     I       NUMBER := 0;
  3     V_STRING VARCHAR2(50) := 'alter system flush buffer_cache';
  4  BEGIN
  5     LOOP
  6        SELECT COUNT(*) INTO I FROM TB_RBOS_20160829_LHR;
  7        EXECUTE IMMEDIATE V_STRING;
  8     END LOOP;
  9  END;
 10  /
```

session 4:(监控)

```
SELECT A.SID,A.BLOCKING_SESSION,A.EVENT,A.P1TEXT,A.P1,A.P2TEXT,A.P2,A.P3TEXT,A.P3,A.WAIT_CLASS
  FROM V$SESSION  A
 WHERE SID IN (190, 5, 68);
```

	SID	BLOCKING_SESSION	EVENT	P1TEXT	P1	P2TEXT	P2	P3TEXT	P3	WAIT_CLASS
1	5		direct path read	file number	1	first dba	164736	block cnt	128	User I/O
2	68		direct path read	file number	1	first dba	162944	block cnt	128	User I/O
3	190		direct path read	file number	1	first dba	161664	block cnt	128	User I/O

由于是11g，满足DPR的特性，所以会走direct path read绕过SGA直接读取数据到PGA的，我们先禁用该特性：

```
SYS@lhrdb> alter system set "_serial_direct_read"=never scope=both sid='*';

System altered.
```

禁用之后继续查询：

	SID	BLOCKING_SESSION	EVENT	P1TEXT	P1	P2TEXT	P2	P3TEXT	P3	WAIT_CLASS
1	5	190	read by other session	file#	1	block#	162048	class#	1	User I/O
2	68	190	read by other session	file#	1	block#	162048	class#	1	User I/O
3	190		db file scattered read	file#	1	block#	162048	blocks	128	User I/O

可以看到等待事件已经变为了read by other session了，当然我们也可以在表上创建索引CREATE INDEX IND_XX_LHR ON TB_RBOS_20160829_LHR(OBJECT_ID) NOLOGGING;然后查询的时候SELECT COUNT(object_id) INTO I FROM TB_RBOS_20160829_LHR;可以走索引，这样

的话模拟出来的也是 read by other session 等待事件了。

我们根据 P1 和 P2 参数值获取到访问的对象名称和类型：

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, OWNER, TABLESPACE_NAME
FROM DBA_EXTENTS
WHERE FILE_ID =1
AND 162048
BETWEEN BLOCK_ID AND BLOCK_ID + BLOCKS - 1;
```

SEGMENT_NAME	SEGMENT_TYPE	OWNER	TABLESPACE_NAME
TB_RBOS_20160829_LHR	TABLE	SYS	SYSTEM

另外，其实我们也可以根据 v$session 的 ROW_WAIT_OBJ#列获取到等待的对象的名称，SQL 如下：

```
SELECT A.ROW_WAIT_OBJ#,
       B.OBJECT_NAME,
        A.SQL_ID,
       A.SID,
       A.BLOCKING_SESSION,
       A.EVENT,
       A.P1TEXT,
       A.P1,
       A.P2TEXT,
       A.P2,
       A.P3TEXT,
       A.P3,
       A.WAIT_CLASS
  FROM V$SESSION A, DBA_OBJECTS B
 WHERE A.ROW_WAIT_OBJ# = B.OBJECT_ID
AND SID IN(190,5,68);
```

ROW_WAIT_OBJ#	OBJECT_NAME	SQL_ID	SID	BLOCKING_SESSION	EVENT	P1TEXT
93621	TB_RBOS_20160829_LHR	97pq51643d7b5	190	5	read by other session	file#
93621	TB_RBOS_20160829_LHR	97pq51643d7b5	68	5	read by other session	file#
93621	TB_RBOS_20160829_LHR	97pq51643d7b5	5		db file scattered read	file#

```
SELECT * FROM V$SQL A WHERE A.SQL_ID='97pq51643d7b5';
```

SQL_TEXT	SQL_FULLTEXT	SQL_ID
SELECT COUNT(*) FROM TB_RBOS_20160829_LHR	<CLOB>	97pq51643d7b5

找到 SQL_ID，剩下的就是优化 SQL 语句了。

1.6.8 local write wait

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('local write wait');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
89	1570123276	local write wait	file#	block#		1740759767	8	User I/O

造成此等待事件的原因：

1) 磁盘损坏

2) 若执行 TRUNCATE 操作很慢，则可能由于表及其表上的索引的初始化值过大，可以通过 SQL 语句 ALTER INDEX IND_BIG_TEMP REBUILD STORAGE (INITIAL 1M);和 ALTER TABLE T_BIG MOVE STORAGE (INITIAL 1M);修改其初始化大小

truncates / reduce cache size

降低 cache size

Basically 'local write' wait happens (as the name indicates) when the session is waiting for its local (means writes pending because of its own operation) write operation. This could happen typically if the underlying disc has some serious problems (one of the member disk crash in RAID-05 - for example, or a controller failure). That is why I might have said ' you never see this

wait in the normal databases!'. You may see this during (rarely) Truncating a large table while most of the buffers of that table in cache. During TRUNCATEs the session has to a local checkpoint and during this process, the session may wait for 'local write' wait. We have not documented this in 'Oracle Wait Interface' as it is very uncommon. May be we can think of adding similar events in the Misc Waits in next edition.

翻译下： 基本上'local write wait' 表示会话在等待自己的写操作。在磁盘发生严重问题时会发生(例如 RAID 5 的一个磁盘崩溃,或者磁盘控制器错误),这在正常的系统中极少发生，在 TRUNCATE 一个大表而这个表在缓存中的时候，会话必需进行一个 local checkpoint,这个时候会话会等待 local session wait. 在开发环境里面 truncate 一些列表，速度奇慢。看了一下 session 等待事件 Local Write Wait

local write wait 主要是在 dbwr 将脏数据写回 dbf 过程中产生的。

可以考虑调整 dbwr 的效率。

另外，从设计上看，可以采用分区表。 把 truncate 操作改成 drop partition 的操作。

Sorry for the delay in my reply. I was traveling and was in Beirut for a week and just returned to Bangalore. You have got some valuable advice (esp the one from Jonathan).

Basically 'local write' wait happens (as the name indicates) when the session is waiting for its local (means writes pending because of its own operation) write operation. This could happen typically if the underlying disc has some serious problems (one of the member disk crash in RAID-05 - for example, or a controller failure). That is why I might have said ' you never see this wait in the normal databases!'. You may see this during (rarely) Truncating a large table while most of the buffers of that table in cache. During TRUNCATEs the session has to a local checkpoint and during this process, the session may wait for 'local write' wait.

We have not documented this in 'Oracle Wait Interface' as it is very uncommon. May be we can think of adding similar events in the Misc Waits in next edition.

During TRUNCATEs the session has to a local checkpoint and during this process, the session may wait for 'local write' wait.

果然有 local checkpoint

1.7 所有 User I/O 类等待事件的总结

类型	名称	P1	P2	P3	原因	处理
User I/O	db file sequential read	Oracle 要读取的文件的绝对文件号即 File#	Oracle 从这个文件中开始读取的起始数据块的 BLOCK 号即 Block#	Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量即 Blocks ,通常这个值为 1，表明是单个 BLOCK 被读取，如果这个值大于 1，则是读取了多个 BLOCK，这种多 BLOCK 读取常常出现在早期的 Oracle 版本中从临时段中读取数据的时候	这一事件通常显示与单个数据块相关的读取操作(如索引读取)。如果这个等待事件比较显著，可能表示在多表连接中，表的连接顺序存在问题，可能没有正确的使用驱动表；或者可能索引的使用存在问题，不加选择地进行索引，并非索引总是最好的选择。 还有一种特殊的情况是，全表扫描过程还会产生单块读的情况有，读 UNDO 块。可以参考最后的老熊文章的例子。对于这种情况的解决办法是加索引，或等大事务执行完成后再执行 SQL。	① 从 AWR 的报告中的"SQL ordered by Reads"部分或者从 V$SQL 视图中找出读取物理磁盘 I/O 最多的几个 SQL 语句 ,优化这些 SQL 语句以减少对 I/O 的读取需求 ② 增大高速缓存区
	db file scattered read	Oracle 要读取的文件的绝对文件号即 File#	Oracle 从这个文件中开始读取的起始数据块的 BLOCK 号即 Block#	Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量。	这个等待事件在实际生产库中经常可以看到，这是一个用户操作引起的等待事件，当用户发出每次 I/O 需要读取多个数据块这样的 SQL 操作时或者说当 Oracle 从磁盘上读取多个 BLOCK 到不连续的高速缓存区的缓存中，会产生这个等待事件，这个事件表明用户进程正在读数据到 Buffer Cache 中，等待直到物理 I/O 调用返回。最常见的两种情况是全表扫描（FTS： Full Table Scan）和索引快速全扫描（IFFS： index fast full scan）。根据经验，通常大量的 db file scattered read 等待可能意味着应用问题或者索引缺失。Oracle 一次能够读取的最多的 BLOCK 数量是由初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT 来决定。	① 找出执行全表扫描（FTS： Full Table Scan）和索引快速全扫描（IFFS： index fast full scan）扫描的 SQL 语句，判断这些扫描是否是必要的，是否导致了比较差的执行计划，如果是，则需要调整这些 SQL 语句 ,结合 v$session_longops 动态性能视图来进行诊断，该视图中记录了长时间(运行时间超过 6 秒的)运行的事务，可能很多是全表扫描操作 ② 调整 Oracle 数据库的多 BLOCK 的 I/O，设置一个合理的 Oracle 初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT ③ 通过对表和索引使用分区、将缓存区的 LRU 末端的全表扫描和 IFFS 扫描的的 BLOCK 放入到 KEEP 缓存池中等方法调整这个等待事件

	db file parallel read	P1 为 files 代表有多少个文件被读取所请求	blocks 代表总共有多少个 BLOCK 被请求	requests 代表总共有多少次 I/O 请求	db file parallel read 物理读等待事件涉及到的数据块均是不连续的，同时可以跨越 extent，这点不像 db file scattered read。 这是一个很容易引起误导的等待事件，实际上这个等待事件和并行操作（比如并行查询，并行 DML）没有关系。这个事件发生在数据库恢复的时候，当有一些数据块需要恢复的时候，Oracle 会以并行的方式把他们从数据文件中读入到内存中进行恢复操作。当 Oracle 从多个数据文件中并行的物理读取多个 BLOCK 到内存的不连续缓冲中(可能是高速缓存区或者是 PGA)的时候可能就会出现这个等待事件。这种并行读取一般出现在恢复操作中或者是从缓冲中预取数据达到最优化(而不是多次从单个 BLOCK 中读取，buffer prefetch 以优化多个单块读)。这个事件表明会话正在并行执行多个读取的需求。注意：在 11g 之前，这个等待事件发生在数据文件的恢复过程中，但 11g 中新增了 prefetch 的特性，所以也可能导致这个等待事件的产生。	如果在等待时间中这个等待事件占的比重比较大，可以按照处理 db file sequential read 等待事件的方法来处理这个事件。 若是由于 prefetch 引起的性能问题，我们可以通过添加隐含参数来解决该问题。 ALTER SYSTEM SET "_db_block_prefetch_quota"=0 SCOPE=SPFILE SID='*'; ALTER SYSTEM SET "_db_block_prefetch_limit"=0 SCOPE=SPFILE SID='*'; ALTER SYSTEM SET "_db_file_noncontig_mblock_read_count"=0 SCOPE=SPFILE SID='*';
	db file single write	需要更新的数据块所在的数据文件的文件号。查询文件号的 SQL 语句是：SELECT * FROM v$datafile WHERE file# = <file#>;	需要更新的数据块号，如果 BLOCK 号不是 1，则可以通过如下查询查出 Oracle 正在写入的对象是什么： SELECT segment_name , segment_type , owner , tablespace_name FROM sys.dba_extents WHERE file_id = <file#> AND <block#> BETWEEN block_id AND block_id + blocks -1 ;	需要更新的数据块数目(通常来说应该等于 1），或 Oracle 写入 file#的数据文件中从 BLOCK#开始写入的 BLOCK 的数量。头一般来说都是 BLOCK1，操作系统指定的文件头是 BLOCK0，如果 BLOCK 号大于 1，则表明 Oracle 正在写入的是一个对象而不是文件头。	这个等待事件通常只发生在一种情况下，就是 Oracle 更新数据文件头信息时（比如发生 Checkpoint）。	当这个等待事件很明显时，需要考虑是不是数据库中的数据文件数量太大，导致 Oracle 需要花较长的时间来做所有文件头的更新操作（checkpoint）。

	direct path read	等待 I/O 读取请求的文件的绝对文件号	等待 I/O 读取请求的第一个 BLOCK 号	以 first block 为起点，总共有多少个连续的 BLOCK 被请求读取	由参数 P1 与 P2 推得访问的数据对象： select s.segment_name, s.partition_name from dba_extents s where between s.block_id and (s.block_id + s.blocks -1) and s.file_id = 直接路径读（direct path read）通常发生在 Oracle 直接读取数据到 PGA 时，这个读取不需要经过 SGA。这类读取通常在以下情况被使用： ① 大量的磁盘排序 IO 操作 在排序操作(order by, group by, union, distinct, rollup,合并连接)时，由于 PGA 中的 SORT_AREA_SIZE 空间不足 ,无法在 PGA 中完成排序，需要利用 temp 表空间进行排序，当从临时表空间中读取排序结果时，会产生 direct path read，从 10g 开始表现为 direct path read temp 等待事件。 ② 大量的 Hash Join 操作，利用 temp 表空间保存 hash 区。使用 HASH 连接的 SQL 语句，将不适合位于内存中的散列分区刷新到临时表空间中。为了查明匹配 SQL 谓词的行，临时表空间中的散列分区被读回到内存中(目的是为了查明匹配 SQL 谓词的行)，ORALCE 会话在 direct path read 等待事件上等待。 ③ SQL 语句的并行查询，并行查询从属进程使用并行扫描的 SQL 语句也会影响系统范围的 direct path read 等待事件。在并行执行过程中，direct path read 等待事件与从属查询有关，而与父查询无关，运行父查询的会话基本上会在 PX Deq:Execute Reply 上等待，从属查询会产生 direct path read 等待事件。 ④ 预读操作 ⑤ 串行全表扫描(Serial Table Scan)，大表的全表扫描，在 Oracle11g 中，全表扫描的算法有新的变化，根据表的大小、高速缓存的大小等信息，决定是否绕过 SGA 直接从磁盘读取数据。而 10g 则是全部通过高速缓存读取数据，称为 table scan(large)。11g 认为大表全表时使用直接路径读，可能比 10g 中的数据文件散列读(db file scattered reads)速度更快，使用的 latch	分析产生该等待事件的原因然后有针对性的解决。对于这一写入等待 ,我们应该找到 I/O 操作最为频繁的数据文件（如果有过多的排序操作，很有可能就是临时文件），分散负载，加快其写入操作。 禁用 direct path read： _serial_direct_read = false 启用 direct path read： _serial_direct_read = true 由 direct path read 事件产生的原因，我们需要判断该事件正在读取什么段(如：散列段、排序段、一般性的数据文件)，由此可判断产生该事件的原因是什么，可使用以下语句进行查询： SELECT a.event, a.sid, c.sql_hash_value hash_vale, decode(d.ktssosegt, 1, 'SORT', 2, 'HASH', 3, 'DATA', 4, 'INDEX', 5, 'LOB_DATA', 6, 'LOB_INDEX', NULL) AS segment_type, b.tablespace_name, b.file_name FROM v$session_wait a, dba_data_files b, v$session c, x$ktsso d WHERE c.saddr = d.ktssoses(+) AND c.serial# = d.ktssosno(+) AND d.inst_id(+) = userenv('instance') AND a.sid = c.sid AND a.p1 = b.file_id AND a.event = 'direct path read' UNION ALL SELECT a.event, a.sid, d.sql_hash_value hash_value,

					也更少。	decode(e.ktssosegt, 1, 'SORT', 2, 'HASH', 3, 'DATA', 4, 'INDEX', 5, 'LOB_DATA', 6, 'LOB_INDEX', NULL) AS segment_type, b.tablespace_name, b.file_name FROM v$session_wait a, dba_temp_files b, v$parameter c, v$session d, x$ktsso e WHERE d.saddr = e.ktssoses(+) AND d.serial# = e.ktssosno(+) AND e.inst_id(+) = userenv('instance') AND a.sid = d.sid AND b.file_id = a.p1 - c.VALUE AND c.NAME = 'db_files' AND a.event = 'direct path read';
	direct path write	等待 I/O 读取请求的文件的绝对文件号	等待 I/O 读取请求的第一个 BLOCK 号	总共有多少个连续的 BLOCK 被请求读取	由参数 P1 与 P2 推得访问的数据对象： select s.segment_name, s.partition_name from dba_extents s where between s.block_id and (s.block_id + s.blocks -1) and s.file_id = 这个等待事件和 direct path read 正好相反，是会话将一些数据从 PGA 中直接写入到磁盘文件上，而不经过 SGA。这类读取通常在以	① 如果是等待的文件是临时表空间的文件，那么需要查看是否存在大量不合理的磁盘排序，优化相应的存在问题的 SQL 语句。 ② 检查是否存在 I/O 消耗很严重的 SQL 语句，如果存在，尝试优化 SQL 语句减少 I/O 的消耗。 ③ 确认一下是否达到了磁盘的 I/O 极限，如果是，则需要考虑更换更好的硬件设备。 ④ 确认异步 I/O 是否配置正确，异步 I/O 不会减少这个等待事件的等待时间但是却可以减少会话所消耗的时间。

					下情况被使用： 1 .直接路径加载 使用 append 方式加载数据、CREATE TABLE AS SELECT） 2．并行 DML 操作 3．磁盘排序使用临时表空间排序（内存不足）	
	read by other session	文件号	块号	类别，class#为 1 代表扫描的是表或者索引，class#为 4 代表"segment header"，class#>=15 则代表 undo 块。	当多个进程访问同一个数据块，而此数据块不在内存中，这时会有一个进程将它从磁盘读到内存时，其它读取此数据块进程的状态就是 read by other session；因为 Oracle 内存不允许多个进程同时读到同一个数据块到内存，其它进程只能等待。 当我们查询一条数据时，Oracle 第一次会将数据从磁盘读入 buffer cache。如果有两个或者多个 session 请求相同的信息 ,那么第一个 session 会将这个信息读入 buffer cache，其他的 session 就会出现等待。	找到并优化相关的 SQL 语句。 我们可以根据 P1 和 P2 参数值获取到等待的对象名称和类型： SELECT SEGMENT_NAME, SEGMENT_TYPE, OWNER, TABLESPACE_NAME FROM DBA_EXTENTS WHERE FILE_ID = FILE# AND BLOCK# BETWEEN BLOCK_ID AND BLOCK_ID + BLOCKS - 1; 另外，其实我们也可以根据 v$session 的 ROW_WAIT_OBJ#列获取到等待的对象的名称，SQL 如下： SELECT A.ROW_WAIT_OBJ#, B.OBJECT_NAME, A.SQL_ID, A.SID, A.BLOCKING_SESSION, A.EVENT, A.P1TEXT, A.P1, A.P2TEXT, A.P2, A.P3TEXT, A.P3, A.WAIT_CLASS FROM V$SESSION A, DBA_OBJECTS B WHERE A.ROW_WAIT_OBJ# = B.OBJECT_ID AND A.EVENT='read by other session';

About Me

.....

- 本文作者：小麦苗，只专注于数据库的技术，更注重技术的运用
- 本文在 ITpub（http://blog.itpub.net/26736162）、博客园(http://www.cnblogs.com/lhrbest)和个人微信公众号（xiaomaimiaolhr）上有同步更新，推荐 pdf 文件阅读或博客园地址阅读
- QQ 群：230161599 微信群：私聊
- 本文 itpub 地址：http://blog.itpub.net/26736162/viewspace-2124417/ 博客园地址：http://www.cnblogs.com/lhrbest/articles/5835420.html
- 本文 pdf 版：http://yunpan.cn/cdEQedhCs2kFz （提取码：ed9b）
- 小麦苗分享的其它资料：http://blog.itpub.net/26736162/viewspace-1624453/
- 联系我请加 QQ 好友(642808185)，注明添加缘由
- 于 2016-08-13 09:00~2016-09-02 19:00 在中行完成
-

……

长按识别二维码或微信客户端扫描下边的二维码来关注小麦苗的微信公众号：xiaomaimiaolhr,学习最实用的数据库技术。