【等待事件】等待事件系列（3+4）--System IO（控制文件）+日志类等待

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 控制文件类等待

② 日志类等待

1.2.2 相关参考文章链接

📖 说明：

【推荐】 等待事件系列（1）--User I/O 类型（下）		http://blog.itpub.net/26736162/viewspace-2124435/
【推荐】 等待事件系列（1）--User I/O 类型（上）		http://blog.itpub.net/26736162/viewspace-2124417/
2016-09-07	【等待事件】System I/O 类 等待事件(3.4) --control file single write	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771471&idx=1&sn=5922a52ac6294acf2802f44e2bb0d724&chksm=fe8bba77c9fc336151a61bdf876cb058df0d61d1404d8450cb7771330b6d44309d86dae4bb54&scene=21#wechat_redirect
2016-09-06	【等待事件】System I/O 类 等待事件(3.3) --control file sequential read	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771468&idx=1&sn=fc7d83d1a9b12911f3c93d3b5b444e9a&chksm=fe8bba74c9fc3362b58717fca9e95c68d45e701fa2f733a643ba01db7969cca668858272fbfc&scene=21#wechat_redirect
2016-09-04	【等待事件】System I/O 类 等待事件(3.2) --control file parallel write	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771458&idx=1&sn=e949dfa5bff65ce4a596005955c5be5a&scene=21#wechat_redirect
2016-09-03	【等待事件】System I/O 类 等待事件(3.1) --db file parallel write	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771454&idx=1&sn=e90248954475dfd2c78bdec592405735&scene=21#wechat_redirect
2016-09-01	【等待事件】User I/O 类 等待事件（2.10） --所有 User I/O 类 等待事件总结	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771447&idx=1&sn=22ae192f0d8a161f65514339ad763985&scene=21#wechat_redirect
2016-08-31	【等待事件】User I/O 类 等待事件（2.9） --local write wait	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771443&idx=1&sn=02b4ad5ca03052013b69ae6bcb7e3487&scene=21#wechat_redirect
2016-08-30	【等待事件】User I/O 类 等待事件（2.8） --read by other session	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771439&idx=1&sn=b3c01eed444cd6e597a63a3ed0687768&scene=21#wechat_redirect
2016-08-29	【等待事件】User I/O 类 等待事件（2.7） --direct path read/write temp	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771429&idx=1&sn=50b5684e699165a34087db88e07edb34&scene=21#wechat_redirect
2016-08-27	【等待事件】User I/O 类 等待事件（2.6） --direct path write(直接路径写、DRW)	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771420&idx=1&sn=458eb18dc26da94debcea62643d15181&scene=21#wechat_redirect
2016-08-26	【等待事件】User I/O 类 等待事件（2.5） --direct path read (直接路径读、DPR)	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771416&idx=1&sn=b26c3135584c5b60ce14cc0749ac58a7&scene=21#wechat_redirect
2016-08-20	【等待事件】User I/O 类 等待事件（2.4） --db file single write	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771403&idx=1&sn=054dd852dac5ac8837fa251f0e84332e&scene=21#wechat_redirect
2016-08-16	【等待事件】User I/O 类 等待事件（2.3） --db file parallel read	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771387&idx=1&sn=0037fb89470d8e6dd5ff72714b18a3b7&scene=21#wechat_redirect
2016-08-15	【等待事件】User I/O 类 等待事件（2.2） --db file scattered read (数据文件离散读)	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771379&idx=1&sn=5887eee02885000c1d293adfd04ee044&scene=21#wechat_redirect

2016-08-14	【等待事件】User I/O类 等待事件（2.1）--db file sequential read（数据文件顺序读）	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771376&idx=1&sn=42de046e73190f4e265f81bbb6e3ae00&scene=21#wechat_redirect
2016-08-13	【等待事件】等待事件概述（1）--等待事件的源起和分类	http://mp.weixin.qq.com/s?__biz=MzIzOTA2NjEzNQ==&mid=2454771373&idx=1&sn=1e55af795aae5f641b2c3cc610814ead&scene=21#wechat_redirect

1.3 System I/O 类型

```
SELECT *
FROM   v$event_name d
WHERE  d.WAIT_CLASS ='System I/O';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	16	2006672057	Clonedb bitmap file write	blkno	size		4108307767	9	System I/O
2	22	2089793129	Log archive I/O	count	intr	timeout	4108307767	9	System I/O
3	23	3107297351	RMAN backup & recovery I/O	count	intr	timeout	4108307767	9	System I/O
4	24	3511162007	Standby redo I/O	count	intr	timeout	4108307767	9	System I/O
5	25	2907891391	Network file transfer	count	intr	timeout	4108307767	9	System I/O
6	58	1867721841	io done	msg ptr			4108307767	9	System I/O
7	60	4120085931	RMAN Disk slave I/O	wait count	wait flags	timeout	4108307767	9	System I/O
8	61	2828390964	RMAN Tape slave I/O	tape operation	operation flags	timeout	4108307767	9	System I/O
9	62	3671110102	DBWR slave I/O	wait count	wait flags	timeout	4108307767	9	System I/O
10	63	1180269184	LGWR slave I/O	wait count	wait flags	timeout	4108307767	9	System I/O
11	64	1260687889	Archiver slave I/O	wait count	wait flags	timeout	4108307767	9	System I/O
12	80	3213517201	control file sequential read	file#	block#	blocks	4108307767	9	System I/O
13	81	2383414886	control file single write	file#	block#	blocks	4108307767	9	System I/O
14	82	4078387448	control file parallel write	files	block#	requests	4108307767	9	System I/O
15	99	765902655	recovery read				4108307767	9	System I/O
16	110	3134783330	RFS sequential i/o				4108307767	9	System I/O
17	111	3089862638	RFS random i/o				4108307767	9	System I/O
18	112	2009048728	RFS write				4108307767	9	System I/O
19	132	549236675	log file sequential read	log#	block#	blocks	4108307767	9	System I/O
20	133	215477332	log file single write	log#	block#	blocks	4108307767	9	System I/O
21	134	3999721902	log file parallel write	files	blocks	requests	4108307767	9	System I/O
22	149	1620694733	db file parallel write	requests	interrupt	timeout	4108307767	9	System I/O
23	150	133155944	db file async I/O submit	requests	interrupt	timeout	4108307767	9	System I/O
24	220	3487232407	flashback log file write	log#	block#	Bytes	4108307767	9	System I/O
25	221	2769438501	flashback log file read	log#	block#	Bytes	4108307767	9	System I/O
26	228	2705202413	cell smart incremental backup	cellhash#			4108307767	9	System I/O
27	230	3260626027	cell smart restore from backup	cellhash#			4108307767	9	System I/O
28	385	1568594048	kfk: async disk IO	count	intr	timeout	4108307767	9	System I/O
29	404	1946591843	cell manager opening cell	cellhash#			4108307767	9	System I/O
30	405	1617821548	cell manager closing cell	cellhash#			4108307767	9	System I/O
31	406	4026210584	cell manager discovering disks	cellhash#			4108307767	9	System I/O

1.3.1 db file parallel write

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

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
150	1620694733	db file parallel write	requests	interrupt	timeout	4108307767	9	System I/O

这个等待事件有 3 个参数：

Requests： 操作需要执行的 I/O 次数（DBWR 写入批量的大小-块数）。

interrupt：（中断）

timeout：等待的超时时间。

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 正在写入的数据文件的数量，P2 代表操作将会写入多少的 BLOCK 数量，P3 在 Oracle9i release2 版本之前代表总共有多少 BLOCK 的 I/O 请求，等于 P2 的值；在 Oracle9i release2 版本之后则代表等待 I/O 完成的超时的时间，单位是百分之一秒。

经过高速缓冲区的所有数据是通过 DBWR 写入到磁盘上的。DBWR 请求写入脏块的 I/O 后，在此工作结束期间等待 db file parallel write 事件。

这是一个后台等待事件，它同样和用户的并行操作没有关系，它是由后台进程 DBWR 产生的，当后台进程 DBWR 向磁盘上写入脏数据时，会发生这个等待。

DBWR 会批量地将脏数据并行地写入到磁盘上相应的数据文件中，在这个批次作业完成之前，DBWR 将出现这个等待事件。 如果仅仅是这一个等待事件，对用户的操作并没有太大的影响，当伴随着出现 free buffer waits 等待事件时，说明此时内存中可用的空间不足，这时候会影响到用户的操作，比如影响到用户将脏数据块读入到内存中。

当出现 db file parallel write 等待事件时，可以通过启用操作系统的异步 I/O 的方式来缓解这个等待。 当使用异步 I/O 时，DBWR 不在需要一直等到所有数据块全部写入到磁盘上，它只需要等到这个数据写入到一个百分比之后，就可以继续进行后续的操作。

这个等待事件是指 Oracle 后台进程 DBWR 等待一个并行写入文件或者是 BLOCK 的完成 ,等待会一直持续到这个并行写入操作完成。这个等待事件即使在总的等待时间中占的比例比较大也不会对用户的会话有很大的影响，只有当用户的会话显示存在大量的等待时间消耗在"write complete waits" 或者是"free buffer waits"上的时候才会影响到用户的会话，较明显的影响是这个写操作的等待会影响到读取同一个磁盘上数据的用户会话的 I/O。

① 与其名称相反，该事件不与任何并行 DML 操作相关。

② 该等待事件属于 DBWR 进程，DBWR 进程负责向数据文件写入脏数据块的唯一进程，即 DBWR 进程执行对使用 SGA 的所有数据库写入。阻塞该进程的是操作系统的 IO 子系统。当然 DBWR 进程的写入操作也会对同一磁盘操作的

其他会话造成影响。

③ DBWR 查找脏块的时机：

>> 每隔三秒一次的查找。

>> 当前台提交需要清除缓冲区内容时。

>> 当满足_DB_LARGE_DIRTY_QUEUE/_DB_BLOCK_MAX_DIRTY_TARGET /FAST_START_MTTR_TARGET 阈值。

④ 缓慢的 DBWR 操作可以造成前台会话在 write complete waits(前台不允许修改正在传输到磁盘的块)或 free buffer waits(DBWR 不能满足释放缓冲区的需求)事件上。通过以下语句可以获知该事件的平均等待时间，如果平均等待时间大小 10cs，则表明 IO 缓慢。如果不存在 db file parallel write 事件，很可能初始化参数 disk_async_io=FALSE，这种情况一般发生在 AIX 和 HPUX 平台上。

```
SELECT s.event, s.time_waited, s.average_wait
FROM v$system_event s
WHERE s.event IN ('db file parallel write', 'free buffer waits',
'write complete waits')
```

相关查询：

```
SELECT *
FROM v$sysstat
WHERE NAME IN ('write clones created in background',
'write clones created in foreground')
```

⑤ 操作说明：DBWR 将一组脏数据编成"写入批量组"，然后发布多个 IO 请求以将"写入批量组"写入数据文件，然后以此事件等待直到 IO 请求都完成。但是，当使用异步 IO 时，DBWR 不等待整个批量写入完成，仅等待一定百分比的 IO 操作完成后，就将空闲缓冲区推到 LRU 链以使其可用。

⑥ 解决方法：

>> 如果平均等待时间长，要选择使用正确的 IO 操作。如果数据文件在裸设备上，并且平台支持异步 IO，请应该使用异步 IO。如果数据文件位于文件系统上，则应该使用同步写入和直接 IO。相关的初始化参数是 DISK_ASYNCH_IO 和 FILESYSTEMIO_OPTIONS。

>> 如果重做位于裸设备上，而数据文件位于文件系统上，则可以设置 DISK_ASYNCH_IO=TRUE，FILESYSTEMIO_OPTIONS=DIRECTIO。使用这种方法可以获得对于裸设备使用异步 IO，而对于文件系统使用直接 IO 的效果。

>> 使用 DB_WRITER_PROCESSES 选项产生多个 DBWR 进程。

1、I/O 系统的性能缓慢时

db file parallel write 等待的发生原因和解决方法如下：

如果 DBWR 进程上 db file parallel write 等待时间表现得过长，就可以判断为 I/O 系统上有问题。如果 DBWR 上的 db file parallel write 等待时间延长，服务器进程就会接连经历 free buffer waits 事件或 write complete waits 事件的等待。这个问题可以通过改善 I/O 系统解决，改善 I/O 性能的方法如下：

（1）组合使用裸设备和异步 I/O 是目前为止的最好方法。

（2）OS 级上使用 Direct I/O。若 CPU 数量充足，可以调整 db_writer_processes 参数值，将 DBWR 数量增加。多个 DBWR 具有模拟异步的效果。oracle 推荐的 DBWR 进程是 CPU_COUNT/8。

2、I/O 工作过多时

频繁发生检查点时，DBWR 的活动量过多，可能导致 DBWR 的性能降低。DBWR 的性能与整个系统的性能有直接的联系。将 fast_start_mttr_target（MTTR 指平均恢复时间，数据库进行崩溃恢复需要的秒数。）参数值设定过小时，将频繁发生增量检查点工作。日志文件过小时，将频繁发生日志文件的转换，因此检查点工作将增加。因 Parallel Query 发生 direct path read 时，在 truncate、drop、hot backup 时也发生检查点。如果 I/O 系统上不存在性能问题，但还是广泛出现 db file parallel write 等待，就应该检查是否存在给 DBWR 带来不必要的负荷的因素。

3、不能有效使用高速缓存区时

间接改善 DBWR 性能的另一种方法是合理使用多重缓冲池。与其说这个方法能改善 I/O 系统的性能，不如说是因为不必要的写入工作减少，进而减少了 DBWR 的负担。

1.3.2 控制文件 I/O 等待事件

```
SELECT A.* FROM V$EVENT_NAME A WHERE NAME LIKE 'control file%';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	80	3213517201	control file sequential read	file#	block#	blocks	4108307767	9	System I/O
2	81	2383414886	control file single write	file#	block#	blocks	4108307767	9	System I/O
3	82	4078387448	control file parallel write	files	block#	requests	4108307767	9	System I/O
4	83	2365673131	control file backup creation				4166625743	3	Administrative
5	580	40893507	control file heartbeat				1893977003	0	Other
6	581	3061031534	control file diagnostic dump	type	param		1893977003	0	Other

这一类等待事件通常发生在更新控制文件时，例如日志切换、检查点等发生时，需要更新控制文件内的 System Change Number(SCN)所引起的相关等待事件，以下将为这些控制文件所引起的等待事件做详尽的介绍。

1.3.2.1 control file parallel write-控制文件并行写

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('control file parallel write');
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	82	4078387448	control file parallel write	files	block#	requests	4108307767	9	System I/O

这个等待事件包含三个参数：

files：Oracle 要写入的控制文件个数。

block#：写入控制文件的数据块数目。

requests：写入控制文件请求的 I/O 次数。

在 V$SESSION_WAIT 这个视图里面，这个等待事件有三个参数 P1、P2、P3，这三个参数都设置为同样的值，代表控制文件对 I/O 的请求数量。当 Oracle 更新控制文件的时候是同时更新所有控制文件并写入同样的信息。

这个等待事件表明服务器进程(Server Process)在更新所有的控制文件的时候等待 I/O 的完成。因为控制文件所在的磁盘的 I/O 过高引起无法完成对所有控制文件的物理写入，写入控制文件的这个会话会拥有 CF 队列，因此其他的会话都会在这个队列中等待。

一般环境下，因为更新控制文件的次数不多，因此不怎么发生 control file parallel write 等待现象。但如下情况下可能发生与控制文件相关的争用。

1） 日志文件切换经常发生时：

日志文件过小时，将经常发生日志文件的切换。每当发生日志文件切换时，需要对控制文件进行更新，所以 LGWR 进程等待 control file parallel write 事件的时间将延长。

2） 检查点经常发生时：

MTTR 设定得过短或频繁发生人为的检查点时，CKPT 进程等待 control file parallel write 事件的时间将延长。

3） nologging 引起频繁的数据文件修改时：

对数据文件在 nologging 选项下执行修改工作时，为了修改 unrecoverable SCN 需要更新控制文件。这时，服务器进程将等待 control file parallel write 事件。

4） I/O 系统的性能缓慢时：

最好是将控制文件位于独立的磁盘空间上，使用裸设备或 direct I/O。

control file parallel write 等待，通常与 control file sequential read 等待或 enq: CF - contention 等待一同出现的情况较多。enq: CF - contention 等待是在多个会话为了同时更新控制文件获得 CF 锁的过程中发生的。control file parallel write、control file sequential read、CF - contention 等待，全是因为过多的控制文件更新或 I/O 系统的性能问题引发的。

当 server 进程更新所有控制文件时，这个事件可能出现。如果等待很短，可以不用考虑。如果等待时间较长，检查存放控制文件的物理磁盘 I/O 是否存在瓶颈。

多个控制文件是完全相同的拷贝，用于镜像以提高安全性。对于业务系统，多个控制文件应该存放在不同的磁盘上，一般来说三个是足够的，如果只有两个物理硬盘，那么两个控制文件也是可以接受的。在同一个磁盘上保存多个控制文件是不具备实际意义的。

当数据库中有多个控制文件的拷贝时，Oracle 需要保证信息同步地写到各个控制文件当中，这是一个并行的物理操作过程，因为称为控制文件并行写，当发生这样的操作时，就会产生 control file parallel write 等待事件。

控制文件频繁写入的原因很多，比如：

- 日志切换太过频繁，导致控制文件信息相应地需要频繁更新。当系统出现日志切换过于频繁的情形时，可以考虑适当地增大日志文件的大小来降低日志切换频率。

- 系统 I/O 出现瓶颈，导致所有 I/O 出现等待。

如果在等待时间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：

- 在确保控制文件不会同时都丢失的前提下，将控制文件的数量减小到最少，降低控制文件的拷贝数量(在确保安全的前提下)。
- 如果系统支持异步 I/O，则推荐尽量使用异步 I/O，这样可以实现真正并行的写入控制文件。
- 将控制文件移动到负载比较低，速度比较快的磁盘上去。
- 将控制文件的拷贝存放在不同的物理磁盘上的方式来缓解 I/O 争用。

1.3.2.2 control file sequential read

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('control file sequential read');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
80	3213517201	control file sequential read	file#	block#	blocks	4108307767	9	System I/O

这个等待事件有三个参数：

File#：要读取信息的控制文件的文件号。

Block#：读取控制文件信息的起始数据块号。

Blocks：需要读取的控制文件数据块数目。

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 代表正在读取的控制文件号，通过下面的 SQL 语句可以知道究竟是具体是哪个控制文件被读取：

```
SELECT * FROM X$KCCCF WHERE INDX = <file#>;
```

P2 代表开始读取的控制文件 BLOCK 号，它的 BLOCK 大小和操作系统的 BLOCK 大小一样，通常来说是 512K，也有些 UNIX 的是 1M 或者 2M，P3 代表会话要读取 BLOCK 的数量。一般来说使用参数 P1、P2 来查询 BLOCK，当然也可以包括参数 P3，但是那样最终就变成了一个多 BLOCK 读取，因此我们一般都忽略参数 P3。

Wait Time: The wait time is the elapsed time of the read

Parameter	Description
file#	The control file from which the session is reading

block#	Block number in the control file from where the session starts to read. The block size is the physical block size of the port (usually 512 bytes, some UNIX ports have 1 or 2 Kilobytes).
blocks	The number of blocks that the session is trying to read

控制文件连续读/控制文件单个写对单个控制文件 I/O 存在问题时，这两个事件会出现。如果等待比较明显，检查单个控制文件，看存放位置是否存在 I/O 瓶颈。

当数据库需要读取控制文件上的信息时，会出现这个等待事件，因为控制文件的信息是顺序写的，所以读取的时候也是顺序的，因此称为控制文件顺序读，它经常发生在以下情况：

- 备份控制文件
- RAC 环境下不同实例之间控制文件的信息共享
- 读取控制文件的文件头信息
- 读取控制文件其他信息

Reading from the control file. This happens in many cases. For example, while:

1、Making a backup of the control files

2、Sharing information (between instances) from the control file

3、Reading other blocks from the control files

4、Reading the header block

读取控制文件的时候遇到 I/O 等待就会出现这个等待事件，例如备份控制文件的时候、读取 BLOCK 头部都会引起这个等待事件，等待的时间就是消耗在读取控制文件上的时间。

如果这个等待事件等待的时间比较长，则需要检查控制文件所在的磁盘是否很繁忙，如果是，将控制文件移动到负载比较低，速度比较快的磁盘上去。如果系统支持异步 I/O，则启用异步 I/O。对于并行服务器来说，如果这种等待比较多，会造成整个数据库性能下降，因为并行服务器之间的一些同步是通过控制文件来实现的。

解决方式与 control file parallel write 的解决方式一样。

首先，该等待事件并不表明数据库有问题。一个健康的系统，物理读事件应是除空闲等待事件外的最大等待事件。而该事件在 RAC 中尤其明显，依照经验来看，在一个正常的 RAC 集群中，该事件应该排在 top10 中，因为实例间共享同一份控制文件，对控制文件读取是很频繁的，如果被其他等待事件挤出前 10 了，那就得看看是哪些等待事件了。

其次，可以查看 AWR 报告该事件的等待次数，平均等待时间，最大等待时间等信息进行进一步确认。看看这些信息比起日常 AWR 报告信息是否有明显的异常。

1.3.2.3 control file single write

```
SELECT A.*
  FROM V$EVENT_NAME A
 WHERE NAME IN ('control file single write');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
81	2383414886	control file single write	file#	block#	blocks	4108307767	9	System I/O

P2 代表开始读取的控制文件 BLOCK 号，它的 BLOCK 大小和操作系统的 BLOCK 大小一样，通常来说是 512K，也有些 UNIX 的是 1K 或者 2K，P3 代表会话要读取 BLOCK 的数量。一般来说使用参数 P1、P2 来查询 BLOCK，当然也可以包括参数 P3，但是那样最终就变成了一个多 BLOCK 读取，因此我们一般都忽略参数 P3。

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 代表正在读取的控制文件号，通过下面的 SQL 语句可以知道究竟是具体是哪个控制文被读取：

```
SELECT * FROM X$KCCCF WHERE INDX = <file#>;
```

这个等待事件出现在写控制文件的共享信息到磁盘的时候，这是个自动操作，并且通过一个实例来保护的，如果是并行的数据库服务器，那么对于并行服务器来说也只能有一个实例能够执行这个操作。这个事件的等待事件就是写操作所消耗的时间。

尽管这个事件的是 single write，事实上也会出现多 BLOCK 写的情况，即 P3>1。使用参数 P1、P2 来查询检测 BLOCK 而不用去考虑 P3 的值。

如果这个等待事件等待的时间比较长，则需要检查控制文件所在的磁盘是否很繁忙，如果是，将控制文件移动到负载比较低，速度比较快的磁盘上去。如果系统支持异步 I/O，则启用异步 I/O。对于并行服务器来说，如果这种等待比较多，会造成整个数据库性能下降，因为并行服务器之间的一些同步是通过控制文件来实现的。

解决方式与 control file parallel write 的解决方式一样。

1.4 日志相关等待--联机重做日志文件 I/O 等待事件

REDO 对于数据库来说非常重要，有一系列等待事件和日志相关，通过 V$EVENT_NAME 视图可以找到这些等待事

件。

我们以 11g 为主：

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE '%log%';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	2	3934444552	logout restrictor				3875070507	4	Concurrency
2	129	2836458162	LNS ASYNC archive log				2723168908	6	Idle
3	131	3378470826	LNS ASYNC end of log				2723168908	6	Idle
4	132	549236675	log file sequential read	log#	block#	blocks	4108307767	9	System I/O
5	133	215477332	log file single write	log#	block#	blocks	4108307767	9	System I/O
6	134	3999721902	log file parallel write	files	blocks	requests	4108307767	9	System I/O
7	137	3357856061	log buffer space				3290255840	2	Configuration
8	138	2867289651	log file switch (checkpoint incomplete)				3290255840	2	Configuration
9	139	114164561	log file switch (private strand flush incomplete)				3290255840	2	Configuration
10	140	681532028	log file switch (archiving needed)				3290255840	2	Configuration
11	141	3845123846	switch logfile command				4166625743	3	Administrative
12	142	3834950329	log file switch completion				3290255840	2	Configuration
13	143	1328744198	log file sync	buffer#	sync scn		3386400367	5	Commit
14	144	2915106572	simulated log write delay				2723168908	6	Idle
15	209	3314391977	LogMiner reader: log (idle)	Session ID	Thread	Sequence	2723168908	6	Idle
16	220	3487232407	flashback log file write	log#	block#	Bytes	4108307767	9	System I/O
17	221	2769438501	flashback log file read	log#	block#	Bytes	4108307767	9	System I/O
18	223	157547587	flashback log file sync				1740759767	8	User I/O
19	343	881521144	Streams capture: waiting for archive log	Is GoldenGate	Is XStream	TYPE	2723168908	6	Idle
20	541	164764783	ges RMS0 retry add redo log				1893977003	0	Other
21	573	3480025058	gcs log flush sync	waittime	poll	event	1893977003	0	Other
22	609	2370101988	ARCH wait for archivelog lock				1893977003	0	Other
23	623	916189195	MRP wait on archivelog arrival				1893977003	0	Other
24	624	649270296	MRP wait on archivelog archival				1893977003	0	Other
25	625	1722088478	log switch/archive	thread#			1893977003	0	Other
26	648	1561425187	log file switch (clearing log file)				1893977003	0	Other
27	685	260587712	enq: SB - logical standby metadata	name|mode	0	0	1893977003	0	Other
28	694	3609173507	enq: XR - database force logging	name|mode	operation	0	1893977003	0	Other
29	712	2825667152	recovery area: computing applied logs				1893977003	0	Other
30	720	3383573524	enq: FL - Flashback database log	name|mode	Log #	zero	1893977003	0	Other
31	727	2085703061	enq: FD - Flashback logical operations	name|mode	Internal	Internal	1893977003	0	Other
32	728	1729254775	flashback free VI log				1893977003	0	Other
33	729	3594374201	flashback log switch				1893977003	0	Other
34	1067	3902700176	log write(odd)	group#			1893977003	0	Other

1.4.1 log file switch（日志文件切换）

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE 'log file switch %';
```

	EVENT#	EVENT_ID	NAME	PA	PAR	P/	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	138	2867289651	log file switch (checkpoint incomplete)				3290255840	2	Configuration
2	139	114164561	log file switch (private strand flush incomplete)				3290255840	2	Configuration
3	140	681532028	log file switch (archiving needed)				3290255840	2	Configuration
4	142	3834950329	log file switch completion				3290255840	2	Configuration

当数据库日志文件发生切换时出现，LGWR 需要关闭当前日志组，切换并打开下一个日志组，在这个切换过程中，数据库的所有 DML 操作都处于停顿状态，直至这个切换完成。

这个等待事件是指执行日志文件切换命令的时候等待日志文件切换完成，Oracle 数据库会每隔五秒钟就检测一次是否超时。如果出现这个等待事件，表明花费了很长的时间去切换重做日志文件，此时我们需要去检查数据库的告警日志文件查看 Oracle 后台进程 LGWR 是否正常在工作。

log file switch 引起的等待都是非常重要的，如果出现就应该引起重视，并由 DBA 介入进行及时处理。

log file switch 包含 5 个子事件：

1. log file switch (archiving needed)，即日志切换（需要归档）

这个等待事件出现时通常是因为日志组循环写满以后，在需要覆盖先前日志时，发现日志归档尚未完成，出现该等待。由于 Redo 不能写出，该等待出现时，数据库将陷于停顿状态。

这个等待事件是指当前的重做日志文件准备切换到下一重做日志文件，但是当前重做日志文件因为没有被归档而导致等待，这个等待事件只出现于采用了归档方式的 Oracle 数据库中。

如果出现这个等待事件，首先应该查看 Oracle 数据库的告警日志文件，看是否因为写归档日志文件错误导致归档进程停止，其次，可以增加归档进程的数量或者将归档日志文件存放到 I/O 速度比较快的磁盘上，还可以通过增大和增加重做日志文件的大小和数量来给予归档更多的时间。

出现该等待，可能表示 I/O 存在问题、归档进程写出缓慢、日志切换太快，也有可能是日志组设置不合理、日志文件太小、redo 生成太多等原因导致。针对不同原因，可以考虑采用的解决方法有：

① 可以考虑增大日志文件和增加日志组；

② 移动归档文件到快速磁盘；

③ 调整 log_archive_max_processes 参数等；

2. log file switch (checkpoint incomplete)，即日志切换（检查点未完成）

当一个在线日志切换到下一个在线日志时，必须保证要切换到的在线日志上的记录的信息（比如一些脏数据块产生的 redo log）被写到磁盘上（checkpoint），这样做的原因是，如果一个在线日志文件的信息被覆盖，而依赖这些 redo 信息做恢复的数据块尚未被写到磁盘上（checkpoint），此时系统 down 掉的话，Oracle 将没有办法进行实例恢复。当所有的日志组都写满之后。LGWR 试图覆盖某个日志文件，如果这时数据库没有完成写出由这个日志文件所保护的脏数据时（检查点未完成），该等待事件出现。该等待出现时，数据库同样将陷于停顿状态。

在日志切换时，会完成一个检查点操作，如果此检查点完成的过于缓慢，就会造成此事件的等待，检查点为什么会缓慢呢?可能是 buffer cache 太大因此容纳的脏块太多，DBWR 进程太少，调整检查点频率的参数设置频率太低等原

因造成的.

在 v$log 视图里记录了在线日志的状态。 通常来说，在线日志有三种状态。

Active：这个日志上面保护的信息还没有完成 checkpoint。

Inactive： 这个日志上面保护的信息已完成 checkpoint。

Current： 当前的日志。

Oracle 在做实例恢复时，会使用状态为 current 和 Active 的日志进行实例恢复。

如果系统中出现大量的 log file switch (checkpoint incomplete) 等待事件，原因可能是日志文件太小或者日志组太少，DBWR 写出速度太慢或者 I/O 存在问题，所以解决的方法是，考虑增加额外的 DBWR 或者增加日志文件的大小或者增加日志组的数量。

同时警告日志文件中会记录如下信息：

```
Fri Nov 18 14:26:57 2005
Thread 1 cannot allocate new log, sequence 7239
Checkpoint not complete
Current log# 5 seq# 7238 mem# 0: /opt/oracle/oradata/hsmkt/redo05.log
```

增加日志：

```
alter database add logfile thread 1  group 3 ('/oradata/backera3/redo03.log') size 256M;
alter database add logfile thread 2  group 4 ('/oradata/backera3/redo04.log') size 256M;
```

3. log file switch completion

这个等待事件是指由于当前重做日志文件已经被写满了而 Oracle 后台进程 LGWR 需要完成写完当前重做日志文件并且要打开一个新的重做日志文件而导致的重做日志文件切换的等待，或者是其他请求需要切换重做日志文件导致等待。

如果当前的重做日志写满了，这个时候 Oracle 数据库就需要切换重做日志文件来提供足够的磁盘空间给重做日志写日志缓存。但是由于一些其他的进程也同样可以引起重做日志的切换，Oracle 数据库不会同时去切换重做日志两次，因此，就出现了这个等待事件，在 Oracle 数据库早期的版本中还有

log_file_switch_checkpoint_incomplete、log_file_switch_archiving_needed、

log_file_switch_clearing_log_file 的等待事件。

当一个日志文件满了，oracle 要打开另一个日志文件，写完上一日志文件，准备好下一日志文件，这之间的等待就是此等待事件了，简单点说，就是为了完成日志文件切换而发生的等待.

4. log file switch (clearing log file)

这发生在 DBA 发布 alter system clear log file 命令.且 LGWR 正需要切换到被清空的日志文件.等待时间是 1 秒.很少见。

5. log file switch (private strand flush incomplete)

很少见。

1.4.2 log file sync（日志文件同步）

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIke 'log file sync';
```

EVENT#	EVENT_ID	NAME	PARAME	PARAME	PARAMET	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
143	1328744198	log file sync	buffer#	sync scn		3386400367	5	Commit

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 buffer#代表在日志缓冲区中需要被写入到重做日志文件中的缓存的数量，即 redo buffer 中需要被写入到磁盘中的 buffer。写入的同时会确认事务是否已经被提交，并且保留提交信息到实例意外中断之前，因此必须等待 LGWR 将 P1 数量的缓存写入重做日志文件为止。P2、P3 属于无用的参数。

更多请参考 metalink：1626301.1，地址：http://blog.itpub.net/26736162/viewspace-2124856/

文档 1626301.1 故障排除 log file sync 等待 (文档 ID 1626301.1) .mhtml

英文版：Troubleshooting: 'Log file sync' Waits (文档 ID 1376916.1)

此等待事件用户发出提交或回滚声明后，等待提交完成的事件，提交命令会去做日志同步，也就是写日志缓存到日

志文件， 在提交命令未完成前,用户将会看见此等待事件,注意,它专指因提交,回滚而造成的写缓存到日志文件的等待.当发生此等待事件时,有时也会伴随 log file parallel write.因为此等待事件将会写日志缓存，如果日志的 I/O 系统较为缓慢的话,这必将造成 log file parallel write 等待.当发生 log file sync 等待后,判断是否由于缓慢的日志 I/O 造成的,可以查看两个等待事件的等待时间,如果比较接近,就证明日志 I/O 比较缓慢或重做日志过多,这时,造成 log file sync 的原因是因为 log file parallel write,可以参考解决 log file parallel write 的方法解决问题,如果 log file sync 的等待时间很高,而 log file parallel write 的等待时间并不高,这意味着 log file sync 的原因并不是缓慢的日志 I/O,而是应用程序过多的提交造成的.

当一个用户提交或回滚数据时，LGWR 将会话期的重做由日志缓冲区写入到重做日志中，LGWR 完成任务以后会通知用户进程。日志文件同步过程（Log File Sync）必须等待这一过程成功完成。对于回滚操作，该事件记录从用户发出 Rollback 命令道回滚完成的时间。如果该等待过多，可能说明 LGWR 的写出效率低下，或者系统提交过于频繁。针对该问题，可以通过 log file parallel write 等待事件或 User Commits、User Rollback 等统计信息来观察提交或回滚次数。

这是一个用户会话行为导致的等待事件，当一个会话发出一个 commit 命令时，LGWR 进程会将这个事务产生的 redo log 从 log buffer 里面写到磁盘上，以确保用户提交的信息被安全地记录到数据库中。

当一个用户会话提交，会话的重做信息需要从内存刷新到重做日志文件，使其永久化。

这个等待事件是指等待 Oracle 的前台的 COMMIT 和 ROLLBACK 操作进程完成 ,有时候这个等待事件也会包括等待 LGWR 进程把一个会话事务的日志记录信息从日志缓冲区中写入到磁盘上的重做日志文件中。因此，当前台进程在等待这个事件的时候，LGWR 进程同时也在等待事件 log file parallel write。理解什么造成这个等待事件的关键在于对比这个等待事件和 log file parallel write 等待事件的平均等待时间：如果它们的等待时间差不多，那么就是重做日志文件的 I/O 引起了这个等待事件，则需要调整重做日志文件的 I/O，这个在之后会有详细的讲述。如果 log file parallel write 等待事件的平均等待时间明显小于 log file sync 等待事件的等待时间，那么就是一些其他的写日志的机制在 COMMIT 和 ROLLBACK 操作的时候引起了等待，而不是 I/O 引起的等待，例如重做日志文件的 latch 的竞争，会伴随着出现 latch free 或者 LGWR wait for redo copy 等待事件。

在提交时,用户会话会通知 LGWR 把日志缓冲区中的信息写到重做日志文件(当前所有未被写入磁盘的 redo 信息,包括本次会话的 redo 信息)。当 LGWR 完成写操作后,它会通知用户会话。当等待 LGWR 通知确认所有 redo 已经安全的保存到磁盘的过程时,用户会话会等待'log file sync'。

用户会话显示等待'log file sync',是用户会话通知 LGWR 和 LGWR 通知用户的写操作完成之间的时间。

需要注意的是,如果已有一个正在进行的同步,其他需要提交的会话(为了保存日志信息)也需等待 LGWR,进而也将等待'log file sync'?

当系统中出现大量的 log file sync 等待事件时,应该检查数据库中是否有用户在做频繁的提交操作。这种等待事件通常发生在 OLTP 系统上。 OLTP 系统中存在很多小的事务,如果这些事务频繁被提交,可能引起大量的 log file sync 的等待事件。

如果这个等待事件在整个等待时间中占了比较大的比重,可以从以下几个方面来调整这个等待事件:

1).**调整 LGWR 进程使其具有更好的磁盘 I/O 吞吐量**,尽量使用快速磁盘,不要把 redo log file 存放在 RAID5 的磁盘上;RAID5 对于频繁写入得系统会带来较大的性能损失,可以考虑使用文件系统直接输入/输出,或者使用裸设备(raw device),这样可以获得写入的性能提高。

2). **使用批量提交** 如果存在很多执行时间很短的事务,可以考虑将这些事务集合成一个批处理事务以减少提交的次数,因为每次提交都需要确认相关的日志写入重做日志文件,因此使用批处理事务来减少提交的次数是一种非常行之有效的减少 I/O 的方法。

3). **适当使用 NOLOGGING/UNRECOVERABLE 等选项**,查看是否一些操作可以安全的使用 NOLOGGING 或者 UNRECOVERABLE 选项,这样可以减少日志的产生。

用户应该搜集那些信息,来初步分析'log file sync'等待事件?

初步分析等待'log file sync',下面的信息是有帮助的:

- 没有'log file sync'等待的类似时间的 AWR 报告,作为用于比较的性能基线
- 'log file sync'等待发生期间的 AWR 报告 注:2 个报告应在 10-30 分钟之间。

- LGWR 日志文件　当'log file parallel wait'高的时候，LGWR 日志文件将会显示警告信息

什么原因造成了很高的'log file sync'等待？

'log file sync' 可以在用户会话通知 LGWR 写日志，和 LGWR 写完日志后通知用户会话，及用户会话被唤醒间的任何一个点发生。

更多详情，请参照文档：

Document:34592.1 WAITEVENT: "log file sync"

其中的最常见的原因：

- **影响 LGWR 的 I/O 性能问题**
- **过多的应用程序 commit**

这些原因以及如何解决它们详情概述如下：

影响 LGWR 的 IO 性能问题

我们在这里回答的主要问题是“是否 LGWR 写入磁盘慢？”，下面的步骤可以帮助确定是否是这个导致的。

比较'log file sync'和'log file parallel write'的平均等待时间。

等待事件'log file parallel write'表示 LGWR 正在等待写 redo 操作。该事件的持续时间就是等待 IO 操作部分的时间。关于'log file parallel write'的更多信息，请参阅：

Document:34583.1 WAITEVENT: "log file parallel write" Reference Note

结合事件“log file sync”看同步操作消耗在 IO 的时间，由此推断，有多少处理时间消耗在 CPU 上。

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
log file sync	868,667	62,544	72	58.29	Commit

Foreground Wait Events

- s - second, ms - millisecond - 1000th of a second
- Only events with Total Wait Time (s) >= .001 are shown
- ordered by wait time desc, waits desc (idle events last)
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% DB time
log file sync	868,667	0	62,544	72	1.01	58.29

Background Wait Events

- ordered by wait time desc, waits desc (idle events last)
- Only events with Total Wait Time (s) >= .001 are shown
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
log file parallel write	93,144	0	5,495	59	0.11	40.31

上面的例子显示了'log file sync' 和 'log file parallel write' 都有很高的等待时间

如果'log file sync'的时间消耗在'log file parallel write'上的比例高，那么大部分的等待时间是由于 IO（等待 redo 写入）。应该检查 LGWR 在 IO 方面的性能。作为一个经验法则，'log file parallel write'平均时间超过 20 毫秒，意味着 IO 子系统有问题。

建议：

1. 与系统管理员一起检查重做日志所在的文件系统的位置，以提高 IO 性能。
2. 不要把重做日志放在需要额外计算的 RAID 上，比如 RAID-5 或者 RAID-6
3. 不要把重做日志放在 Solid State Disk (SSD)

   虽然通常情况下，SSD 写入性能好于平均水平，他们可能会遇到写峰值，从而导致大量的增加'log file sync'

   等待（关于这一点您需要详尽的测试，因为我们也碰到一些 SSD 的性能可以接受的系统）

   (Engineered Systems (Exadata, SuperCluster 和 Oracle Database Appliance) 除外，因为在

   这些系统上已经为使用 SSD 来存放重做日志而做了额外的优化）
4. 监控其他可能需要写到相同路径的进程，确保该磁盘具有足够的带宽，足以应付所要求的容量。如果不能满足，

   移动这些进程或 redo。
5. 确保 LOG_BUFFER 不要太大，一个非常大的 log_buffer 的不利影响就是刷新需要更长的等待时间。当缓冲

   区满了的时候，它必须将所有数据写入到重做日志文件。LGWR 将一直等待，直到最后的 I/O 完成。

间歇性物理 IO 缓慢对 'log file sync' 等待事件的影响：

LGWR 倾向于做很多小的 IO 操作，而不是大块的 IO 操作。大部分的磁盘配置并不能在这种场景下很好的工作，可能会发生间歇性物理 IO 缓慢。但是从平均等待时间来看，IO 等待的时间并不长，磁盘设备提供商据此断定没有磁盘问题。 因为系统里还有其它的 IO 操作，所有这些正常的 IO 操作的等待时间很短，所有这些 IO 操作平均起来的等待时间并不长，这就掩盖了间歇性物理 IO 缓慢的问题。 但是间歇性物理 IO 缓慢的问题会造成很高的'log file sync', 虽然平均的'log file parallel write'是处于正常性能的范围

如果你发现系统的'log file sync'很高，但是'log file parallel write'是处于正常的范围，那么这可能是由于间歇性物理 IO 缓慢导致的。你需要使用一些像 OSWatcher 一样的工具(参照 Document 301137.1)来确定是否系统中存在间歇性物理 IO 缓慢。如果可以确定存在间歇性物理 IO 缓慢，那么你需要与磁盘提供商一起来解决这个问题。

检查 LGWR trace

尽管'log file parallel write'的平均等待时间可能在一个合理的区间范围内，在峰值时刻写操作时间还是可能会很长进而影响’log file sync’的等待时间。从 10.2.0.4 开始如果写操作超过 500 毫秒我们会在 LGWR 的 trace 中写警告信息。这个阀值很高所以就算没有警告也不代表没有问题。警告信息如下：

```
   *** 2011-10-26 10:14:41.718
Warning: log write elapsed time 21130ms, size 1KB
(set event 10468 level 4 to disable this warning)

*** 2011-10-26 10:14:42.929
Warning: log write elapsed time 4916ms, size 1KB
(set event 10468 level 4 to disable this warning)
```

注意：上面的峰值如果时间间隔的很远，可能不会对'log file parallel wait'有大的影响。 但是，如果有 100 个会话等待'log file parallel wait'完成，'log file sync'总等待可能就会很高，因为等待时间将被乘以会话的个数 100。因此，值得探讨日志写 IO 高峰的原因。

请参阅：

Document:601316.1 LGWR Is Generating Trace file with "Warning: Log Write Time 540ms, Size 5444kb" In 10.2.0.4 Database

建议：

- 与系统管理员一起检查其他正在发生的可能会导致 LGWR 写磁盘峰值的操作
- 当 LGWR 进程慢的时候，对其进行 Truss 操作会帮助确定时间消耗在什么地方

注意：这些警告信息对于预防潜在问题的发生很有帮助。就算平均等待时间没问题，通过找到 I/O 性能峰值点，DBA 可以知道 LGWR 会间歇性的遇到性能问题，进而在它引发更大问题前将其解决。

检查在线重做日志是否足够大

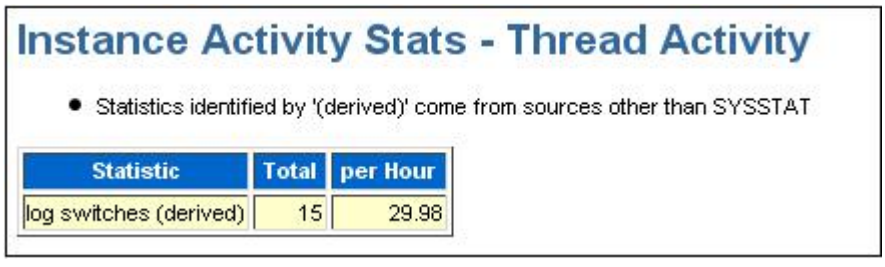
每次重做日志切换到下一个日志时，会执行'log file sync'操作，以确保下一个日志开始之前信息都写完。 标准建议是日志切换最多 15 至 20 分钟一次。 如果切换比这更频繁，那么将发生更多的'log file sync'操作，意味着更多的会话等待。

- 检查 alert.log 日志文件切换的时间

```
   Thu Jun 02 14:57:01 2011
Thread 1 advanced to log sequence 2501 (LGWR switch)
Current log# 5 seq# 2501 mem# 0: /opt/oracle/oradata/orcl/redo05a.log
Current log# 5 seq# 2501 mem# 1: /opt/oracle/logs/orcl/redo05b.log
Thu Nov 03 14:59:12 2011
Thread 1 advanced to log sequence 2502 (LGWR switch)
Current log# 6 seq# 2502 mem# 0: /opt/oracle/oradata/orcl/redo06a.log
Current log# 6 seq# 2502 mem# 1: /opt/oracle/logs/orcl/redo06b.log
Thu Nov 03 15:03:01 2011
Thread 1 advanced to log sequence 2503 (LGWR switch)
Current log# 4 seq# 2503 mem# 0: /opt/oracle/oradata/orcl/redo04a.log
Current log# 4 seq# 2503 mem# 1: /opt/oracle/logs/orcl/redo04b.log
```

在上面的例子中，我们看到每 2 到 4 分钟进行日志切换，这比建议值的 5 倍还高。

您也可以检查 AWR 报告日志切换的平均时间



Instance Activity Stats - Thread Activity

- Statistics identified by '(derived)' come from sources other than SYSSTAT

Statistic	Total	per Hour
log switches (derived)	15	29.98

在上面的例子中基于 AWR 中的信息，每小时有 29.98 次重做日志切换：每 2 分钟切换一次。这个比每 15-20 分钟切换一次的建议值要高,并将影响前台进程需要等待'log file sync'完成的时间，因为发起同步操作的开销比必要的多。

建议：增加 redo logs 的大小

Document:602066.1 How To Maintain and/or Add Redo Logs
Document:779306.1 How To Add/Increase The Size Of Redo Log Files In Rac Environment?

应用程序提交过多

在这种情况下，要回答的问题是“是否应用程序 commit 过于频繁？”。

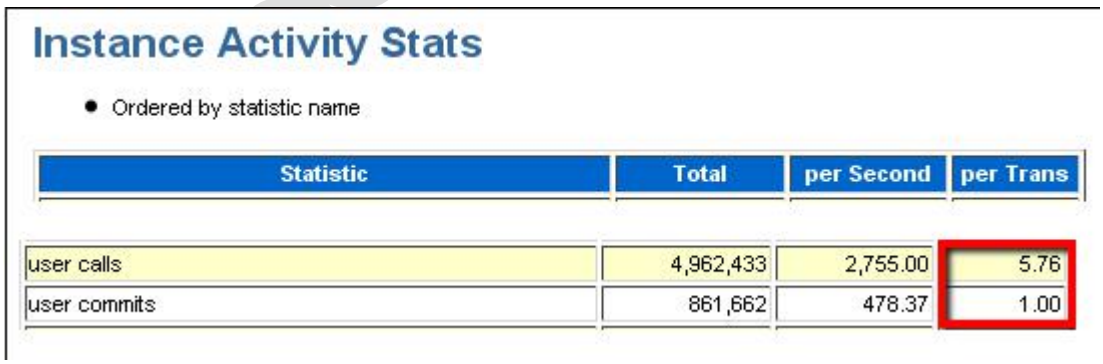
如果是，那么过多的 commit 活动可能会导致性能问题，因为把 redo 从日志缓冲区刷新到重做日志可能会导致等待'log file sync'。

如果’log file sync’的平均等待时间比’log file parallel write’高很多，这意味着大部分时间等待不是由于等待 redo 的写入，因而问题的原因不是 IO 慢导致。剩余时间是 CPU 活动，而且是过多的 commit 导致的最常见的竞争。

此外，如果'log file sync'的平均等待时间低，但等待次数高，那么应用程序可能 commit 过于频繁。

比较 user commit/rollback 同 user calls 比值的平均值：

在 AWR 或 Statspack 报告中，如果每次 commit/rollback 的平均 user calls（"user calls/(user commits+user rollbacks)"）小于 30，表明 commit 过于频繁



Instance Activity Stats

- Ordered by statistic name

Statistic	Total	per Second	per Trans
user calls	4,962,433	2,755.00	5.76
user commits	861,662	478.37	1.00

在上面的例子中，我们看到，平均每 5.76 次 user calls 就会有一次 commit，大约高出建议值 5 倍。基于经验，我们期望 user calls/user commit 至少是 25。当然，这取决于应用程序。

建议：

- 如果有很多短事务，看是否可能把这些事务组合在一起，从而减少 commit 操作。 因为每一个 commit 都必须收到相关 REDO 已写到磁盘上的确认，额外的 commit 会显著的增加开销。虽然 Oracle 可以将某些 commit 组合在一起，通过事务的批处理来减少 commit 的总体数量还是可以带来非常有益的效果。
- 看看是否有操作可以使用 COMMIT NOWAIT 选项 （务必在使用前应明白语义）。
- 看看是否有操作可以安全地使用 NOLOGGING/ UNRECOVERABLE 选项完成。

其他可能相关的等待事件：

检查 AWR 报告，看是否有跟 LGWR 相关的，显示占用了显著数量时间的其他事件，因为这可能会给出导致这个问题的一个线索。前台和后台事件都应该进行检查。

例如下面的 AWR 显示某些其他前台和后台等待事件等待高，意味着传输重做日志到远程位置的问题，这可能会导致 fore gorund 进程等待"log file sync"。

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
log file sync	32,108	32,902	1025	94.09	Commit
log file switch completion	35	1,008	28813	2.88	Configuration
DB CPU		511		1.46	
buffer busy waits	874	267	305	0.76	Concurrency
control file sequential read	473,353	105	0	0.30	System I/O

After 'log file sync', the next wait event is 'log file switch completion'.
Each switch takes an average of 28813ms to complete.

Background Wait Events

- ordered by wait time desc, waits desc (idle events last)
- Only events with Total Wait Time (s) >= .001 are shown
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
LNS wait on SENDREQ	22,290	0	3,426	154	0.71	46.77
LGWR-LNS wait on channel	324,024	91	3,334	10	10.32	45.51

High percentage of time is spent on events realted to transporting redo logs.

Adaptive Log File Sync

11.2 中引入了 Adaptive Log File sync，由参数 _use_adaptive_log_file_sync 控制，在 11.2.0.1 和 11.2.0.2 默认设置为 false。

从 11.2.0.3 开始默认是 true。 当启用时，Oracle 可以在两种方法之间切换：

- Post/wait，传统发布写重做日志完成的方法
- Polling，一种新的方法，其中前台进程会检查 LGWR 是否已写完成。

更多关于这个特性的信息，请参阅：

Document 1541136.1 Waits for "log file sync" with Adaptive Polling vs Post/Wait Choice Enabled

Redo Synch Time Overhead：

统计值'redo synch time overhead'在 11.2.0.4 和 12c 被引入，记录了理想的 log file sync 时间以及真正的 log file sync 时间的差值。

如果这个差值很小，说明 log file sync 等待次数可能是 log file parallel write 等待之外的原因导致的

当一个用户提交或回滚数据时，LGWR 将会话期的重做由 Log Buffer 写入到重做日志中，

LGWR 完成任务以后会通知用户进程。日志文件同步等待（Log File Sync）就是指进程等待

LGWR 写完成这个过程；对于回滚操作，该事件记录从用户发出 rollback 命令到回滚完成的时

间。

如果该等待过多，可能说明 LGWR 的写出效率低下，或者系统〃交过于频繁。针对该问

题，可以关注 log file parallel write 等待事件，或者通过 user commits,user rollback 等统计信息

观察〃交或回滚次数。

可能的解决方案主要有：

l 〃高 LGWR 性能，尽量使用快速磁盘，不要把 redo log file 存放在 RAID5 的磁盘上；

l 使用批量〃交；

l 适当使用 NOLOGGING/UNRECOVERABLE 等选项。

可以通过如下公式计算平均 Redo 写大小：

avg.redo write size = (Redo block written/redo writes)*512 bytes

如果系统产生 Redo 很多，而每次写的较少，一般说明 LGWR 被过于频繁地激活了。可

能导致过多的 Redo 相关 Latch 的竞争，而且 Oracle 可能无法有效地使用 piggyback 的功能。

从一个 Statspack 报告中〃取一些数据来研究一下这个问题。

Report 概要信息如下：

```
DB Name DB Id Instance Inst Num Release OPS Host
------------ ----------- ------------ -------- ----------- --- ------------
DB 1222010599 oracle 1 9.1.7.4.5 NO sun
```

```
Snap Id Snap Time Sessions
------- ------------------ --------
Begin Snap: 3473 13-Oct-04 13:43:00 540
End Snap: 3475 13-Oct-04 14:07:28 540
Elapsed: 24.47 (mins)
Cache Sizes
~~~~~~~~~~~
db_block_buffers: 102400 log_buffer: 20971520
db_block_size: 8192 shared_pool_size: 600M
Load Profile
~~~~~~~~~~~~ Per Second Per Transaction
--------------  ---------------
Redo size: 28,459.11 2,852.03
```

等待事件如下：

```
Event  Waits Timeouts Time (cs) (ms) /txn
---------------------------- ------------ ---------- ---------- ------ ------
log file sync  14,466 2 4,150 3 1.0
db file sequential read  17,202 0 2,869 2 1.2
latch free 24,841 13,489 2,072 1 1.7
direct path write    121 0 1,455 120 0.0
db file parallel write  1,314 0 1,383 11 0.1
log file sequential read 1,540 0 63 0  0.1
log file switch completion 1 0  3 30 0.0
refresh controlfile command 23 0 1 0 0.0
LGWR wait for redo copy  46 0  0 0 0.0
log file single write 4 0  0 0 0.0
```

注意以上输出信息，这里 log file sync 和 db file parallel write 等等待事件同时出现，那么

可能的一个原因是 I/O 竞争导致了性能问题，实际用户环境正是日志文件和数据文件同时存放

在 RAID5 的磁盘上，存在性能问题需要调整。

统计信息如下：

```
Statistic Total per Second per Trans
--------------------------------- ---------------- ------------ ------------
……
redo blocks written 93,853 63.9 6.4
redo buffer allocation retries 1 0.0 0.0
redo entries 135,837 92.5 9.3
redo log space requests 1 0.0 0.0
redo log space wait time 3   0.0 0.0
redo ordering marks 0 0.0 0.0
redo size 41,776,508 28,459.1 2,852.0
redo synch time 4,174 2.8 0.3
redo synch writes 14,198 9.7 1.0
```

```
redo wastage 4,769,200 3,249.8 325.6
redo write time 3,698 2.5 0.3
redo writer latching time   0 0.0 0.0
redo writes 14,572 9.9 1.0
……
sorts (disk) 4 0.0 0.0
sorts (memory) 179,856 122.5 12.3
sorts (rows) 2,750,980 1,874.0 187.8
……
transaction rollbacks 36 0.0 0.0
transaction tables consistent rea 0 0.0   0.0
transaction tables consistent rea 0 0.0 0.0
user calls 1,390,718 947.4 94.9
user commits 14,136 9.6 1.0
user rollbacks   512 0.4 0.0
write clones created in backgroun 0 0.0 0.0
write clones created in foregroun 11 0.0 0.0
-------------------------------------------------------------
```

根据统计信息可以计算平均日志写大小：

```
avg.redo write size = (Redo block written/redo writes)*512 bytes
= ( 93,853 / 14,572 )*512
= 3KB
```

这个平均值过小了，说明系统的〃交过于频繁。从以上的统计信息中，可以看到平均每

秒数据库的〃交数量是 9.6 次。如果可能，在设计应用时应该选择合适的〃交批量，从而〃高

数据库的效率。

```
Latch Sleep breakdown for DB: DPSHDB Instance: dpshdb Snaps: 3473 -3475
-> ordered by misses desc
Get Spin &
Latch Name Requests Misses Sleeps Sleeps 1->4
-------------------------- -------------- ----------- ----------- ------------
row cache objects 12,257,850 113,299 64 113235/64/0/0/0
shared pool 3,690,715 60,279 15,857 52484/588/6546/661/0
library cache 4,912,465 29,454 8,876 23823/2682/2733/216/0
cache buffers chains 10,314,526 2,856 33 2823/33/0/0/0
redo writing  76,550 937   1 936/1/0/0/0
session idle bit   2,871,949 225 1 224/1/0/0/0
messages   107,950 159 2 157/2/0/0/0
session allocation 184,386 44 6 38/6/0/0/0
checkpoint queue latch 96,583   1 1 0/1/0/0/0
-------------------------------------------------------------
```

由于过度频繁的〃交，LGWR 过度频繁的激活，看到这里出现了 redo writing 的 latch 竞

争。

以下是一则 ASH 报告中显示的 Log File Sync 等待信息，注意到其 Parameter 1 是 Buffer#，Parameter 2 代表 Sync SCN，也就是同步的 SCN。Log File Sync 以 SCN 为节点，以 Buffer 号为起始，不断将 Log Buffer 的内容写出到日志文件上来：

Top Event P1/P2/P3 Values

Event	% Event	P1 Value, P2 Value, P3 Value	% Activity	Parameter 1	Parameter 2	Parameter 3
db file sequential read	10.81	"7","809389","1"	1.35	file#	block#	blocks
		"8","1749501","1"	1.35			
		"9","350240","1"	1.35			
log file sync	10.81	"2268","1849513965","0"	1.35	buffer#	sync scn	NOT DEFINED
		"5060","1849596977","0"	1.35			
		"5982","1849465171","0"	1.35			
log file parallel write	8.11	"1","2","1"	5.41	files	blocks	requests
		"1","3","1"	1.35			
		"1","49","1"	1.35			

1.4.3 log file parallel write

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIke 'log file parallel write';
```

EVENT#	EVENT_ID	NAME	PARAME	PARAME	PARAMET	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
134	3999721902	log file parallel write	files	blocks	requests	4108307767	9	System I/O

这个等待事件有三个参数：

Files： 操作需要写入的文件个数。

Blocks： 操作需要写入的数据块个数。

Requests：操作需要执行的 I/O 次数

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 代表正在被写入的重做日志文件组中的重做日志文件号，P2 代表需要写入重做日志组中每个重做日志文件的重做日志 BLOCK 数量，P3 代表 I/O 请求的次数，需要被写入的 BLOCK 会被分成多次分别请求。

从 Log Buffer 写 Redo 记录到日志文件，主要指常规写操作（相对于 Log File Sync）。如果 Log Group 存在多个组成员，当 Flush Log Buffer 时，写操作是并行的，这时候此等待事件可能出现。尽管这个写操作并行

处理，直到所有 I/O 操作完成该写操作才会完成(如果你的磁盘支持异步 IO 或者使用 IO SLAVE，那么即使只有一个 redo log file member,也有可能出现此等待)。后台进程 LGWR 负责将 log buffer 当中的数据写到 REDO 文件中，以重用 log buffer 的数据。 如果每个 REDO LOG 组里面有 2 个以上的成员，那么 LGWR 进程会并行地将 REDO 信息写入这些文件中。

这个等待事件出现在当 LGWR 后台进程从日志缓冲区写日志信息到磁盘上的重做日志文件的时候。只有启用了异步 I/O 的时候 LGWR 进程才会并行写当前日志组内的重做日志文件，否则 LGWR 只会循环顺序逐个的写当前日志组重做日志文件。LGWR 进程不得不等待当前日志组所有的重做日志文件成员全部写完，因此，决定这个等待事件的等待时间长短的主要因素是重做日志文件所在磁盘的 I/O 读写的速度。

这个参数和 log file sync 时间相比较可以用来衡量 log file 的写入成本。通常称为同步成本率。

如果数据库中出现这个等待事件的瓶颈 ,主要的原因可能是磁盘 I/O 性能不够或者 REDO 文件的分布导致了 I/O 争用，比如同一个组的 REDO 成员文件放在相同的磁盘上。

如果是当前的 LGWR 进程写的速度不够快导致了这个等待事件，可以通过查看一些和重做日志相关的统计值判定当前的 LGWR 进程是否效率很低，具体的可以查看"redo writes"、"redo blocks written"、"redo write time"、"redo wastage"、"redo size"统计值，这些都是和 LGWR 进程性能直接相关的一些统计值。

如果这个等待事件占用的等待时间比较多，可以从以下几个方面来进行调整：

- 对能使用 UNRECOVERABLE/NOLOGGING 的操作尽量使用这两个选项来减少重做日志的产生。
- 在保证不会同时丢失重做日志文件的前提下尽量减少重做日志组中的成员的个数，减少每次写重做日志组文件的时间。
- 除非在备份的情况下，否则不要在将表空间置于热备的模式下，因为表空间处于热备的模式下会产生更多的重做日志文件。
- 对于使用 LogMiner、Logical Standby 或者 Streams，在能够满足要求功能的前提下，尽量使用最低级别的追加日志以减少重做日志的产生。
- 尽量将同一个日志组内的重做日志文件分散到不同的硬盘上，减少并行写重做日志文件的时候产生的 I/O 竞争。

- 不要将重做日志文件放置在 RAID-5 的磁盘上，最好使用裸设备来存放重做日志文件。
- 如果设置了归档模式，不要将归档日志的目的地设置为存放重做日志存放的磁盘上面，避免引起 I/O 竞争。

当日志缓存到日志文件时,这是一个主要的等待事件.虽然这个时间的名字中有"并行"(parallel)字样,但即使日志缓存并没有使用并行写,因日志缓存的写出而造成的等待仍然是此等待事件.

我们可以通过 v$system_event 来了解下某一个阶段内,此等待事件的平均等待时间.通过此时间值,来评估我们的日志 I/O 是否正常.有资料介绍当 log file parallel write 的平均等待时间大于 10 毫秒时.有可能就表明着日志的吞吐量缓慢.我认为这只是一个参考值,在不同的系统上要根据不同的情况来决定.记录一些在正常情况下 log file parallel write 等待事件的平均等待时间,当出现问题后,以此时间作为是否有问题的标准.这种方法也是可取的.

当日志 I/O 确实有问题时,减少重做产生的数量,确实能够缓解 log file parallel write 的等待时间.但有时,重做信息的数量是无法减少的.根据情况,将日志 I/O 转移到更快速的磁盘上,也是解决问题的方法之一.

日志缓存的大小,有时候也会对此等待事件产生影响.如果你的日志缓存更大,会降低 LGWR 刷新缓存到磁盘的次数,增大日志的缓存,也会有助于缓解此等待事件.但过大的日志缓存,有可能会造成 LGWR 间歇性的拥堵.因为 LGWR 被触发的条件之一是日志缓存满 1/3,如果日志缓存过大,1/3 的日志缓存数量可能过多,每次 LGWR 被触发,不得不写大量数据,这造成 LGWR 间歇性的停顿与拥堵,这也会增加此等待事件的等待时间.我们可以通过设置隐藏参数 _log_io_size 来改变日志缓存满 1/3 才触发 LGWR 的阈值.通过设置此参数,我们即可以拥有较大的日志缓存,又避免了 LGWR 间歇性的停顿或拥堵.

我没有在生产库中使用过这个参数,因为他毕竟是一个隐藏参数.虽然据说他不会带来什么 bug.在我的测试机上,通过调节这个参数,确实可以对性能略有提升.但这些都是为数据库的"微调".不可能带来大幅度的性能提升.

LGWR 在刷新缓存时,需要 redo allocation 和 redo writing 闩,并且 LGWR 需要等待一些 redo copy 闩的完成.因此,如果这些闩的争用较高,则不要减少 _log_io_size 此隐藏参数,因为减少它,将会使 LGWR 更为频繁的刷新缓存.这会进一步加剧这 3 个闩的争用.减缓 LGWR 完成工作的速度.

**小小结:日志缓存到底应该设置为多大??_log_io_size 参数的值应该定为多少??这没有一个统一的标准,

只有通过多做测试才能决定.

从 log buffer 写 Redo 记录到日志文件，主要指常规写操作（相对于 log file sync）。如果每个日志组存在多个组成员，当 flush log buffer 时，写操作是并行的，这时此等待事件可能出现。尽管这个写操作并行处理，直到所有 I/O 操作完成该写操作才会完成（如果磁盘支持异步 IO 或者使用 IO SLAVE，那么即使只有一个 redo log file member，也有可能出现此等待）。这个参数和 log file sync 时间相比较可以用来衡量 log file 的写入成本，通常称为同步成本率。当数据库产生日志的速度比 LGWR 的写出速度快，或者是当日志切换（log switch）太慢时，就会发生这种等待。这个等待出现时，通常表明 redo log buffer 过小，为解决这个问题，可以考虑增大日志文件的大小，或者增加日志缓冲区的大小。

另外一个可能的原因是磁盘 I/O 存在瓶颈，可以考虑使用写入速度更快的磁盘。在允许的条件下，可以考虑使用裸设备来存放日志文件，〃高写入效率。在一般的系统中，最低的标准是，不要把日志文件和数据文件存放在一起，因为通常日志文件只写不读，分离存放可以获得性能〃升，尽量使用 RAID10 而不是 RAID5 磁盘来存储日志文件。以下是一个 log buffer 存在问题的 Statspack Top5 等待事件的系统：

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~ Wait % Total
Event Waits Time (cs) Wt Time
-------------------------------------------- ------------ ------------ -------
log file parallel write 1,436,993 1,102,188 10.80
log buffer space       16,698 873,203  9.56
log file sync 1,413,374 654,587 6.42
control file parallel write 329,777 510,078 5.00
db file scattered read 425,578 132,537 1.30
-------------------------------------------------------------
```

Log Buffer Space 等待事件出现时，数据库将陷于停顿状态，所有和日志生成相关的操作全部不能进行，所以这个等待事件应该引起充分的重视。

1.4.4 log buffer space（日志缓冲空间）

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIke 'log buffer space';
```

	EVENT#	EVENT_ID	NAME	PARAME	PARAME	PARAMET	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	137	3357856061	log buffer space				3290255840	2	Configuration

当数据库产生日志的速度比 LGWR 的写出速度快，或者当日志切换太慢时，就会发生这种等待。这个等待出现时，通常表明 Redo log buffer 过小，为解决这个问题，可以考虑增大日志文件的大小或者增加日志缓冲器的大小。

另一个可能的原因是磁盘 I/O 存在瓶颈，可以考虑使用写入速度更快的磁盘。在允许的条件下设置，可以考虑使用裸设备来存放日志文件，提高写入效率。在一般的系统中，最低的标准是，不要把日志文件和数据文件存放在一起，因为通常日志文件只写不读，分离存放可以获得性能提升，尽量使用 RAID10 而不是 RAID5 磁盘来存储日志文件。当 log buffer 中没有可用空间来存放新产生的 redo log 数据时，就会发生 log buffer space 等待事件。 如果数据库中新产生的 redo log 的数量大于 LGWR 写入到磁盘中的 redo log 数量，必须等待 LGWR 完成写入磁盘的操作，LGWR 必须确保 redo log 写到磁盘成功之后，才能在 redo buffer 当中重用这部分信息。

如果数据库中出现大量的 log buffer space 等待事件，可以考虑如下方法：

（1） 增加 redo buffer 的大小。

（2） 提升磁盘的 I/O 性能

服务器进程生成重做记录的速度快过 LGWR 写出重做记录的速度，因而发生等待．日志 I/O 缓慢是 log buffer space 等待的主要原因之一．还有一点，如果日志缓存区过小，也容易出现此等待事件．将日志缓存设置的大一些，对于缓解此事件的等待会有帮助．但是，过大的日志缓存，又会降低 LGWR 刷新缓存的频率，这可能会使提交时必须刷新的缓存数量增多．从而造成 log file sync 等待．日志缓存具体应该设置为多大，这就多进行测试咯．不同的环境下，不可能有一个标准．为了缓解 log buffer space 等待事件，将日志缓存调节的比较大之后，可以通过 _log_io_size 参数来提高 LGWR 刷新缓存的频率．这样做既可以减少 log buffer space 的等待，也可以减少 log file sync 等待．但这样的隐藏参数 应该小心使用．

1.4.5 log file sequential read

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE 'log file sequential read';
```

	EVENT#	EVENT_ID	NAME	PARAME	PARAME	PARAMET	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	132	549236675	log file sequential read	log#	block#	blocks	4108307767	9	System I/O

Waiting for the read from this logfile to return. This is used to read redo records

from the log file.

这个等待事件包含三个参数：

Log#： 发生等待时读取的 redo log 的 sequence 号。

Block#： 读取的数据块号。

Blocks： 读取的数据块个数。P3 的值为 1，一般来说都是在读取日志文件头。

```
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;
```

这个等待事件通常发生在对 redo log 信息进行读取时，比如在线 redo 的归档操作，ARCH 进程需要读取 redo log 的信息，由于 redo log 的信息是顺序写入的，所以在读取时也是按照顺序的方式来读取的。

这个等待事件是指等待读取重做日志文件中的日志记录，等待的时间就是耗费在完成整个读取日志记录的物理 I/O 操作的时间。

等待从日志文件中读，一般 ARC 进程会遭遇此事件，如果 P3 参数为 1，证明等待发生在读日志文件头，否则，P3 代表要读出的日志块的数量.

1.4.6 log file single write

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE 'log file single write';
```

	EVENT#	EVENT_ID	NAME	PARAME	PARAME	PARAMET	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	133	215477332	log file single write	log#	block#	blocks	4108307767	9	System I/O

这个等待事件包含三个参数：

Log#： 正在被写入的重做日志文件组的组号。

Block#：写入的数据块号。

Blocks：写入的数据块个数。

这个等待事件是指等待写重做日志文件操作完成，常常是在等待写重做日志文件头，例如在增加一个新的重做日志组成员的时候，Oracle 数据库就会往这个重做日志文件头写入相应的 sequence 号。

这个等待事件发生在更新 redo log 文件的文件头时，当为日志组增加新的日志成员时或者 redo log 的 sequence 号改变时，LGWR 都会更新 redo log 文件头信息。

因为 single write 通常都是在写或者重写日志文件头的时候出现，因此开始的 block 号总是为 1。一般如果出现这个等待事件，应该对重做日志文件尽量使用裸设备，避免将多个日志文件放在同一个磁盘上，减少产生 I/O 竞争的可能。

日志文件写等待，注意，这里所指的写，并不是从日志缓存写到日志文件，这里的写并不涉及日志缓存，此事件只代表写日志文件头时发生的等待.有两种情况日志文件头被写：当添加新的成员文件或日志序列号增加.应对日志文件尽量使用裸设备.或避免将日志文件放在同一磁盘上，以减少此事件产生的可能。

该事件仅与写日志文件头块相关，通常发生在增加新的组成员和增进序列号（Log switch）时。头块写单个进行，因为头块的部分信息是文件号，每个文件不同。更新日志文件头这个操作在后台完成，一般很少出现等待，无需太多关注。

1.4.7 LGWR wait for redo copy

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE '%LGWR wait for redo copy%';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	646	4266849434	LGWR wait for redo copy	copy latch #			1893977003	0	Other

LGWR 将要写一组日志块，但它必须等待直到服务器进程完成任意当前的拷贝操作，这些拷贝操作影响将要被写出的缓存。

1.4.8 switch logfile command

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE '%switch logfile command%';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	141	3845123846	switch logfile command				4166625743	3	Administrative

执行日志文件切换命令的时候等待日志文件切换完成.超时时间为 5 秒.

1.4.9 log switch/archive

```
SELECT * FROM V$EVENT_NAME A WHERE A.NAME LIKE '%log switch/archive%';
```

	EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
1	625	1722088478	log switch/archive	thread#			1893977003	0	Other

当 DBA 手动输入命令 alter system archive log change<SCN>时,可能会等待此事件.

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
- 本文 itpub 地址：http://blog.itpub.net/26736162/viewspace-2125065/
- 本文博客园地址：http://www.cnblogs.com/lhrbest/p/5878100.html
- 本文 pdf 版：http://yunpan.cn/cdEQedhCs2kFz （提取码：ed9b）
- 小麦苗云盘地址：http://blog.itpub.net/26736162/viewspace-1624453/
- QQ 群：230161599　　微信群：私聊
- 联系我请加 QQ 好友(642808185)，注明添加缘由
- 于 2016-09-13 10:00~ 2016-09-17 11:20 在公寓完成
- 文章内容来源于小麦苗的学习笔记，部分整理自网络，若有侵权或不当之处还请谅解！


小麦苗