# DB File Sequential Read

## 发生场景：

当一个进程需要从磁盘中读取数据块到数据库的缓冲区时，就会产生“DB File Sequential Read”。

最常见的情况有执行索引的访问（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。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。

这个等待事件有三个参数：

File#： 要读取的数据块锁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）。

## 工作原理：

此事件表示读取数据块的过程是顺序的，即一次读取一个块。

通常，这涉及到快速的单块I/O操作，因为索引结构设计为提供快速的数据访问路径。

## 可能原因：

1. 磁盘I/O性能： 如果磁盘I/O速度慢，这个等待事件的时间会增加。
2. 索引设计： 索引可能不是最优的，或者没有被查询有效利用。
3. 数据分布： 数据可能在磁盘上分布不均匀，导致访问某些数据时I/O需求增加。
4. 缓存命中率低： 如果所需的数据块不在数据库缓冲区中，就需要从磁盘读取，增加了I/O操作。

## 优化策略：

1. 检查索引： 确保查询使用了最合适的索引。
2. 重新构建或优化索引： 有时重新构建索引或更改索引结构可以提高性能。
3. SQL语句优化： 改进查询逻辑，以减少依赖于索引查找的操作。
4. 硬件升级： 如果I/O是瓶颈，考虑使用更快的磁盘或增加I/O子系统的能力。
5. 数据分区： 对于大型表，使用分区可以改善数据的物理组织，减少单个索引查找的时间。
6. 缓存调整： 调整数据库的缓存设置，以增加缓冲区命中率。

# DB File Scattered Read

## 发生场景：

当一个查询操作需要从磁盘上读取多个数据块到SGA（系统全局区）的缓冲区时发生。

与“DB File Sequential Read”不同的是，“DB File Scattered Read”通常涉及一次性读取多个块并将它们“分散”到缓冲区的不同部分。

最常见的两种情况是全表扫描（FTS： Full Table Scan）和索引快速扫描（IFFS： index fast full scan）。在 Full Table Scan 或 Fast Full Scan 等访问方式下使用。

## 工作原理：

这个过程用于处理大量数据，如全表扫描或批量数据操作。

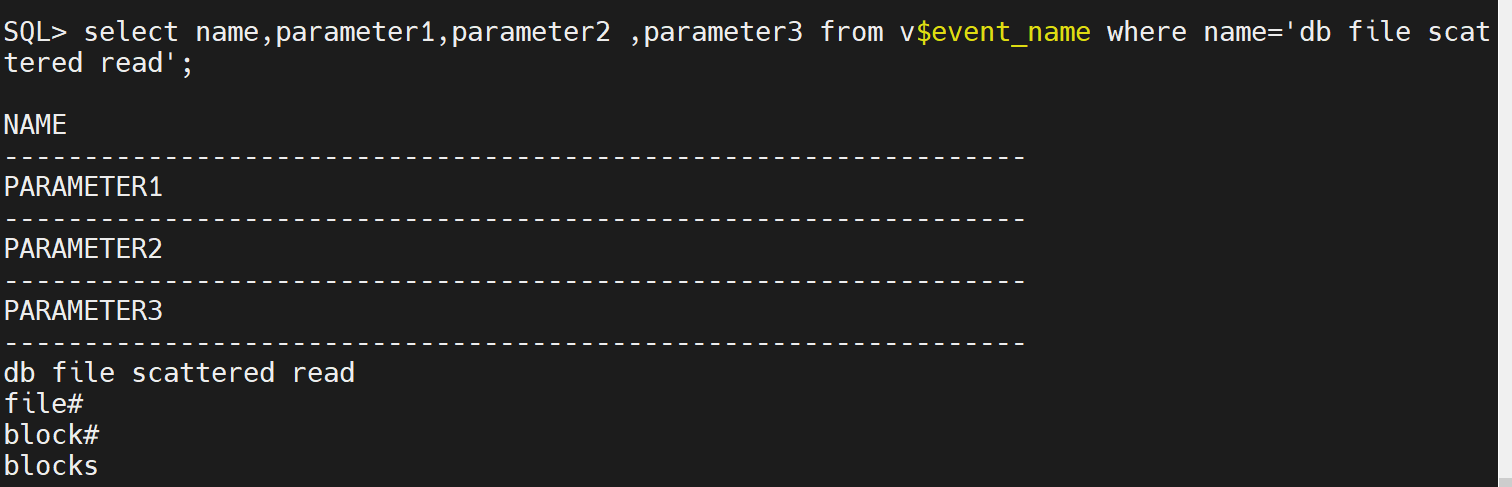
“分散”读取意味着从磁盘读取的数据块被放置到SGA的不同位置，以便于并行处理和优化内存利用。

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：需要读取的数据块数目。



## 可能原因：

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。

全表扫描： 当没有有效的索引可用时，数据库可能执行全表扫描。

大范围索引扫描： 即使使用了索引，如果需要检索的数据范围很大，也可能引发此事件。

磁盘I/O性能： 如果磁盘读取速度慢，这会导致等待时间增加。

缓存效率低： 如果所需数据块不在缓冲区，就需要频繁地从磁盘读取。

## 优化策略：

优化索引： 确保查询有效地使用索引来减少对全表扫描的依赖。

改进查询逻辑： 重新设计查询，以减少需要检索的数据量。

分区： 对大表进行分区，以减少单次查询需要扫描的数据量。

硬件改进： 提高磁盘I/O性能，比如使用更快的硬盘或配置更优的存储解决方案。

增加缓冲区大小： 调整SGA设置，增加缓冲区大小，以便存储更多的数据块，减少磁盘I/O操作。

# Log File Sync

## 发生场景：

这是一个用户会话行为导致的等待事件，当一个事务进行提交时，会话发出一个commit，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。会话发出的commit指令后，需要等待LGWR将这个事务产生的redo成功写入到磁盘之后，才可以继续进行后续的操作，这个等待事件就叫作log file sync。

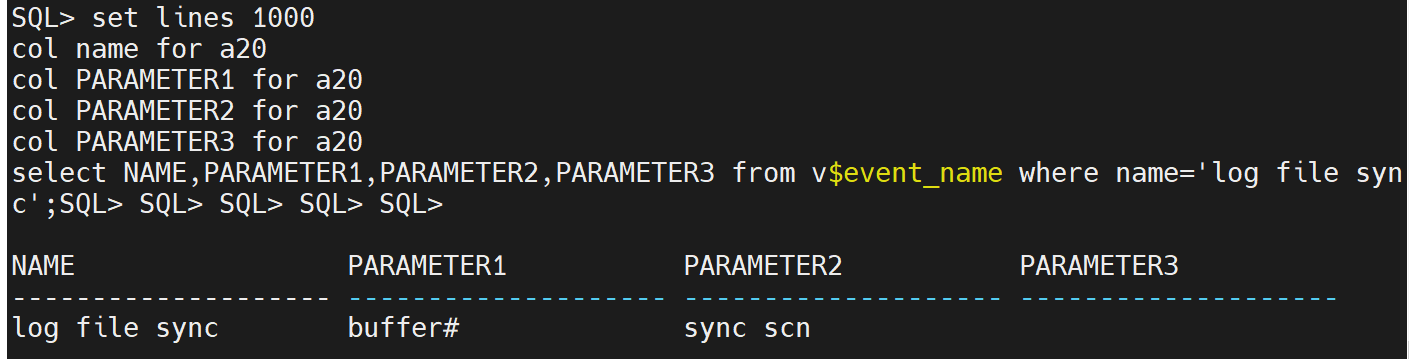
这个过程确保了数据库的持久性和一致性，即使在系统崩溃的情况下也能恢复事务。

## 工作原理：

事务提交时，相关的重做日志信息（记录了事务所做的更改）需要从日志缓冲区写入到磁盘上的重做日志文件。

"Log File Sync" 等待事件发生在这个写入过程中，即会话等待日志写入操作完成。

这个等待事件的参数



## 可能原因：

频繁的事务提交： 如果有大量的小事务频繁提交，会导致多次的日志同步操作。

I/O 性能问题： 如果写入重做日志文件的速度慢，可能是因为磁盘I/O性能不足。

日志文件的配置： 日志文件大小、位置或数量的不合理配置可能导致性能问题。

系统资源争用： 在高并发环境中，多个会话可能同时竞争写入日志文件，导致等待。

## 优化策略：

优化事务大小： 减少小事务的数量，通过合并多个小更改为更大的事务来减少提交频率。

改善I/O性能： 使用更快的磁盘，如SSD，或优化存储系统的配置。

调整日志文件配置： 优化重做日志文件的大小和数量，以及它们的物理位置。

应用层优化： 在应用程序层面上减少不必要的事务提交。

监控和调整系统资源： 监控系统资源使用情况，确保足够的资源来处理并发的日志写入请求。

# Log File Parallel Write

## 发生场景：

在事务处理过程中，所有的更改（如插入、更新、删除操作）都会首先记录在重做日志缓冲区中。

LGWR 进程定期将这些事务记录从缓冲区写入到在线重做日志文件，以确保事务的持久性。

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。

所以"Log File Parallel Write" 等待事件指的是这个写入磁盘的过程，这是一个异步的、并行的I/O操作。

## 工作原理：

重做日志信息需要被快速且可靠地写入到日志文件，以确保在发生故障时能够恢复数据。

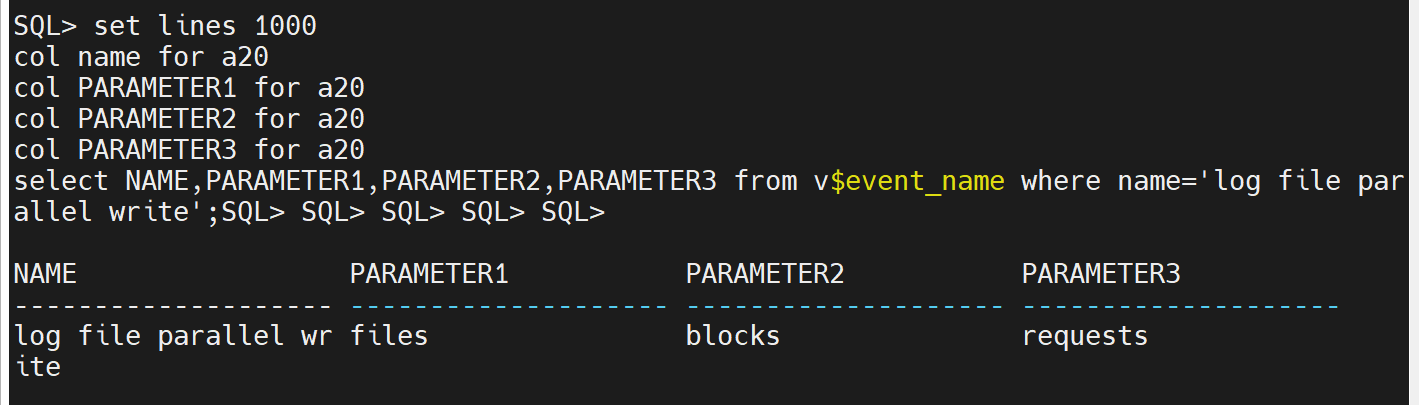
该过程通常是并行执行的，尤其在有多个重做日志文件成员的配置中。

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



## 可能原因：

I/O性能： 如果磁盘I/O速度慢，特别是重做日志文件所在的磁盘，这可能导致等待时间增加。

系统资源争用： 在高并发系统中，可能会出现多个进程同时尝试写入日志文件，导致性能瓶颈。

日志文件配置： 重做日志文件的数量、大小和分布对写入性能有重要影响。

## 优化策略：

提升I/O性能： 使用更快的磁盘（如SSD）来存储重做日志文件，或优化现有磁盘的性能。

合理配置重做日志文件： 适当增加重做日志文件的大小和数量，以及优化它们在磁盘上的布局。

负载均衡： 确保重做日志文件分布在不同的物理磁盘上，以减少单个磁盘的I/O负载。

监控系统性能： 定期监控系统的I/O性能指标，确保足够的资源处理日志写入。

# Enqueue

## 发生场景：

当多个会话试图对同一个数据库对象（如表、行、索引等）进行修改时，可能会发生锁争用。

例如，一个会话正在更新一行数据，而另一个会话也尝试更新相同的行。

## 工作原理：

表示在获取或释放enqueue锁时的等待时间。enqueue 是一种保护共享资源的锁定机制。该锁定机制保护共享资源，这些资源可以是数据结构、数据库对象或内存结构。

当一个会话请求一个资源的锁时，如果该资源已被另一个会话锁定，并且锁类型之间存在冲突，请求会话将进入队列等待状态，直到锁被释放。

当我们在AWR 报告中发现长时间的enqueue 等待事件时，说明数据库中出现了阻塞和等待，可以关联AWR报告中的enqueue activity部分来确定是哪一种锁定出现了长时间等待。

这个等待事件有2个参数：

Name： enqueue 的名称和类型。

Mode： enqueue的模式。

## 可能原因：

数据修改冲突： 当多个会话同时尝试修改相同的数据时，可能导致锁争用。

长事务： 某些会话持有锁的时间过长，导致其他会话无法获取锁。

不合理的应用设计： 应用程序设计不当可能导致频繁的锁争用。

## 优化策略：

事务管理： 确保事务尽可能短，以减少持有锁的时间。

应用程序优化： 改善应用程序逻辑，减少不必要的锁请求和长时间的事务。

避免热点数据： 通过设计变更，分散对特定热点数据的访问。

锁升级和降级策略： 合理管理锁的级别，尽可能使用更细粒度的锁。

监控和诊断： 使用Oracle的监控工具诊断锁争用的原因，并据此进行优化。

# Buffer Busy Waits

## 发生场景：

多个会话试图读取或修改同一个数据块，比如在高并发环境中频繁更新同一表的特定行。

它还可能发生在索引叶节点的分裂过程中，或者当多个会话尝试插入同一表的不同行时，如果这些行恰好在同一个数据块中。

## 工作原理：

在 Oracle 数据库中，数据是以块的形式在缓冲区中管理的。当一个会话需要访问某个数据块时，它会尝试在缓冲区中锁定这个块。

Oracle中，数据以块的方式在缓存区中管理，即使你只修改一条记录，也需要对这条记录所在的这个数据块做操作。

当你对这个数据块做修改时，其他的会话将被阻止对这个数据块上的数据做修改（即使其他用户修改的不是当前用户修改的数据），但是可以以一致性的方式读取这个数据块（from undo）。当前的用户修改完这个数据块后，将会立即释放掉加在这个数据块上的排他锁，这样另一个会话就可以继续修改它。修改操作是一个非常短暂的时间，这种加锁的机制我们叫Latch。

当一个会话修改一个数据块时，是按照以下步骤来完成的：

1. 以排他的方式获得这个数据块（Latch）
2. 修改这个数据块。
3. 释放Latch。

Buffer busy waits等待事件常见于数据库中存在的热块的时候，当多个用户频繁地读取或者修改同样的数据块时，如果这个块已被另一个会话锁定（例如，进行写入操作），其他会话必须等待直到锁被释放，这种等待就被记录为 "Buffer Busy Waits"。

## 可能原因：

* 高并发数据修改： 当许多会话都试图修改相同的数据块时。

当一个会话视图修改一个数据块，但这个数据块正在被另一个会话修改时。

当一个会话需要读取一个数据块，但这个数据块正在被另一个会话读取到内存中时

（在新的版本中，第二种情况已经被独立出来，以read by other session取代。）

* 索引热点： 在索引的特定部分（如叶节点）发生高并发插入操作。
* 表的全表扫描： 当多个会话同时执行针对同一表的全表扫描时。
* 不足的缓冲区空间： 如果缓冲区大小不足，可能导致更频繁的数据块加载和卸载，增加了锁的竞争。

## 优化策略：

重新设计数据访问： 分散对同一数据块的访问，例如通过应用程序逻辑更改或数据库设计调整。

增加ITL槽： 在频繁更新的块中增加事务插槽（ITL - Interested Transaction List）的数量。

分区和索引管理： 对于频繁更新的大表，使用分区可以帮助分散访问。同时，优化索引以减少对特定数据块的访问。

增加缓冲区大小： 增加数据库缓冲区的大小可以减少数据块的重复加载。

监控和调整： 使用Oracle提供的工具和视图监控缓冲区的使用情况，并根据需要进行调整。

# Row Lock Waits 行锁等待

## 发生场景：

当一个事务试图修改（例如，更新或删除）一行数据，而这行数据已经被另一个事务锁定，当前事务就会进入等待状态，直到锁被释放。

这种情况通常发生在高并发的应用中，尤其是当多个用户或进程需要同时修改同一数据集时。

## 模拟行锁等待

会话1：

create table test(x number,y char(20),z date,q varchar2(4000));

Table created.

create index t\_idx on test(q,z);

Index created.

begin

for x in 1..1500 loop

insert into test values(1162,'1060000abcdefg', sysdate, rpad('x',2000,'x'));

end loop;

end;

/

PL/SQL procedure successfully completed.

会话2

begin

for x in 1..1500 loop

insert into test values(1162,'1060000abcdefg', sysdate, rpad('x',2000,'x'));

end loop;

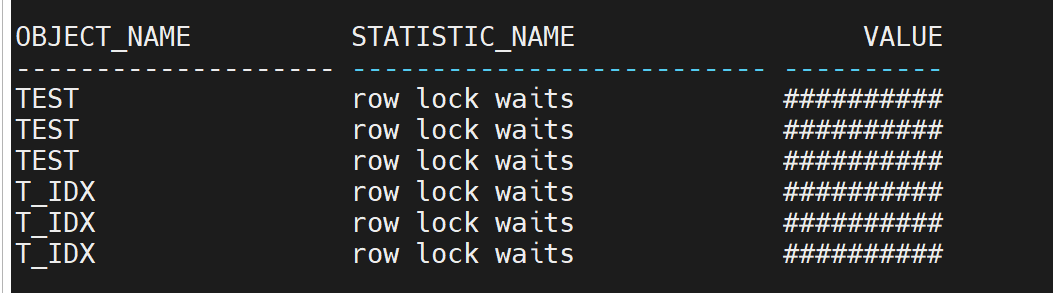
end;

/

PL/SQL procedure successfully completed.

会话3

|  |
| --- |
| set lines 1000  set pages 1000  col OBJECT\_NAME for a20  col STATISTIC\_NAME for a26  col VALUE for a20  select object\_name, STATISTIC\_NAME, value from v$segment\_statistics  where object\_name in ('TEST', 'T\_IDX') and STATISTIC\_NAME = 'row lock waits'  order by 1; |



## 工作原理：

Oracle使用行级锁来保证事务的隔离性和一致性。

当一个事务修改某行数据时，它会在这行数据上放置一个锁，防止其他事务同时修改相同的数据。

如果另一个事务也试图修改这些已被锁定的行，它将不得不等待直到第一个事务完成（提交或回滚）并释放锁。

## 可能原因：

长事务： 如果一个事务持有行锁的时间过长，其他试图访问这些行的事务将不得不等待。

并发冲突： 在高并发环境中，多个事务试图同时修改同一行可能会导致争用。

应用设计问题： 有时候，应用程序的设计导致不必要的行级锁竞争，例如，通过不合理的事务设计或不恰当的数据访问模式。

## 优化策略：

缩短事务长度： 保持事务尽可能短，以减少锁定时间，避免长事务。

避免热点数据： 重新设计应用逻辑，分散对特定行的访问，避免创建数据热点。

应用程序和数据库设计优化： 优化数据访问模式和数据库设计，如合理使用索引，减少不必要的全表扫描，这些都可以降低行锁争用的可能性。

监控和分析： 使用Oracle提供的监控工具，如自动工作负载仓库（AWR）报告，来识别和分析行锁等待的问题。

# SQL\*Net more data to client

## 发生场景：

当一个客户端执行了一个返回大量数据的查询（例如，选择大型表的多行），数据库服务器需要将这些数据发送回客户端。

如果数据量大到无法一次性发送完毕，服务器会分批发送。在等待网络完成这些大量数据传输的过程中，可能会记录“SQL\*Net more data to client”等待事件。

## 工作原理：

Oracle 使用 SQL\*Net（或Net8）协议在数据库服务器和客户端之间传输数据。

数据通过网络包分批传输。当一个包被发送后，数据库进程可能需要等待网络层的确认，或等待准备下一个数据包。

这个等待时间包含两个参数：

Driver id: 服务器端和客户端连接使用的协议信息。

#bytes: 服务器端向客户端发送消息的字节数。

## 可能原因：

大量结果集： 执行返回大量数据的查询，特别是当这些数据需要跨多个网络包发送时。

网络带宽和延迟： 限制了数据传输的速度，导致数据发送过程中出现等待。

客户端处理速度： 客户端处理接收到的数据的速度可能影响整个数据传输过程。

## 优化策略：

优化查询： 减少查询返回的数据量，例如，通过更精确的 WHERE 子句或分页技术。

提升网络性能： 改善数据库服务器和客户端之间的网络连接，例如，增加带宽或减少网络延迟。

客户端性能优化： 确保客户端应用程序能够高效处理接收到的数据。

调整批大小： 在某些情况下，调整网络批的大小可以提高效率。

# Latches

## 发生场景：

当多个数据库进程需要访问和修改共享的内存结构（如缓冲区缓存、共享池等）时，为了防止数据损坏和保持一致性，Oracle 使用 Latches 来控制访问。

Latches 主要用于短期操作，如修改内存中的数据结构。

## 工作原理：

Latches 的工作原理类似于锁，但它们是非阻塞的，并且设计用于非常短的操作。

当一个进程需要访问一个受 Latch 保护的结构时，它会尝试获取相应的 Latch。如果该 Latch 已被其他进程占用，进程会进行短暂的自旋（spin），尝试重新获取 Latch，而不是立即进入等待状态。

这个等待事件有三个参数：

1. Address： 会话等待的latch地址
2. Number： latch号，通过这个号，可以从v$latchname视图中找到这个latch的相关信息。

SQL> select \* from v$latchname where latch#=number;

1. Tries: 会话尝试获取Latch 的次数。

在10g之前的版本里，latch free 等待事件代表了所有的latch等待，在10g以后，一些常用的latch事件已经被独立了出来：

|  |
| --- |
| * 11gr2:   select name from v$event\_name where name like 'latch%' order by 1;  NAME  ----------------------------------------------------------------  latch activity  latch free  latch: Change Notification Hash table latch  latch: In memory undo latch  latch: MQL Tracking Latch  latch: PX hash array latch  latch: Undo Hint Latch  latch: WCR: processes HT  latch: WCR: sync  latch: cache buffer handles  latch: cache buffers chains  latch: cache buffers lru chain  latch: call allocation  latch: change notification client cache latch  latch: checkpoint queue latch  latch: enqueue hash chains  latch: gc element  latch: gcs resource hash  latch: ges resource hash list  latch: lob segment dispenser latch  latch: lob segment hash table latch  latch: lob segment query latch  latch: messages  latch: object queue header operation  latch: parallel query alloc buffer  latch: redo allocation  latch: redo copy  latch: redo writing  latch: row cache objects  latch: session allocation  latch: shared pool  latch: undo global data  latch: virtual circuit queues  已选择33行。 |
| * 10gr2 rac:   select name from v$event\_name where name like 'latch%' order by 1;  NAME  --------------------------------------------------  latch activity  latch free  latch: Change Notification Hash table latch  latch: In memory undo latch  latch: KCL gc element parent latch  latch: MQL Tracking Latch  latch: Undo Hint Latch  latch: cache buffer handles  latch: cache buffers chains  latch: cache buffers lru chain  latch: checkpoint queue latch  latch: enqueue hash chains  latch: gcs resource hash  latch: ges resource hash list  latch: library cache  latch: library cache lock  latch: library cache pin  latch: messages  latch: object queue header heap  latch: object queue header operation  latch: parallel query alloc buffer  latch: redo allocation  latch: redo copy  latch: redo writing  latch: row cache objects  latch: session allocation  latch: shared pool  latch: undo global data  latch: virtual circuit queues  29 rows selected.   * 19crac   NAME  --------------------------------------------------------------------------------  latch activity  latch free  latch: AQ OPT Background Master Latch  latch: AQ Sharded subscriber statistics latch  latch: Change Notification Hash table latch  latch: Column stats entry latch  latch: GCS logfile block  latch: GCS logfile write queue  latch: ILM access tracking extent  latch: ILM activity tracking latch  latch: IM area sb latch  latch: IM area scb latch  latch: IM emb latch  latch: IM seg hdr latch  latch: IMFS defer write list  latch: In memory undo latch  latch: MGA asr alloc latch  latch: MGA heap latch  latch: MGA pid alloc latch  latch: MGA shared context latch  latch: MGA shared context root latch  latch: MQL Tracking Latch  latch: PX hash array latch  latch: SGA Logging Bkt Latch  latch: SGA Logging Log Latch  latch: Undo Hint Latch  latch: WCR: processes HT  latch: WCR: sync  latch: active service list  latch: cache buffer handles  latch: cache buffers chains  latch: cache buffers lru chain  latch: call allocation  latch: change notification client cache latch  latch: checkpoint queue latch  latch: enqueue hash chains  latch: gc element  latch: gcs resource hash  latch: ges resource hash list  latch: java patching  latch: kjci process context latch  latch: kjci request sequence latch  latch: kjoeq omni enqueue hash bucket latch  latch: kjoer owner hash bucket  latch: ksm sga alloc latch  latch: ksolt lwth alloc  latch: last service list  latch: lob segment dispenser latch  latch: lob segment hash table latch  latch: lob segment query latch  latch: messages  latch: obj/range reuse redo processing  latch: object queue header operation  latch: parallel query alloc buffer  latch: pdb enqueue hash chains  latch: redo allocation  latch: redo copy  latch: redo writing  latch: row cache objects  latch: service drain list  latch: session allocation  latch: shared pool  latch: undo global data  latch: virtual circuit queues |

## 可能原因：

高并发访问： 当多个进程同时竞争相同的内存结构。

过度的数据共享： 某些共享数据结构可能由于设计或配置问题，面临过度的并发访问。

不恰当的应用设计： 某些情况下，应用程序可能导致不必要的共享数据结构访问，增加 Latches 等待。

## 优化策略：

减少共享数据结构的竞争： 识别并减少对共享内存结构的高并发访问。

优化内存配置： 调整 SGA 和相关内存结构的配置，如增加共享池的大小。

应用和数据库优化： 优化 SQL 查询和数据库操作，减少对共享内存的访问。

监控和调优： 使用 Oracle 提供的监控工具（如 AWR、ASH 报告）分析 Latches 等待，并根据结果调整系统。

# Library Cache Lock

## 发生场景：

当多个会话试图同时执行、编译或解析相同的 SQL 语句或 PL/SQL 程序时，为了保持一致性和同步，Oracle 需要在这些对象上放置锁。

比如，一个会话可能正在重新编译一个包，而另一个会话试图执行同一个包中的过程。

## 工作原理：

库缓存是 Oracle SGA 的一部分，用于存储共享的 SQL 和 PL/SQL 代码。

当一个会话需要修改库缓存中的对象（如重新编译过程）时，它会在该对象上放置一个排他锁。

如果另一个会话同时试图访问这个对象（例如执行一个相关的 SQL 查询），它将不得不等待直到锁被释放。

这个等待事件包含四个参数：

Handle address： 被加载的对象的地址

Lock address： 锁的地址

Mode： 被加载对象的数据片段

Namespace： 被加载对象在v$db\_objects\_cache视图中的namespace名称。

|  |
| --- |
| * 10gr2 rac:   select name from v$event\_name where name like 'library%' order by 1;  NAME  --------------------------------------------------  library cache load lock  library cache lock  library cache pin  library cache revalidation  library cache shutdown   * 19c-rac   SQL> select name from v$event\_name where name like 'library%' order by 1;  NAME  --------------------------------------------------------------------------------  library cache load lock  library cache lock  library cache pin  library cache revalidation  library cache shutdown  library cache: bucket mutex X  library cache: dependency mutex X  library cache: mutex S  library cache: mutex X  9 rows selected. |

## 可能原因：

高并发执行或编译： 同时尝试执行、解析或编译相同代码的多个会话可能导致锁争用。

动态 SQL 和频繁重新编译： 频繁修改和重新编译存储过程或函数可能导致其他会话等待。

不恰当的应用设计： 应用程序设计中对数据库对象的不恰当使用可能增加锁竞争。

## 优化策略：

减少动态 SQL 使用： 尽量减少动态 SQL 的使用和频繁重新编译。

代码优化： 优化 PL/SQL 代码，减少需要编译的次数。

应用和数据库调整： 调整应用程序逻辑，避免高并发访问同一数据库对象。

库缓存调整： 调整库缓存的大小，确保足够的空间存储共享的 SQL 和 PL/SQL 对象。

监控和诊断： 使用 Oracle 工具（如 AWR、ASH 报告）监控库缓存的使用情况，并根据需要进行调整。