

# 逻辑备库应用案例

中国 Oracle 用户组

作者：范向荣

<http://www.acoug.org>

| 版本  | 发布时间       |
|-----|------------|
| 1.0 | 2010/06/06 |
|     |            |
|     |            |

**文章摘要：**本文旨在探讨逻辑备库的可应用性，实施逻辑备库时的注意事项、有可能遇到的问题以及如何解决这些问题。内容涵盖 10g 和 11g，以 10g 为主，对于 11g 中的一些变动会特别加以说明。

## 目录

|                               |        |
|-------------------------------|--------|
| 1. 引言 .....                   | - 2 -  |
| 2. 逻辑备库的限制 .....              | - 3 -  |
| 3. 逻辑备库的架构 .....              | - 4 -  |
| 4. 逻辑备库的参数 .....              | - 5 -  |
| 5. 逻辑备库的故障分析 .....            | - 7 -  |
| 6. 逻辑备库应用中的一些问题 .....         | - 10 - |
| 7. 逻辑备库实际应用中的性能表现 .....       | - 14 - |
| 8. 附录 A-各版本支持的数据类型及存储类型 ..... | - 14 - |

## 1. 引言

逻辑备库(logical standby)作为 Data Guard 的一种，在企业架构中占有重要的一席之地。

首先我们来将它和其他类型的 Standby 数据库做一个简单对比。

相比较传统的物理备库(physical standby)来说，逻辑备库的优点是它处在打开状态，可以提供查询。其缺点是在使用上有一些限制，实际中遇到的问题会比物理备库多，管理成本偏高。数据文件不能用于主库的恢复，另外 SQL Apply 的效率也不如物理备库 block apply 效率高。

相比较 11g 引入的 Active Data Guard，其优势在于更加灵活，比方说可以在逻辑备库上建立索引以供报表查询使用，另外逻辑备库是完全免费的，而 Active Data Guard 是需要 license 的，对于企业应用来说，成本也是考虑方案的重要因素之一。其缺点也是 SQL Apply 的效率问题以及数据文件不能用于主库的恢复。

相比较其他的逻辑复制软件例如 shareplex 来说，其优点在于，逻辑备库对主库产生的开销比较小，配置上相对简单，新增加表时不需要维护其配置文件。其缺点在于，使用上不如 shareplex 灵活，在某些特性的需求下不能满足，比方说多个数据库复制到一个数据库，或者只是复制表的部分列。Shareplex 支持级联而逻辑备库不支持，另外在对大事务的处理上逻辑备库的性能不如 shareplex。

总体来说，没有哪个产品是完美的。特别是逻辑复制的产品，多多少少都存在着这样那样的问题。那么关键是选择符合自己需求的产品，并在实施过程中去克服其不足之处。

## 2. 逻辑备库的限制

在实施前让我们首先来了解一下逻辑备库有哪些限制:

10g 版本逻辑备库对下面的数据类型不支持:

**BFILE**

**Collections (including VARRAYS and nested tables)**

**Encrypted columns**

**Multimedia data types (including Spatial, Image, and Context)**

**ROWID, UROWID**

**User-defined types**

**XMLType**

**另外不支持压缩表。**

11g 提供了对 Encrypted columns 的支持以及对压缩表的支持。但对一些新增加的功能如 binary xml 数据类型和下面的一些存储类型不支持:

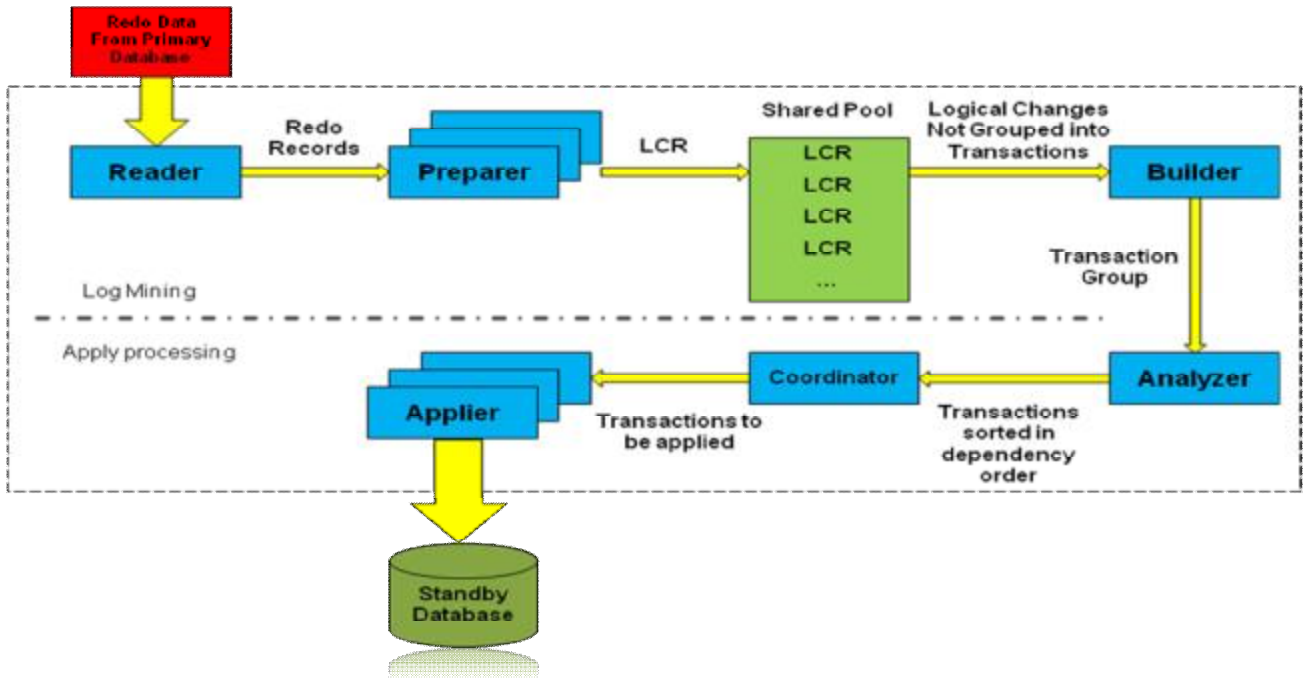
**Tables containing LOB columns stored as SecureFiles (unless the compatibility level is set to 11.2 or higher)**

**Tables with virtual columns**

**Tables using hybrid columnar compression**

10g 和 11g 支持的数据类型和存储类型详细列表参见附录 A。从列表上来看, 10g 和 11g 的版本已经对常见的大多数数据类型和存储类型提供了支持。另外虽然文档上说 context index 不支持, 但是 context index 分为两种类型, CTXCAT 类型和 CONTEXT 类型, 对于 CTXCAT 类型的 context index, 其实逻辑备库是支持的, 因为 CTXCAT 类型的 context index 会自动维护, 就和普通索引一样。只是在创建的时候会重复创建, 需要忽略掉第二次操作。我们目前就有一些逻辑备库在使用 CTXCAT 类型的 context index。

### 3. 逻辑备库的架构



如上图所示，主库只是负责将 redo 数据传输到备库，不做任何解析的工作，所以逻辑备库对主库产生的开销比较小，传输方法和物理备库相同。另外，因为逻辑备库在主库上没有复杂的操作（例如 shareplex 在主库上做分析操作，生成传输队列），所以 redo 信息会被快速地传递到备库。从灾备的角度上来看，如果主库发生意外，损失的数据相对来说也会比较少。

**Reader 进程**负责从备库的 standby redo logs 或者 archived logs 中读取 redo 信息，一个逻辑备库有且只能有一个 Reader 进程。

**Preparer 进程**将读出的 Redo 信息转换为 LCR（logical change record 逻辑修改记录），LCR 是最小单位，可以对应于一行数据的修改或者 LOB CHUNK 的一次修改或者是一个 DDL 语句。在生成 LCR 的过程中，会使用存储在备库上的主库数据字典信息用来生成 DML 或者 DDL。Preparer 进程可以有多个，可以通过逻辑备库参数 PREPARER\_SERVERS 来调整其个数。

**Builder 进程**将一个个单独的 LCR 组合成事务块（transaction chunk），一个事务块中可能不包含 commit，具体要由事务中包含的 LCR 的数量和逻辑备库的参数 EAGER\_SIZE 来决定，如果单个事务中所含的 LCR 的数目大于 EAGER\_SIZE（默认值是 201），那么这个事务就由多个事务块组成，该事务被定义为 eager 类型的事务，后面我们会讲到什么是 eager 类型的事务。另外，一个逻辑备库有且只能有一个 Builder 进程。

**Analyzer 进程**用于统计各个事务块之间的依赖关系，比方说 A 和 B 修改了同一行数据，那么 A 和 B 之间必须依照次序来执行，否则数据就不对了。这个依赖关系和 PRESERVE\_COMMIT\_ORDER 的含义不同，即使 PRESERVE\_COMMIT\_ORDER 设置为 false，这个依赖关系依然存在。但是 Oracle 如何知道两个事务是否修改了同一行数据呢？这里就要提到另一个逻辑备库参数 HASH\_TABLE\_SIZE，这些参数的意义会在逻辑备库参数一章里进行详细解释。另外 DDL 和 DML 不同，当 mining 时发现 DDL，逻辑备库会设置一个 DDL 屏障 (barrier)，mining 停止工作。在应用该 DDL 之前，所有在此 DDL 之前的事务都需要应用完成，而所有在此之后的事务都不能被应用。一个逻辑备库有且只能有一个 Analyzer 进程。

**Coordinator 调度进程**将事务块按照次序传给 Apply 进程。

**Apply 进程**将对应的修改在备库上执行，如果该事务块依赖于其他的 chunk，则在提交前会等待其他 Apply 进程提交。一般 Apply 进程会有多个，可以通过 APPLY\_SERVERS 参数来设置其数目。在后面我会提到如何根据系统统计信息来调整 APPLY 进程的数目。

## 4. 逻辑备库的参数

逻辑备库的参数存储在 DBA\_LOGSTDBY\_PARAMETERS 中。

```
SQL> desc dba_logstdby_parameters;
```

| Name  | Null? | Type           |
|-------|-------|----------------|
| NAME  |       | VARCHAR2(30)   |
| VALUE |       | VARCHAR2(2000) |

11g 中这个表多增加了一些列。在 11g 中如果 DYNAMIC=YES 表示该参数可以动态设定，不需要重启 SQL APPLY 。

```
SQL> desc dba_logstdby_parameters;
```

| Name    | Null? | Type           |
|---------|-------|----------------|
| NAME    |       | VARCHAR2(64)   |
| VALUE   |       | VARCHAR2(2000) |
| UNIT    |       | VARCHAR2(64)   |
| SETTING |       | VARCHAR2(64)   |

DYNAMIC

VARCHAR2(64)

## 1、PRESERVE\_COMMIT\_ORDER, TRANSACTION\_CONSISTENCY 参数

首先要说的是 TRANSACTION\_CONSISTENCY & PRESERVE\_COMMIT\_ORDER 参数，这两个参数意思一样，控制逻辑备库的事务提交次序是否需要和主库一致。9i 前使用的是 TRANSACTION\_CONSISTENCY，10g 后使用的是 PRESERVE\_COMMIT\_ORDER，可能是因为 TRANSACTION\_CONSISTENCY 的名字会造成误解，让大家产生一种数据不安全的感。设置 PRESERVE\_COMMIT\_ORDER=FALSE 可以大大提高逻辑备库应用的性能。虽然事务应用的次序可能和主库的次序不一致，但是不会导致数据出现错误，而且会达到最终一致。也就是说在切换或者激活为主库前，所有事务还是会处于一致的状态。另外如果两个事务处理的是同一数据，还是会有先后关系，不完全是相互独立的状态。

MAX\_SGA 参数是用来控制 LCR cache 的大小，这部分内存是从 shared pool 中分配的。

```
SQL> select * from v$$sgastat where name like '%LCR%';
```

| POOL        | NAME           | BYTES     |
|-------------|----------------|-----------|
| shared pool | Logminer LCR c | 349716032 |

这个参数最大可以设置为 4GB。可考虑设大一点，太小的话会影响性能。

## 2、MAX\_SERVERS, PREPARE\_SERVERS, APPLY\_SERVERS 参数

这三个参数用来控制 slaves 的数目。如果只设定 MAX\_SERVERS，那么 oracle 会根据一定的比例关系调整 preparer 和 apply 进程的数目，默认是 1:20。在 10g 版本中，MAX\_SERVERS 要小于 PARALLEL\_MAX\_SERVERS 参数，11g 无此限制。在我们的系统中，PREPARE\_SERVERS 的值设置为 5，APPLY\_SERVERS 的数量和系统的压力有关。Coordinator 进程在分配事务时总是从第一个 Applier 开始，如果第一个忙就分配给第二个，以此类推。那么如果所有的 APPLY 进程所分配到的事务数目差不多，也就是说第一个 Applier 和最后一个 Applier 一样忙，就说明 APPLY 进程数目可能不足。可以通过下面的 SQL 来查看为每个 Apply 进程分配了多少事务。

```
SQL> select min(pct_applied) pct_applied_min
2 , max(pct_applied) pct_applied_max
3 , avg(pct_applied) pct_applied_avg
4 , count(server_id) number_of_appliers
5 from ( select server_id
6 , (greatest(nvl(s.total_assigned,0),0.00000001) /
7 greatest(nvl(c.total_assigned,1),1)) * 100 pct_applied
```

```
8 from v$streams_apply_server s
```

```
9 , v$streams_apply_coordinator c);
```

```
PCT_APPLIED_MIN PCT_APPLIED_MAX PCT_APPLIED_AVG NUMBER_OF_APPLIERS
```

```
-----
```

```
372236673 1.5131083 . 833333436 120
```

从以上查询可以看到最忙的 **Applier** 和最闲的 **Applier** 相差还是蛮大的，所以 **Applier** 的数目是足够的，如果系统资源不足的话可以考虑减少 **Applier**。如果 **MAX** 和 **MIN** 所分配到的事务数差不多并且系统资源充足的话，可以考虑增加 **Applier** 数目来提高性能。

### 3、\_EAGER\_SIZE 参数

如果一个事务中包含有多个 **LCR** 且 **LCR** 的数目大于 **\_EAGER\_SIZE** 的值（默认为 201），则该事务被认为是 **eager** 事务。如果是 **eager** 事务，那么 **apply** 不需要等待所有的 **LCR** 准备完毕，可以一边 **APPLY** 一边 **Build**。这样做对于大事务而言有两个好处，一个是减小 **LCR cache** 的内存占用，另一个是可以使大事务应用的更快一些。但是从实践来看，不管 **\_EAGER\_SIZE** 大小如何，大事务总是会影响整个系统的应用性能。如果主库有大量的大事务发生，那么逻辑备库可能会有长时间延迟。针对此种情况，我们要做的就是尽量把大事务变成小事务。比如说删除作业原本是 **Bulk delete, array size=1000, commit rows =1000**，我们可以将其改为 **array size=100, commit rows=100**，这样可以提高逻辑备库的性能。再比如说，如果每晚在固定时候有 **SQL Load** 任务，则可以在逻辑备库中忽略该表，然后通过相同的脚本在逻辑备库同时 **Load** 数据。

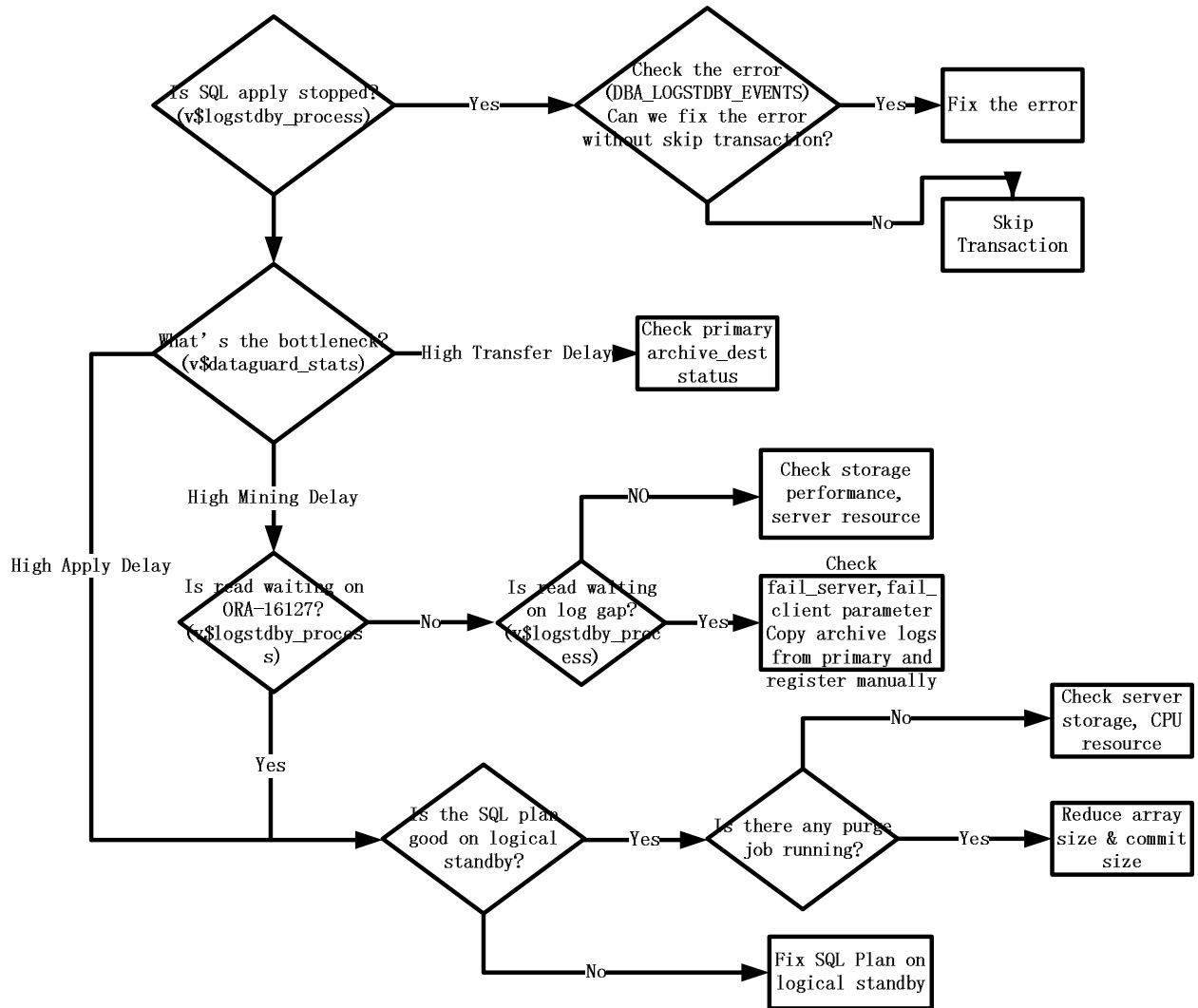
### 4、\_HASH\_TABLE\_SIZE 参数

这是一个隐藏参数，默认值为 1000000，最高可以设置为 32000000。10g 中设置该参数后可以在 **DBA\_LOGSTDBY\_PARAMETERS** 中看到，11g 中必须到基表(**X\$DGLPARAM**)中查看。

前面我们在讲 **Analyzer** 进程的时候提到过这个参数。这个参数的作用是设置内部 **Hash** 表的 **bucket** 数目。我们如何知道事务 **A** 和事务 **B** 是否在修改同样的数据呢？**oracle** 会根据 **PK** 和 **unique index** 算出一个 **hash** 值，然后来比较这个 **hash** 值是否一致。这就有可能不同的行对应到同一个 **hash** 值的情况，可能会导致不必要的依赖关系。那么如果 **Hash** 表中的 **bucket** 数目越多，这种情况的发生机率就会越小。在我们的系统上，这个值通常被设为最高的 32000000。

## 5. 逻辑备库的故障分析

下图是我平时处理故障时的一些思路，供大家参考。



上图大致的步骤是：首先查看 SQL APPLY 进程是否因为错误被停止了。逻辑备库遇到任何错误都会停止工作，在 alert.log 和 DBA\_LOGSTDBY\_EVENTS 中会有相应的记录。如果是错误导致停止，纠正错误或者通过 DBMS\_LOGSTDBY.skip\_transaction 忽略该事务。DBA\_LOGSTDBY\_EVENTS 中有三个列（XIDUSN，XIDSLT，XIDSQN）记录了相对应的事务。

如果 SQL APPLY 还在继续工作，只是比较慢，我们可以通过查询 v\$dataguard\_stats 来查看瓶颈在哪里。如果是传输的问题，可以查看主库的 log\_archive\_dest 参数是否设置正确，逻辑备库上的 listener 是否配置正确，密码文件是否存在。如果是因为网络速度导致传输较慢，可以尝试设置 buffer size:

在逻辑备库的 listener.ora 中加入 SEND\_BUF\_SIZE,RECV\_BUF\_SIZE 和 SDU 参数:

```

TEST_LISTENER_LG=
  (ADDRESS_LIST=
  (ADDRESS =
  (PROTOCOL = TCP)
  
```

```
(HOST = xxxxxxxx)
(PORT = 1600)
(QUEUESIZE=40)
(SEND_BUF_SIZE=1048576)
(RECV_BUF_SIZE=1048576)
))
```

SID\_LIST\_TEST\_LISTENER\_LG =

```
(SID_LIST =
(SID_DESC =
(SDU = 32767)
(SID_NAME = xxxxx)
(ORACLE_HOME = xxx)
))
```

同样在 log\_archive\_dest 中加入该参数:

```
Alter system set
log_archive_dest_2="service=(DESCRIPTION=
(SDU=32767)(SEND_BUF_SIZE=1048576)(RECV_BUF_SIZE=1048576)
(ADDRESS=(PROTOCOL=TCP)(HOST=xxxx)(PORT=1600))(CONNECT_DATA=(SID=xxx)))" reopen=5 lgwr async";
```

如果是使用的 TNS name, 可以添加到 tnsnames.ora 中:

```
TEST=
(DESCRIPTION =
(SDU=32767)
(SEND_BUF_SIZE=1048576)
(RECV_BUF_SIZE=1048576)
(ADDRESS =
(PROTOCOL = TCP)
(Host = xxxx)
(Port = 1600))
(CONNECT_DATA = (SID = xxxx)))
```

如果是 11g, 则可以在 `log_archive_dest` 中设置 `compression=enable`, 这样会将 redo 进行压缩然后传输, 在网络不太好的情况可考虑使用该参数, 压缩比和 redo 内容有关。在我的测试中, 普通的数据类型压缩后大约是原来的 24%。但值得注意的是, 使用 oracle 11g 高级压缩功能是需要额外的 license 的。

如果看到 APPLY 和 Mining 的延迟都很高并且 Reader 进程等待在 `ORA-16127 "stalled waiting for additional transactions to be applied"`上, 那么基本上可以确定是 APPLY 进程的问题。能引起 APPLY 性能问题的状况比较多, 但大体上可以从以下几个方面来入手:

**1、逻辑备库的 SQL 执行有没有性能问题。** 比如说执行计划不对。

**2、主库上是否存在大事务导致逻辑备库性能降低。** 这个可以通过查询 `vstreams_apply_server` 中的 `MESSAGE_SEQUENCE` 的最大值来判断。如果 `MESSAGE_SEQUENCE` 很大, 则说明事务中包含的 LCR 数目很多。这种情况下可以看看能不能将主库的大事务变成若干的小事务, 比如说减小 `array size`, `commit` 频繁一点等等。

**3、APPLY 进程数目太少。** 上面我们提高过这个问题, 可以通过查询 APPLY 进程所分配到的事务数来判断。如果太少了并且系统还有足够的资源, 就可以通过增加 APPLY 进程来提高性能。

**4、是否达到系统瓶颈。** 比如说 CPU 的使用率过高, 存储的响应时间过高。这种情况尤其在主库和备库硬件配置不一样的时候很有可能会发生。

如果 Reader 进程在等待归档日志 (从 `V$LOGSTDBY_PROCESS` 中可以看到), 就要分析为什么日志没有被成功地传送过来, 为什么逻辑备库没有去主库抓取所需要的日志(`fail_client, fail_server` 参数是否设置正确)。必要时需要手工从主库拷贝归档日志到备库, 然后在逻辑备库注册(`ALTER DATABASE REGISTER LOGICAL LOGFILE 'XXXX'`)。

## 6. 逻辑备库应用中的一些问题

### 1、对于索引的创建

逻辑备库可以自动复制主库 `CREATE INDEX` 语句, 不过在 10G 版本中, APPLY 进程不能并行的执行 DDL 语句, 11g 版本有所改进, 可以并行执行 DDL 语句, 但是对于大的索引还是要很长时间。前面提到过 `DDL barrier`, 索引的创建属于 DDL, 所以他会阻塞其他的事务。如果一个索引创建需要几个小时, 那么逻辑备库延迟就会达到几个小时。

为了达到减少延迟的目的, 对于大的索引, 我们需要手工在逻辑备库上创建。然后通过设置

DBMS\_LOGSTDBY.SKIP 来忽略主库复制过来的 CREATE INDEX 语句。

```
EXEC DBMS_LOGSTDBY.SKIP('CREATE INDEX','SCHEMA_NAME','INDEX_NAME')
```

然后再在主库上建立索引。

如果索引很小的话，可以由逻辑备库自动复制过来应用。

## 2、逻辑备库 standby logs 不能被重用的问题

在实践中，我们遇到过几次 standby logs 不能被重用的问题。

| THREAD# | SEQUENCE# | BYTES      | USED      | STATUS | FIRST_TIME                    |
|---------|-----------|------------|-----------|--------|-------------------------------|
| 1       | 60745     | 1049656832 | 432919552 | ACTIVE | 28-JUL-2009 01:41:30          |
| 1       | 60213     | 1049656832 | 288615936 | ACTIVE | 22-JUL-2009 20:12:53 -- stale |
| 1       | 60242     | 1049656832 | 198030848 | ACTIVE | 23-JUL-2009 02:52:05 -- stale |
| 1       | 60720     | 1049656832 | 127185920 | ACTIVE | 27-JUL-2009 18:49:15 -- stale |

四个 standby logs 中只有第一个是最新的日志文件，其他三个都是很早之前的日志文件。因为只有一个 standby log 可用，所以下一个 standby log 会直接写到归档目录中而不是 standby log 中。这会导致逻辑备库不能实时地应用日志。解决办法就是 DROP 掉三个不能重用的 standby log 然后再重新添加。

## 3、逻辑备库的补丁包

逻辑备库的补丁包一定要打，Bug 7937113 - 10.2.0.4 Data Guard Logical Recommended Patch Bundle #1 [ID 7937113.8]。这个补丁包对下列问题进行了修正：

Bug:6120004 Concurrent eager transactions may cause logical standby apply to hang

Bug:6181488 Logical standby can fail ith ORA-4030

Bug:6265559 Apply spins after sequence altered at primary

Bug:6268409 ORA-29275 error when querying the sql\_redo/undo columns of V\$LOGMNR\_CONTENTS

Bug:6392076 Streams does not handle Java DDL correctly

Bug:6413089 Restarting a logminer session can be slow if the session has fallen behind

Bug:6451626 Dump selecting from V\$LOGMNR\_CONTENTS or FLASHBACK\_TRANSACTION\_QUERY

Bug:6452375 ORA-26687 in Streams from DROP TABLE

Bug:6596564 Logminer ad-hoc query does not handle thread events in RAC

- Bug:6599920 Streams capture aborts with ORA-26744 and ORA-26773 on LOB columns
- Bug:6615740 ORA-4030 in logical standby ("knas:shtrans")
- Bug:6650256 PDDL causes logminer spill
- Bug:6683178 Capture performance degrades due to lots of DDL (eg truncate of empty tables)
- Bug:6851438 OERI [kcrldr.6] during dictionary build on new primary
- Bug:6903051 Logical standby hang (with RAC primary)
- Bug:6926448 Logical standby cannot identify new primary after failover to physical standby
- Bug:6987790 Large trace files from logical standby
- Bug:6988017 "krvutld" ORA-1280 on logical standby / Streams capture
- Bug:6994160 Capture reader process constantly writing messages to trace file
- Bug:7033630 OERI:knldqm2usr:4 from SQL Apply
- Bug:7036453 Logical standby upgrade may appear to hang
- Bug:7043989 Logminer started with CONTINUOUS\_MINE stops without reason
- Bug:7125408 Dump [krvbsdml\_SuppressDml] in logical standby apply process
- Bug:7175513 ORA-26773 / ORA-26767 from Streams after SPLIT PARTITION
- Bug:7189645 Logical standby fails with OERI:knacpsm\_ProcessSlaveMessage250
- Bug:7196532 Bad session created after logfile registration failure
- Bug:7219752 ORA-26773 malformed redo on capture of LONG
- Bug:7315642 Reinstatement of failed primary to logical standby may error
- Bug:7331867 "LOGMNR WARNING: LogMnr ckptUnsafeCnt .." warnings
- Bug:7341598 Logical standby auto purge may leave some logs undeleted
- Bug:7345904 Streams capture slow processing certain DDLs
- Bug:7356443 Gradual slowdown of user defined transformations and Streams DML handlers
- Bug:7393804 OERI [kcrldr.6] / OERI [krrcomp.6] on Logical Standby
- Bug:7432514 ORA-4068 on DBMS\_INTERNAL\_LOGSTDBY with logical standby
- Bug:7499353 Logminer checkpointing may cause capture performance problems
- Bug:7523787 Dump (krvrgtl\_GenTokLcr) in logical standby / streams
- Bug:7527650 Data divergence during rollback inline LOB DML
- Bug:7553884 DML done immediately after DDL lost on logical standby

Bug:7693128 Internal Enhancement for logminer

Bug:8239142 Streams capture / logical standby aborts with ORA-26744 / ORA-26766

Bug:8267348 Logminer reports committed distributed transaction as rolled back

这个补丁包修正了许多问题，包括 LOB column 的复制，主从库不能成功切换的问题。

#### 4、不能正常 STOP APPLY 的问题

我们遇到过几次 ALTER DATABASE STOP LOGICAL STANDBY APPLY 不能正常关闭的情况。都是因为 READER 进程 hang 住了，从 V\$LOGSTDBY\_PROCESS 中查看 READER 进程的 STATUS，发现再读取 SEQUENCE#=0 的 log，将 READER 进程用 kill -9 命令杀掉就可以正常关闭了。

#### 5、如何忽略 DBA 创建的临时表

有时候 DBA 需要创建一些临时表，比方说为了做删除操作，创建一个临时表用来存放 ROWID 和 PK。这些临时表是不需要被复制到逻辑备库的，我们可以通过设定 SKIP 规则来忽略这些表的复制。SKIP 支持通配符和 escape character，所以我们可以很灵活的来配置。比如说，忽略所有以 TMP\_ 和 TEMP\_ 开头的表和索引：

```
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'CREATE TABLE', SCHEMA_NAME=>%',object_name =>'TMP\%',esc=>'\');
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'CREATE INDEX', SCHEMA_NAME=>%',object_name =>'TMP\%',esc=>'\');
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'CREATE TABLE', SCHEMA_NAME=>%',object_name =>'TEMP\%',esc=>'\');
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'CREATE INDEX', SCHEMA_NAME=>%',object_name =>'TEMP\%',esc=>'\');
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'DML', SCHEMA_NAME=>%',object_name =>'TMP\%',esc=>'\');
EXECUTE DBMS_LOGSTDBY.SKIP (stmt =>'DML', SCHEMA_NAME=>%',object_name =>'TEMP\%',esc=>'\');
```

#### 6、SKIP PROCEDURE 的使用

SKIP PROCEDURE 可以在 DBMS\_LOGSTDBY.SKIP 和 DBMS\_LOGSTDBY.SKIP\_ERROR 中使用。满足 SKIP 或者 SKIP\_ERROR 条件时被触发。在某些特定的情况下需要通过创建 SKIP PROCEDURE 来满足需求。在我的 blog 中 [Ignore sequence grant error on logical standby](http://www.dbafan.com/blog/?p=363) 有其使用案例[<http://www.dbafan.com/blog/?p=363>]。

## 7. 逻辑备库实际应用中的性能表现

对于典型的 OLTP 系统来说，逻辑备库还是蛮适合的。如果系统的事务都是小事务，逻辑备库的应用性能还是很高的。

在我们的系统（Sun T5120, Hitachi 9990）上可以应用每秒 6MB 的日志量，TPS 达到每秒 1300 的事务数。如果使用 shareplex 来实现同样的吞吐量，必须要把表拆分到不同的队列中，对于特别繁忙的表可能还要进行水平分割(horizontal split)。而逻辑备库只要调整一些参数就可以了，配置上更为方便。

在 10g 版本中，正常情况下应用延迟在 30 秒内，和 shareplex 相比实时性差点，shareplex 差不多是在一两秒以内。如果是实时性要求比较高的系统（比如说 2 秒内），逻辑备库不太适合。11g 在实时性方面有所提高，应用的延迟大部分在 5 秒以内，一般不会超过 10 秒。

但是如果存在大事务，逻辑备库的性能就会大大下降。所以要提高性能还是要从应用入手，将大事务变成小事务。

总体上，对于一款免费的逻辑复制软件来说，逻辑备库方案的表现还是挺不错的。如果要实施逻辑复制，可以尝试一下。

## 8. 附录 A-各版本支持的数据类型及存储类型

### 10g 支持的数据类型：

BINARY\_DOUBLE

BINARY\_FLOAT

BLOB

CHAR

CLOB and NCLOB

DATE

INTERVAL YEAR TO MONTH

INTERVAL DAY TO SECOND

LONG

LONG RAW

NCHAR

NUMBER

NVARCHAR2

RAW

TIMESTAMP

TIMESTAMP WITH LOCAL TIMEZONE

TIMESTAMP WITH TIMEZONE

VARCHAR2 and VARCHAR

### 10g 支持的存储类型:

Cluster tables (including index clusters and heap clusters)

Index-organized tables (partitioned and nonpartitioned, including overflow segments)

Heap-organized tables (partitioned and nonpartitioned)

### 11g 支持的数据类型:

BINARY\_DOUBLE

BINARY\_FLOAT

BLOB

CHAR

CLOB and NCLOB

DATE

INTERVAL YEAR TO MONTH

INTERVAL DAY TO SECOND

LONG

LONG RAW

NCHAR

NUMBER

NVARCHAR2

RAW

TIMESTAMP

TIMESTAMP WITH LOCAL TIMEZONE

TIMESTAMP WITH TIMEZONE

VARCHAR2 and VARCHAR

XMLType stored as CLOB

LOBs stored as SecureFile

**11g 支持的存储类型:**

Cluster tables (including index clusters and heap clusters)

Index-organized tables (partitioned and nonpartitioned, including overflow segments)

Heap-organized tables (partitioned and nonpartitioned)

OLTP table compression (COMPRESS FOR OLTP) and basic table compression (COMPRESS BASIC)

## 范向荣的个人简介



范向荣，网名 eagle\_fan

eBay 中国营运中心首席数据库工程师

ITPUB 专题深入讨论版版主

擅长研究数据库高压力高并发下的企业级解决方案。在大型数据库管理、数据库调优和数据库故障诊断方面有丰富的经验。

个人 blog: <http://www.dbafan.com>