|  |  |  |
| --- | --- | --- |
| **Troubleshooting 'enq: TX - index contention' Waits (Doc ID 873243.1)** | [IMG_256](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4%26_afrLoop=179744812545361)  [To Bottom](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4&_afrLoop=179744812545361) | IMG_257 |

IMG_258

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **In this Document**   |  |  | | --- | --- | |  | [Goal](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4&_afrLoop=179744812545361" \l "GOAL) |  |  |  | | --- | --- | |  | [Solution](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4&_afrLoop=179744812545361" \l "FIX) |  |  |  | | --- | --- | |  | [Community Discussions](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4&_afrLoop=179744812545361" \l "aref_section21) |  |  |  | | --- | --- | |  | [References](https://support.oracle.com/epmos/faces/SearchDocDisplay?_adf.ctrl-state=139gln6twy_4&_afrLoop=179744812545361" \l "REF) |    Applies to: Oracle Database - Enterprise Edition - Version 10.2.0.1 to 11.2.0.4 [Release 10.2 to 11.2]  Information in this document applies to any platform. Goal This document explains the how to troubleshoot and resolve 'enq: TX - index contention' waits. Solution When running an OLTP systems, it is possible to see high TX enqueue contention on index associated with tables, which are having high concurrency from the application.  This usually happens when the application performs lot of INSERTs and DELETEs concurrently. For RAC system, the concurrent INSERTs and DELETEs could happen from all the instances .   The reason for this is the index block splits while inserting a new row into the index. The transactions will have to wait for TX lock in mode 4, until the session that is doing the block splits completes the operations.  A session will initiate a index block split, when it can't find space in an index block where it needs to insert a new row. Before starting the split, it would clean out all the keys in the block to check whether there is enough sufficient space in the block.deleted  Splitter has to do the following activities:       o          Allocate a new block.      o          Copy a percentage of rows to the new buffer.      o          Add the new buffer to the index structure and commit the operation.   In RAC environments, this could be an expensive operation, due to the global cache operations included. The impact will be more if the split is happening at a branch or root block level.  Causes:   Most probable reasons are:      o          Indexes on the tables which are being accessed heavily from the application.     o          Indexes on table columns which are monotonically growing. In other words, most of the index insertions occur only on the right edge of an index.     o          Large data purge has been performed, followed by high concurrent insert  Identifying the Hot index:   The indexes which are having contention can be identified from the AWR reports taken during the time of the issue.  ***Top 5 Timed Events:   Event                       Waits      Time(s)   Avg Wait(ms)  % Total Call  Time Wait Class   en: TX - index contention   89,350     40,991    459           63.3          Concurrency   db file sequential read     1,458,288  12,562    9             19.4          User I/O   CPU time                               5,352                   8.3     Instance Activity Stats:   Statistic                Total     per Second    per Trans   branch node splits       945       0.26          0.00   leaf node 90-10 splits   1,670     0.46          0.00   leaf node splits         35,603    9.85          0.05***  And the objects can be found either from V$SEGMENT\_STATISTICS or from 'Segments by Row Lock Waits' or 'Segments by ITL Waits' or 'Service ITL Waits' of the AWR reports.  ***Segments by Row Lock Waits:   Owner     Tablespace  Object Name            Obj.Type   Row Lock Waits  % of Capture   ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_PK   INDEX      3,425           43.62   ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_ST   INDEX      883             11.25   ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_DT   INDEX      682             8.69***    Segments by ITL Waits   Owner   Tablespace Name Object Name     Subobject Name  Obj. Type       ITL Waits       % of Capture  ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_PK                  INDEX              6       50.00  ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_ST                  INDEX              3       25.00  ACSSPROD  ACSS\_IDX03  ACSS\_ORDER\_HEADER\_DT                  INDEX              3       25.00    **Solutions:**  Solution here is to tune the indexes avoid heavy access on a few set of blocks.  Following are the options we could try:  o        Rebuild the index  as reverse key indexes or hash partition the indexes which are listed in the 'Segments by Row Lock Waits' of the AWR reports  For example:  CREATE INDEX <index name> ON <column> REVERSE;  From the Performance Tuning Guide -   Reverse key indexes are designed to eliminate index hot spots on insert applications.  These indexes are excellent for insert performance.  But the downside of it is that, it may affect the performance of index range scans.  [http://download.oracle.com/docs/cd/B19306\_01/server.102/b14211/design.htm#sthref112](http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/design.htm" \l "sthref112)  The hash method can improve performance of indexes where a small number leaf blocks in the index have high contention in multiuser OLTP environment. In some OLTP applications, index insertions happen only at the right edge of the index. This could happen when the index is defined on monotonically increasing  columns. In such situations right edge of the index becomes a hotspot because of contention for index pages, buffers, latches for update, and additional index  maintenance activity, which results in performance degradation.  [http://download.oracle.com/docs/cd/B19306\_01/server.102/b14211/data\_acc.htm#i2678](http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/data_acc.htm" \l "i2678)  It's recommended to test the application performance,  after rebuilding the indexes as  reverse key or hash partitioned.  **o        Consider increasing the CACHE size of the sequences**  alter sequence <owner>.<seq name> cache <required value>;  When we use monotonically increasing sequences for populating column values, the leaf block which is having high sequence key will be changing with every  insert, which makes it a hot block and potential candidate for a block split.   With CACHE SIZE (and probably with NOORDER option), each instance would use start using the sequence keys with a different range reduces the index keys getting insert same set of leaf blocks.  **o       Rebuild or shrink associated index after huge amount of data purge**  If there is a huge amount of data purge (delete) has been done, rebuild or shrink associated index should help to reduce the wait via alter index rebuild or alter index shrink command.  **o       Increase PCT\_FREE for the index**   Community Discussions   Still have questions? Use the communities window below to search for similar discussions or start a new discussion on this subject.   Note: Window is the **LIVE** community not a screenshot.   Click [here](https://community.oracle.com/community/support/oracle_database/database_-_rac_scalability) to open in main browser window. |