Scaling Indexes for very large and very busy databases

Saibabu Devabhaktuni

January 2015

Who Am I

1) Director of DB engineering at eBay Inc (PayPal)
2) Managing Oracle databases since 1998
3) Blog at http://sai-oracle.blogspot.com
4) Can be contacted at sadevabhaktuni@paypal.com

Scope

1) Only B-tree indexes are covered, applicable to IOT also.
2) No Domain indexes
3) Applicable for 11g/12c database versions
4) Applicable for Web scale OLTP and bulk workloads
5) Some of the observations here can be incorrect or may change in future versions

Agenda

1) B-tree index concepts
2) Scalability challenges
3) Index contention
4) Scalability options
5) Online index build

B-tree Index Concepts

1) Minimum of one key column entries in each index block
2) Conceptual analogy, new index tree level when power(single block entries, next level) reached
3) http://sai-oracle.blogspot.com/2006/03/how-to-predict-index-blevel-and-when.html for more information
4) Index is balanced at all level's of the tree
5) Uniqueness using rowid is enforced when non unique index created
6) Leaf blocks can be traversed in either direction at the leaf level

Oracle’s Implementation of B-tree Indexes

1) Root block address always stay same [1]
2) Last leaf block pointer key value in far right branch block defined as max value
3) Leaf blocks are split with 50/50 key entries unless max value is reached
4) New leaf block created with one key value entry when max value reached [2]
5) Least possible unique key values stored in branch blocks for leaf block pointers
6) ITL entry markers always cloned during leaf block split (optimized in 12c after ER 16075761)
7) Optimistic space management, i.e. freed leaf blocks are not unlinked right away from the tree

Index Design Criteria

1) Non partitioned simple B-tree index is the most optimal index to support all flavors of sql queries with least possible cost per execution, especially if optimizer is relying on index to avoid sort
2) Above index is also typically most space efficient and incur lesser physical I/O, but least scalable
3) Most scalable index designed for high DML's can restrict type of sql queries or increase execution cost
4) Above index pattern is typically less space efficient and incur higher physical I/O.
5) Information life cycle management (ILM) pose another challenge in balancing between scaling for DML's versus optimizing for reads.
6) Best possible optimization for primary key based queries leads to an IOT index, use it very sparingly as it causes secondary index based queries to be much more expensive.

Index Scalability Upfront Design Challenges

1) Unknown peak DML rate at design time
2) Evolving DML rate or access pattern changes over time
3) Evolving key value combination inter column relationship
4) Possible burst of DML after slow down up the transaction or application call stack
5) Possible ineffective partition pruning for any top sql
6) Possible conflict with ILM, i.e. forcing local indexes
7) Ability to identify hot indexes proactively
8) Cost of rebuilding indexes in future, i.e. it may force conservative and restrictive design upfront

Types of Index Contention

1) TX: Index contention due to block splits (DML limit can vary from 500 to 3000 monotonically increasing column inserts per second based on hardware and the system load)
2) Buffer busy waits at higher call stack (high DML)
3) Row lock contention for unique indexes (duplicate data inserts)
4) ITL contention (only 169 ITL slots available in 8k block)
5) Recursive space management operations after mass delete and insert (http://sai-oracle.blogspot.com/2009/04/beware-of-index-contention-after-mass.html , fixed in 12c, bug 8446989)
6) High water mark enqueue contention (flashback can aggravate it)

Monitoring and Detection of Index Contention

1) Goal should be to detect index contention at early stage before it grows to an outage
2) Real time monitoring of active sessions for index contention and group by current_row_obj#
3) Real time monitoring of ASH samples, group it by current_obj# for top indexes
4) Monitor segment statistics (v$segstat) for ITL contention
5) AWR report, look for top wait events and top segments
6) Check the rate of sequence gets for any indexes on sequence based columns
7) Profile increase in top DML executions for identifying any indexes prone for contention.

Index Scalability Principles

1) Carefully choose order of index key columns to reduce inserting at max value
2) Randomizing data arrival order of index key columns at application level if possible, i.e. combination of sequence, machine id, process id, etc.
3) Try to partition index key column data at application level, i.e. key column value sharding by using another column and adding it to index key column list, i.e. (mod(key1, n), key1)
4) Convert ordered sequences to unordered when creating indexes on those columns
5) Target for fewer DML's on index key columns (i.e. updates)
6) Target for index key value combination length to be as low as possible
7) Avoid global indexes if possible, i.e. for using partition operations to enforce ILM

Index Scalability Options at DB level

1) Reverse key index
   1) Pro: Up to 100 insertion points at any point in time for sequence based column values
   2) Con: Increased physical I/O due to data distribution and need exact query predicate values
2) Global hash partitioned index
   1) Pro: Better data affinity than reverse key and support range queries
   2) Con: Inefficient partition operations (ILM) and key column length can't be changed online
3) Function based index, i.e. using (mod (key column, x), key column) or virtual columns
   1) Pro: Better data affinity and efficient ILM partition operations
   2) Con: Require application query change and union all based range queries
4) Key column data arrival randomization at application level
   1) Pro: No changes at DB level, application queries based on data arrival pattern
   2) Con: Organizational challenges and less data affinity leading to more physical I/O
5) Using no ordered sequences based column indexes in RAC
   1) Pro: Scales well in RAC
   2) Con: Application dependency on sequence order and some queries can become expensive

Index Scalability Options (new in 12c)

1) Standard_hash function based index for large key columns
    1) Pro: key value length is fixed at 128 bytes (for 64 bit), no application code change is required
    2) Con: Only useful for the column value length much higher than 128 bytes
2) Deferred global index maintenance leading to better adoption (Implemented with ER 8677124)
    1) Pro: ILM partition operations are much faster (set event 43820 for space management bug)
    2) Con: Higher redo due to offline index entry deletes and higher physical I/O due to larger index
3) Partial indexes for partitioned tables
    1) Pro: Faster data loads and enable read optimized index for less active partitions
    2) Con: All query patterns need to be understood
4) Partitioned sequences (undocumented), i.e. using it for index key column
    1) Pro: Meant for scaling indexes in RAC environment
    2) Con: Need application compatibility for out of order sequences and slower order by queries
5) Online drop index operation incase if any index type or structure change needed

Online Index Build Scalability

1) Best designed online feature among all Oracle database HA features
2) http://www.nocoug.org/download/2012-02/Internals%20of%20online%20index%20build.pdf for internals of online index build
3) No DML contention during prepare or merge phase
4) Speed of index build is less influenced by rate of DML's compared to older implementation
5) Journal IOT table is automatically cleared up after aborting online index

ER’s for Further Scaling of Indexes

1) ER 12979221: Reducing Index Contention during branch/leaf block splits
2) ER 16075761: Initiate leaf block split when ITL limit reached
3) ER 9912950: Contention on underlying journal IOT table when scalable index is being build (i.e. reverse key or global hash partitioned index)
4) Bug 10038517: dbms_repair.online_index_clean waits for exclusive table lock
5) ER 8759587: Ability to create composite partitioned local index while the table is only partitioned at top level (i.e. table partition by range and local index composite partition by range and hash)
6) Bug 18715233: ORA-600 with online index build when MSSM tablespace is used

Summary

1) Consider designing index upfront for future scalability needs
2) Prefer index scaling through application changes
3) Prefer enforcing ILM policies through optimal index design
4) All operations of index scaling options at DB level are online as of 12c
5) Proactively monitor ASH data and awr reports for any signs of index contention
6) Don’t shy away from using online index build feature to fix index scaling
7) When designed properly Oracle indexes are capable of handling web scale workloads.