Andrzej BARCZAK,
Dariusz ZACHARCZUK,
Damian PLUTA

Siedlce University of Natural Sciences and Humanities,
Institute of Computer Science,
ul. 3 Maja 54, 08-110 Siedlce, Poland

Methods of optimization of distributed databases in oracle – part 1

Abstract. The paper presents actions required to tune a distributed database to obtain optimal the performance. Each method and its impact on the performance of a database depending on the environment or configuration is discussed. The last part of the paper describes the stages of the process and data to be collected to obtain positive results.

Keywords: optimization, distributed databases, Oracle

1. Introduction

Nowadays, we use distributed databases on regular basis, e.g. when withdrawing cash from ATMs, booking tickets to the cinema or scheduling a visit at a service station. The problem of performance improvement applies to every database, and is a complex issue, especially in case of a distributed database. It is required that a distributed database be optimized as a whole in order to obtain maximum performance. For this reason, concentration on just one area or node of a distributed database is not advisable. Studying such areas as communication between remote nodes, performance of replication mechanism, or a method of extracting data from remote nodes is necessary. All the above mentioned aspects will be discussed in this paper.

The authors assume that the reader has basic knowledge of the architecture of a distributed database and related issues. Terms will be explained briefly to present the problem being discussed. Some mechanisms (e.g. partitioning) may require the Enterprise version of the database.

1.1. Basic definitions

Key elements that make up distributed database architecture include: [1],[2],[4]

- database service name – specifies one or more names, which clients use to connect to an instance of a database; an instance registers the names in the process called LISTENER,
- database links – connects two physical databases and enables the user to access them as one logical database,
- materialized views also called snapshots – are, simply speaking, a copy of a remote table made at a certain point of time, in the context of distributed databases. In distributed systems, materialized views are used for synchronizing updates of data from remote nodes, including methods of solving conflicts,
- synonyms – may be aliases of each table, perspective, materialized view, sequence, procedure, function, package, user-defined object or another synonym,
- perspective also called view – is a logical representation of data contained in one or more base tables.
- service processes (dedicated – default, shared) – are created by a database to handle the requests of processes of users connected to an instance.

1.2. Advanced replication

More attention should be devoted to the mechanism of replication. Advanced replication uses the technology of distributed databases to share data between multiple nodes. It is worth emphasizing that a replicated database is not the same thing as a distributed database. In a distributed database, data are available at different locations, but a given table is at one site, while replication means that the same data are in different sites. The main reasons for using replication include [3]:

- Accessibility and performance – replication provides fast local access to shared data as the load is divided between many nodes.
- Offline processing – materialized view is a complete or partial copy (replica) of a source table made at a certain point of time, and enables the user to work on a subset of a database, which may be temporarily unavailable.
- Reduction of network load – replication may be used for distribution of appropriate data subsets to regional nodes of the system.

1.2.1. Multimaster replication

Multimaster replication environment consists of multiple equivalent master nodes that participate in the process of updating data. Changes made to one master node are propagated to all master nodes that participate in the process of replication. Multimaster replication increases availability of data and ensures equal load distribution. There are two types of multimaster replication: asynchronous and synchronous. Asynchronous replication captures local changes, stores them in a queue, and, at regular intervals propagates these changes to remote nodes. In synchronous replication, any changes are applied to data as in a transaction. This type of replication ensures data consistency at all nodes in real-time.

1.2.2. Snapshot replication

A materialized view is a source table replica made at a certain point of time. A source may be both a table at a master node and a materialized view at a snapshot node. Materialized views are updated by snapshot nodes that use updates also called refreshments incoming at regular intervals. Snapshot replication has the following advantages as it allows for:

- reduction of network load
- division of data into subsets
- offline work.

Oracle system provides a few types of materialized views, used in various replication environments:

- Primary Key materialized views are the default type of materialized view, and are updatable if the materialized view was created as part of a materialized view group, and FOR UPDATE was specified when defining the materialized view.
- ROWID materialized views are based on the physical row identifier in a source table. They can be used for materialized views based on source tables that do not have a primary key, or for materialized views that do not include all primary key columns.
- Complex materialized views may not be refreshed incrementally. A materialized view is considered complex when the defining query for a materialized view contains the following:
    - a CONNECT BY clause
    - INTERSECT, MINUS, UNION or UNION ALL set operations
    - the DISTINCT or UNIQUE keyword
    - Aggregating functions (with some exceptions): COUNT, AVG, SUM, VARIANCE and STDEV
    - Joins

The simplest method to determine whether a materialized view satisfies all the restrictions for incremental refreshment, is defining the materialized view with REFRESH FAST option. Oracle returns an error if the materialized view violates any restrictions for simple materialized views.

To perform a complete refresh of a materialized view, the materialized view's defining query is executed, which re-creates the materialized view. If a complete refresh is performed on a master materialized view, then a complete refresh must be performed on any dependent views.

Fast refreshes are more efficient than complete refreshes when there are few changes to the base object and when each column in join statement in the materialized view's defining query has an index on it.

2. Optimization of distributed databases

This chapter discusses issues that directly affect the efficiency of a distributed database, including mechanisms of communication between nodes of a distributed database, handling user requests and mechanisms to access data at nodes of a distributed database.

2.1. Optimization of service processes

The starting point for optimization of service processes is when one suspects that the system is inefficient or observes lower efficiency. In case of shared service processes, the first step involves determining whether dispatchers are capable of handling the amount of incoming requests and whether service processes are capable of handling the volume of requests. If a system is based on dedicated service processes, the first step is verification whether the number of processes is sufficient and whether the hardware resources are adequate to handle the processes.

Tuning shared service processes involves specifying the optimum number of dispatchers and service processes required for stable and efficient work of the system. To determine whether the number of dispatchers is sufficient for handling a given volume of requests, a dynamic system perspective V$DISPATCHE and V$QUEUE that selects the following data from the perspectives may be used:

```
D.NAME AS DISPATCHER_NAME,
ROUND((D.BUSY/(D.BUSY+D.IDLE))*100, 2) AS TOTAL_BUSY_RATE,
ROUND(Q.WAIT/Q.TOTALQ, 2) AS AVERAGE_WAIT
…
WHERE Q.TYPE = 'DISPATCHER' AND Q.PADDR = D.PADDR;
```

When the load of a database varies depending on, e.g. the time of day, the load ought to be determined in the peak time. Based on the returned results, it may be observed whether the dispatcher's processes are overloaded or not. If the busy rate (TOTAL_BUSY_RATE) exceeds 50% or wait times (AVERAGE_WAIT) assume significant values, the number of dispatchers requires increasing, e.g.

```
ALTER SYSTEM SET DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=5)";
```

Too many tasks in the queue of processes to be handled (information available in the system view V$QUEUE.ITEMS_QUEUED) and long wait time (V$QUEUE.AVERAGE_WAIT), requires considering increasing the number of service processes using initialization parameters SHARED_SERVERS and MAX_ SHARED_SERVERS, and continue monitoring the content of the queue of tasks to be handled. It is also worth checking the status of virtual circuits (V$CIRCUIT view) used by users to connect to the database via dispatchers and service processes. The values of QUEUE column store information about the current status of a virtual circuit: [5], [8]

- COMMON – waiting in a queue of tasks to be handled by the server process;

- DISPATCHER – waiting for a dispatcher, too high value may indicate server overloading;
- SERVER – being handled by a server's process;
- NONE – idle.

To conclusively determine whether the load on server processes is too high, one may use another request using V$SHARED_SERVER view and the following instruction:

```
ROUND((SS.BUSY/(SS.BUSY+SS.IDLE))*100, 2)||'%' AS TOTAL_BUSY_RATE
```

If the busy rate exceeds 50%, increasing the number of service processes and further monitoring of this aspect of database efficiency is recommended.

2.2. Sharing connections

Let us consider a system that is capable of handling 950 connections by each dispatcher process and a situation, which requires simultaneous handling of 4000 users connected via TCP/IP protocol, and 2500 users connected via TCP/IP protocol with SSL. Such a situation requires at least five dispatchers for TCP/IP protocol and at least three dispatchers for TCP/IP protocol with SSL:

```
DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=5)"
DISPATCHERS="(PROTOCOL=tcps)(DISPATCHERS=3)"
```

Sharing connections allows each dispatcher to handle more than 950 sessions. Let us assume that in this case each dispatcher is capable of handling 4000 sessions connected through TCP/IP and 2500 sessions connected through TCP/IP with SSL. Such a configuration reduces the number of required dispatchers to one for each protocol, and allows for more efficient use of dispatcher processes:

```
DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=1)(POOL=on)(TICK=1)
 (CONNECTIONS=950)(SESSIONS=4000)"
DISPATCHERS="(PROTOCOL=tcps)(DISPATCHERS=1)(POOL=on)(TICK=1)
 (CONNECTIONS=950)(SESSIONS=2500)"
```

However, a decision to enable connections sharing ought to be preceded by an analysis of shared service processes and dispatchers processes load. Connections sharing is efficient when connected clients are idle most of time, which allows the dispatcher to handle increased number of sessions.

2.3. Database links tuning

There are two types of links: shared and non-shared. An argument for using shared database links is a situation in which the number of users is much larger than the number of connections to the local database. An exception: a shared database link in a scenario with one user leads to a greater number of network connections. Therefore, using the link is recommended when many users require access to the same link.

2.4. Query optimization

In order to optimize SQL queries, the following tools may be used: EXPLAIN PLAN, SQL Trace or TKPROF1[1] to learn how queries are partitioned and distributed to individual nodes.

The most efficient method of distributed queries optimization is accessing remote databases only to obtain the required data. Let us consider a situation, which requires referring to two remote tables located at the same remote node. To improve the efficiency of such a query, the query ought to refer to the remote database only once, and a filter ought to be applied to the remote database, which reduces the amount of data sent to a local database to minimum. Such a query may be constructed using a collocated inline view. To describe this construction it is necessary to explain a few terms: [9]

- inline view – a query placed in FROM clause of another query that replaces the table in the master query:

  ```
  SELECT … FROM (SELECT empno, ename FROM emp@orc1.world) e …
  WHERE …;
  ```

- Collocated inline view – an inline view that selects data from multiple tables in a single database. This solution reduces the number of times the database is accessed, improving the performance of a distributed query.

It is worth noting that Oracle cost-based optimizer has the capability to rewrite multiple distributed queries into a form that uses a collocated inline view. Queries may also be constructed using subquery factoring, which allows for repeated use of the same query and improves its clarity.

2.4.1. Using cost-based optimization

The main optimization task is rewriting a distributed query into a form that uses a collocated inline view. The optimization consists of three stages:

1) All mergeable views are merged.
2) The optimizer performs a collocated inline view block test.
3) The optimizer rewrites a query into a form that uses a collocated inline view.

Cost-based optimization optimizes distributed queries according to the statistics for tables, to which the query refers, and calculations performed by the optimizer. For example, the query below was cost-optimized:

[1] „Tools and methods for optimization of databases in Oracle 10g. Part 2 – Tuning of hardware, applications and SQL queries", Studia Informatica, Volume 1-2(18), Publishing House of University of Natural Science

```
SELECT … FROM local l, remote1 r1, remote2 r2
WHERE l.c = r.c AND r1.c = r2.c AND r1.e > 300;
```

And, after optimization, takes the following form:

```
SELECT … FROM (  SELECT … FROM remote1 r1, remote2 r2
WHERE r1.c = r2.c AND r1.e > 300) v, local l
WHERE l.c = r1.c;
```

A collocated inline view reduces the volume of data to be processed in a remote node, and, as a result, the amount of data sent to a local node, and costly network traffic. Better performance of a query can be achieved by moving the condition r1.e > 300 which results in narrowing the dataset in the remote database.

2.4.2. Optimizer hints

For queries that require further optimization, optimizer hints may be used to obtain better results of cost-optimization [3]:

- NO_MERGE: used when a distributed query is based on expert knowledge and already uses an inline view. This hint is also useful when the distributed query contains aggregations, subqueries or complex SQL as a query that contains any of the elements mentioned above cannot be rewritten by the optimizer. Therefore, skipping optimization stage saves time.
- DRIVING_SITE: forces the execution of a query in a node other than the node selected by Oracle, e.g. in a remote node selected on the basis of expert knowledge.
- ORDERED: instructs the optimizer that the join of tables in the FROM clause should be performed in the order specified in the clause.
- LEADING: instructs the optimizer to use in the order of the join the set of tables specified in the hint first. Oracle recommends using this hint instead of ORDERED.

2.5. Materialized views tuning

Materialized views are an extremely useful mechanism, which allows for reduction of redundant i/o operations. In some situations they are irreplaceable. Despite their usefulness, there is one problem with materialized views, namely, ensuring that the materialized views reflect the current data from their source tables (materialized views may be based on multiple tables). Mechanism of materialized views in the Oracle database was optimized by the producer. Therefore, direct optimization is not possible. However, there are a few methods to improve the performance of this mechanism.

The first step to accelerate refresh process of a materialized view is the analysis and optimization of a query that defines a materialized view.

Base table partitioning may significantly shorten refresh time of materialized views. If a key, based on which the table was partitioned corresponds to the key used in the WHERE clause of the query that defines the materialized view, and, additionally, the refresh is performed in the complete model.

Another method to shorten materialized view refresh time, is to create indexes on base tables, which, to a great degree, correspond to the conditions in the WHERE clause of the query that defines the materialized view. Benefits of using indexes are particularly visible during refreshes performed in the complete mode, when the materialized view presents only a small portion of data from a base table, i.e. smaller than 15%. Otherwise, use of indexes may be inefficient.

Simultaneous refresh makes multiple processes simultaneously refresh a materialized view, which, consequently, accelerates the refresh process. Ideally, breaking the refresh process of a materialized view into 4 parallel processes should shorten the refresh time of the materialized view in question 4 times. In reality, the refresh may take longer, as the disk subsystem is not capable of handling four parallel processes with the same speed as one. Despite this, parallel processing is significantly faster. Prior to making a decision about parallelizing the refresh process of materialized views, an analysis of use of hardware resources of a database is required.

2.5.1. Comparison of simple and complex materialized views

Depending on a materialized view characteristic, one ought to consider using a few simple materialized views that may be refreshed incrementally instead of one complex materialized view that cannot be refreshed in this mode as it does not satisfy criteria for simple views. Let us consider two variants based on a view in a local database and one remote database with two base tables: [10]

a) A complex materialized view based on a complex materialized view which ensures high performance of queries in a local database, as the join operation was performed during materialized view refresh. However, due to the complexity of the materialized view, refresh process is realized in the complete mode, and, consequently, takes longer than incremental refresh.

b) A simple materialized view based on two simple materialized views in a local database, where the join is also executed. For this method, the performance of queries is lower than in variant A, which uses a materialized view. However, simple materialized views may be refreshed more efficiently if a materialized view log and incremental refresh are used.

To sum up, if the data are not refreshed often and higher performance of queries is required, then a complex materialized view is a better solution.

2.6. Size of refresh groups

Oracle is optimized for large refresh groups. Therefore, the refresh time of the same number of materialized views in a large refresh group is shorter than the refresh time of these materialized views grouped in small groups. During the refresh of such a group, each materialized view in a group is locked at a snapshot node for the amount of of time required for refreshing all views that belong to the same refresh group. With smaller refresh groups, the all materialized views in the group are locked for a shorter amount of time, required for performing a refresh. Network connectivity must be maintained while performing a refresh. Otherwise, all changes are rolled back so that the database remains consistent. Therefore, in cases where the network connectivity is difficult to maintain for a longer period of time, using smaller refresh groups is recommended.

3. The testing process

Testing the performance of a distributed database using the optimization methods presented above is not an easy task. Moreover, the results of tests only apply to the topology and configuration being tested. Such parameters as expert knowledge of a logical model of the optimized base, distribution of data in tables, or knowledge of technical conditions do not allow for testing one general solution. Change of any of the above mentioned parameters requires repeat tests. The role of each administrator of a distributed database is to pass through all the stages and apply the methods individually, and, based on results of analysis, decide on the best solution.

The process of testing a distributed database should incorporate the following [7]:

1) Tests of dedicated and shared service processes that involve simulation of loading a database using the above mentioned processes. The test provides information on CPU utilization, SGA and PGA utilization, number of logical and physical I/O operations, number of sessions, buffer hit rate, average query execution times and virtual circuits status.

2) Tests of public and shared database joins provide information on dispatchers processes load, server processes load, query execution times, rate of session wait events .

3) Tests of distributed queries using SQL Trace and TKPROF provide us with plans and real times of query execution. These tests include:

   − Test of collocated inline views and cost-based optimization

   − Test of optimizer hints

4) Test of materialized views

   − Test of base tables partitioning involves using SQL Trace and TKPROF to analyse the plan and time of query execution.

   − Tests of indexes on base table: see above

   − Test of simple and complex materialized views: depending on refresh mode, the amount of modified data in base tables requires consideration. Data upload times require analysis.

- Test of “small” and “big” refresh groups: refresh times of groups in sequential and parallel modes as well as refresh time of the whole set of materialized views require analysis.

4. Summary

With the results of all the required tests, tuning the distributed environment is a formality. However, it is worth emphasizing that optimization of databases is a broad topic and this paper does not exhaust the subject. Additionally, as it was already mentioned in the previous chapter, a key may be a case, which is being optimized, with solutions selected for this particular case. The path taken in this paper is based on a concrete practical problem. This allows to present partial results of tests performed during optimization of a real problem. Presentation of full results with related analysis is not possible due to their extensiveness.

The paper does not exhaust all tools and solutions available in the newest databases. ADDM (Automatic Database Diagnostic Monitor) mechanisms may be used to analyse data contained in AWR (Automatic Workload Repository) to identify potential bottlenecks. More papers may be written on this subject as it is still valid.

References

1. Alapati S. R.: Expert Oracle Database 10g Administration, Apress, 1276 pages, 2005.
2. Barczak A., Florek J., Sydoruk T.: Bazy danych, Wydawnictwo Akademii Podlaskiej, 280 pages, 2007.
3. Burleson D. K.: Oracle Tuning: The Definitive Reference, Rampant TechPress, 1200 pages, 2010.
4. Dye Ch.: Oracle Distributed Systems, O'Reilly, 552 pages, 1999.
5. Greenwald R., Stackowiak R., Stern J.: Oracle Essentials: Oracle Database 10g, 3rd Edition, O’Reilly, 400 pages, 2004.
6. Lonley K., Bryla B.: Oracle Database 10g Podręcznik administrator baz danych, Helion, Gliwice, 760 pages, 2008.
7. Oracle PL/SQL Database Code Library and Resources, [online] http://psoug.org/reference/library.html.
8. Price J.: SQL i Programowanie, Oracle Database 11g, Helion, 672 pages, 2009.

9. Urman S., Hardman R., McLaughlin M.: Oracle Database 10g. Programowanie w języku PL/SQL, Helion, 752 pages, 2007.

10. Wrembel R., Bębel B.: Oracle Projektowanie rozproszonych baz danych, Helion, 304 pages, 2003.