Performance Optimization of Oracle Distributed Databases

Shailesh Paliwal and Vinoth Babu Subash
Infosys Technologies Limited

It has been observed that applications accessing Oracle distributed databases can run into potential performance issues which often lead to customers spending long hours on their business operations, spending money on new design and additional hardware. The reason behind these problems is that the applications designed for extracting data from distributed databases require special performance considerations when compared against the local standalone databases. This paper covers the distributed databases key Concepts, Architecture, identification of key performance root causes and optimization techniques for distributed queries of Oracle's homogenous distributed databases. These techniques are for improving the application response time and throughput numbers. The paper also focuses on indicative performance comparison numbers for techniques which we had experienced in our various performance optimization exercises.

WHY DISTRIBUTED DATABASES?

Today's competitive business scenarios need their IT enterprises to operate across the globe. The enterprise databases play a very key role here as information is the backbone for organizations. The information is dispersed across the various databases in the form of transactional and data warehousing databases. Generally the business need is to have single consolidated data view from the transactional and data warehousing databases for proactive and faster decision making at the enterprise level and to get an edge over competitors. In Oracle environments these types of dispersed databases are connected through distributed database architecture and database links.

The paper starts with a generic discussion on the distributed databases and then narrows down to Oracle homogenous distributed databases, the concepts and architecture. The paper focuses on the performance optimization techniques for the Oracle homogeneous distributed databases which authors have experienced during their various performance engagements.

DISTRIBUTED DATABASE – CONCEPTS

DEFINITION

The databases maintained in physically separated locations connected over a network are referred to as distributed databases. The application users in this environment have access to their database and other remote databases transparently with the help of distributed database architecture.

In figure 1 databases 1, 2 and 3 are hosted on different computers connected over network for different distributed database setup. Any user connected to one of the database can either access data from his local database or consolidated data from one or more databases.

Oracle supports two types of distributed databases: homogenous and heterogeneous. In a homogenous distributed database system, each database is an Oracle database. In a heterogeneous, distributed database system, at least one of the databases is a non-Oracle database. The figure 1 below shows the illustration of Oracle homogeneous and heterogeneous distributed databases.

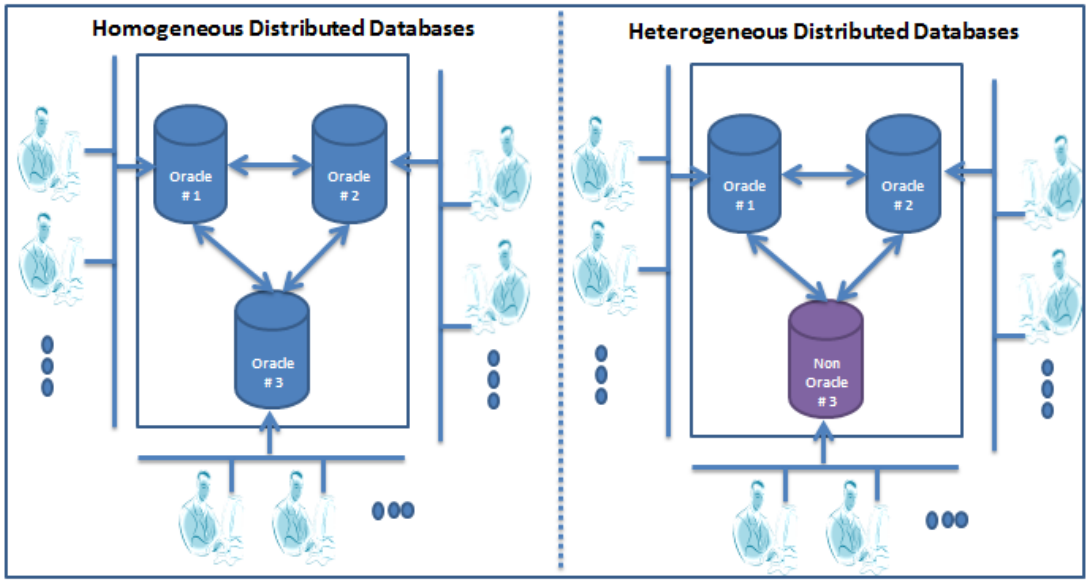

Figure.1. Oracle Homogeneous & Heterogeneous Distributed Databases

ARCHITECTURE

The physical server hosting the Oracle database software is the database server and the client is the one which is accessing the data from the server.

The physical servers in the network can host one or more databases. The distributed databases are connected with each other through database link which is unidirectional. The figure 2 shows the two types of queries which are sent to the database for fetching the data. The first type of query, direct query is fetching the data locally and the second type of query, in-direct query is fetching/manipulating the data remotely through database link. All these queries in the figure are part of a transaction which can either commit or rollback as a unit in all the associated distributed databases. This is referred as a two-phase commit mechanism.

Database link enables distributed databases. It defines a one-way communication path between two physical database servers. The figure 2 contains an example of a user connecting to a local database

for accessing a table in the remote database. This is done transparently with the help of the database link created in the local database.

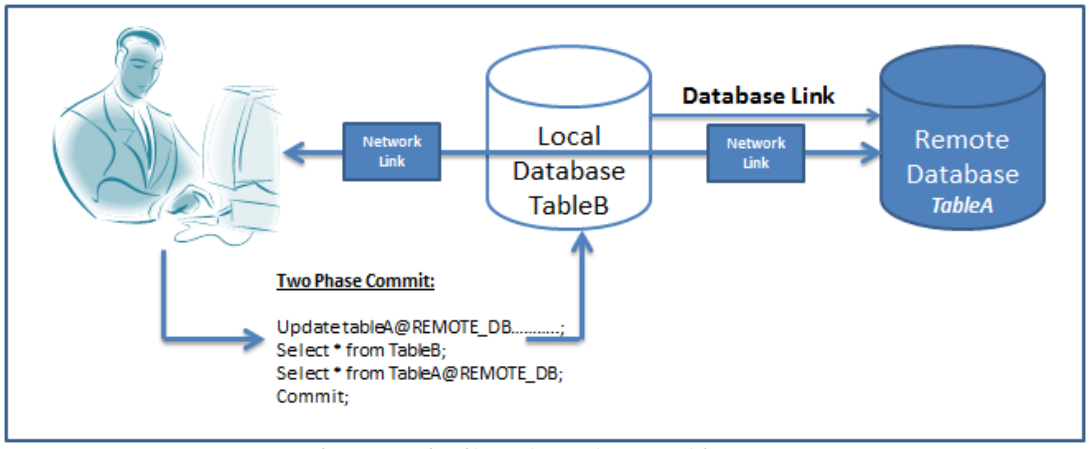

Figure.2. Distributed Database Architecture

WHAT ARE THE ROOT CAUSES OF PERFORMANCE PROBLEMS?

The authors have come across different performance problems while working on distributed queries of Oracle Homogeneous distributed databases, for various customers. They have classified the most important performance problems and their root causes into five major categories. The problems encountered do not include problems related to network configuration and bandwidth. The figure 3 below depicts the same:

Figure.3. Performance problem root causes in the distributed database environment

HOW TO RESOLVE THE PERFORMANCE PROBLEMS?

The authors have come up with a set of distributed query performance optimization techniques based on their extensive performance tuning experience to handle each kind of performance problem and its root cause. These performance improvement techniques can be applied based on each problem scenario. These techniques are unconventional ways of writing the distributed database queries. The figure 4 below shows the performance problems and the high level distributed query optimization techniques to address root causes.

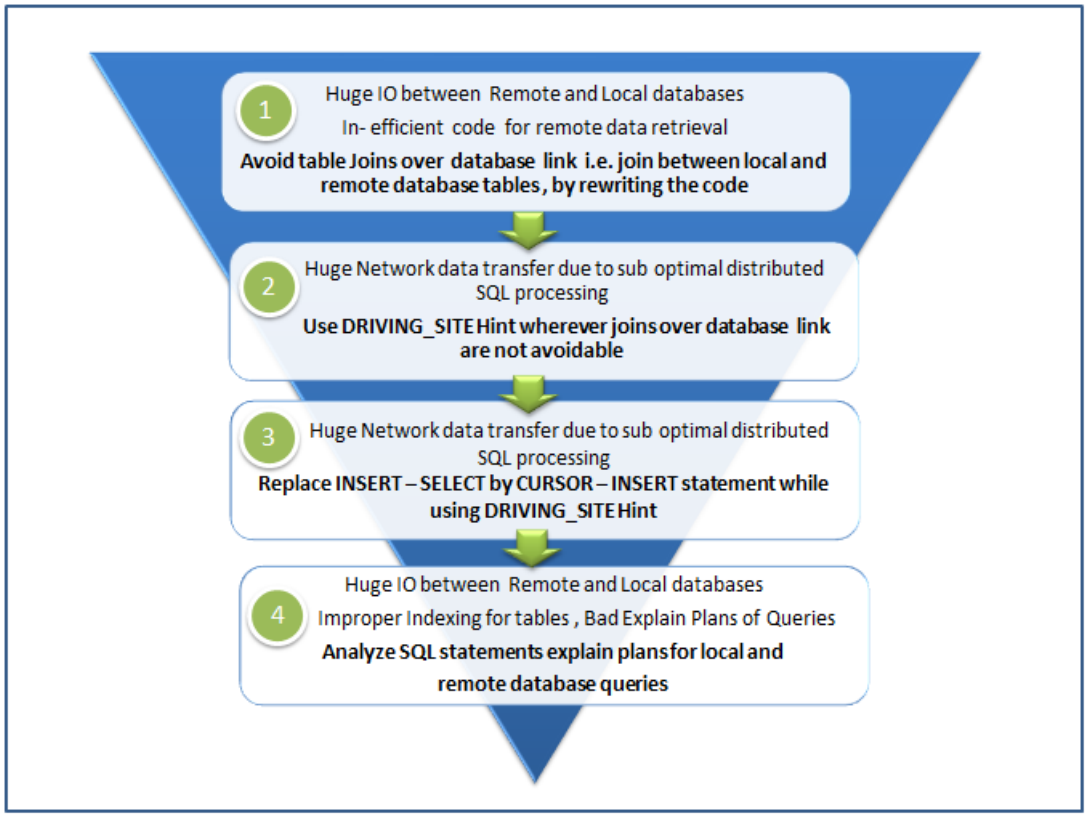

Figure.4. High level distributed query optimization techniques to address performance root causes

In order to ensure accurate results, we made sure the following prerequisites are in place before the implementation of the optimization techniques:

i. Performance test environment to remain same as the replica of production environment in terms of processing power, main memory and data volumes. It is recommended to have the test environment very close to production environment.

ii. The distributed databases, database link and network setups are exactly same as production environment and Performance test environment.

iii. Oracle database statistics are up to date and there are no stale statistics present in the databases. This is a highly recommended for any Oracle performance optimization exercise,

as Oracle optimizer totally relies on existing statistics present in the databases for accurate and optimum query execution paths. Oracle stale database statistics lead to in-accurate and in-consistent performance results and your whole performance exercise will go into soup.

Oracle performance tools and metrics used for performance analysis and optimization techniques implementation exercises are -

- SQL Plus Tool
- Database Alert log file
- Automatic Workload Repository (AWR)
- Automatic Database Diagnostic Monitor (ADDM)
- TKPROF
- Oracle Trace files
- Query execution plan

In order to address root causes, the database server performance was analyzed with the help of the above performance tools and metrics. The database server performance was made optimal before getting into the code level optimization. The server side optimization is not elaborated in the paper as it is not part of the scope. But it should be noted that the following database configuration parameters generally have an effect on the database server performance and thus, need optimization.

- db_cache_size
- pga_aggregate_target
- shared_pool_size
- optimizer_index_caching
- optimizer_index_cost_adj

The optimal parameter value depends upon the environment and application workload. So the values have to be configured after properly analyzing the database performance metrics.

The performance optimization techniques implementation are elaborated with relevant illustrations from the real world performance tuning engagements along with the performance benefits achieved after applying each technique.

OPTIMIZATION TECHNIQUES

OPTIMIZATION TECHNIQUE # 1

Avoid table Join over Database link i.e. join between local and remote database tables

PROBLEM SCENARIO

A critical patient claim category query was taking too long for a healthcare customer

ANALYSIS AND OBSERVATION

It has been observed that a table join over a database link, i.e. a join between a local and remote database tables, is very costly and drastically affects the application performance. Please find below for the representative existing code and the optimized code for this scenario.

EXISTING CODE

```
SELECT  P.PATIENT_LNAME, P.PATIENT_FNAME, H.HOSPITAL_NAME, C.CLAIM_ID,
        C.CLAIM_NAME, C.CLAIM_DATE, CP.CLAIM_PROVIDER_NAME
FROM    PATIENT P, HOSPITAL H,
        CLAIM@REMOTE_DB C, CLAIM_PROVIDER@REMOTE_DB CP
WHERE P.PATIENT_ID = C.PATIENT_ID
AND     P.PATIENT_ID = H.PATIENT_ID
AND     H.ADMIT_DATE = C.CLAIM_DATE
AND     C.PATIENT_ID = 1000
AND     C.CLAIM_DATE = SYSDATE
AND     C.CLAIM_PROVIDER_ID = CP.CLAIM_PROVIDER_ID;
```

In the existing scenario, the local database (LOCAL_DB) tables, namely PATIENT and HOSPITAL, were joined with the Remote database (REMOTE_DB) tables CLAIM and CLAIM_PROVIDER and it was taking around 297 seconds for a single execution.

OPTIMIZATION TECHNIQUE IMPLEMENTED

The joins over the database link was removed by rewriting the logic using temp tables and procedural code wherever possible.

OPTIMIZED CODE

In the optimized scenario, the existing code is broken into two sections. In the first section, the data is fetched from the remote database and inserted into the global temporary table (GTT) of the local database. In the second section, the GTT is joined with the other tables locally. The combined timings of both SQLs from sections were taking only 22 seconds. Please find below for the section i and ii SQL code:

i.
```
INSERT INTO CLAIM_GTT_TEMP@LOCAL_DB
(PATIENT_ID, CLAIM_ID, CLAIM_NAME, CLAIM_DATE, CLAIM_PROVIDER_NAME)
SELECT C.PATIENT_ID, C.CLAIM_ID, C.CLAIM_NAME, C.CLAIM_DATE,
        CP.CLAIM_PROVIDER_NAME
FROM    CLAIM C, CLAIM_PROVIDER CP
WHERE C.PATIENT_ID = 1000
AND     C.CLAIM_DATE = SYSDATE
AND     C.CLAIM_PROVIDER_ID = CP.CLAIM_PROVIDER_ID;
```

ii.
```
SELECT P.PATIENT_LNAME, P.PATIENT_FNAME,
        H.HOSPITAL_NAME, CGT.CLAIM_ID, CGT.CLAIM_NAME,
        CGT.CLAIM_DATE, CGT.CLAIM_PROVIDER_NAME
FROM   PATIENT P, HOSPITAL H,
        CLAIM_GTT_TEMP CGT
```

```
WHERE  P.PATIENT_ID = C.PATIENT_ID
AND    P.PATIENT_ID = H.PATIENT_ID
AND    H.ADMIT_DATE = C.CLAIM_DATE
AND    CGT.PATIENT_ID = 1000
AND    CGT.CLAIM_DATE = SYSDATE
AND    C.CLAIM_PROVIDER_ID = CP.CLAIM_PROVIDER_ID
```

PERFORMANCE RESULTS

The implementation of the technique resulted in around 13.5x performance improvement as shown in the graph below in figure 5.

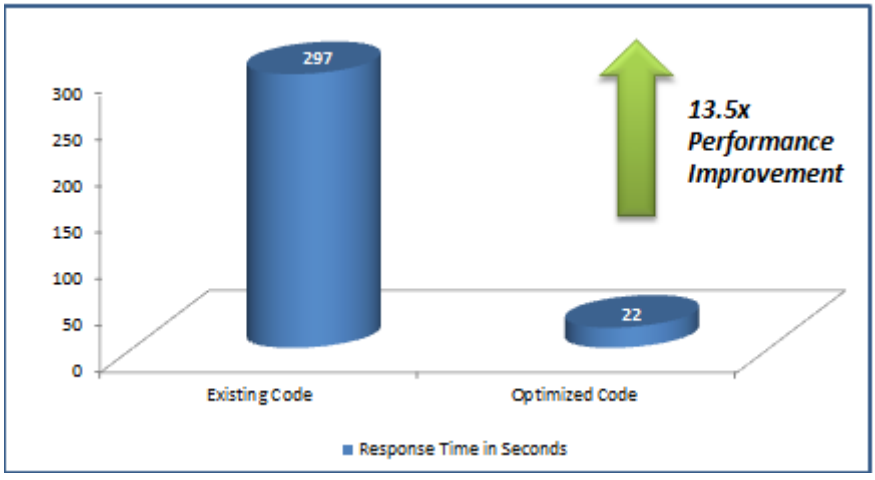

Figure.5. Optimization technique # 1 Performance Benefits

OPTIMIZATION TECHNIQUE # 2

Use DRIVING_SITE Hint wherever joins over database link are not avoidable

A brief about driving site Hint:

The driving site hint is generally used in the performance tuning of distributed databases. The hint tells the optimizer to execute the query from the mentioned site. This is used when the developer has a good understanding of the application and knows how the data should be processed within the application to reduce network data transfer between the distributed databases.

Please find below for an example of the SQL statement with a driving site hint:

```
SELECT /*+ DRIVING_SITE(rem) */*
FROM table loc,
        tableA@REMOTE_DB rem
WHERE loc.column_name = rem.column_name
AND     loc.column_name_2= 1889;
```

PROBLEM SCENARIO

The remote data access over the network for a critical transaction was exceeding the performance service level agreement (SLA) timings.

ANALYSIS AND OBSERVATION

The joins over database link was unavoidable due to business constraints. It was observed that while joining tables over the database link, response time was increasing exponentially. Furthermore, it was found that a huge amount of network data transfer was happening between the local to the remote database. Please find below an example of the existing code and the optimized code for this scenario.

EXISTING CODE

```
SELECT PP.PATIENT_ID, H.HOSPITAL_NAME, NM.DRUG_NAME, PSM.SERVICE_NAME
FROM   PATIENT_PRESCRIPTION PP, HOSPITAL H, NDC_MASTER@REMOTE_DB NM,
       PATIENT_SERVICE_MASTER@REMOTE_DB PSM
WHERE PP.PATIENT_ID = H.PATIENT_ID
AND    NM.DRUG_ID = PP.DRUG_ID
AND    PSM.SERVICE_ID = PP.SERVICE_ID
AND    H.ADMIT_DATE = SYSDATE
AND    PP.PATIENT_ID = 1000;
```

In the existing scenario, the local database (LOCAL_DB) tables, namely PATIENT_PRESCRIPTION and HOSPITAL, were joined with Remote database (REMOTE_DB) tables NDC_MASTER and PATIENT_SERVICE_MASTER in a query which was taking around 1080 seconds for a single execution.

OPTIMIZATION TECHNIQUE IMPLEMENTED

In this special scenario, the Oracle DRIVING_SITE Hint was used on the driving table on remote database table to reduce network data transfer between the databases and for optimal processing.

OPTIMIZED CODE

The remote database (REMOTE_DB) table NDC_MASTER was identified as a driving site to process the actual join. With this hint implemented in the optimized code, the amount of data getting transferred from the local to the remote database was reduced because the "patient_id=1000" clause limits the query to a great extent. The optimized below query completed in 90 seconds.

```
SELECT /*+ DRIVING_SITE (NM) */
       PP.PATIENT_ID, H.HOSPITAL_NAME, NM.DRUG_NAME, PSM.SERVICE_NAME
FROM PATIENT_PRESCRIPTION PP, HOSPITAL H, NDC_MASTER@REMOTE_DB NM,
       PATIENT_SERVICE_MASTER@REMOTE_DB PSM
WHERE PP.PATIENT_ID = H.PATIENT_ID
AND    NM.DRUG_ID = PP.DRUG_ID
AND    PSM.SERCIE_ID = PP.SERVICE_ID
```

```
AND     H.ADMIT_DATE = SYSDATE
AND     PP.PATIENT_ID = 1000;
```

PERFORMANCE RESULTS

The implementation of the technique resulted in around 12x performance improvement by applying this performance technique as shown in graph below in figure 5

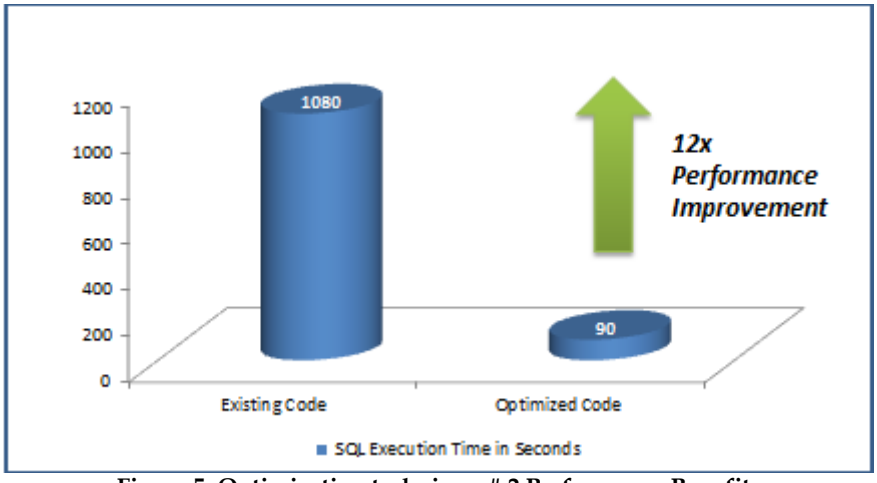

Figure.5. Optimization technique # 2 Performance Benefits

OPTIMIZATION TECHNIQUE # 3

Replace INSERT – SELECT by CURSOR – INSERT statement while using DRIVING_SITE Hint

PROBLEM SCENARIO

A healthcare application transaction involving data insertion into a local table based on the remote querying of the data was taking too long to complete.

ANALYSIS AND OBSERVATION

The table joins over the database link was unavoidable and the query was taking a long time even with the DRIVING_SITE hint. The Oracle optimizer was ignoring the DRIVING_SITE hint in the query, so the distributed DML statement wasn't executing on the database where DML resides.

Oracle Metalink reference – 5517609: DRIVING_SITE HINT IS IGNORED FOR INSERT AS SELECT

A query joining two tables using driving_site hint is performing as expected. Insert into a local table using the same query is ignoring driving_site hint. This is not a bug. A distributed DML statement must execute on the database where the DML target resides. The DRIVING_SITE hint cannot override this.

EXISTING CODE

In the existing scenario, the local database (LOCAL_DB) table PATIENT_DRUG_AND_SERVICE is populating from the query output of local tables (PATIENT_PRESCRIPTION and HOSPITAL) joined with Remote database (REMOTE_DB) tables (NDC_MASTER and PATIENT_SERVICE_MASTER) for the Patient drug and services. The driving site was used in the query and it was completing in 595 seconds.

```
INSERT INTO PATIENT_DRUG_AND_SERVICE
(PATIENT_ID, HOSPITAL_NAME, DRUG_NAME, SERVICE_NAME)
SELECT /*+ DRIVING_SITE (NM) */
        PP.PATIENT_ID, H.HOSPITAL_NAME, NM.DRUG_NAME, PSM.SERVICE_NAME
FROM PATIENT_PRESCRIPTION PP, HOSPITAL H, NDC_MASTER@REMOTE_DB NM,
        PATIENT_SERVICE_MASTER@REMOTE_DB PSM
WHERE PP.PATIENT_ID = H.PATIENT_ID
AND     NM.DRUG_ID = PP.DRUG_ID
AND     PSM.SERCIE_ID = PP.SERVICE_ID
AND     H.ADMIT_DATE = SYSDATE
AND     H.PATIENT_ID = 1000;
```

OPTIMIZATION TECHNIQUE IMPLEMENTED

In the optimized scenario, the logic was changed from conventional INSERT - SELECT into CURSOR--INSERT and Oracle server started considering the DRIVING_SITE Hint.

OPTIMIZED CODE

In the optimized scenario, a Cursor named DRUG_SERV_CUR was introduced for fetching the data and it was then inserted into the PATIENT_DRUG_AND_SERVICE of local database (LOCAL_DB) for better performance. In this case, the query was using the DRIVING_SITE and it was getting executed in 98 seconds.

Please find below for the representative optimized code snippet for this scenario:

```
FOR DRUG_SERV_CUR IN
       (SELECT /*+ DRIVING_SITE (NM) */
                PP.PATIENT_ID, H.HOSPITAL_NAME,
                NM.DRUG_NAME, PSM.SERVICE_NAME
        FROM  PATIENT_PRESCRIPTION PP , HOSPITAL H,
                NDC_MASTER@REMOTE_DB NM,
                PATIENT_SERVICE_MASTER@REMOTE_DB PSM
         WHERE PP.PATIENT_ID = H.PATIENT_ID
         AND     NM.DRUG_ID = PP.DRUG_ID
         AND     PSM.SERCIE_ID = PP.SERVICE_ID
         AND     H.ADMIT_DATE = SYSDATE
         AND     H.PATIENT_ID = 1000);
LOOP
      INSERT INTO PATIENT_DRUG_AND_SERVICE
     (PATIENT_ID, HOSPITAL_NAME, DRUG_NAME, SERVICE_NAME)
      VALUES (DRUG_SERV_CUR.PATIENT_ID, DRUG_SERV_CUR.HOSPITAL_NAME,
                DRUG_SERV_CUR.DRUG_NAME, SERVICE_NAME) ;
END LOOP;
```

PERFORMANCE RESULTS

The implementation of the technique resulted in around 6x performance improvement by applying this performance technique as shown in graph below in figure 6.

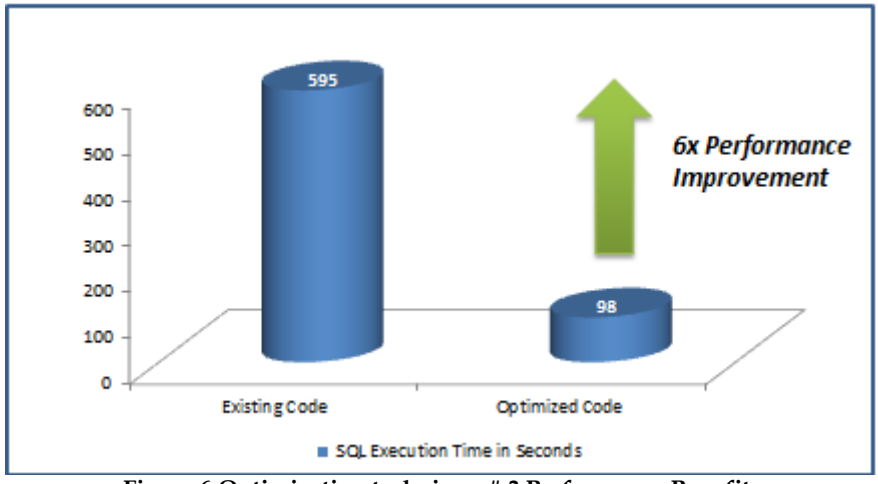

Figure.6 Optimization technique # 3 Performance Benefits

OPTIMIZATION TECHNIQUE # 4

Analyze SQL Statements explain plan to optimize performance.

PROBLEM SCENARIO

A critical business transaction for a healthcare customer was rendering sub-optimal throughput.

ANALYSIS AND OBSERVATION

The transaction was drilled down to the SQL statements, which were taking 95% of the total transaction time. The SQL explain plan was analyzed and it was observed that the SQL was not picking up the index. It was found that the index was in a disabled status; it was disabled for a migration activity.

SQL STATEMENT

```
SELECT PATIENT_ID, NAME, LOCATION FROM PATIENT_DRUG A
WHERE PATIENT_ID = (SELECT PATIENT_ID FROM DRUG
                    WHERE DRUG_ID = :B1 AND DRUG_CATEGORY_ID = :B2)
```

ORIGINAL EXPLAIN PLAN

```
0    SELECT STATEMENT OPTIMIZER=CHOOSE (COST=2489 CARD=1 BYTES=10)
1  0   TABLE ACCESS (FULL) OF 'PATIENT_DRUG' (COST=2487 CARD=1 BYTES=10)
2  1     TABLE ACCESS (BY INDEX ROWID) OF 'DRUG' (COST=2 CARD=1 BYTES=12)
3  2       INDEX (UNIQUE SCAN) OF 'PK_DRUG' (UNIQUE)  (COST=1 CARD=1)
```

OPTIMIZATION TECHNIQUE IMPLEMENTED

The primary key index on the table "PATIENT_DRUG" table was enabled.

OPTIMIZED EXPLAIN PLAN

```
0     SELECT STATEMENT OPTIMIZER=CHOOSE (COST=4 CARD=1 BYTES=8)
1   0  TABLE ACCESS (BY INDEX ROWID) OF 'PATIENT_DRUG' (COST=2 CARD=1 BYTES=8)
2   1    INDEX (UNIQUE SCAN) OF 'PK_PATIENT_DRUG' (UNIQUE)  (COST=1 CARD=1)
3   2      TABLE ACCESS (BY INDEX ROWID) OF 'DRUG' (COST=2 CARD=1 BYTES=10)
4   3        INDEX (UNIQUE SCAN) OF 'PK_DRUG' (UNIQUE)  (COST=1 CARD=1)
```

PERFORMANCE RESULTS

The SQL execution time has decreased to 10 milliseconds from 1700 milliseconds after the optimization technique was applied. The throughput of the transaction had increased by 170 times. The figure 7 shows the performance benefits chart.

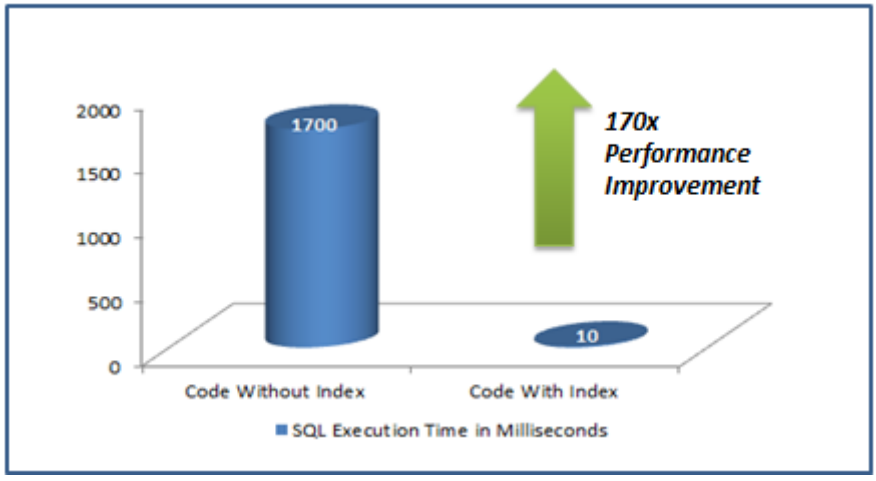

Figure.7 Optimization technique # 4 Performance Benefits

This standard technique has to be followed for all the SQL statements of the application as a performance sanity check. The performance analysis of SQL explain plans can solve a majority of the performance problems. In our experience, 80% of the performance problems are because of not properly analyzing the SQL explain plans during the development stages. The common pitfalls observed are missing indexes, improper indexing strategy and improper coding techniques.

OVERALL PERFORMACE IMPROVEMENT

We had applied all the above mentioned techniques for three critical batches in one of the performance tuning assignments. The throughput of the batch programs have increased by three folds to twenty folds for the different batches. The figure 8 below shows the performance improvement folds of the three batches individually:

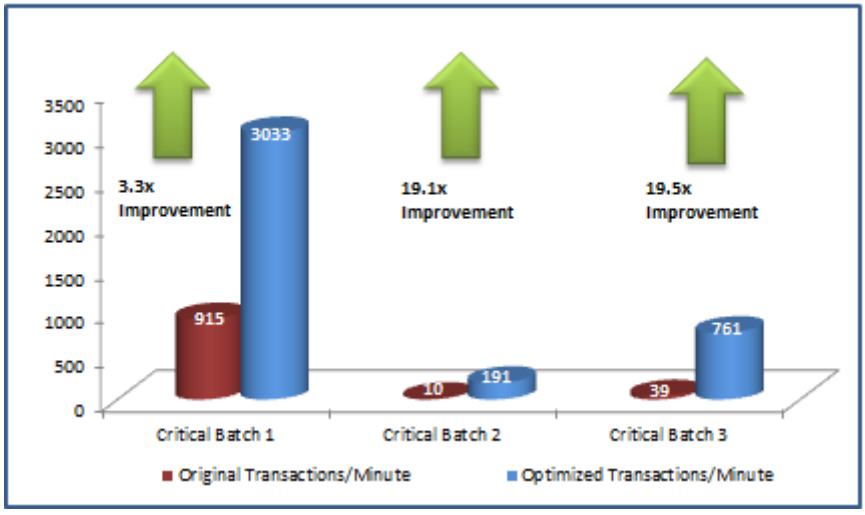

Figure.8 Overall Benefits using all techniques

SUMMARY

This paper emphasizes on following simple but effective performance optimization techniques for distributed database queries performance tuning:

1. Avoid table Join over Database link i.e. join between local and remote database tables
2. Use DRIVING_SITE hint wherever joins over database link are not avoidable
3. Replace INSERT – SELECT by CURSOR – INSERT statement while using DRIVING_SITE hint
4. Analyze SQL Statements explain plan to optimize performance

The performance benefits from implementation of these techniques are enormous and may vary depending upon the environment.

DISCLAIMER

The performance results for the optimization techniques can vary depending upon the environment and the data. We strongly recommend testing the techniques in the sandbox environment and analyzing the performance metrics for the performance numbers before implementing the changes in the production environment.

The authors have taken great care while coming up with the contents of this paper, but any and all responsibility for any loss, damage or destruction of data or any other property which may arise from relying on the paper is explicitly disclaimed. The authors are not liable for monetary damages arising from such loss, damage or destruction.

REFERENCES

1. Oracle Database Administrator's Guide 10g Release 2
2. Oracle Database Performance Tuning Guide 10g Release 2
3. Haroun Rababaah. 2005. Distributed Databases fundamentals and research
4. Mark L. Gillenson. 2004. Fundamentals of Database Management Systems. Wiley E-Books. www.wiley.com/
5. My Oracle support - https://support.oracle.com/CSP/ui/flash.html

ACKNOWLEDGEMENT

We greatly appreciate the guidance, valuable feedback and insights from our Mentor **Somasekhar Pamidi**, Principal Architect - Infosys Technologies Limited for this paper.

AUTHOR'S PROFILE

Vinoth Babu Subash is Technology Architect at Infosys's "High Performance and Cloud Computing - Enterprise Technology Modernization" Practice of System Integration Unit.
He has around 10 years of IT experience in performance engineering and optimization areas in very large database systems. He specializes on performance management areas in Oracle databases, SQL Server databases, Oracle Applications ERP and Siebel CRM Suites. He has worked extensively on applications workload management, Database and SQL tuning for database systems for clients in various businesses.
He is an Oracle 8i, 9i and 10g Database Administrator Certified Professional.
He can be contacted at vinothbabu_s@infosys.com

Shailesh Paliwal is a Senior Technology Architect at Infosys's "High Performance and Cloud Computing - Enterprise Technology Modernization" Practice of System Integration Unit.
He has over 14 years of IT experience articulating the next steps in the evolution of information technology toward strategic business applications and services that deliver performance coupled with intelligence throughout organizations. He has specialized expertise in the areas of Data and Information Architecture, Database Design, Data warehouse design , Data Modeling, Dimension Modeling, NFR Validation, Workload Modeling, Performance Tuning , Master Data Management and Business Intelligence.
He is a DAMA Certified Data Management Professional and TOGAF certified Enterprise Architect.
He can be contacted at shailesh_paliwal@infosys.com