PART 1 (40 min)

- What do we mean by Query Performance
- Integration of Query Performance with Application life cycle
- Defining Query Performance Problem Statement

PART 2 (55 min)

- Models for Predicting Query Performance for Data Volume
- Models for Predicting Query Performance for Concurrent workload, new workload etc.
- Future Direction of Research
Take Away

• Why SQL performance prediction is needed?

• Various modelling techniques for predicting a SQL performance.

• Integration of SQL Performance Prediction model with Application life cycle.

• A performance prediction model for SQLs in Application development.

• Future research scope for Query performance prediction
Architecture & Performance Metrics

- Query Elapsed Response Time
- Query Execution Time
- Query Cost
- Query IO Time
- Query Computation Time
- Query Performance in isolation
- Query Performance in concurrent workload
Varying Data size and Workloads

Production System
Data Size=X

After 1 year

Production System
Data Size=2*X

After 3 years

Production System
Data Size=10*X
Performance Violation
Problem Statement

Predict performance of structured query for production system before its deployment while being transparent to the underlying hardware and database system to ensure query performance Service Level Agreement (SLA) compliance

Predict Query Performance for-
- Varying data size
- Varying mix of queries
- Varying number of concurrent queries
- Varying type of Workload

Input: SQL, Data Size, System Configuration

Output: SQL Execution Time
Motivation

- SLA Compliance
- Large Data Loading Time
- Large number of queries mix/degrees for testing
- Large testing time -> delayed deployment
- Unavailability of Resources
- Highly expensive
- Prospective capacity planning for application server maintenance
- Avoid query starvation, fair performance to users - Analytics
- May use for Query optimization , query scheduling, reduce application benchmarking cycle time.
Challenges

- Limited availability of the Production System
- Unavailability of the projected data size DB system
- Estimation of complex query cardinality.
- Transparency to the underlying Hardware Subsystem
- Transparency to the underlying data management server - DB Server (Oracle, Postgres), Big data architectures
Challenges

• Limited availability of the Production System

• Unavailability of the projected data size DB system

• Estimation of complex query cardinality.

• Transparency to the underlying Hardware Subsystem

• Transparency to the underlying data management server - DB Server (Oracle, Postgres), Big data architectures

Data semantics is the key to most of these challenges
Approach

Database Application Life Cycle Phases:

- Application Requirement Analysis
- Query Design and Development
- Query Testing
- Application/Query Deployment
- Application in Production

We will focus on Read query right now
<table>
<thead>
<tr>
<th>DB Application Cycle</th>
<th>Performance Cycle</th>
<th>Performance Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design and development of DB and queries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB queries testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application deployment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application in Production</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Requirement Analysis

• Model data value ranges
  – min and max value during application life cycle? e.g. person age = 0, 125

• Model data distribution?
  – Does an applications stores all suppliers of a nation together (111122222333..) or at least one supplier from each nation (123..123…123..)

• Model data relation
  – find relation across different data objects/types. This helps in building correct data generator for the application testing. e.g. a supplier can supply at most 80 parts.

• Model data growth
  – how fast data is growing . This helps in doing capacity planning for the DB system and defining the SLA compliance time period. e.g. with addition of one supplier, partsupply data increases by 80.

• Capture business queries with their SLA
Design and Development of DB and Queries

• Model for optimal Database Schema design
  – number of tables, columns in the table and Indexes

• Model for designing tuned Query
  – Use of hints to force type of join
  – Order of tables in the join
  – can be tuned for better performance
Query Testing in Isolation

- Optimal DB server settings to get better query(ies) performance

- Query plan viewer to get optimal execution plan for query.
  - This could be feedback to DB design changes as well

- Small database is created and query is tested for SLA compliance in isolation.

- Different operator and IO access models could be build based on measurements.

- Statistical analysis of different concurrent queries.
Application Deployment

- Different queries of the application should meet SLA on deployment and subsequently
- Unavailability of information about history of queries executions.
- Queries can be executed multiple times with different DB settings and data sizes
- Complete flexibility on the testing system
- Predict queries performance with increase in data size
- Predict query performance in different types/sizes of concurrent workload.
Application in Production

• System should work without failing or violating performance guarantee for any query.

• Limited flexibility to do any measurements.

• Application is in operation, past query execution history could be collected and statistical model could be built

• Current database statistics could be pulled in to understand the data ranges and relation.

• Predict performance of a query before scheduling it for execution – for DW community

• Generate advanced alerts when frequent running queries will fail SLA – for DBA

• Monitor progress of query execution – for DBA
Varying Workload Mix

Production System
Data Size=X

New set of queries arriving on system
<table>
<thead>
<tr>
<th><strong>DB Application Cycle</strong></th>
<th><strong>Performance Cycle</strong></th>
<th><strong>Performance Model</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Analysis</td>
<td>DB Capacity planning</td>
<td>1. Data distribution, relation and growth pattern</td>
</tr>
<tr>
<td>Design and development of DB and queries</td>
<td>Tune database design</td>
<td>2. Business queries</td>
</tr>
<tr>
<td>DB queries testing</td>
<td>Tune queries and DB settings for best performance</td>
<td>Database schema and queries design</td>
</tr>
<tr>
<td>Application deployment</td>
<td>1. Tune queries for SLA compliance on large data</td>
<td>Plan viewer and Tool to get database system and schema statistics</td>
</tr>
<tr>
<td>Application in Production</td>
<td>2. Tune query for concurrent workload</td>
<td>Model to predict query performance with data and workload growth</td>
</tr>
<tr>
<td></td>
<td>1. Predict performance of incoming query before scheduling</td>
<td>1. Model to predict query performance in production system</td>
</tr>
<tr>
<td></td>
<td>2. Generate Alert for performance failure</td>
<td>2. Model to predict data size on which a query will fail SLA</td>
</tr>
<tr>
<td></td>
<td>3. Monitor performance of query</td>
<td>3. Tool to show progress of query</td>
</tr>
</tbody>
</table>
Performance Violation -> Data Volume
Query Performance Prediction Challenges

- Limited availability of the Production System
- Unavailability of the data of projected data size on DB system
- Unavailability of the query execution steps and cardinality
- Transparency to the underlying Hardware Subsystem
- Transparency to the underlying data management server - DB Server (Oracle, Postgres), Big data architectures
Query Performance Prediction Challenges Handling

• Limited availability of the Production System
  – One time measurements

• Unavailability of the projected data size DB system
  – Emulation of large data volume DB

• Unavailability of the query execution steps and cardinality
  – Cost based optimizer helps in getting this information

• Transparency to the underlying Hardware Subsystem
  – Use one time measurements to model hardware system
  – Measurements on ‘X’ system are extrapolated for ‘Y’ system

• Transparency to the underlying data management server - DB Server, Big data architectures
  – Cost based Relational Database System such as Oracle, Postgres
  – Capture database operations behaviour on large data size
Query Execution Plan

SELECT STATEMENT

SORT AGGREGATE

NESTED LOOPS

FULL TABLE SCAN (SUPPLIER)

INDEX RANGE SCAN (PARTSUPP_SK)
Traditional Approaches I

Linear extrapolation

- Measure query performance on small data size and linearly extrapolate to estimate the query performance for higher size.
- Query ERT on large size ‘L’ using measurement on small size “S”

\[
\text{Query } ERT_L = \text{Query } ERT_S \times \frac{L}{S}
\]

Pifalls

- Query plan may change
- Operator may spill over to disk
- Non uniform growth of data
BREAK!!
Traditional Approaches II (CBO based DB)...

Cost based Optimizer
- Generates optimal query execution plan before actual execution
- Plan shows ‘cost’ signifying resources required during query execution
- Uses DB statistics to get optimal execution plan

Using ‘Cost’ for extrapolation
- Given by Optimizer
  \[
  Query\ ERT_L = Cost_L \times \text{Time per Single IO}
  \]
- Query ERT on large size ‘L’ with using measurement on small size “S”
  \[
  Query\ ERT_L = Query\ ERT_S \times \frac{Cost_L}{Cost_S}
  \]

Pitfalls
- DB statistics for projected database size are not available.
- Cost calculation is pessimistic
- Require updated statistics
- Caching effect not captured e.g nested loop join
- Estimated Cardinality may be different
Traditional Approaches II (CBO based DB)

Using ‘Cost’ for very very large data sizes.

- Execution Time is IO predominant whether table access or temporary tablespace IO
- Calculate correct number of IO and time per IO

\[ \text{Query } ERT_L = Cost_L \times \text{Time per Single IO} \]

Pitfalls

- Size of IO may depend on type of operator
- Cost is miscalculated for repetitive Index Scans especially in nested loop join
- Correct measurement of time per IO.
Analytical Approach using Queuing Model

Model Database system as multi queue system (MVA) having as many CPU blocks as number of cores and multiple disk service centres.
Analytical Approach using Queuing Model

• Prediction Needs
  – Average Cpu service time
  – Average Disk service time
  – Number of CPU visits
  – Number of Disk visits
  – Queuing time is zero for query in isolation

• Query $ERT_L = Cpu \ Visits_L \times \text{avg Cpu service time} + Disk \ Visits_L \times \text{avg disk service time}$

Challenges
• Query has both random and sequential IOs
• Query IOs may be of different sizes for operator, index access and full table access
• Memory access time not accounted for
• Estimating cpu and disk visits
Analytical Approach using Queuing Model

• Measurements using calibrated queries to calculate cpu mean service time for different operators, index access and IO access
• Measurements using calibrated queries to calculate disk mean service time for random and sequential IO access
• Query Execution plan to get number of random IOs and sequential IO

\[ CO = ns \cdot cs + nr \cdot cr + nt \cdot ct + ni \cdot ci + no \cdot co \]

e.g. \( Ct \) calculated using \( \text{Select count(*) from table} \) and \( \text{Select * from table} \)

Reference paper for Postgres.
Query Performance Prediction Model

Use Measurements, Analytical models, Cost and Linear extrapolation together

Table Stats, Column Stats, Index Stats, Database Stats, System Stats

DB Schema & Projected Size

Model

Elapsed time

Execution Time

Net time

IO time

CPU time

Query Performance Measurements

SQL, OS (traces)
DB (ERT, Plan, traces)
Broad Approach

- Emulate the projected size database
- Get Query Execution Plan
- Predict ERT for each step
  - Get total IO access, IO access with DB cache miss, overheads per miss
  - Get SQL operator spill over cost, overhead per spill over
- Merge ERT of all steps
- Add network time (either constant or linearly extrapolated)
Database Emulator

- Capture database statistics either through application requirement specifications or from the production system.
- Linear extrapolation of database statistics
  - Table Statistics – num rows, num blocks etc.
  - Column Statistics- min, max, density
  - Index Statistics- num rows, density, clustering factor etc.

Refer:
Query Components

Select sum(s_acctbal) from supplier
where s_suppkey=ps_suppkey

SELECT STATEMENT
SORT AGGREGATE
NESTED LOOPS

FULL TABLE SCAN (SUPPLIER) INDEX RANGE SCAN (PARTSUPP_SK)

Mapped Components are:

1. Select /*+ FULL */ s_acctbal, s_suppkey from supplier
2. Select /*+ index(partsupp_sk) */ count(*) from partsupp where ps_suppkey >0
3. Nested join Matching
4. Aggregate Time
5. Network Time

Validations

1. Matching number of logical reads and physical reads
2. Matching number of rows outputted
3. Matching execution time
Two Major Components

- Table access IO Time
- Operator Time
Query Taxonomy for IO Access

Refer:
R. Singhal and M. Nambiar, “Measurement based model to study the affect of increase in data size on query response time”, Rekha Singhal, Manoj Nambiar, Performance and Capacity CMG 2013, La Jolla, California, November 2013. Also Published in TactTics 2013.
IO Access Time

1. Data Access Pattern from DB perspective
2. Requesting data in same data access pattern: OS cache and Disk access
3. Linearly extrapolated from ‘point of measurements’ for block accesses if temporal distances between them is 1. e.g. Full table scan, Primary index scan
4. Non Linear when temporal distance between repeated accesses > 1. e.g. Select * from table where col > 9 (index scan on non unique index on ‘col’)
Full Table Scan (FTS)

- Data Access Pattern file
  - system read of size database block size (8K)
  - 4 pairs of system read calls of sizes 64K and 56K and 1MB subsequently.
  - last system read call of remaining size.
- Calculate $LR$ (approx. 1 MB or more). The inflexion size is where, $LR > 95\%$.
- Linear extrapolation from inflexion size
Index Scan

- **Input**: Query \((\text{with } \text{key} < \text{val}, \text{key} < \text{val} \text{ and } \text{key} = \text{val})\)
- **Steps**
  - Emulate large size database
  - Calculate number of matched key values \((MKV)\)
  - Total qualified leaf nodes, \(QL = Lsize \times MKV\)
  - The inflexion size is where computation (CPU cost) is significant
Fast Index Scan (AFIS)

- \((Root + branch\ nodes)\) reads are generated each starting from a random address. This is followed by \(QL-1\) sequential accesses.
Primary Index Scan (PIS)

- $(1+\text{branch nodes})$ system reads
- $QL-1$ sequential accesses interspersed with $Dsize$ sequential accesses.
- leaf and data blocks addresses are random to each other.
**SQL Operators**

- **Hash Join (T1, T2)**
  - Time = Access and Hashing (T1) + Access (T2) + Probing(T2)

- **Sort Merge Join (T1, T2)**
  - Time = Access and Sort(T1) + Access and Sort(T2) + Merge (T1+T2)

- **Nested Loop join**
  - Time = Access (T1) + Index Scan of T2 for Numrows(T1) times + Matching Time(T1)

- **Aggregate (N rows)**
  - Time = linear function of N.
Linear Model for Operator Time

• Linear for Hash join if the number of rows in the first table (hashed) does not grow with data size and T2 grows uniformly

    select * from T1 t1, T2 t2 where t1.c1 = t2.c1 and t1.c2 < 10;

• Linear for Nested Loop join if T1 grows uniformly and it is primary index scan for sorted data from T1.
  – ‘c1’ is primary key T1 and there is index on t2.c1 as well which returns sequential data blocks.

• Linear for Sort Merge join if T1 and T2 grows uniformly and sort does not spill over.
  – Check from execution plan if spill over occurs
Model for Non Linear Growth in Execution Time

• Non linear IO access
  – Overheads of DB cache miss, OS cache miss
  – Number of DB cache misses.

• Non Linear Hash Join
  – Hashing spill over – depends on ‘cost’ and memory available

• Non Linear Nested Loop
  – Number of DB cache misses for table and index accesses.

• Non linear Sort Merge Join
  – Number of spill overs i.e. extra disk reads/writes
  – overhead per spill over
Benchmarks Available for Validations

- TPC-DS for decision support application
- TPC-C for OLTP applications
- TPC-H for DW application with capability for generating different size of data.
Experimental Setup

- Intel quad core server with 4GB RAM, 1 TB SAN
- DB server - Oracle 11g
- The database schema and data are generated using an open source dbgen utility based on TPC-H[11] benchmarks
- Lab implementation of Real life application Vehicle Insurance application is also used for validation for size upto 50GB.
Full Table Scan Results

select * from supplier (full output)

select sum(s_acctbal) from supplier

[Graphs showing ERT (execution time) vs. Table Size (MB) for both queries]
Non Linear Growth in IO access Time

select /*+ index(customer cust_nk) */ sum(c_acctbal) from customer where c_nationkey>0;

select /*+ index(partsupp partsupp_suppkey) */ sum(ps_availqty) from partsupp where ps_suppkey>0;

select /*+ index(customer cust_nk) */ sum(c_acctbal) from customer where c_nationkey>0;
Error % Analysis

Prediction Error % for TPCH Queries

![Graph showing Error % vs. Database Size (GB)]

- FullScan
- FullAgg
- PrimidxScan
- NUScan-fixed
- NUScan-growing
Linear Model vs Proposed Model

```
select /*+ USE_NL(s p) ordered */ sum(s_acctbal)+sum(ps_availqty) from supplier s, partsupp p where p.ps_suppkey = s.s_suppkey;
```
TPCH Benchmarks Query Results

![Graph showing Error (%) for TPCH Benchmarks]

- Error (%) on the y-axis
- Query ID on the x-axis
- Two lines representing 4 GB and 128 GB scenarios
VINS Application Query Results

Error % for VINS Queries

Query ID

Error (%)

40 GB

10 GB
MORE ON QUERY PERFORMANCE
Query Tuning using its Plans

• PICASSO: Query Plan viewer
  – Jayant Haritsa IIS bangalore

  - Different filter values in where clause leads to different selectivity and hence query execution plans
Query Performance Prediction of New Query

- Performance Prediction for a new query/schema on a given system in isolation
  - Statistical model approach by Ganpathi, Surajit Chowdhary, Mert Akdere

**Training**

Data Collection: Query Execution → Data Extraction: Collecting Query Features and Execution Times → Model Building via Feature Selection, Cross-Validation → Prediction Models and Accuracy Estimates

**Testing**

Query Planning: Execution Plan Generation → Feature Extraction: Compute Query Features → Model-based Prediction: Estimate Query Performance

- Operator model: Does not capture operator propagation and concurrent use of CPU and disk
- Plan model: No query with unforeseen plan
Concurrent Query Performance Prediction

• Prediction in concurrent workload for RDBMS
  – Interaction between queries is modeled – Mumtaz Ahmed
  – Bayseian approach – Muhamad at waterloo
  – Cost and IO interaction Model, pipelineing – wisconsin

  – Analytical approach by Duggan, they proposed model which is transparent to hardware system.
    • Interaction between queries
    • Worst case execution time – available resources reduced by degree of concurrency
    • Best execution time: Isolation execution
Advances in Query Performance Prediction

• Prediction for MR applications in Hadoop clusters

• Prediction for No SQL data and on Cloud
Future Direction in Query Performance Prediction

Big Data Real Time Analytics
• Data streaming
• Parallel Database Systems.

Query Performance Prediction in Cloud
• VM wares
• Multi tenant databases

Big Data in HPC
• Different MR framework
• MR on GPUs
• MR on different file systems

Big Data Architectures
• Hive/Hadoop
• MongoDB
• NoSQL queries
• Hbase/BigTable
Thanks

Questions?

rekha.singhal@tcs.com