A Measurement based Model to Predict Query IO Access Time with Data Growth

Rekha Singhal/rekha.singhal@tcs.com

October 2013

Tata Consultancy Services
Contents

Problem Statement
Approach to Problem Solving
Query Taxonomy
Algorithms
Validation
Conclusions
Contents

Problem Statement

Approach to Problem Solving

Query Taxonomy

Algorithms

Validation

Conclusions
Problem Statement

How to **predict** query Input-Output access time and hence query **elapsed response time** at application **development stage** for **production** environment such that the methodology for prediction shall be “**transparent**” to the underlying DB system?

**Transparency with respect to:**
- Hardware system
- Database Server
- Storage Subsystem
Query Elapsed Response Time (ERT)

ERT = t2 - t1

SQL Query at time=t1

Output at time=t2

ERT = IO Time + CPU Time

IO Time = Disk Time + Network Time

PRODUCTION SYSTEM
Production System
Data Size=X

Production System
Data Size=2*X

Production System
Data Size=10*X

After 1 year
Centralization
After 3 years
Centralization
Motivation

- SLA Compliance
- Data Loading Time for large data
- Unavailability of Resources
- Query optimization
Related Work

- **Statistical Modeling**
  - Build a model based on data collected from past queries and predict for a new query based on near neighbourhood.

- **Analytic modeling**
  - Use queuing and software process engineering (SPE) model to derive an equation for ERT and use that for prediction.

- **Query progress prediction**
  - Require changes in optimizer to incorporate query progress prediction.

- **Closely related**
  - CODD, Jayant Haritsa
Contents

Problem Statement

Approach to Problem Solving

Query Taxonomy

Algorithms

Validation

Conclusions
Overall Broad Approach

• Get Query Execution Plan
• Predict ERT for each step
• Add up ERT of all the steps
• Add up time to send output data to the client.
**Query Execution Plan**

```sql
select /*+ USE_NL(s p) */ sum(s.s_acctbal) from supplier s, partsupp p where s.s_suppkey = p.ps_suppkey;
```

**PLAN_TABLE_OUTPUT**

|   0 | SELECT STATEMENT | 
| 1 | SORT AGGREGATE |
| 2 | NESTED LOOPS |
| 3 | TABLE ACCESS FULL SUPPLIER |
| 4 | INDEX RANGE SCAN PARTSUPP_SK |
Architecture and Assumptions

- Two tier architecture
- Test environment = Production system.
- Data features
  - Uniformly distributed
  - Uniformly grows
  - Not organized/deleted
- Validated for
  - ANSI compliant SQL,
  - Nonpartitioned tables
  - Row storage databases.
  - No Parallelism
Measurement Based Approach

Application DB Schema & Production System Stats

SQL Query
Development DB System
Measurements
Prediction Tool

Cost Estimation Utility
Emulated Production DB System

Query Elapsed Response Time
Measurements

• Database level measurements
  • Trace 10046 for understanding time spent by an SQL for physical reads, cpu time and waiting time.
  • Trace 10053 to understand the query execution plan. There are other traces available to understand the join operators.

• OS traces
  • strace to understand the pattern in accessing the physical blocks.
  • profiler tools, atop and sar, to understand the system utilization such as memory, disk and cpu during the execution of query.

• Disk traces
  • blktrace to understand the actual blocks accessed from the disk.
  • iostat to measure number of disk requests..
Large Size Database Emulation

- Linear extrapolation of database statistics
- Table Statistics
  - Number of Rows, Number of Blocks, Average Row Length
- Column Statistics
  - Density, Maximum value, Minimum value, Distinct, Histogram, Column Length
- Index Stats
  - Blevel (Height of tree) Clustering Factor (access pattern of rows from the index perspective), Rows, Density
Contents

Problem Statement

Approach to Problem Solving

Query Taxonomy

Algorithms

Validation

Conclusions
Query Taxonomy

- Full Table Scan
- Fast Index Scan
- Primary Index Scan
- Non Primary Index Scan
Contents

Problem Statement
Approach to Problem Solving
Query Taxonomy
Algorithms
Validation
Conclusions
Algorithms (Basic Steps)

1. Benchmark Testing Database
   a) Execute the given query with/without database traces such as tkprof
   b) Execute the calibrated queries.
   c) Run micro benchmark tools (strace) for network time details.

2. Production Database
   a) Disk Time: Generate Data Access patterns \( <\text{size of data to read, offset}> \)
   b) Network Time: Calculate the number of data packets for each system read call and Intersperse the system read calls with total delays due to network.
   c) Play the above generated data access pattern.
   d) Calculate the total time taken which corresponds to the IO access time of the query.
   e) For very large size database, the IO time is calculated by linear extrapolation from the measurements on the small size database (this is referred as inflexion size)

FTS, FIS, PIS, NUIS differ in
- Generation of data access patterns
- Inflexion size for conducting measurements.
Full Table Scan (FTS)

- Data Access Pattern file
  - system read of size database block size (8K)
  - maximum 4 pairs of system read calls of sizes 64K and 56K and 1MB until \( S < 1\text{MB} \).
  - last system read call of remaining size.
- Calculate number of system read calls of large size (approx. 1 MB or more)- \( LR \). The inflexion size is the one with high LR such as \( LR > 95\% \).
Index Scan

- **Input**: Query (with key < val, key < val and key = val), Table Schema, Size of the table or (number of rows & average size of row)
- **Steps**:
  - Measurements on small database.
  - Emulate the large size database
    - Get database block size (Bsize), minimum value of the indexed column (min), maximum value of the column (max), number of data blocks per key (Dsize), number of leaf nodes per key (Lsize) and height of the tree (Hsize).
  - Calculate the number of matched key values (MKV) as
    - 1 for key=val
    - (val – min) for key < val
    - (max – val) for key > val
  - Total qualified leaf nodes, $QL = Lsize \times MKV$
  - The inflexion size is one where the cpu computation is significant which can be observed from CPU cost of the query.
Fast Index Scan (AFIS)

- Data access pattern with all the offsets are of $Bsize$.
  - $Hsize+1$ system reads are generated each starting from a random address. This is followed by $QL-1$ sequential accesses.
Primary Index Scan (PIS)

• Data access pattern with offsets are of $B_{size}$.
  
  • $H_{size}+1$ system reads starting from a random address.
  
  • $QL-1$ sequential accesses interspersed with $D_{size}$ sequential accesses between two leaf nodes access. Kindly note that starting block addresses for leaf and data blocks are different, so their addresses are random to each other.
Non-Unique Index Scan (NUIS)

• Data access pattern with the offsets are of Bsize.
  • Hsize+1 system reads are generated each starting from a random address.
  • QL-1 sequential leaf node accesses interspersed with Dptr accesses between two leaf node accesses.
• Repeat this step for MKV iterations for table size > database cache, however, keeping the same set of block addresses as generated in the first iteration.
Contents

Problem Statement
Approach to Problem Solving
Query Taxonomy
Algorithms
Validation
Conclusions
Experimental Setup

Intel quad core server with 4GB RAM.

1 TB SAN

DB server- Oracle 11g

Query client as ‘sqlplus’ on the same machine.

The database schema and data are generated using an open source dbgen utility based on TPC-H[11] benchmarks
Workload Details

**Full Table Scan Query 1:** select * from supplier (full output)

**Full Table Scan Query 2:** select sum(s_acctbal) from supplier (aggregated output)

**Fast Index Scan Query 3:** select /*+ index(supplier pk_supplier) */ s_suppkey from supplier

**Primary Index Scan Query 4:** select /*+ index(supplier pk_supplier) */ sum(s_acctbal) from supplier where s_suppkey >10;

**Non-Unique Index Scan Query 5:** select /*+ index(supplier supp_nk) */ sum(s_acctbal) from supplier where s_nationkey=3;
Error % Analysis

![Prediction Error % for Queries](image)
Analysis

• For smaller size - error rate seems to be high
  • the time reported for these queries on these sizes is in \textit{milliseconds}, a small variation in milliseconds leads to high error rate
  • The small variation is attributed to \textit{measurement anomaly} which could be reduced by increasing the number of measurements and averaging.
• For higher sizes, the error rate is bounded by 10\% for Full table scan, Fast Index Scan and Primary Index Scan
  • The data access is \textbf{never repeated} in these types of queries.
  • We generate data access pattern and play it; this methodology incorporate the behaviour of OS and disk subsystem caches.
• For Non-Unique Index Scan query (Q5), the error rate is increasing with increase in data size.
  • The \textbf{data accesses are repeated}.
  • OS and DB cache has role to play which may influence each other as well. This influence and DB cache affect is not captured in this model.
  • We have devised a model to predict the behaviour of OS and DB cache with growth in database and use that to correctly predict the IO time
Conclusions

- A measurement based model to predict Query ERT with data growth.
- A query taxonomy based on the mode of table access.
- AFTS, AFIS, APIS and ANUIS, to emulate IO access pattern of full table scan, fast index scan, primary index scan and non-unique index scan queries respectively.
- Model is validated on Oracle 11g using synthetic queries on TPC-H schema.
- Prediction is within 10% average error rate for Full table scan, Fast Index scan and Primary Index Scan queries.
- Non Unique Index Scan queries has higher error rate due to repeated accesses; OS and DB cache impact query IO time.
Thank You

rekha.singhal@tcs.com
<table>
<thead>
<tr>
<th>Aggregate Output</th>
<th>Full Table Scan</th>
<th>Primary Index Scan</th>
<th>Non Unique Index Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>O/p from Index Block only (Fast Index Scan)</code></td>
<td><code>O/p from Table Block only</code></td>
<td><code>O/p from Index blocks only (Fast Index Scan)</code></td>
<td><code>O/p from Table Blocks only</code></td>
</tr>
<tr>
<td><code>Select sum(s_acctbal) from supplier;</code></td>
<td><code>Select /*+index (pk_supp) */ sum(s_suppkey) from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (pk_supp) */ sum(s_acctbal) from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (sup_nk) */ sum(s_acctbal) from supplier where s_nationkey&gt;3</code></td>
</tr>
<tr>
<td><code>Select * from supplier</code></td>
<td><code>Select /*+index (pk_supp) */ s_suppkey from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (pk_supp) */ s_suppkey from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (sup_nk) */ * from supplier where s_nationkey&gt;3</code></td>
</tr>
<tr>
<td><code>Select * from supplier</code></td>
<td><code>Select /*+index (pk_supp) */ s_suppkey from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (pk_supp) */ s_suppkey from supplier where s_suppkey&gt;3</code></td>
<td><code>Select /*+index (sup_nk) */ * from supplier where s_nationkey&gt;3</code></td>
</tr>
</tbody>
</table>
Full Table Scan Results
Fast Index Scan Results

Fast Full Scan Query 3

ERT (Seconds)

Table Size (MB)

ERT
ERT Predicted
Primary Index Scan Results

Primary Index Scan Query 4

ERT(Seconds)

Table Size (MB)

ERT
ERT Predicted
Non-Unique Index Scan Results

Non-Unique Index Scan Query 5

ERT (Seconds)

Table Size (MB)

ERT
ERT Predicted