Performance Testing and Monitoring SQL Queries for Rebuild or Reorganize Operations

Mr. Sudhakar Panigrahy¹, Mr. Pragnyaban Mishra², Mr. Murali Krishna Senapaty³
Dept of CSE, Gandhi Institute of Engineering & Technology, Gunupur, Rayagada, India¹,²,³

Abstract: A database system must be able to respond to requests for information from the user—i.e. process queries. Obtaining the desired information from a database system in a predictable and reliable fashion is Query Processing. To extract these results in a timely manner is possible with the technique of Query Optimization. For effective query processing the data must not be much fragmented. In this paper we focussed on the poor performance of query when the date fragmentation percentage is more and we have analysed how the fragmentation can be reduced using rebuild and re-organize techniques. In the last section, we have taken a sample data for query processing before rebuild and after that to have a comparative analysis.

Keywords: Indexing, Fragmentation, rebuild, re-organize.

1. INTRODUCTION

1.1 Index:
Index is used to speed up the query retrieval process. Index uses lookup table the search engine of the database uses to improve the performance of select statement of SQL query. As index stores data in sorted order hence it is also useful in order by clause. However index reduces the DML (Data Manipulation Language) operations (Insert, Update and Delete operations). Hence index is avoided in following situations:
- If number of rows in table is less (less than 1000).
- If the column not used in where clause of SQL statement.
- The number of distinct values of index column is less (Like the gender column contains M/F).
- The column contains more number of NULL values.
- If the column that is frequently updated.

Use of indexes is one of the best ways to improve performance of database application. The index uses the data structure B-Tree for indexes. The searching operation is first as B Tree uses logarithmic time for look up, insertion and deletion. The correct use of index requires careful analysis, benchmarking and testing. SQL query tuning is used to identify the useful of index[2].

The performance of the query is very important when the database size if large so there exist so many techniques for tuning the query by which the performance of the query can be improved[1].

Now it is important to know, if we are looking for a specific value how it retrieves other values. Index also stores pointer to the corresponding rows in the table. Hence it retrieves data as per specific column value used where clause.

1.2 FRAGMENTATION:
Storing of data non-contiguously on a disk is called as fragmentation.

We can classify fragmentation into two types:

Types of Fragmentation
- **Internal Fragmentation:** When records are stored non-consecutively inside the page, then it is called internal fragmentation. Internal fragmentation will occur if there is unused space between records in a page. This fragmentation occurs due to the process of data modifications (INSERT, UPDATE, and DELETE statements) that are made against the table many times and also to the indexes defined on the table. When the modifications are not equally distributed among the rows of the table and indexes then the fullness of each page can vary. Due to these unused spaces it causes poor cache utilization and more I/O, which finally leads to poor query performance.
- **External Fragmentation:** When the extents of a table are not physically stored contiguously on disk, switching from one extent to another causes higher disk rotations, and this is called Extent Fragmentation.
- **Logical Fragmentation:** Index pages also maintain a logical order of pages inside the extent. Every index page is linked with previous and next page in the logical order of column data. However, because of Page Split, the pages turn into out-of-order pages. An out-of-order page is a page for which the next physical page allocated to the index is not the page pointed to by the next-page pointer in the current leaf page. This is called Logical Fragmentation.

Advantages of Fragmentation
- **Usage:** Generally, applications work with views more than entire relations. That is why, for data distribution, it seems appropriate to work with subsets of relation as the unit of distribution.
- **Efficiency:** Data is stored close to where it is most frequently used. The data that is not needed by local applications is not needed to store locally.
Parallelism: With the fragments as the unit of distribution, a transaction can be divided into several sub queries that operate on fragments. This allows concurrency, thereby allowing transactions that can do so safely to execute in parallel.

Disadvantages of the Fragmentation:
- **Performance**: The performance of global application that requires data from several fragments located at different sites may be slower.
- **Integrity**: Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.

2. REORGANIZE AND REBUILD INDEXES

The Database Engine maintains indexes whenever insert, update, or delete operations are made to the underlying data. Over time these modifications cause the information in the index to become scattered in the database. Fragmentation exists when indexes have pages in which the logical ordering, based on the key value, does not match the physical ordering inside the data file. More fragmented indexes can degrade query performance and cause your application to respond slowly[5].

We can remedy index fragmentation by reorganizing or rebuilding an index. For partitioned indexes built on a partition scheme, you can use either of these methods on a complete index or a single partition of an index.

2.1 Rebuild: Rebuilding an index drops and re-creates the index. This removes fragmentation, reclaim disk space by compacting the pages based on the specified or existing fill factor setting, and then reorders the index rows in contiguous pages. When ALL is specified, all indexes on the table are dropped and rebuilt in a single transaction.

2.2 Reorganize: Reorganizing an index uses minimal system resources. It defragments the leaf level of clustered and non-clustered indexes on tables and views by physically reordering the leaf-level pages to match the logical, left to right order of the leaf nodes. Reorganizing also minimizes the index pages. The Compaction is also based on the existing fill factor value.

3. EXPERIENTIAL SETUP

3.1 DETECTING FRAGMENTATION

The first step in deciding which defragmentation method to use is to analyse the index to determine the degree of fragmentation. By using the system function sys.dm_db_index_physical_stats, we can detect fragmentation in a specific index, all indexes on a table or indexed view, all indexes in a database, or all indexes in all databases [4].

Query-1: The following query is used to detect the fragmentation exist in all indices of specific database

```sql
SELECT OBJECT_NAME(OBJECT_ID), index_id, index_type_desc, index_level, avg_fragmentation_in_percent, avg_page_space_used_in_percent, page_count FROM sys.dm_db_index_physical_stats (DB_ID(N'MySms'), NULL, NULL, NULL, 'SAMPLED') ORDER BY avg_fragmentation_in_percent DESC
```

Output:

Fragmentation Before rebuild:

(Figure-3.1)
From the above output it is observed that the fragmentation exist in many indices of the database.

TABLE 3.1

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg_fragmentation_in_percent</td>
<td>The percent of logical fragmentation (out-of-order pages in the index).</td>
</tr>
<tr>
<td>fragment_count</td>
<td>The number of fragments (physically consecutive leaf pages) in the index.</td>
</tr>
<tr>
<td>avg_fragment_size_in_pages</td>
<td>Average number of pages in one fragment in an index.</td>
</tr>
</tbody>
</table>

3.2 Constructing complex query:
For measuring the performance of the query we used following tables:
1. smsStudent(no. Of records 15000).
2. smsAttendance_Master(no. Of records 118023).
3. smsAttendance_Details(no. Of records 6504012).
5. Proctor.Student_Details(no of records 65230).
6. smsEmployee(no.of records 792)

From the above tables is has been seen that the larger tables are: smsAttendance_Master, smsAttendance_Details

Query-2: The query is used to find the attendance percentage of the students in specified date:

```sql
select smsStudent vcRoll_No AS 'Roll No', smsStudent vcStudent_Name as Name, smsStudent vcStudent_Mobile_No as 'Student Mobile No',
vcFathers_No as 'Fathers Mobile No', smsEmployee vcEmp_Name as 'Proctor',
smsAttendance_Master vcAttendance_No as 'Total Classes',
SUM(intpresent) as Presnt_count(smsAttendance_Master vcAttendance_No) - SUM(intpresent) as Absent,
cast((cast(SUM(intPresent) * 100 as decimal(18,2))/COUNT(smsAttendance_Master vcAttendance_No)) as decimal(10,2)) as Atte_Per
from smsAttendance_Master, smsAttendance_Details, Proctor.Student_Master, Proctor.Student_Details, smsEmployee
where smsAttendance_Master vcAttendance_No = smsAttendance_Details vcAttendance_No and
smsAttendance_Details vcRoll_No = smsStudent vcRoll_No and
Proctor.Student_Master vcProctor.Student_Code = Proctor.Student_Details vcProctor.Student_Code and
Proctor.Student_Details vcRoll_No = smsStudent vcRoll_No and
Proctor.Student_Master vcProctor.Employee_Code = smsEmployee vcEmp_code and
smsStudent vcBranch_Id = 'CSE' AND smsStudent intSemester = 6 and vcSection = 'A' and
smsStudent vcStatus = 'ACTIVE' AND
smsAttendance_Master vcStatus = 'ACTIVE' and smsAttendance_Master dtDateAttendance_Date = '12/13/2016 12:00:00 AM' and
smsAttendance_Master dtDateAttendance_Date <= '12/13/2016 12:00:00 AM'
group by smsStudent vcRoll_No, smsStudent vcStudent_Name, smsStudent vcStudent_Mobile_No, vcFathers_No, smsEmployee vcEmp_Name
having count(smsAttendance_Master vcAttendance_No) - SUM(intpresent) >= 1
order by smsStudent vcRoll_No
```

Query-3: The query is used to rebuild all indices of the database.

```sql
use MySms:// MySms is name of the database
SET NOCOUNT ON;
DECLARE @objectid int;
DECLARE @indexid int;
DECLARE @partitioncount bigint;
DECLARE @schemaname nvarchar(130);
DECLARE @objectname nvarchar(130);
DECLARE @indexname nvarchar(130);
DECLARE @partitionnum bigint;
DECLARE @partitions bigint;
DECLARE @frag float;
DECLARE @command nvarchar(4000);
```
-- Conditionally select tables and indexes from the sys.dm_db_index_physical_stats function
-- and convert object and index IDs to names.

```
SELECT
    object_id AS objectid,
    index_id AS indexid,
    partition_number AS partitionnum,
    avg_fragmentation_in_percent AS frag
INTO #work_to_do
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL, NULL, 'LIMITED')
WHERE avg_fragmentation_in_percent > 10.0 AND index_id > 0;

-- Declare the cursor for the list of partitions to be processed.
DECLARE partitions CURSOR FOR
    SELECT *
    FROM #work_to_do;

-- Open the cursor.
OPEN partitions;

-- Loop through the partitions.
WHILE (1=1)
BEGIN

    FETCH NEXT FROM partitions
    INTO @objectid, @indexid, @partitionnum, @frag;

    IF @@FETCH_STATUS < 0 BREAK;

    SELECT @objectname = QUOTENAME(o.name),
           @schemaname = QUOTENAME(s.name)
    FROM sys.objects AS o
    JOIN sys.schemas as s ON s.schema_id = o.schema_id
    WHERE o.object_id = @objectid;

    SELECT @indexname = QUOTENAME(name)
    FROM sys.indexes
    WHERE object_id = @objectid AND index_id = @indexid;

    SELECT @partitioncount = count(*)
    FROM sys.partitions
    WHERE object_id = @objectid AND index_id = @indexid;

    -- 30 is an arbitrary decision point at which to switch between reorganizing and rebuilding.
    IF @frag < 30.0
        SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'. ' + @objectname + N' REORGANIZE';
    ELSE
        SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'. ' + @objectname + N' REBUILD';

    IF @partitioncount > 1
        SET @command = @command + N' PARTITION=' + CAST(@partitionnum AS nvarchar(10));

    EXEC (@command);
    PRINT N'Executed: ' + @command;

END;

-- Close and deallocate the cursor.
CLOSE partitions;
DEALLOCATE partitions;

-- Drop the temporary table.
DROP TABLE #work_to_do;

```

Output:
```
Executed: ALTER INDEX [PK__smsEmplo__99AAB30420C1E124] ON [dbo].[smsEmployee] REBUILD
Executed: ALTER INDEX [IX_smsStudent_Branch] ON [dbo].[smsStudent] REORGANIZE
Executed: ALTER INDEX [IX_smsStudent_Semester] ON [dbo].[smsStudent] REORGANIZE
Executed: ALTER INDEX [IX_smsStudent_Section] ON [dbo].[smsStudent] REBUILD
(Note: All rows of the output are not shown here)
```

From the study [3] it is noticed that rebuild is not useful when the fragmentation is less than 30%. So the query is designed to reorganize or rebuild as per the fragmentation percentage as below:
Fragmentation after rebuild all indices (Execution of the Query-3)

### TABLE-3.2

<table>
<thead>
<tr>
<th>avg_fragmentation_in_percent value</th>
<th>Corrective statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 5% and &lt; = 30%</td>
<td>ALTER INDEX REORGANIZE</td>
</tr>
<tr>
<td>&gt; 30%</td>
<td>ALTER INDEX REBUILD WITH (ONLINE = ON)*</td>
</tr>
</tbody>
</table>

### TABLE-3.3

<table>
<thead>
<tr>
<th>(No column name)</th>
<th>index_id</th>
<th>index_type_desc</th>
<th>index_level</th>
<th>avg_fragmentation_in_percent</th>
<th>avg_page_space_used_in_percent</th>
<th>page_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>smsEmployee</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>66.66666667</td>
<td>94.0490981</td>
<td>3</td>
</tr>
<tr>
<td>Proctor_Student_Details</td>
<td>7</td>
<td>NONCLUSTERED INDEX</td>
<td>0</td>
<td>50</td>
<td>93.74845565</td>
<td>12</td>
</tr>
<tr>
<td>smsStudent</td>
<td>36</td>
<td>NONCLUSTERED INDEX</td>
<td>0</td>
<td>45.45454545</td>
<td>97.1258216</td>
<td>11</td>
</tr>
<tr>
<td>smsStudent</td>
<td>35</td>
<td>NONCLUSTERED INDEX</td>
<td>0</td>
<td>35.71428571</td>
<td>99.59494687</td>
<td>14</td>
</tr>
<tr>
<td>smsStudent</td>
<td>37</td>
<td>NONCLUSTERED INDEX</td>
<td>0</td>
<td>27.27272727</td>
<td>91.87951569</td>
<td>11</td>
</tr>
<tr>
<td>smsSend_Sms_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>4.301075269</td>
<td>98.38045466</td>
<td>93</td>
</tr>
<tr>
<td>smsAttendance_Master</td>
<td>12</td>
<td>NONCLUSTERED INDEX</td>
<td>0</td>
<td>0.625</td>
<td>99.41423277</td>
<td>160</td>
</tr>
<tr>
<td>smsAttendance_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0.319488818</td>
<td>99.14720781</td>
<td>626</td>
</tr>
<tr>
<td>smsDPR_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0.091911765</td>
<td>98.34676303</td>
<td>1088</td>
</tr>
<tr>
<td>smsProctor_Cycle_Plan_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>94.84609587</td>
<td>76</td>
</tr>
<tr>
<td>smsExamSubject_Registration_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>12.96021745</td>
<td>1</td>
</tr>
<tr>
<td>smsHolidays</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>1.470224858</td>
<td>1</td>
</tr>
<tr>
<td>smsExamSubject_Registration_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>62.54427971</td>
<td>3</td>
</tr>
<tr>
<td>smsStudentRegistration</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>13.24437855</td>
<td>1</td>
</tr>
<tr>
<td>smsDPRComment_From_Authorities</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>5.564615765</td>
<td>5</td>
</tr>
<tr>
<td>smsSetLeave</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>86.75150729</td>
<td>3</td>
</tr>
<tr>
<td>Superviser_Faculty_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>14.05979738</td>
<td>1</td>
</tr>
<tr>
<td>smsSubject_Registration_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>90.22424018</td>
<td>4</td>
</tr>
<tr>
<td>smsExamination_Mark_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>99.15221151</td>
<td>395</td>
</tr>
<tr>
<td>smsSubject_Registration_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>75.20610329</td>
<td>11</td>
</tr>
<tr>
<td>smsExam_Semester_Master</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>89.28836175</td>
<td>8</td>
</tr>
<tr>
<td>smsAttendance_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>98.7805535</td>
<td>10100</td>
</tr>
<tr>
<td>smsBranch</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>0</td>
<td>0</td>
<td>9.340252039</td>
<td>1</td>
</tr>
<tr>
<td>smsExam_Semester_Details</td>
<td>0</td>
<td>HEAP</td>
<td>0</td>
<td>0</td>
<td>95.30226093</td>
<td>68</td>
</tr>
</tbody>
</table>
4. PERFORMANCE MEASUREMENT OF REBUILD OR REORGANIZE OPERATION

Execution plan is as below
4.1 Tools identified to measure the performance:
The following tools are used to trace the performance result and necessary actions:

4.1.1: Estimated Execution Plan
4.1.2: Client Statistics
4.1.3: Measurement of execution time using query

4.1.1: Estimated Execution Plan:
It displays the execution plan, resources (execution time and space) of the current query and also it necessary actions require to improve the performance of the query. Following is the output of the execution plan after rebuild or reorganize operations (execution of the query-3)
4.1.2: Client Statistics

The client statistics report displays the performance result in different trials about Query Profile Statistics, Number of INSERT, DELETE and UPDATE statements, Rows affected by INSERT, DELETE, or UPDATE statements, Number of SELECT statements, Rows returned by SELECT statements, Number of transactions, Network Statistics, Number of server roundtrips, TDS packets sent from client, TDS packets received from server, Bytes sent from client, Bytes received from server, Time Statistics, Client processing time, Total execution time, Wait time on server replies.

The output is as below:

<table>
<thead>
<tr>
<th>Client Execution Time</th>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
<th>Trial 4</th>
<th>Average</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Query Profile Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of INSERT, DELETE and UPDATE statements</td>
</tr>
<tr>
<td>Rows affected by INSERT, DELETE, or UPDATE statements</td>
</tr>
<tr>
<td>Number of SELECT statements</td>
</tr>
<tr>
<td>Rows returned by SELECT statements</td>
</tr>
<tr>
<td>Number of transactions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of server roundtrips</td>
</tr>
<tr>
<td>TDS packets sent from client</td>
</tr>
<tr>
<td>TDS packets received from server</td>
</tr>
<tr>
<td>Bytes sent from client</td>
</tr>
<tr>
<td>Bytes received from server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client processing time</td>
</tr>
<tr>
<td>Total execution time</td>
</tr>
<tr>
<td>Wait time on server replies</td>
</tr>
</tbody>
</table>

4.1.3: Measurement of execution time using query

DECLARE @EndTime datetime
DECLARE @StartTime datetime
SELECT @StartTime=GETDATE()/* The Query 2 need paste here to measure the execution time*/
SELECT @EndTime=GETDATE()

SELECT DATEDIFF(ms,@StartTime,@EndTime)AS [Duration in millisecs]

--This will return execution time of your query
SELECT DATEDIFF(ms,@StartTime,@EndTime)AS [Duration in millisecs]

Output is as below: 60 rows are retrieved.
RESULT ANALYSIS

5.1: Execution Time:
Following table represents the execution time in milli seconds different executions. The execution time is measured before and after the rebuild operations:

<table>
<thead>
<tr>
<th>Execution Time before Rebuild (in milli sec)</th>
<th>execution1</th>
<th>execution2</th>
<th>execution3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time after Rebuild (in milli sec)</td>
<td>696</td>
<td>699</td>
<td>690</td>
</tr>
</tbody>
</table>

5.2: Fragmentation percentage comparison:
After rebuild and reorganize the fragmentation level also reduced refer the table-4.3 and figure-4.1
It is observed that the fragmentation of different tables whose page size is more, the fragmentation percentage is reduced about to 0% (table name : smsAttendance_master, the fragmentation of the non cluster index for date field is reduced to 0.4632% )
From the tables 4.3 it is observed that after rebuild or reorganize of indices the fragmentation still exist in the tables: smsStudent, smsEmployee. The reason for the fragmentation is the page size of the specific tables is less. Hence there is no distinct improvement of fragmentation of the specified tables.[6]

6. CONCLUSION

One of the most important functional requirements of a database system is its ability to process queries in a timely manner. This is particularly true for very large database applications. There are different types of indices are there like cluster index and non-cluster index. The performance of the query decreases after couple of DML statements like insert, update, delete which increases the fragmentation. So in experiment it is observed that the performance of the query decreases drastically due to the fragmentation. Hence rebuild and reorganize have to be performed periodically after identifying the fragmentations. The rebuild is necessary when the fragmentation is more than 30% and reorganize is necessary when the fragmentation is <30%. The rebuild operations takes the database to offline mode where the reorganize operation reduces the fragmentation in online mode operations. Thus, a great deal of research and resources is spent on creating smarter, highly efficient query optimization engines. The rebuild is simplest and frequent used method for optimized query processing.

REFERENCES

[3] “Introduction to Query Processing and Optimization” Michael L. Rupley, Jr. Indiana University at South Bendmrupleyj@iusb.edu