Overview

- Databases are growing rapidly. Both in terms of data and in terms of use.

- DB2 has many options for handling big systems through “Divide and conquer” approach:
  - Data partitions features (DPF)
  - Table partitioning
  - MDC

- New approaches introduced in V10 and above

- This session aims to highlight the main options of handling big systems and discuss the best case scenario for each one
Agenda

• Database partitioning Feature (Refresher)
• Table Partitioning (Refresher)
• MDC (Refresher)
• Data partition best practice, monitoring and tuning
• Table partitions best practice and enhancements
• V10 and v10.5 improvements for storage and partitioning
Database Partition Feature (DPF)
**DPF - Database Partitions (Hash)**

- **Share-Nothing-Architecture**
- **Allow partition a database within a single server or across a cluster of servers based on a hash function**

### Partitioning Map

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>...</th>
<th>...</th>
<th>4094</th>
<th>4095</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitions</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>1000 Partitions</td>
<td>ID</td>
</tr>
</tbody>
</table>
DPF – Terms and Definitions

• Catalog Database Partition
  • Contains SYSCATSPACE
  • Catalog look-up required for compiling queries

• Coordinator Partition
  • DB2 agent to which application connects/communicates

• Partition Groups
  • Group of one or more database partitions
  • Tablespaces assigned to partition groups
  • Default partition groups include: IBMCATGROUP, IBMTEMPGROUP and IBMDEFAULTGROUP
  • Default groups exist even for non-partitioned databases
DPF - Benefits

• Scalability
  • As workload / data volume increases, add more resources proportionally for preserving the same response time
  • Grow vertically and/or horizontally

• Response Time
DPF - Benefits

• Scalability
  • As workload / data volume increases, add more resources proportionally for preserving the same response time
  • Grow vertically and/ or horizontally

• Response Time
DPF - Benefits

• Scalability
  • As workload / data volume increases, add more resources proportionally for preserving the same response time
  • Grow vertically and/ or horizontally

• Response Time
DPF - Benefits

• Scalability

• Response Time
  • Provide effective way to reduce the data set need to scan. Result in less IO needed.
  • Better utilize resources in Parallelism
  • Reduce maintenance windows
  • OLAP/DW and OLTP
Table Partitions (Range)

- Storing a table in more than one physical object, across one or more table spaces
- Each table space contains a range of the data that can be found very efficiently
Table Partitioning - Benefits

- Increase table capacity limit

- Increase large table manageability
  - Roll-in/Roll-out - Attach and Detach a partition instantaneously
  - Can do maintenance work at a partition level - runstats, reorg, backup etc. Assume that each partition has its own tablespace

- Improve SQL performance through partition elimination

- Minimize lock escalation because lock escalation happens at the partition level

- Extremely flexible - ability to auto generate partitions, exclusive, inclusive, auto increment and more
Multi Dimension Cluster (MDC)
MDC – Multiple Dimension Cluster

- Data is organized on disk in storage blocks (of extent size) according to dimension (clustering) values
- Rows are placed on disk with other rows having the same key values
- Each combination has its own “cell” of data
MDC - Benefits

• Physically cluster data in multiple dimensions

• Improve performance of queries and prefetching on n separate indexes

• Clustering automatically and dynamically maintained over time

• Reorg not necessary for re-clustering

• Indexes on dimensions are BLOCK based which are much smaller
Hybrid partitioning

Database Partitioning
- 1000 Partitions

Jan  Feb  Mar  Apr

Fact_ID

Start_Time

MDC

Database, User
Database Partition - Best Practice, Monitoring and Tuning
DPF – Best Practice

• Even Distribution - same number of rows for a partitioned table on each database partition

• Co-location - A collocated query has all the data required to complete the query on the same database partition
  • Less communication between partitions
  • Less overhead
DPF – Even Distribution

CREATE TABLE sales
(sale_id int,
store_id char(1))
Distribute by(sale_id);

CREATE TABLE sales
(sale_id int,
store_id char(1))
Distribute by(store_id);

Good – all data partitions hold the same amount of data

Bad – some partitions are fully loaded while others are empty
DPF – co-location in execution plans

Collocated joins

Broadcast and Directed joins
DPF – Other Best Practice

• Partitions Number
• Partition Groups
• Tables
• General
DPF Best Practice – Partitions Number

• Never have more partitions than CPUs

• Consider whether to dedicate partitions for catalog and coordinator

• Fewer partitions (more CPUs per partitions, more data per partition)
  • the ability to handle more concurrent queries
  • longer single-query run times (since there's more data per partition)
  • Better response times due to less communication between partitions

• More partitions (less data per partition)
  • faster single-query performance
  • faster utility performance (BACKUP, RUNSTATS, REORG, etc.)
  • higher OLTP throughput, but possible worse response times due to inter-partitions communications
DPF Best Practice – Partition Groups

- Create multiple partition groups based on the size of the tables and their use
- Create partition groups that span one, all or part of the partitions
Tables

• Choose partition key wisely
  • Highest cardinality and low data skew
  • Few columns as possible
  • Column are not usually updated
  • Compact data type : integer -> characters  -> decimals

• Partition related tables on common join key
• Group related tables based on joins / access and place them in the same partition group
• Place smaller tables into a single database partition group.
• Replicate smaller tables that are frequently used in joins and infrequently updated across all database partitions
DPF Best Practice - General

• Use a similar configuration for all partitions and systems - hardware, software, and database layout.

• Consider logical database partition servers
  • Can respond well to workload / data skews
  • Can use memory-based inter-partition communication

• For physical partitions - have a high-speed communications interface between the partitions
Analyze data distribution

• Physical storage at each partition

`db2_all`
“`db2 connect to <dbname> && db2 list tablespace containers for <NN> show detail`”

• Row count per partition – table level

Select
  `DBPARTITIONNUM(col) DBP, COUNT(1)`
From `<table>`
Group by `DBPARTITIONNUM(col)`

• Row count per partition map entry – table level

Select
  `HASHEDVALUE(col) HV, COUNT(1)`
From `<table>`
Group by `HASHEDVALUE(col)`
Analyze data distribution

• Check IO is distributed evenly between partitions

Select
DBPARTITIONNUM,
(POOL_DATA_P_READS + POOL_XDA_P_READS) as data_physical_reads,
(POOL_INDEX_P_READS) as index_physical_reads
From table(SNAP_GET_DB(current server,0))
Union all
Select
DBPARTITIONNUM,
(POOL_DATA_P_READS + POOL_XDA_P_READS) as data_physical_reads,
(POOL_INDEX_P_READS) as index_physical_reads
From table(SNAP_GET_DB(current server,1))
Union all
...

...
Partition Communication

• Related monitoring functions
  – MON_GET_FCM, ON_GET_FCM_CONNECTION_LIST
  – SNAP_GET_FCM, SNAP_GET_FCM_PART

• Ensure enough memory exist
  • FCM Buffers and Channels
  • Configured automatically by default since DB2 version 9.5

Select
  member,
  (buff_free*100.00000) / buff_total percent_free_buffers,
  (ch_free*100.00000) / ch_total percent_free_channels,
  buff_free_bottom,
  ch_free_bottom
From table(mon_get_fcm(-2))
Partition Communication

• Check inter-partition communication

Select
  member,
  sum(TOTAL_BUFFERS_SENT) buffers_sent,
  sum(TOTAL_BUFFERS_RCVD) buffers_recv
From table(MON_GET_FCM_CONNECTION_LIST(-2))
Group by member

• Monitor over time
Tune Relevant Queries

• Find out which queries wait for communication the most

Select
  member,
  stmt_type_id,
  (TOTAL_ACT_TIME/1000.00) as TOTAL_EXEC_TIME_S,
  TOTAL_ACT_TIME / (NUM_EXECUTIONS + 1) as avg_time,
  TOTAL_CPU_TIME/1000000.00 as total_cpu_time_s,
  (TOTAL_ACT_WAIT_TIME / 1000.00 ) as total_wait_time_s,
  (FCM_RECV_WAIT_TIME + FCM_SEND_WAIT_TIME)*100.00
  / (TOTAL_ACT_WAIT_TIME+ 1) as percent_fcm_wait,
  substr(stmt_text,1,CAST(2048 AS INT)) AS sql_text
From table(MON_GET_PKG_CACHE_STMT ( NULL , NULL, NULL, -2))
Where NUM_EXEC_WITH_METRICS <> 0
Order by 7 desc

• Run explain to get execution plan and analyze the join strategies
DPF – best case scenario

• Queries that scan all or large part of a billion rows table
  • Select count(*) from MyVeryBigTable

• Large table joined with several small tables that are updated infrequently

• Tables that are usually joined together and have the same partitioning key
Table Partitioning
Best practice
Table partitioning – Best Practice

- Assign each partition to an individual table space so that you can perform table space backups

- Keep the numbers of partitions in hundred not thousands to minimize number of system catalog entries

- Use local indexes to speed up roll-in and roll-out of data

- Facilitate a multi-temperature data solution by moving data partitions from one temperature of storage tier to another as data ages
Multi-temperature Storage

- Temperature of the data is a measure of its age, importance and frequency of use
Multi-temperature Storage in DB2 v10.1

• Storage Groups
  • Logical grouping of automatic storage paths
  • Each SG represent a storage tier that has the same performance characteristics of latency and read rate

• Table space is associated with a storage group.

• Association of Table Space to Storage Group can be dynamic

• Actual data movement is done asynchronously
Multi-temperature Storage in DB2 v10.1

- Minutes_TS
- Days_TS
- Weeks_TS
- Months_TS
- Years_TS
- Hot_SG
- Warm_SG
- Cold_SG
- Dormant_SG
- SSD
- FC 15RPM
- FC 10RPM
- SATA
Multi-temperature Storage - Benefits

• Performance
  - Fast device allow shorter response time
  - DB2 optimizer can factor in the I/O properties of the various storage tier into the costing model, resulting in a more accurate access plans

• Increased ROI
  - Ensure the fastest storage is assigned only to the queries requiring short response time
  - Does not require sophisticated storage arrays
Multi-temperature Storage – Best Practice

• Organize data partitions into table spaces based on how granularity you plan to move data from one temperature tier to another
  • Example: if you plan to move the data from hot to warm on a quarterly basis, create a table space for each quarter

• Be sure to time the move of tablespace between storage groups correctly so not all will run at the same time
Multi-temperature Storage – How to

• Create the storage groups

Create STOGROUP sghot on '/db2/hot01' device READ RATE 350
  OVERHEAD 0.75
Create STOGROUP sgwarm on '/db2/warm01', '/db2/warm02', '/db2/warm03'

• Create table space according to the granularity

Create TABLESPACE tbsp2011q2 using STOGROUP sgwarm
Create TABLESPACE tbsp2011q3 using STOGROUP sgwarm
Create TABLESPACE tbsp2011q4 using STOGROUP sgwarm
Create TABLESPACE tbsp2012q1 using STOGROUP sghot
Multi-temperature Storage – How to

• Create the range partition table
  CREATE TABLE sales (orderdate DATE, orderid INT, custid BIGINT)
    PARTITION BY RANGE (orderdate)
      (PART "2011Q2" STARTING ('2011-04-01') ENDING ('2011-06-30')
        in "tbsp2011q2",
        PART "2011Q3" STARTING ('2011-07-01') ENDING ('2011-09-30')
        in "tbsp2011q3",
        PART "2011Q4" STARTING ('2011-10-01') ENDING ('2011-12-31')
        in "tbsp2011q4",
        PART "2012Q1" STARTING ('2012-01-01') ENDING ('2012-03-31')
        in "tbsp2012q1");

• Automate the tier movement
  ALTER TABLESPACE tbsp2012q1 USING STOGROUP sgwarm
V10 improvements
Pure Scale

• A DB2 pureScale environment consists of multiple “instances” called members.

• Queries are automatically routed to different members, based on member workload.

• One storage partition shared by all members

• Provide high availability

• ideal for short transactions where there is little need to parallelize each query
Columnar Tables

- Data is organized on disk by columns, rather than by rows.
- All processing is done in-memory but DB2 does not require all of the table data to be in-memory.
- Compression – save space while maintaining the ability to filter the data without having to decompress.
- Data skipping – Only reads page if it has relevant data.
- Take advantage of new chips design - parallelize on multiple processors.
- No other objects required for good performance (no index, MDC, MQT…).
- Implemented as part of the DB2 kernel. All regular utilities work the same.
Summary

• Data is ever-growing and systems are ever changing

• Finding a model that will fit current and future needs is the challenge

• DB2 provides a lot of options to choose from and they can sometimes be mixed

• Use the one most suit for your application
Thank You

Anat Dror
Anat.Dror@software.dell.com