IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

Advanced Performance Diagnostics for SQL

David Kalmuk
IBM

Session code: D04
Monday November 5th, 16:30

Db2 for Linux, Unix, Windows

Objectives

- Learn how to pinpoint your most expensive SQL statements using the package cache table functions
- Learn how to analyze where a problem query is spending its time using time spent metrics
- Learn how to monitor query sort memory usage and spilling
- Learn how to use the runtime explain capabilities and activity event monitor to capture both the actual cardinalities and the new object metrics for a problem query.
- Take away practical examples you can try out in your own environment.

Agenda

- A quick review of the core DB2 monitoring capabilities
- Identifying high impact SQL statements
- Analyzing queries using Time Spent
- Monitoring query sort memory usage and spilling
- Advanced diagnostics using Runtime Explain and Section Actuals

IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

A Quick Review of Db2's Monitoring Capabilities

DB2 Monitoring Basics

- How do I monitor what DB2 is doing?
  - Real-time in-memory metrics using SQL functions
  - Historical data captured using event monitors
- Also of note
  - Snapshot monitoring
  - db2pd
- For this session we'll focus on the latest generation of monitoring capabilities introduced starting in DB2 9.7

Lightweight Monitoring Functions

- MON_* SQL functions introduced starting in DB2 9.7
- Less impact / more efficient then snapshot functions
  - Direct in-memory access through trusted routines (not fenced wrappers over snapshot apis)
  - Much less latch contention
    - Uses new infrastructure that pushes data up to accumulation points rather than forcing monitor queries to do extensive drilldown
  - Lower CPU consumption
  - Significantly faster response time
  - Less FCM resource usage / internode traffic
- Monitoring data collection carries low overhead – is enabled by default on new databases

Monitoring Perspectives and Dimensions

- DB2 allows monitoring metrics to be accessed through a number of different reporting dimensions
- Allows more effective drilldown, and different perspectives on the data to help isolate problems
- Three main dimensions, each consisting of a number of reporting points with corresponding routines
- System / Request
  - Provide total perspective of application work being done by database system
  - Aggregated through the WLM infrastructure
- Data objects
  - Provide perspective of impact of all activity occurring with the scope of data objects
  - Aggregated through data storage infrastructure
- **Activity / Query**
  - **Provide perspective of work being done by specific SQL statements**
  - **Aggregated through the package cache infrastructure**

In this session we will spend our time here

Access Points: Activity Perspective

- MON_GET_PKG_CACHE_STMT
  - Both static and dynamic SQL (historical)
- MON_GET_PKG_CACHE_STMT_DETAILS
  - XML based output
- MON_GET_ACTIVITY
  - Information on current executing activities / queries
- MON_GET_ACTIVITY_DETAILS
  - XML based output

Some Additional Tips

- Monitoring data is accumulated and maintained in-memory from point of database activation until de-activation
  - Explicitly activate your database to ensure consistent availability of monitoring metrics
- Monitoring metrics are incremented globally at each of the reporting levels and do not reset
  - To compute changes in metrics over a specific period of time take an initial baseline sample and compute deltas from that (eg. compute I/O a particular SQL statement has driven over the past 5 mins)
- Event monitors can be utilized to capture and persist event based data for historical analysis
  - Package cache event monitor for aggregate statement data
  - Activity event monitor for individual statement executions

Identifying High Impact SQL Statements

MON_GET_PKG_CACHE_STMT()

- Ideal entry point for analyzing query problems
- Query a wealth of metrics for any statement that is active in the package cache
  - Rank and order by any of these metric
  - Aggregate metrics accumulated after each statement execution
  - Both static and dynamic SQL
  - Metrics collected by default
  - Low overhead
- Retains significant workload information with a modest `PCKCACHESZ`
- Package Cache Event Monitor can be configured in cases where cache evictions are causing information to be lost
- Only limitation is that it doesn't track individual executions

Finding High Impact Queries

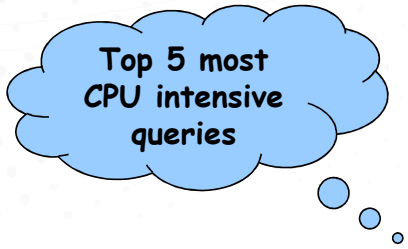
Top 5 queries by statement execution time in server

```
select stmt_exec_time, num_executions, stmt_text
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
order by stmt_exec_time desc fetch first 5 rows only
```

Statement with most execution time in the server

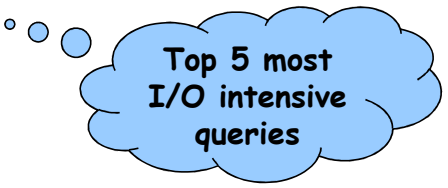
STMT_EXEC_TIME	NUM_EXECUTIONS	STMT
3951764	2218111	SELECT s_quantity, s_dist_01, s_dist_02, …
902078	195866	SELECT c_balance, c_delivery_cnt …
619547	212999	DECLARE CUST_CURSOR1 CURSOR FOR SELEC …
480681	221873	SELECT w_tax, c_discount, c_last, c_credit …
441494	20124	SELECT count(distinct S_I_ID) INTO :H …

More High Impact Queries




```
select stmt_exec_time, num_executions,
       (total_cpu_time / 1000) as cpu_time,
       stmt_text
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
order by cpu_time desc fetch first 5 rows only
```

```
select stmt_exec_time, num_executions,
       (pool_read_time + pool_write_time +
        direct_read_time + direct_write_time) as io_time
 from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as t
order by io_time desc fetch first 5 rows only
```



Queries with the Worst Relative Velocity

```
select total_act_time, total_act_wait_time,
   (case when total_act_time > 0
   then ((total_act_time - total_act_wait_time) * 100
         / total_act_time)
    else 100
    end) as relvelocity,
   stmt_text
from table (mon_get_pkg_cache_stmt(null,null,null,-2)) as t
order by relvelocity fetch first 5 rows only
```

Compute percentage of query time where we're processing

Relative velocity shows the degree to which progress of the query is impacted by waits

Majority of query time spent in waits!

TOTAL_ACT_TIME	TOTAL_ACT_WAIT_TIME	RELVELOCITY	STMT_TEXT
1481597	1457690	1	DECLARE READ_ORDERLI …
228	223	2	create view dbtimeme …
28	27	3	alter table activity …
30	29	3	create event monitor …
35	33	5	create event monitor …

Queries with the Least Efficient Plans

```
select rows_returned, rows_read,
       (case when rows_returned > 0
        then rows_read / rows_returned
        else 0
       end) as ratio,
       stmt_text as stmt
from table(mon_get_pkg_cache_stmt(null,null,null,-2))
as p
order by ratio desc
fetch first 10 rows only
```

Ratio of rows read to rows returned

This query shows us how much data we processed to produce a single row of results

ROWS_RETURNED	ROWS_READ	RATIO	STMT
2	11137814	5568907	select count(*) from acti…
1	5568907	5568907	select min(time_completed
3	9	3	select * from syscat.WORK…
9	9	1	select substr(serviceclas…
9	9	1	select * from dbtimedelta…
2843729	2843729	1	DECLARE CUST_CURSOR1 CURS…
2843729	2843729	1	SELECT w_street_1, w_stre…
29599464	29599528	1	SELECT s_quantity, s_dist…
0	14	0	alter table control drop…
0	13	0	create view dbtimemetrics…

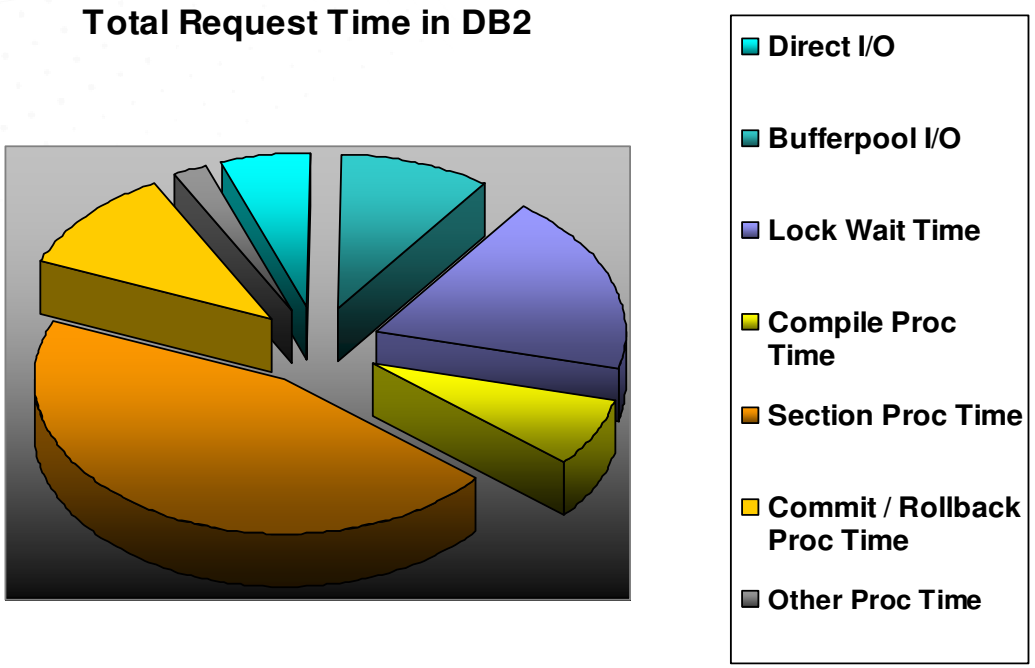
Analyzing Queries using Time Spent

Time Spent Metrics

- Set of metrics in Db2 that break down where time is spent within the server
  - Sum of time spent by each agent thread in the system (foreground processing)
  - Provides *relative* breakdown of time spent
    - Which areas are the most expensive during request / query processing
  - Available in both the system and activity perspectives
    - This presentation will focus on analysis from the activity perspective
  - Can be used for rapid identification and diagnosis of performance problems

- Times are divided into:
  - **Wait times**
    - Time agent threads spend blocking on I/O, network communications, etc
  - **Processing times**
    - Time spent processing in different component areas when the agent was not stuck on a wait
  - **Summary / total times**
    - Total time spent in a particular component area including both processing + wait times

"Time Spent" Metrics: Breakdown of Wait + Processing Times in Db2

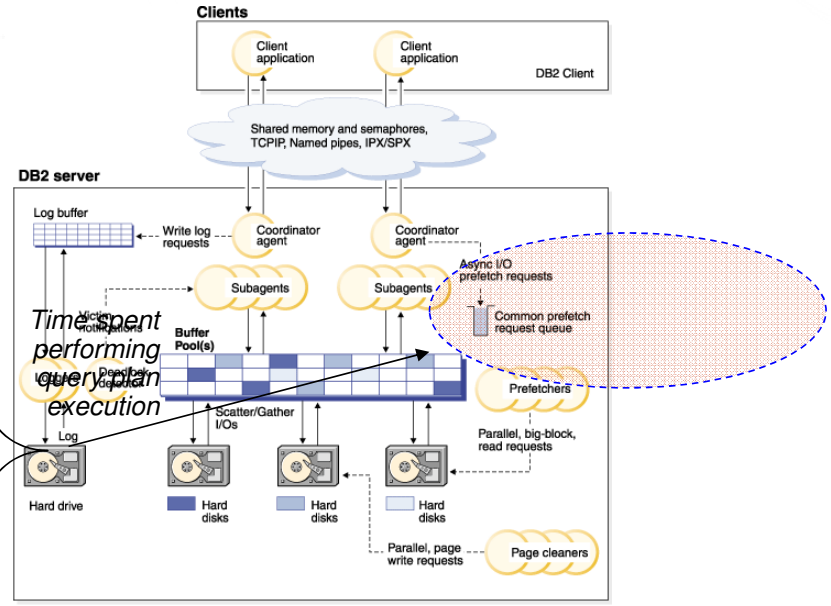


Activity Time Spent Hierarchy

"Time spent" metrics are mutually exclusive and in aggregate form a hierarchy (shown below) that breaks down the time spent executing queries in the database server on behalf of the client. Below we show the hierarchy for the activity perspective.

SQL Statement Execution

WLM_QUEUE_TIME_TOTAL
STMT_EXEC_TIME
- TOTAL_ACT_WAIT_TIME
  - LOCK_WAIT_TIME
  - LOG_BUFFER_WAIT_TIME
  - LOG_DISK_WAIT_TIME
  - FCM_SEND/RECV_WAIT_TIME
  - DIAGLOG_WRITE_WAIT_TIME
  - POOL_READ/WRITE_TIME
  - DIRECT_READ/WRITE_TIME
  - (…)
- TOTAL_SECTION_PROC_TIME
  - TOTAL_SECTION_SORT_PROC_TIME
  - **TOTAL_COL_PROC_TIME [new]**
- TOTAL_ROUTINE_NON_SECT_PROC_TIME
  - TOTAL_ROUTINE_USER_CODE_PROC_TIME
- **TOTAL_INDEX_BUILD_PROC_TIME [new]**
- *(Any nested query processing)*



Analyzing Individual Queries Using Time Spent

- Once we have pinpointed our statements of interest, our next step is to drill down to understand where they are spending their time
- By understanding where the time is being spent during query execution we can identify where the database server is spending effort, and look for opportunities for tuning
- We can use the `EXECUTABLE_ID` value from problem statements identified via previous examples to lookup detailed time metrics for statements of interest and perform more in depth analysis
  - Uniquely identifies each query plan in the package cache

"Where is my time being spent?"

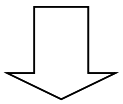
Show me the full hierarchy of waits + processing times for a particular statement

```
select p.executable_id, r.metric_name, r.parent_metric_name,
       r.total_time_value as time, r.count, p.member
from
(select stmt_exec_time, executable_id
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
order by stmt_exec_time desc fetch first row only) as stmts,
table(mon_get_pkg_cache_stmt_details(null,
                                     stmts.executable_id,
                                     null,
                                     -2)) as p,
table(mon_format_xml_times_by_row(p.details)) as r
order by stmts.executable_id, total_time_value desc
```

Find statement with most time in server

Executable ID for our statement(s) of interest

Format XML details to produce row based format for time spent metrics



(continued)

EXEC_ID	METRIC_NAME	PARENT_METRIC_NAME	TIME	COUNT	MEMBER
x'00000001…'	STMT_EXEC_TIME	-	6676617	110191	0
x'00000001…'	**TOTAL_ROUTINE_NON_SECT_PROC_TIME**	**STMT_EXEC_TIME**	**6008956**	**110191**	**0**
x'00000001…'	**TOTAL_ROUTINE_USER_CODE_PROC_TIME**	**TOTAL_ROUTINE_NON_S**	**6008956**	**110191**	**0**
x'00000001…'	POOL_READ_TIME	TOTAL_ACT_WAIT_TIME	372754	52135	0
x'00000001…'	TOTAL_ACT_WAIT_TIME	STMT_EXEC_TIME	372754	-	0
x'00000001…'	TOTAL_SECTION_PROC_TIME	STMT_EXEC_TIME	294907	0	0
x'00000001…'	WLM_QUEUE_TIME_TOTAL	-	0	0	0
x'00000001…'	FCM_TQ_RECV_WAIT_TIME	FCM_RECV_WAIT_TIME	0	0	0
x'00000001…'	FCM_MESSAGE_RECV_WAIT_TIME	FCM_RECV_WAIT_TIME	0	0	0
x'00000001…'	FCM_TQ_SEND_WAIT_TIME	FCM_SEND_WAIT_TIME	0	0	0
x'00000001…'	FCM_MESSAGE_SEND_WAIT	FCM_SEND_WAIT_TIME	0	0	0
x'00000001…'	LOCK_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	DIRECT_READ_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	DIRECT_WRITE_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…`	LOG_BUFFER_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	LOG_DISK_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	POOL_WRITE_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	AUDIT_FILE_WRITE_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	AUDIT_SUBSYSTEM_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	DIAGLOG_WRITE_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	FCM_SEND_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	FCM_RECV_WAIT_TIME	TOTAL_ACT_WAIT_TIME	0	0	0
x'00000001…'	TOTAL_SECTION_SORT_PRO	TOTAL_SECTION_PROC_T	0	0	0

…

Common Statement Bottlenecks

- I/O bottlenecks
  - Large bufferpool read / write times may indicate excessive table scans occurring, spilling to temps, or a poorly tuned I/O subsystem
  - Unexpected direct read / write times may indicate lobs that aren't inlined properly, or unexpected usage of temps in query plan
- Locking bottlenecks
  - Large lock wait times indicate contention problems in your workload are affecting your query performance
- Routine bottlenecks
  - Large routine times may indicate inefficiencies or problems with procedures or user defined functions
- Reclaim wait bottlenecks [PureScale]
  - Large reclaim wait times indicate cross member page contention is impacting your query execution
- Diagnostic or audit bottlenecks
  - Diag log or audit wait times may indicate cases where diagnostic or audit related logging is unexpectedly impacting query performance

Latest Time Spent Metrics

- TOTAL_BACKUP_TIME / TOTAL_BACKUP_PROC_TIME / TOTAL_BACKUPS
  - New time spent category for online backups
- TOTAL_INDEX_BUILD_TIME / TOTAL_INDEX_BUILD_PROC_TIME / TOTAL_INDEXES_BUILT
  - New time spent category for index creation / recreations
- TOTAL_COL_TIME / TOTAL_COL_PROC_TIME
  - Time spent in the columnar runtime

Example: Assessing Efficiency of Columnar Query

```
SELECT TOTAL_SECTION_TIME, TOTAL_COL_TIME,
       DEC((FLOAT(TOTAL_COL_TIME)/
            FLOAT(NULLIF(TOTAL_SECTION_TIME,0)))*100,5,2)
       AS PCT_COL_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-1)) AS T
WHERE STMT_TEXT = 'SELECT * FROM TEST.COLTAB A, TEST.ROWTAB B WHERE A.ONE
= B.ONE'
```

TOTAL_SECTION_TIME	TOTAL_COL_TIME	PCT_COL_TIME
5	4	80.00

Majority of processing occurred in the highly optimized columnar runtime

Compute the ratio of columnar processing time to overall section processing time to see how much we're leveraging the columnar runtime

IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

Monitoring Query Working Memory Usage and Spilling

Monitoring Sort Memory Usage

- Sort memory can be monitoring through the following metrics
  - SORT_SHRHEAP_ALLOCATED (current)
  - SORT_SHRHEAP_TOP (high watermark)
  - SORT_CONSUMER_SHRHEAP_TOP (per consumer hwm)
- Accessible at multiple levels of reporting
  - MON_GET_DATABASE (Database level)
  - MON_GET_PKG_CACHE_STMT (Query level)
  - MON_GET_SERVICE_SUBCLASS_STATS (Subclass level)
  - Others
- Example:

```
SELECT SORT_SHRHEAP_ALLOCATED,
       SORT_SHRHEAP_TOP
FROM TABLE(MON_GET_DATABASE(-1))
```

Obtain current and maximum sort usage for the database

Monitoring Sort Consumers

- Total individual sort consumer counts including
  - TOTAL_SORT_CONSUMERS (overall total)
  - TOTAL_HASH_GRPBYS
  - TOTAL_HASH_JOINS
  - TOTAL_OLAP_FUNCS
  - TOTAL_SORTS
  - TOTAL_COL_VECTORS_CONSUMERS

- Memory throttling and overflow / spill counts
  - POST_THRESHOLD_HASH_GRPBYS / HASH_GRPBY_OVERFLOWS
  - POST_THRESHOLD_HASH_JOINS / HASH_JOIN_OVERFLOWS
  - POST_THRESHOLD_OLAP_FUNCS / OLAP_FUNC_OVERFLOWS
  - POST_THRESHOLD_SORTS / SORT_OVERFLOWS
  - POST_THRESHOLD_COL_VECTOR_CONSUMERS

Monitoring Sort Consumers

- Active sort consumer counts and high watermarks
  - ACTIVE_SORT_CONSUMERS / ACTIVE_SORT_CONSUMERS_TOP
  - ACTIVE_HASH_GRPBYS / ACTIVE_HASH_GRPBYS_TOP
  - ACTIVE_HASH_JOINS / ACTIVE_HASH_JOINS_TOP
  - ACTIVE_OLAP_FUNCS / ACTIVE_OLAP_FUNCS_TOP
  - ACTIVE_SORTS / ACTIVE_SORTS_TOP
  - ACTIVE_COL_VECTORS_CONSUMERS / ACTIVE_COL_VECTOR_CONSUMERS_TOP
- Also accessible at multiple levels of reporting
  - **MON_GET_DATABASE** (Database level)
  - **MON_GET_PKG_CACHE_STMT** (Query level)
  - **MON_GET_SERVICE_SUBCLASS_STATS** (Subclass level)
  - **Others**

Monitoring for Spilling

```
with ops as
( select
   (total_sorts + total_hash_joins + total_hash_grpbys)
    as sort_ops,
   (sort_overflows + hash_join_overflows + hash_grpby_overflows)
    as overflows,
    sort_shrheap_top as shr_sort_top
from table(mon_get_database(-2)))
select sort_ops,
       overflows,
       (overflows * 100) / nullif(sort_ops,0) as pctoverflow,
       shr_sort_top
from ops;
```

Extract percentage of sort operations that have spilled and high watermark sort usage

SORT_OPS	OVERFLOWS	PCTOVERFLOW	SHR_SORT_TOP
1200	300	25	12777216

About 25% of our sort operations overflowed and spilled indicating some tuning may be worthwhile

If SORT_HEAP_TOP is near the configured SHEAPTHRES_SHR it indicates that our SORTHEAP is overconfigured relative to our concurrency limits

Monitoring Query Sort Usage and Consumers

```
SELECT SORT_SHRHEAP_TOP,
       SORT_CONSUMER_SHRHEAP_TOP,
       ACTIVE_SORT_CONSUMERS_TOP,
       NUM_EXECUTIONS,
       (TOTAL_SORTS +
        TOTAL_HASH_JOINS +
        TOTAL_HASH_GRPBYS +
        TOTAL_COL_VECTOR_CONSUMERS) AS SORT_OPS,
       (SORT_OVERFLOWS +
        HASH_JOIN_OVERFLOWS +
        HASH_GRPBY_OVERFLOWS) AS SORT_OVERFLOWS,
       (POST_THRESHOLD_SORTS +
        POST_THRESHOLD_HASH_JOINS +
        POST_THRESHOLD_HASH_GRPBYS +
        POST_THRESHOLD_COL_VECTOR_CONSUMERS) AS THROTTLED_SORT_OPS,
       SUBSTR(STMT_TEXT,1,255) AS STMT_TEXT
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-2))
```

Sort usage + biggest individual consumer + active consumers

Aggregate sort operators

Aggregate sort overflows / spills

Aggregate throttled sort operators

Total memory usage

Biggest single operator

4 distinct sort consumers in this query

```
SORT_SHRHEAP_TOP SORT_CONSUMER_SHRTHEAP_TOP ACTIVE_SORT_CONSUMERS_TOP … STMT_TEXT
---------------- --------------------------- -------------------------- ------------------------
          262144                      131072                          4 WITH OPS AS (SELECT …
```

IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

Advanced Diagnostics Using Runtime Explain and Section Actuals

Introducing Runtime Explain

- There may be cases when a more detailed analysis of query execution is required than basic monitoring metrics such as time spent can provide
- In these cases the tool we typically turn to is the EXPLAIN feature of DB2 – which we will refer to herein as the "SQL Compiler EXPLAIN"
  - Compiles an input SQL statement and allows you to format and view the query plan
  - Generally accurate approximation of the query you actually executed
  - Possible differences due to compilation environment and/or table statistics from when your query was compiled
- For this reason we introduced the ability to perform a "Runtime" EXPLAIN (explain from section) which is generated directly from a compiled query plan in the engine.
- Allows you to generate plan output from the actual section you were executing
- Enables additional diagnostic features like section actuals and object metrics

Explain from Section Procedures

- Set of stored procedures provided that allow you generate an explain from a runtime section
  - EXPLAIN_FROM_CATALOG
  - EXPLAIN_FROM_SECTION
  - EXPLAIN_FROM_ACTIVITY
  - EXPLAIN_FROM_DATA
- Explain table content can then be processed using the standard explain tools (eg. db2exfmt)
- Output can be generated from any of the following sources:
  - Static or dynamic statement entries in the package cache
  - Any cache entry captured by the new package cache event monitor
  - Static statement from the catalog tables
  - Statement execution captured with section by the activity event monitor

Section Actuals + Object Metrics

- One key benefit of explain from section is the ability to capture and format "section actuals" and "object metrics"
  - All EXPLAIN output will contain cardinality estimates for individual operators in the plan
  - Explains generated from captured activity data (EXPLAIN_FROM_ACTIVITY) will also contain actual cardinalities and metrics per-data object within the query
- Examining this output gives you a detailed indication of what actually happened during the query execution
  - How closely actual cardinalities matched estimates
  - What activity occurred on individual data objects
- In order to examine these metrics we will need to capture an execution of our SQL statement of interest using the activity event monitor

Capturing Activities to Obtain Detailed Explain Metrics

- The **activity event monitor** in Db2 allows the capture of execution details for individual SQL statements as well as several other recognized activities (eg. Load)
- It can be configured to capture a variety of different metrics as well as the section data which includes **actual cardinalities** and **object metrics**
- Since the capture of individual activities is quite granular we offer a fair degree of flexibility allowing the following data capture options:
  - Capture data for all activities running in a particular WLM workload
  - Capture data for all activities running in a particular WLM service class
  - Capture data for activities that violate a particular WLM threshold
- We can also enable the capture of activities run by a specific application using the `WLM_SET_CONN_ENV` procedure
- Our final example will demonstrate how to capture a statement of interest using the activity event monitor and then obtain the detailed explain metrics

Step I: Prereq Setup Steps

```
call sysproc.sysinstallobjects('EXPLAIN','C',null,null)
```

Create the explain tables…

```
create event monitor actEvmon for activities write to table
activity ( table activity, in monitorTBS ),
activityvals ( table activityvals, in monitorTBS ),
activitystmt ( table activitystmt, in monitorTBS ),
activitymetrics ( table activitymetrics, in monitorTBS ),
control ( table control, in monitorTBS )
manualstart
```

Create the activity event monitor

Step II: Capturing the Activity Data

```
set event monitor actEvmon state 1
call wlm_set_conn_env(null,
    '<collectactdata>WITH DETAILS,
  SECTION</collectactdata>
    <collectactpartition>ALL</collectactpartition>
    <collectsectionactuals>BASE</collectsectionactuals>')
```

Enable the event monitor and setup to capture a statement on my connection

```
select t1.ident, sum(t1.data) as data,
       sum(t2.moredata) as moredata
from t1,t2
where t1.ident=t2.ident
group by t1.ident
```

Execute the statement I'm interested in

```
call wlm_set_conn_env(null,
  '<collectactdata>NONE</collectactdata>
   <collectsectionactuals>BASE</collectsectionactuals>')
set event monitor actEvmon state=0
```

Disable collection and the event monitoring once I am done

Step II: Another approach

```
set event monitor actEvmon state 1
update db cfg using section_actuals base
alter service class sysdefaultsubclass under
  sysdefaultuserclass
    collect activity data on all database partitions with
  details,section
```

Enable the event monitor on the default subclass, and collect details and section data

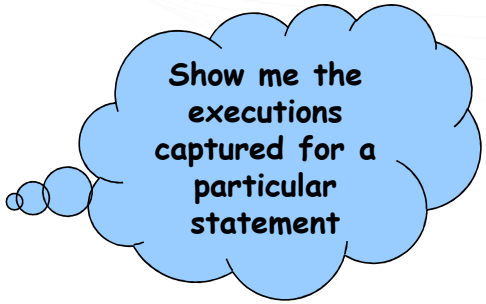
(Queries of interest run and are captured…)

```
alter service class sysdefaultsubclass under
  sysdefaultuserclass
    collect activity data none
update db cfg using section_actuals none
set event monitor actEvmon state 0
```

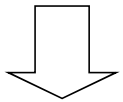
Disable the event monitor once I am done

Step III: Locating the activity of interest

```
select a.appl_id, a.uow_id, a.activity_id, a.appl_name,
       s.executable_id, s.stmt_text
from activity as a,
     activitystmt as s
     where a.appl_id = s.appl_id and
           a.uow_id = s.uow_id and
           a.activity_id = s.activity_id and
           s.stmt_text like 'select * from t1%'
```



Identifiers for the activity



```
APPL_ID                    UOW_ID ACTIVITY_ID  EXECUTABLE_ID    STMT_TEXT                    APPL_NAME
-------------------------- ------ ------------ ---------------- ---------------------------- ----------
*LOCAL.davek.100917004844 62     1            x'010000…1E00' select * from t1,t2 where…  db2bp
```

Step III: An alternate approach

```
select a.appl_id, a.uow_id, a.activity_id, a.appl_name,
       m.total_cpu_time, s.executable_id, s.stmt_text
from
       activity as a,
       activitystmt as s,
       activitymetrics as m
where a.appl_id = s.appl_id and
      a.uow_id = s.uow_id and
      a.activity_id = s.activity_id
and
      a.appl_id = m.appl_id and
      a.uow_id = m.uow_id and
      a.activity_id = m.activity_id
order by total_cpu_time desc fetch first 5 rows only
```

Find the captured activities with the largest CPU time

Statement executable id

Identifiers for the activity

APPL_ID	UOW_ID	ACTIVITY_ID	APPL_NAME	TOTAL_CPU_TIME	EXECUTABLE_ID	STMT_TEXT
*LOCAL.davek.100917004844	62	1	db2bp	30500	x'0100…01E00'	select t1.ident, sum(t1.d …
*LOCAL.davek.100917004844	64	1	db2bp	5360	x'0100…00900'	CALL wlm_set_conn_env(?,? …
*LOCAL.davek.100917001050	105	1	db2bp	4603	x'0100…04A00'	CALL wlm_set_conn_env(?,? …
*LOCAL.davek.100919015109	20	1	db2bp	444	x'0100…05000'	SELECT TABNAME, TABSCHEMA …
*LOCAL.davek.100919015109	25	1	db2bp	406	x'0100…05000'	SELECT TABNAME, TABSCHEMA …

Step III: Notes on MPP and PureScale

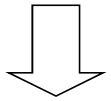
- In **MPP**, statement execution is distributed across multiple partitions
  - Activity data must be collected on all partitions to capture the work done by the query
  - Each partition involved in the query will generate an activity record and section actuals corresponding to that partition's contribution
  - The explain process will amalgamate information across partitions automatically
- In **PureScale** statement execution is local to a particular member
  - Only the coordinator member will execute the query plan and generate section actuals
  - Note that the statement execution may still involve contention on global resources that are being contended for by other members

Step IV: Performing and Formatting the Explain from Section

Identifiers for the activity

```
call explain_from_activity('*LOCAL.davek.100715194643',
                           85,1, 'ACTEVMON', null,
                           ?,?,?,?,?)
```

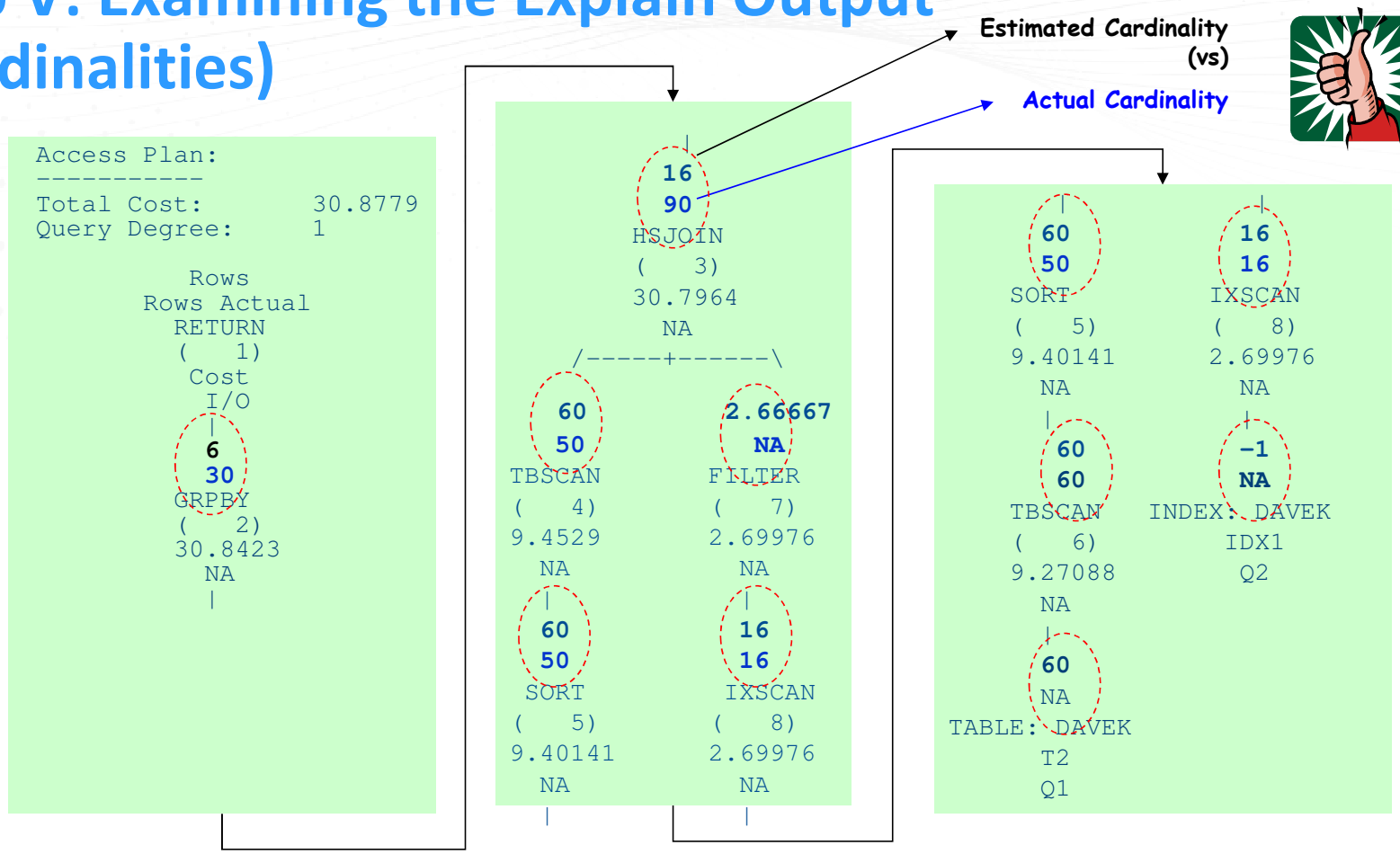
Perform an explain on the activity of interest…



```
db2exfmt -d sample -w -1 -n %% -# 0 -s %% -o explain.txt
```

Now format the most recent data in the explain tables to a readable text file

Step V: Examining the Explain Output (Cardinalities)



Step V: Examining the Explain Output (Object Metrics)

```
Runtime statistics for objects used in Access Plan:
---------------------------------------------------

Schema: DAVEK
Name:     IDX1
Type:     Index

          Metrics
          --------------------
          object_index_l_reads:5
          object_index_lbp_pages_found:5
          object_index_gbp_indep_pages_found_in_lbp:5

Schema: DAVEK
Name:     T2
Type:     Table

          Metrics
          --------------------
          rows_reads:60
          object_data_l_reads:10
          object_data_lbp_pages_found:10
          object_data_gbp_indep_pages_found_in_lbp:10
```

Index Object

Index I/O metrics

Table Object

Table activity + I/O

Per-object metrics available through EXPLAIN

- Tables
  - Rows inserted / updated / deleted / read
  - Overflow creates / accesses
  - Lock wait time + lock escalations
  - Direct reads / writes
  - Bufferpool metrics for data, xda, columnar storage
- Indexes
  - Bufferpool metrics for index storage

Questions?

DB2 Monitoring Resources

- Tuning and Monitoring Database System Performance
  - https://www.ibm.com/developerworks/community/wikis/form/anonymous/api/wiki/0fc2f498-7b3e-4285-8881-2b6c0490ceb9/page/24f2e298-60e3-4a19-8da8-0d92b775ed66/attachment/2a8df9be-3958-47b1-b92c-11d66745176e/media/DB2BP_System_Performance_0813.pdf
- DB2 Monitoring Enhancements for BLU Acceleration
  - http://www.ibm.com/developerworks/data/library/techarticle/dm-1407monitor-bluaccel/index.html

IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

David Kalmuk
IBM
dckalmuk@ca.ibm.com

Session code: D04

Please fill out your session evaluation before leaving!