



Monitoring BLU Acceleration In Depth

David Kalmuk

IBM

Session Code: D19
Friday May 8th, 8:00 – 9:00
Platform: DB2 for LUW



The BLU Acceleration Technology introduced in DB2 10.5 brings with it a host of new monitoring considerations and metrics. This session will introduce you to the new monitoring capabilities that support BLU Acceleration and through practical examples will show you how you can leverage these in your analytics environment.



Objectives

- Learn how to select a workload for BLU, and how to monitor columnar query execution via time spent metrics and runtime explain
- Learn how to monitor table size and compression rates
- Learn how to monitor bufferpool and prefetcher performance for columnar workloads
- Learn how to track BLU memory consumption and performance via new sort monitoring metrics
- Learn how to monitor the new default workload management concurrency controls

Learn how to select a workload for BLU, and how to monitor columnar query execution via time spent metrics and runtime explain

Learn how to monitor table size and compression rates

Learn how to monitor bufferpool and prefetcher performance for columnar workloads

Learn how to track BLU memory consumption and performance via new sort monitoring metrics

Learn how to monitor the new default workload management concurrency controls

Agenda

- A quick review of the BLU Acceleration technology
- Selecting a workload and monitoring columnar query execution
- Monitoring columnar table size and compression rates
- Monitoring bufferpools and I/O performance for columnar workloads
- Monitoring BLU sort memory consumption and performance
- Monitoring the default workload management for analytic workloads



IDUG DB2 North America Tech Conference
Philadelphia, Pennsylvania | May 2015

#IDUGDB2

A Quick Review of the BLU Acceleration Technology



The March to Valley Forge. William E. Trigo (1883)

Introducing BLU Acceleration



IBM Research & Development Lab Innovations

- Dynamic In-Memory**

In-memory columnar processing with dynamic movement of data from storage data



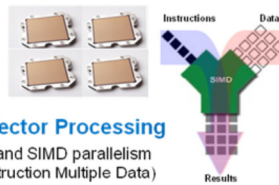
- Actionable Compression**

Patented compression technique that preserves order so that the data can be used without decompressing



- Parallel Vector Processing**

Multi-core and SIMD parallelism (Single Instruction Multiple Data)



- Data Skipping**

Skips unnecessary processing of irrelevant data





DB2 with BLU Acceleration

1. Next generation database

- Fast (query performance)
- Small (storage savings)
- Simple (load-and-go)

Same "old"
SQL, "newly"
implemented!

2. Seamlessly integrated

- Built seamlessly into DB2
 - Rich, robust capabilities for security, availability, cloud computing.
 - Very mature query compiler, storage and caching infrastructure.
- Consistent SQL, language interfaces, administration
- Dramatic simplification

3. Hardware optimized

- Memory optimized
- CPU-optimized
- I/O optimized

Selecting a Workload and Monitoring Columnar Query Execution



Will your workload benefit from BLU?

Probably:

- Analytical workloads, data marts
- Grouping, aggregation, range scans, joins
- Queries touch only a subset of the columns in a table
- Star or Snowflake Schema
- SAP Business Warehouse

Probably not:

- OLTP
- Point access to 1 or few rows
- Insert, Update, Delete of few rows per transaction
- Queries touch many or all columns in a table
- Heavy use of XML, Temporal, LOBs

BLU is aimed at analytical query processing and these are the workloads where you are going to see benefit from this technology. Above we categorize some of the common characteristics of workloads that are a good fit for BLU.

In the vast majority of cases you will want to make an assessment for the overall workload and decide whether to transition it entirely to BLU or not.

IBM Optim Query Tuner

Advisor identifies candidate tables for conversion to columnar format.

Analyzes SQL workload and estimates execution cost on row- and column-organized tables.

Review Workload Advisor Recommendations

This page shows the recommendations from the advisors that you ran.

Database connection: ✔ TPCSDANv10.2hotel67 (DB2 for Linux, UNIX, and Windows V10.5.0)

▶ Status/Description

Statements | Summary | **Table organization** | Candidate Table Organization

Estimated performance improvement: 83.44 %
 Number of tables referenced in the workload: 11 Number of tables recommended for conversion: 11

Show DDL Script Test Candidate Table Organization Filter by: Tables to be converted

Table	Creator	Current Organization	Recommended Organization	Conversion Warning
HOUSEHOLD_DEMOG...	TPCDS	ROW	COLUMN	Indexes will be remove
DATE_DIM	TPCDS	ROW	COLUMN	Indexes will be remove
WEB_SALES	TPCDS	ROW	COLUMN	Indexes will be remove
STORE	TPCDS	ROW	COLUMN	Indexes will be remove
STORE SALES	TPCDS	ROW	COLUMN	Indexes will be remove

If you are dealing with a database that handles a mixed operational / analytic environment OQWT offers a workload advisor that can help you identify which tables are the best candidates for conversion to columnar format.

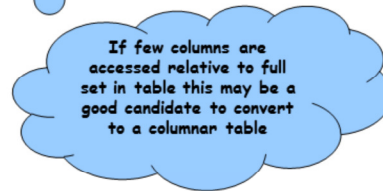
Looking at Table Column Selectivity

- MON_GET_TABLE function includes new metrics to help assess columns accessed per query

```
SELECT SECTION_EXEC_WITH_COL_REFERENCES AS NUM_QUERIES,
       (NUM_COLUMNS_REFERENCED /
        NULLIF(SECTION_EXEC_WITH_COL_REFERENCES, 0))
       AS AVG_COLS_REF_PER_QUERY
FROM TABLE(MON_GET_TABLE('MYSHEMA', 'MYTABLE', -1))
```



NUM_QUERIES	AVG_COLS_REF_PER_QUERY
655	2



Column selectivity is a useful metric in helping to assess which tables may benefit by being in columnar format. We have added new monitoring metrics with DB2 BLU to allow you to query the column selectivity for any tables on the database. In the example above we show how to compute the average columns referenced per query for a specific table. You can use this metric in combination with other factors to help decide which tables might be best suited for columnar data (or use a tool like OQWT which uses this metric internally as part of its advisors).

Time Spent Metrics

- Provide a hierarchical breakdown of where time is spent in the DB2 engine
- Available at multiple reporting levels
 - Eg. MON_GET_DATABASE, MON_GET_PKG_CACHE_STMT
- New time spent elements for measuring amount of columnar processing
 - TOTAL_COL_TIME / TOTAL_COL_PROC_TIME

```
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

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

Time spent metrics were a new concept introduced in DB2 9.7 to provide a hierarchical breakdown of where time is spent in the DB2 engine. The time spent breakdown is available at numerous reporting levels from database, to connection, to service class, all the way down to individual queries / activities.

With DB2 BLU we have introduced new time spent metrics that allow you to differentiate how much of the section execution time for a query was spent in the columnar runtime. This allows you to quickly assess the efficiency of columnar workloads and queries by determining whether the majority of their execution time is truly spent in the columnar runtime. At the system level this will indicate to what percentage your workload is actually leveraging the optimized column-oriented processing. By drilling down to the query level you can identify which queries you may want to do further analysis on to identify constructs or data objects may be preventing them from fully leveraging column-oriented processing.



Explain / Runtime Explain

- Explains represent our most granular form of diagnostics for query performance analysis
- New CTQ operator denotes boundary between row and column oriented processing for DB2 BLU
- Key points for analyzing column-oriented processing with explains
 - Position of CTQ in the plan indicates how much we are able to leverage the columnar runtime
 - Also look at the cardinalities (including section actuals) for number of rows flowing out of the CTQ into row-oriented processing
 - Look for incompatible query constructs, or row oriented data objects that may cause portions of the plan to revert to row-oriented processing

Explains represent our most granular form of query diagnostics and come in two flavors; compile time and runtime explains. With DB2 BLU the new CTQ operator in the plan denotes the boundary between row and column oriented processing for DB2 BLU

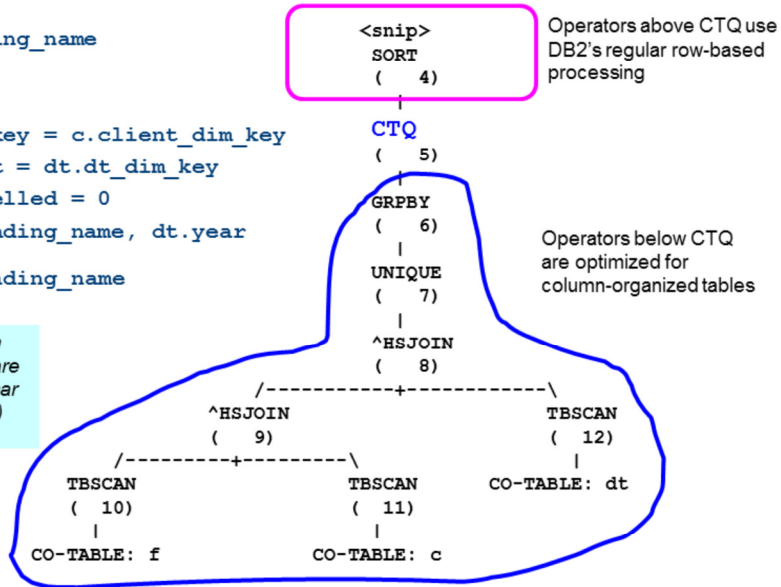
When analyzing plans the position of the CTQ and the cardinality flowing out of the CTQ are indicators for how optimal our query plan is and how much of the processing is being done in the column-oriented runtime. In cases of suboptimal plans look for incompatible query constructs or row oriented data objects that may be causing portions of the plan to revert to row-oriented processing.

In the next slides we'll look at an example.

Example of an Optimal Execution Plan

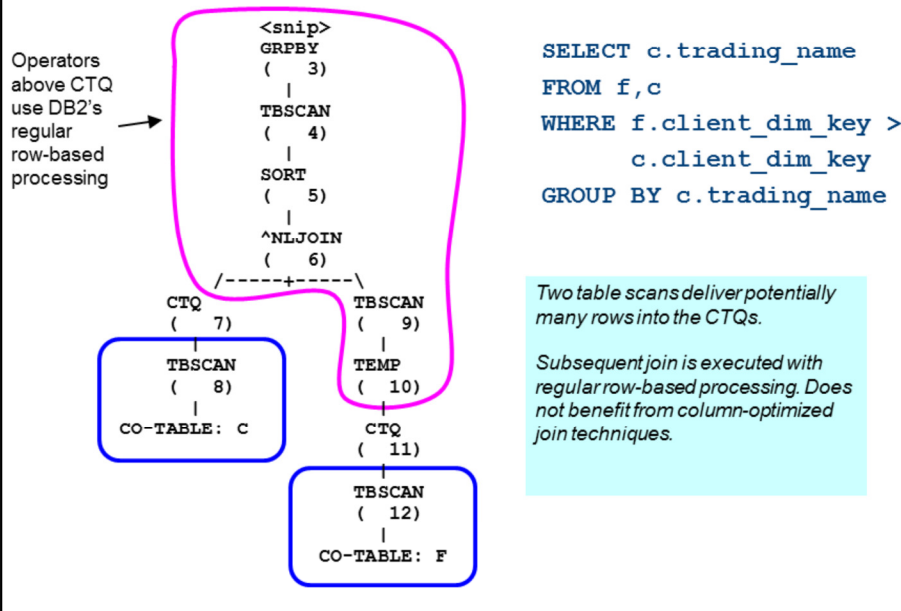
```
SELECT c.trading_name
FROM f, c, dt
WHERE
f.client_dim_key = c.client_dim_key
AND f.trade_dt = dt.dt_dim_key
AND f.is_cancelled = 0
GROUP BY c.trading_name, dt.year
ORDER BY c.trading_name
```

All table scans, hash joins, and grouping are performed in columnar query runtime (good)



Here we are processing a query that includes several joins, grouping and ordering. In the output above you can see that most of this processing occurs below the CTQ operator. The CTQ operator is the entry point into the columnar runtime. Any processing occurring below this operator is occurring in native columnar format, while any processing above it is being performed in conventional row format. Above only the final ordering is performed in row format data, while most of the query operations are performed in the columnar runtime. This means this query will see the full benefits of BLU Acceleration.

Example of a Suboptimal Execution Plan

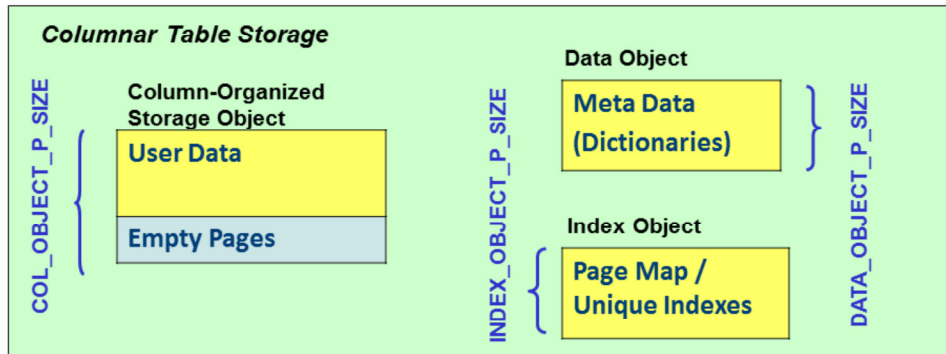


This query is performing a non-equality join which is not supported in the columnar runtime. Since only the table scans are executing below the CTQ in the columnar runtime, this query is effectively scanning column oriented data, converting it to row format and then executing it in the conventional DB2 runtime. As a consequence although this query will execute fine, it's going to see virtually none of the benefits of BLU Acceleration.

Monitoring Columnar Table Size and Compression Rates



Monitoring Table Size



- `ADMIN_GET_TAB_INFO` table function reports
 - `COL_OBJECT_P_SIZE`: Physical size of column-organized data object containing **user data**
 - `DATA_OBJECT_P_SIZE`: Physical size of data object containing **meta data**
 - `INDEX_OBJECT_P_SIZE`: Physical size of index object containing **page map and primary / unique key constraint indexes**

When monitoring table sizes in BLU its important to understand how columnar table storage is broken down in DB2. Columnar tables are stored in three distinct storage areas:

- The column-organized storage where the actual table user data is stored
- Conventional row oriented data storage where the meta-data (primarily dictionaries are stored)
- Index object storage where the page map and any unique constraint indexes are stored

In order to query the total physical table size once can use the `ADMIN_GET_TAB_INFO` SQL function and examine the physical size of these storage objects.

Comparing Space Usage

- Use the ADMIN_GET_TAB_INFO function to compute the physical storage requirements of tables in columnar vs. row format

Columnar Table Storage Footprint


```
select tabschema, tabname,
       (col_object_p_size +
        data_object_p_size +
        index_object_p_size) as space_used_kb
from table(admin_get_tab_info('MYSHEMA','MYCOLTABLE',-1))
```

(User Data +
Meta Data +
Page Map & Unique
Indexes)

Row Table Storage Footprint

```
select tabschema, tabname,
       (data_object_p_size +
        index_object_p_size) as space_used_kb
from table(admin_get_tab_info('MYSHEMA','MYROWTABLE',-1))
```

(User & Meta Data +
Indexes)

- Don't forget to include MQTs as part of the storage cost of tables in row format 

Above we show a simple method to determine a table's storage footprint in columnar and in row format. Recall columnar tables are broken down across three storage areas, while equivalent row tables are broken down across only two.

Don't forget to also include materialized query tables as part of the row footprint since this is part of the storage savings when utilizing columnar tables.

Also a note on synopsis tables; to be 100% technically accurate we'd want to include the size for any synopsis tables as part of the columnar storage footprint calculation, however we've neglected to do that here because synopsis tables typically only occupy 0.1% of the storage footprint of the base table they are associated with.

Monitoring Table Compression

- Check compression rate via PCTPAGESSAVED in SYSCAT . TABLES

$$\begin{aligned} \text{Compression Ratio} &= \text{Uncompressed Size} / \text{Compressed Size} \\ &= 1 / (1 - \text{PCTPAGESSAVED} / 100) \end{aligned}$$

TABSCHEMA='TEST'	TABNAME='TAB1'	PCTPAGESSAVED = 90
Compression Ratio = $1 / (1 - 90/100) = 10x$ compression		

- Other compression statistics do not apply to columnar tables
 - AVGCOMPRESSEDROWSIZE, AVGWROWCOMPRESSIONRATIO, PCTROWCOMPRESSED, AVGWROWSIZE
- ADMIN_GET_TAB_COMPRESS_INFO does not support columnar tables

For column organized tables, table compression rates can be monitored via the PCTPAGESSAVED table statistic in SYSCAT.TABLES. Above we show an example of how to calculate the compression rate based on this metric. To ensure maximum accuracy ensure you table statistics are up to date.

Note that the following compression statistics will all have the value of -1 for N/A for columnar tables.

- AVGCOMPRESSEDROWSIZE
- AVGWROWCOMPRESSIONRATIO
- AVGWROWSIZE
- PCTROWCOMPRESSED

Note also that ADMIN_GET_TAB_COMPRESS_INFO does not currently support columnar tables.

Monitoring Table Dictionary Quality

- Check encoding rate via SYSCAT.COLUMNS



C1	PCTENCODED = 90
C2	PCTENCODED = 75
C3	PCTENCODED = 100



C1	PCTENCODED = 0
C2	PCTENCODED = 10
C3	PCTENCODED = 0

- Poor compression/encoding rate indicates data wasn't loaded properly
- If you see evidence of poor compression you will need to troubleshoot and reload the data
- Remember poor compression also affects query performance

In addition to compression rates you can also examine table dictionary quality by looking at the PCTENCODED field for each column in SYSCAT.COLUMNS.

After deploying and loading data in a BLU environment it's always recommended to perform some sanity validation to ensure the data has been adequately compressed:

- The PCTENCODED field in SYSCAT.COLUMNS will show the percentage of values that were able to be encoded for each column.
- If you see either a poor compression rate or poor encoding rate it's an indication that the data wasn't loaded properly and some troubleshooting is in order.
- Unique keys are expected to compress well due to use of offset coding so a high PCTENCODED is expected.
- Long varchar data is expected to show a relatively lower encoding rate due to its highly randomized nature.

Monitoring Bufferpool and I/O Performance





Monitoring Bufferpool Performance

- Efficient bufferpool access is an important element to achieving maximum performance with BLU
- Column-oriented pages are stored in a separate storage area from row-oriented pages necessitating new monitoring metrics
- New bufferpool metrics allow for monitoring of I/O rates and hit ratios for bufferpool tuning activities
 - POOL_COL_L_READS
 - POOL_COL_P_READS
 - POOL_COL_LBP_PAGES_FOUND
 - POOL_COL_WRITES
 - Also ASYNC variants of the above
- Metrics are available in all standard monitoring interfaces including
 - Eg. MON_GET_DATABASE, MON_GET_BUFFERPOOL, MON_GET_TABLESPACE

An important factor in BLU query performance is the size of the bufferpool memory / efficiency of bufferpool accesses. In terms of factors affecting BLU performance the bufferpool memory is generally second only to sort / working memory.

Because column oriented pages are stored in a separate storage area from row oriented pages, and because the access patterns are different, new bufferpool metrics have been added to allow monitoring of I/O rates and hit ratios in the bufferpool. These metrics are consistent with the existing bufferpool metrics and are available in all the standard interfaces including MON_GET_DATABASE, MON_GET_BUFFERPOOL, MON_GET_TABLESPACE, etc.

Bufferpool Performance: Things to look for

- Ensure your bufferpool size is within the ranges recommended by the BLU best practices
 - 25-40% of system memory
- Check that bufferpool hit ratios look reasonable based on your configuration and data size (note the latest formula)

$$\text{Hit Ratio} = \frac{(\text{LBP_PAGES_FOUND} - \text{ASYNC_LBP_PAGES_FOUND})}{(\text{L_READS} + \text{TEMP_L_READS})}$$



- Keep an eye on excessive temp I/O activity which could be an indication of spilling due to sort memory configuration issues

When monitoring bufferpool performance there are a couple of key things to keep an eye on:

- Ensure the bufferpool is configured within the range prescribed by the BLU Best Practices (25-40% of system memory depending on whether the workload is high concurrency vs. lower concurrency).
- Check that the hit ratios look reasonable based on the size of the active dataset relative to the bufferpool size. Note the update hit ratio calculation that was standardized as of DB2 10.1 to work transparently in both ESE, DPF and PureScale environments.
- Keep an eye on excessive temp I/O activity. This may be an indication of spilling occurring due to a sort memory configuration issue and may be driving unwanted impacts on the I/O subsystem.

Computing Bufferpool Hit Ratios

Compute the standard
hit ratios for columnar,
row, and aggregate
I/O activity

```
SELECT DEC( (FLOAT(PPOOL_DATA_LBP_PAGES_FOUND + PPOOL_COL_LBP_PAGES_FOUND -
PPOOL_ASYNC_DATA_LBP_PAGES_FOUND - PPOOL_ASYNC_COL_LBP_PAGES_FOUND) /
FLOAT(NULLIF(PPOOL_DATA_L_READS + PPOOL_TEMP_DATA_L_READS +
PPOOL_COL_L_READS + PPOOL_TEMP_COL_L_READS,0))) * 100, 5, 2)
AS TOTAL_PAGE_HIT_RATIO,
DEC( (FLOAT(PPOOL_DATA_LBP_PAGES_FOUND - PPOOL_ASYNC_DATA_LBP_PAGES_FOUND) /
FLOAT(NULLIF(PPOOL_DATA_L_READS + PPOOL_TEMP_DATA_L_READS,0))) * 100, 5, 2)
AS ROW_PAGE_HIT_RATIO,
DEC( (FLOAT(PPOOL_COL_LBP_PAGES_FOUND - PPOOL_ASYNC_COL_LBP_PAGES_FOUND) /
FLOAT(NULLIF(PPOOL_COL_L_READS + PPOOL_TEMP_COL_L_READS,0))) * 100, 5, 2)
AS COL_PAGE_HIT_RATIO
FROM TABLE(MON_GET_DATABASE(-2))
```



TOTAL_PAGE_HIT_RATIO	ROW_PAGE_HIT_RATIO	COL_PAGE_HIT_RATIO
79.72	77.70	86.74

Above we show how to compute the standard bufferpool hit ratios for columnar pages, regular row oriented data pages, and overall aggregate I/O activity. As you can see the metrics and calculations are virtually identical to previous releases with the one addition being the columnar page storage. Note the usage of the newer hit ratio formula based on LBP_PAGES_FOUND in order to work transparently across all types of DB2 environments.



Monitoring Prefetcher Performance

- In order to achieve maximum I/O performance it's also important to ensure the prefetcher subsystem is operating efficiently
- We have several metrics for monitoring prefetching of columnar data
 - POOL_COL_P_READS
 - POOL_ASYNC_COL_P_READS
 - SKIPPED_PREFETCH_UOW_COL_P_READS
 - PREFETCH_WAIT_TIME

Another area you will want to look at when monitoring BLU performance is the prefetcher subsystem. BLU queries tend to be complex and operate on large volumes of data so I/O performance is critical.

We've introduced several new metrics for monitoring prefetching of columnar data including columnar page reads, async (or prefetcher driven) page reads, skipped prefetch reads, and prefetch wait time.

Prefetcher Performance: Things to look for

- The prefetch ratio should be close to 100% for scan heavy workloads

$$\text{Prefetch Ratio} = 100 * (\text{POOL_ASYNC_COL_P_READS} / \text{POOL_COL_P_READS})$$

- Check `SKIPPED_PREFETCH_UOW_COL_P_READS`
 - This value indicates how many pages were skipped because the prefetchers did not load the page fast enough and it was read synchronously
- Check `PREFETCH_WAIT_TIME`
 - This time indicates time waited when an agent collided with a prefetcher in the process of reading a page
- Possible corrective steps
 - Check the number of prefetchers configured (AUTO is recommended)
 - Check for problems with storage

When looking at prefetcher performance we recommend looking first at the prefetch ratio which indicates how many page reads were driven asynchronously by the prefetchers vs. synchronously by the agent threads. For scan heavy workloads the prefetch ratio should be close to 100%.

If the ratio is significantly lower than 100% check the `SKIPPED_PREFETCH_UOW_COL_P_READS` and the `PREFETCH_WAIT_TIME`. The `SKIPPED_*_READS` counter indicates how many pages we skipped prefetching for because they were read synchronously by agent threads. When this occurs it indicates the prefetchers are not keeping up with the core workload. `PREFETCH_WAIT_TIME` is a similar secondary metric that indicates how much wait time we incurred because an agent attempted to fetch a page synchronously while it was being read by a prefetcher.

In order to correct prefetcher performance problems you will want to validate the number of prefetchers configured on the database (AUTOMATIC is recommended), and barring an obvious configuration problem you will likely want to perform further investigation in the storage subsystem for possible performance problems.

Examining Prefetch Monitoring Information

```
SELECT POOL_ASYNC_COL_READS AS ASYNC_COL_READS,  
       POOL_COL_P_READS AS COL_P_READS,  
       DEC((FLOAT(PPOOL_ASYNC_COL_READS) /  
             FLOAT(NULLIF(PPOOL_COL_P_READS,0))) * 100, 5, 2)  
       AS COL_PREFETCH_RATIO,  
       SKIPPED_PREFETCH_UOW_COL_P_READS AS SKIPPED_COL_P_READS,  
       PREFETCH_WAIT_TIME  
FROM TABLE(MON_GET_DATABASE(-1))
```

Extract
columnar
prefetch
ratios for the
database



Prefetch ratio of 99% indicates we
are prefetching efficiently

Single skipped read and no
prefetch wait time confirms this

ASYNC_COL_READS	COL_P_READS	COL_PREFETCH_RATIO	SKIPPED_COL_P_READS	PREFETCH_WAIT_TIME
3242134	3274882	99.00	1	0

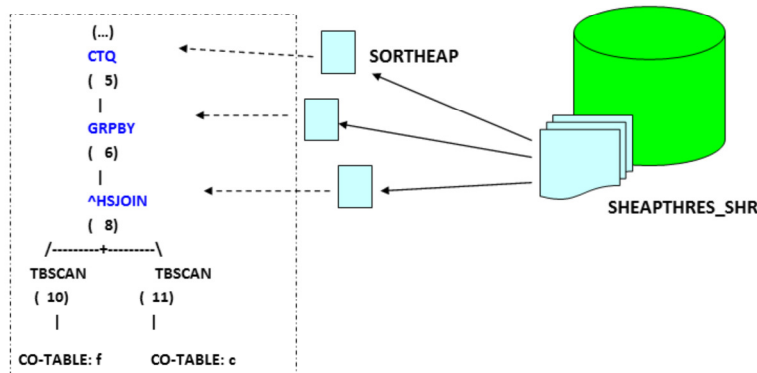
The example above shows how to extract the columnar prefetch ratio and the prefetch wait time. In the example above we can see that the ratio looks good at 99% with only a single page skipped due to synchronous reads. We can also see there has been no prefetch wait time, indicating that our prefetcher subsystem is performing efficiently.

Monitoring BLU Sort Memory Consumption and Performance



BLU Memory Model

- Most working memory for BLU queries comes from database sort memory
 - SHEAPTHRES_SHR (total available memory)
 - SORTHEAP (memory per operator: group-by, join, vector buffering, configured based on expected concurrency)



A key thing to understand about the BLU execution environment are its working memory considerations – having sufficient working memory is the number one factor that’s going to determine how quickly your columnar queries will execute.

Most working memory for BLU queries comes from database sort memory where SHEAPTHRES_SHR defines the total amount of memory on the database available for sort operations, and the SORTHEAP parameter defines how much memory an individual sort operator can consume. Don’t be confused by the “sort” moniker; the naming is historical – today in DB2 this memory area would be more appropriately named “working memory”. In BLU hash join, group-by, and general columnar vector buffering are considered individual sort operators / consumers.

An example is shown in the plan snippet above. Three operators in the plan each sort consumers allocate up to SORTHEAP worth of memory from the overall SHEAPTHRES_SHR. One key thing to be aware of is that SORTHEAP must be configured based on the expected query concurrency on the system,

Sort Memory Performance: Things to look for

- Ensure your sort configuration is within the range recommended by the BLU best practices
 - SHEAPTHRES_SHR at 40-50% of system memory
 - SORTHEAP between 1:5 and 1:20 of SHEAPTHRES_SHR
- Check that the workload has not exceeded the available SHEAPTHRES_SHR on the database
 - When SHEAPTHRES_SHR is exhausted it can lead to unpredictable spilling or even query failures in extreme circumstances
- Check for spilling caused by insufficient SORTHEAP
 - If SORTHEAP is constrained specific large 'sort' operators may be forced to spill
- Possible corrective steps
 - Examine / tune the configured SHEAPTHRES_SHR
 - Adjust the SORTHEAP ratio
 - Adjusting the default workload management in conjunction with this

When monitoring sort performance there are a few key things we want to keep an eye on:

- Ensure your sort memory is within the range recommended by the best practices (depending on whether you anticipate high vs. low query concurrency)
- Check that the workload has not exceeded the available SHEAPTHRES_SHR on the database; this can lead to unpredictable performance impacts
- Check for spilling that may be caused by the SORTHEAP size itself

Possible corrective actions include:

- Adjusting / increasing SHEAPTHRES_SHR
- Adjusting the SORTHEAP ratio to allow more or less memory per operator depending on expected query concurrency.
- Adjusting the default WLM concurrency based on the configured SORTHEAP:SHEAPTHRES_SHR ratio.

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) (DB2 10.5 Cancun+)
- Accessible at multiple levels of reporting
 - MON_GET_DATABASE (Database level)
 - MON_GET_PKG_CACHE_STMT (Query level) (DB2 10.5 Cancun+)
 - MON_GET_SERVICE_SUBCLASS_STATS (Subclass level) (DB2 10.5 Cancun+)
 - Others (DB2 10.5 Cancun+)
- Example:

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

Obtain current and
maximum sort usage for
the database

Sort memory usage can be monitored through several key metrics:

- SORT_SHRHEAP_ALLOCATED indicates the current amount of allocated sort memory,
- SORT_SHRHEAP_TOP indicates the high watermark for sort memory consumption indicating aggregate sort memory demands for the workload and can be compared with SHEAPTHRES_SHR for tuning purposes
- SORT_CONSUMER_SHRHEAP_TOP indicates the high watermark for the largest individual sort consumer and can be compared with SORTHEAP for tuning purposes.

These metrics are available at the database level, and as of DB2 Cancun have been expanded to all other standard reporting levels including connection, query, service class, workload, unit of work, etc.

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 (DB2 10.5 Cancun+)
- 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

In addition to sort memory metrics DB2 also exposes counts for individual sort consumers including:

- Total counts indicating how many sort operators have executed in aggregate across all queries for the database
- Post threshold and overflow counts which indicate respectively how many operator executions hit memory constraints and were throttled, and how many overflowed their available memory and were forced to spill to disk

Monitoring Sort Consumers

- Active sort consumer counts and high watermarks
 - ACTIVE_SORT_CONSUMERS / ACTIVE_SORT_CONSUMERS_TOP (DB2 10.5 Cancun+)
 - ACTIVE_HASH_GRPBY / ACTIVE_HASH_GRPBY_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 (DB2 10.5 Cancun+)
- 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

The counts also include metrics for the number of sort consumers that are currently active as well as high watermarks for concurrently active sort consumers.

As of DB2 Cancun the ACTIVE counts have been expanded to be reported consistently in all the standard monitoring interfaces include query level, connection, subclass, unit of work, etc.

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 sort_heap_top
from table(mon_get_database(-2)))
select sort_ops,
  overflows,
  (overflows * 100) / nullif(sort_ops,0) as pctoverflow,
  sort_heap_top
from ops;
```

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

SORT_OPS	OVERFLOWS	PCTOVERFLOW	SORT_HEAP_TOP
1200	300	25	12777216

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

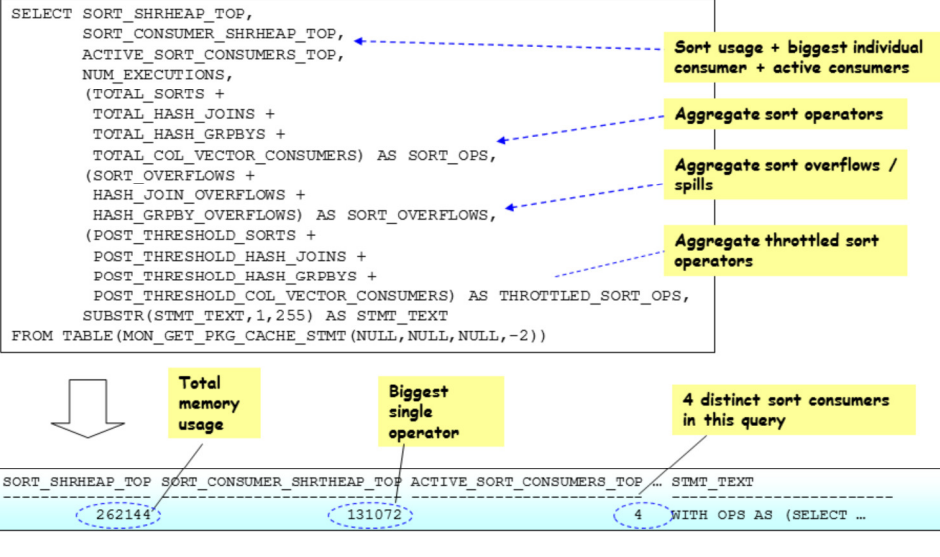
About 25% of our sort operations overflowed and spilled indicating some tuning may be worthwhile

The example above shows how you can monitor both the high watermark sort consumption on your database, and the percentage of operators that are spilling during query execution. Note that we are only including the largest sort consumers in this example as they are generally responsible for the vast majority of sort memory consumption.

This query will allow you to determine two things; first the percentage of operations that are spilling which will tell you how often spilling is impacting your queries and hence whether it's a potential cause of performance problems.

The high watermark will tell you whether the spilling is being triggered because we are exhausting the overall sort memory on the database. If not it's an indication that the SORTHEAP size is too constrained for some queries to execute. Both cases are an indication you might want to consider adjusting the SORTHEAP in conjunction with the WLM concurrency limit (we need to adjust both together because the larger our sortheap is, the fewer queries we can accommodate at a time).

Monitoring Query Sort Usage and Consumers



The example above shows how you can monitor sort memory usage and consumers per query.

- SORT_SHRHEAP_TOP** reports the total memory used for an execution of the query and indicates its impact on SHEAPTHRES_SHR
- SORT_CONSUMER_HEAP_TOP** indicates the largest single sort operator in the query and can be related to the configured SORTHEAP
- ACTIVE_SORT_CONSUMERS_TOP** indicates the total number of concurrent sort operators per execution of the query
- We also include the aggregate counts for all sort operators across all executions of the query as well as the related number of operators throttled, and that overflowed / spilled to disk.

When taken together these metrics give a good indication of the memory footprint of the query and whether it's being constrained due to memory limits in either SHEAPTHRES_SHR or SORTHEAP.

Monitoring Default Workload Management for Analytic Workloads





Resource Usage and Concurrency

- BLU philosophy is to leverage full machine resources (memory, CPU parallelism, etc.) in order to achieve order of magnitude performance benefits
- A consequence of this is that running too many columnar queries at a time can lead to significant resource competition and degrade performance
- Too many queries executing at a time can also have the potential to overload system resources and cause failures
- Some form of query concurrency management is needed to ensure orderly and efficient execution of columnar queries

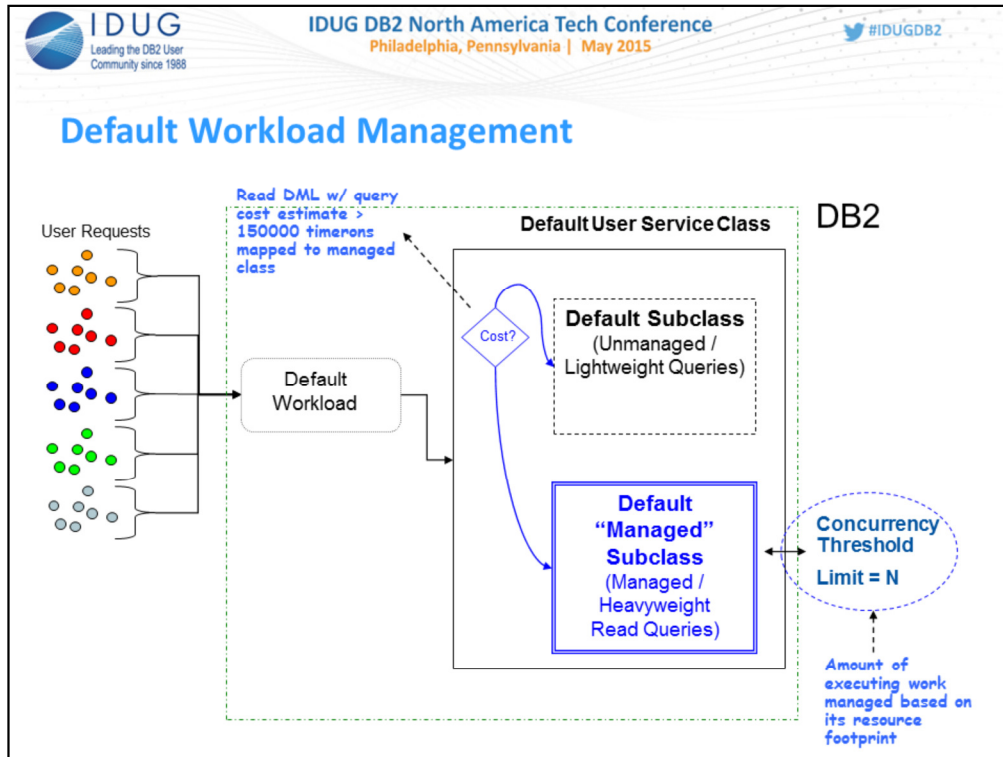
Part of the philosophy underlying the BLU Acceleration technology is that it tries to leverage the full machine resources in order to process as quickly / efficiently as possible and achieve its order of magnitude performance benefits. Basically BLU queries assume they have access to the full machine hardware and will take advantage of this to run fast. As a consequence though, columnar queries are relatively heavy on machine resource usage. Having too many columnar queries running at the same time on a system could theoretically lead to performance degradation, and even push the system beyond its resource limits causing query failures. Some form of control on query execution is required to ensure orderly execution of columnar queries.

Solution: Default Workload Management

- Allow unlimited query concurrency from a user perspective
- Internally manage query execution so that we execute only a limited number of queries at a time
 - ✓ Prevents overload / system remains robust in face of heavy workloads; SORTHEAP can be tuned with predictable concurrency limits in mind
 - ✓ Optimizes performance; ensures that when queries execute we have sufficient resources for them to complete quickly (allows more memory and more intra-query parallelism without causing spilling or overloading the processor run queues)

To avoid these performance and stability problems BLU Acceleration comes with a Default Workload Management scheme enabled. At a high level the concept is relatively straightforward; from a user perspective we allow unlimited query concurrency as usual. Internally though we manage query execution so that we execute only a limited number of queries at a time. This gives us two key benefits:

- First since we know the query concurrency limit we can configure an appropriate default SORTHEAP, and achieve predictable memory consumption which ensures stability even in the face of significant workload spikes.
- Second, allowing only a managed number of queries to execute a time reduces resource contention ensuring queries get the resources they need to execute quickly and generally avoid costly spilling. This means that your throughput will be improved and even considering queue time your queries will likely execute faster.



This slide show the default workload management for BLU in more detail:

- We split statements submitted to the system into two categories; unmanaged and managed.
- Read-only queries with an estimated cost of > 150000 timerons are mapped to the managed class.
- We apply a concurrency limit to the managed class which is computed at database creation time based on the machine hardware and CPU parallelism to ensure orderly execution of heavier weight analytic queries.
- Note that you can recompute this value if your system configuraiton changes by rerunning AUTOCONFIGURE.

The result is that :

- Heavy queries are queued and only N are executed concurrently allowing them to maximize their memory consumption / CPU parallelism and complete more quickly as well as preventing system overload.
- We maintain the response time of lightweight point queries by allowing them to bypass the control and avoid queuing behind large queries; these queries have a much smaller resource impact on the system so we let them pass through as quickly as possible.
- Other activities (DDL, Utilities, ETL) continue to be unmanaged. If managing these is desirable the WLM environment can be customized further.

Monitoring the Default Workload Management Configuration

- Examine the default estimated cost setting

```
SELECT VALUE1 AS EXPENSIVE_QUERY_COST  
FROM SYSCAT.WORKCLASSATTRIBUTES  
WHERE WORKCLASSNAME = 'SYSMANAGEDQUERIES' AND TYPE = 'TIMERONCOST'
```

```
EXPENSIVE_QUERY_COST  
-----  
+1.5000000000000000E+005
```

- Examine the default concurrency threshold settings

```
SELECT MAXVALUE, ENABLED  
FROM SYSCAT.THRESHOLDS  
WHERE THRESHOLDNAME = 'SYSDEFAULTCONCURRENT'
```

```
MAXVALUE          ENABLED  
-----  
11 Y
```

Above we show how to query the settings for the default workload management configuration:

- The default estimated cost setting can be queried from the SYSCAT.WORKCLASSATTRIBUTES catalog table
- The default concurrency threshold setting can be queried from the SYSCAT.THRESHOLDS catalog table

Monitoring the Default Workload Management Behavior

- Examine query cost estimates and wlm queue time

```
SELECT QUERY_COST_ESTIMATE,  
       WLM_QUEUE_TIME_TOTAL, ← Queue time  
       WLM_QUEUE_ASSIGNMENTS_TOTAL,  
       COORD_STMT_EXEC_TIME, ← Total elapsed time  
       NUM_EXECUTIONS,  
       SUBSTR(STMT_TEXT,1,255) AS STMT_TEXT  
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-2))
```

- List all active queries and their queuing state

```
SELECT SUBSTR(STMT_TEXT,1,255) AS STMT_TEXT,  
       ACTIVITY_STATE, ← Queuing state  
       ELAPSED_TIME_SEC  
FROM SYSIBMADM.MON_CURRENT_SQL
```




IDUG DB2 North America Tech Conference
Philadelphia, Pennsylvania | May 2015

 #IDUGDB2

Questions?

DB2 BLU Resources

- **BLU Best Practices Paper**
 - https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Wc9a068d7f6a6_4434_aece_0d297ea80ab1/page/Optimizing%20analytic%20workloads%20using%20DB2%2010.5%20with%20BLU%20Acceleration
- **DB2 Monitoring Enhancements for BLU Acceleration**
 - <http://www.ibm.com/developerworks/data/library/techarticle/dm-1407monitor-bluaccel/index.html>
- **Related IDUG sessions**
 - The Latest in Advanced Performance Diagnostics for SQL (D08)



David Kalmuk

IBM

dckalmuk@ca.ibm.com



D19

Monitoring BLU Acceleration In Depth

*Please fill out your session
evaluation before leaving!*



David Kalmuk is a Senior Architect and IBM STSM with responsibility for the Workload Management, Monitoring, and Systems areas of the DB2 for LUW product. David has contributed to the development of numerous technologies in DB2 over the years including the BLU Acceleration technology in DB2 10.5, Workload Management capabilities in DB2 10, the Performance Monitoring capabilities in DB2 9.7, as well as much of DB2's Processing and Communications architecture. He is currently leading new development efforts related to the BLU Acceleration technology. David has been a member of the DB2 team since 2000.