Scaling up BLU Acceleration with Consistent Performance in a High Concurrency Environment

Russ Perry

TRIDUG Mtg, December 9th

Originally Presented at:

Insight2015

The Premier Data and Analytics Conference

#ibminsight

Please Note

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- A Quick Review of the BLU Acceleration Technology
- Handling High Concurrency Environments
- Tuning the Default Workload Management
- Performance Comparison
- Deploying an Optimized Workload Management Configuration

A Quick Review of the BLU Acceleration Technology

Insight 2015

The Premier Data and Analytics Conference

#ibminsight

DB2 with BLU Acceleration
Rich capability integrated with IBM DB2 10.5

Fast Answers. Simply Delivered.

- In-memory columnar analytic database
- Multiplatform: Supports AIX, Linux, Linux on System Z, Windows
- Ready for Analytics: Cloud, On premises, SAP, Cognos, and more
- Agile warehousing via dashDB

BLU Acceleration

Analyze more data faster and more efficiently

The DB2 with BLU Acceleration Technology

#ibminsight

Dynamic In-Memory

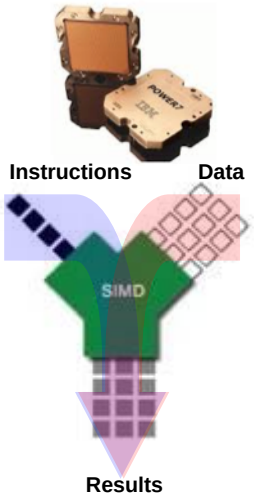
In-memory columnar processing with dynamic movement of data from storage

Actionable Compression

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

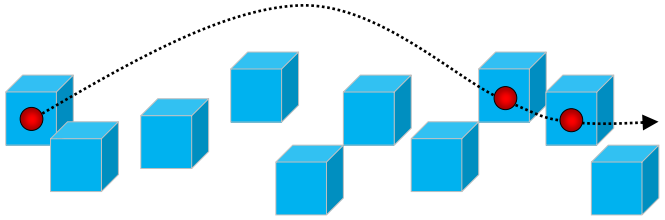
Parallel Vector Processing

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



Data Skipping

Skips unnecessary processing of irrelevant data



Super Fast, Super Easy — *Create, Load and Go!*

No Indexes, No Aggregates, No Tuning, No SQL changes, No schema changes

Simplification of Analytic Operations

Traditional Warehouse
Database Design and Tuning

1. Decide on partition strategies
2. Select Compression Strategy
3. Create Table
4. Load data
5. Create Auxiliary Performance Structures
   - Materialized views
   - Create indexes
     - B+ indexes
     - Bitmap indexes
6. Tune memory
7. Tune I/O
8. Add Optimizer hints
9. Statistics collection

Repeat

AFTER
DB2 with BLU Acceleration

1. Create Table
2. Load data

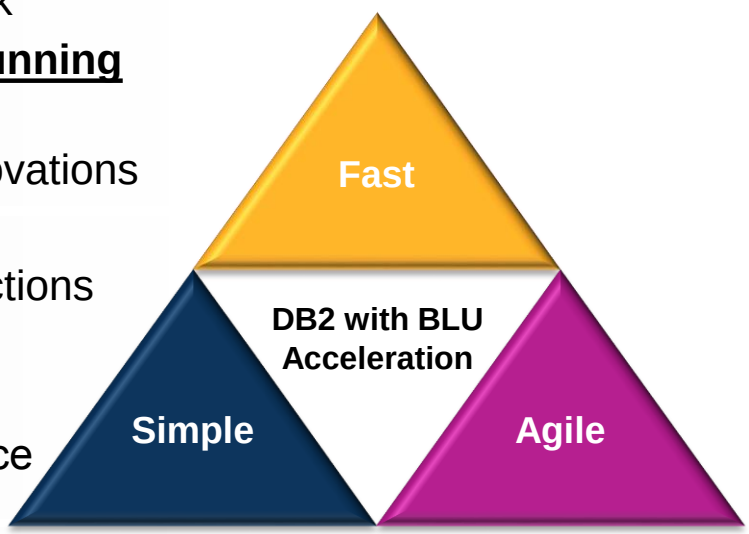
Create
Load
GO!

The Benefits of DB2 with BLU Acceleration for Analytics

Fast Answers. Simply Delivered.

- **Instant insight from real-time operational data** for growing revenue, reducing cost and lowering risk
- **35x to 73x faster analytics, with some queries running more than 1400x faster[1,2]**
- Next generation in-memory with IBM Research innovations

- **Simplified IT landscape** with reporting and transactions in the same system
- No need for indexes, aggregates or tuning
- Operational simplicity with "load and go" performance



- Available for on-premises or via the cloud
- ***"In one of our largest customer databases, we saw a compression ranging from 7x to 20x as compared to the uncompressed tables "*** **- Mike Petkau, Director of Database Architecture & Administration, TMW Systems**
- **Simple, low-risk upgrade** from Oracle Database

1 Based on internal IBM testing of sample client analytic workloads comparing queries accessing row-based tables on DB2 10.1 vs. columnar tables on DB2 10.5 with BLU Acceleration. Performance improvement figures are cumulative of all queries in the workload. Individual results will vary depending on individual workloads, configurations and conditions.

2 Based on internal IBM tests of analytic workloads comparing queries accessing row-based tables on DB2 10.1 vs. columnar tables on DB2 10.5 with BLU Acceleration. Results not typical. Individual results will vary depending on individual workloads, configurations and conditions, including size and content of the table, and number of elements being queried from a given table..

Handling High Concurrency Workloads

Insight 2015

The Premier Data and Analytics Conference

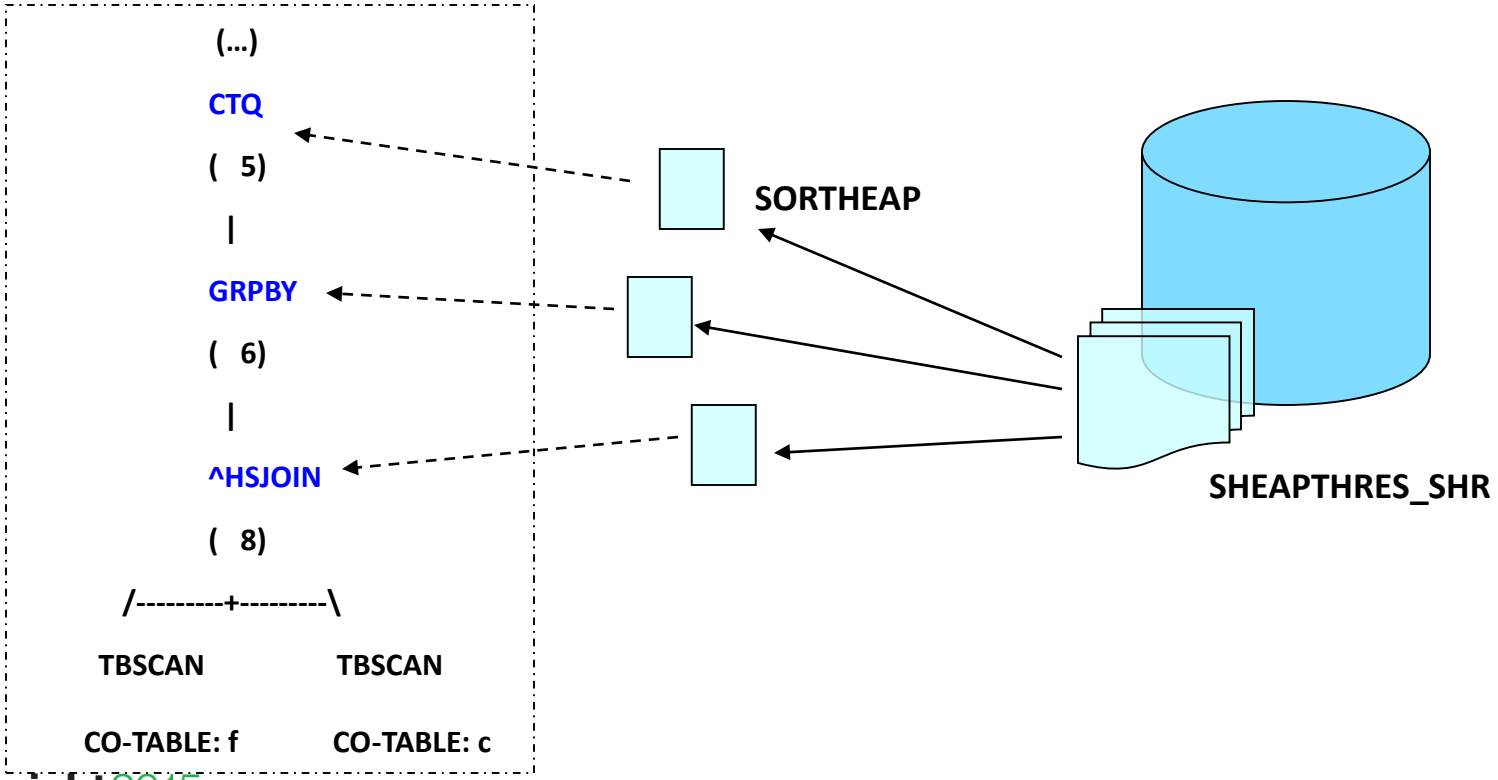
#ibminsight

Resource Usage and Concurrency

- BLU philosophy is to leverage full machine resources (memory, CPU capacity) 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 admission control is needed to ensure orderly and efficient execution of columnar queries(!)

More Detail: Memory Considerations

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



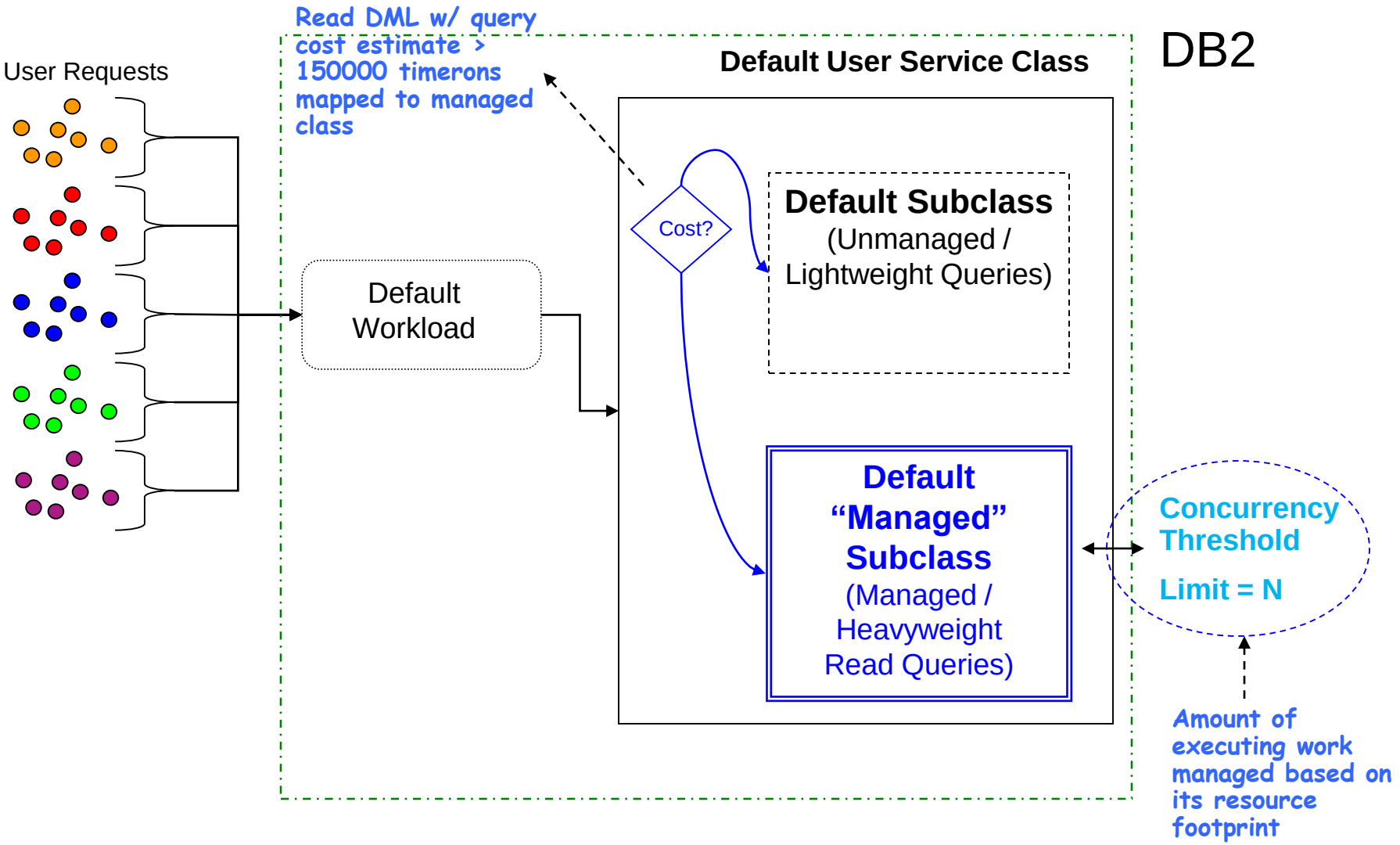
Memory and Concurrency without Workload Management

- Each query consumes 1-3x short heaps for sort, join, buffer operations.
- Concurrent queries multiply memory requirements.
- As multiple queries execute, less memory is available to each operation.
- Operation processing no longer fits in available memory.
- With sort memory exhausted, intermediate results cache to disk, causing ‘spilling’ to occur (significant negative performance impact!)
- If concurrency exceeds level configured for SORTHEAP we can exhaust SHEAPTHRES_SHR. This will cause queries to fail.
- **Result is that allowing unmanaged admission can significantly degrade performance and even put system stability at risk!**

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)

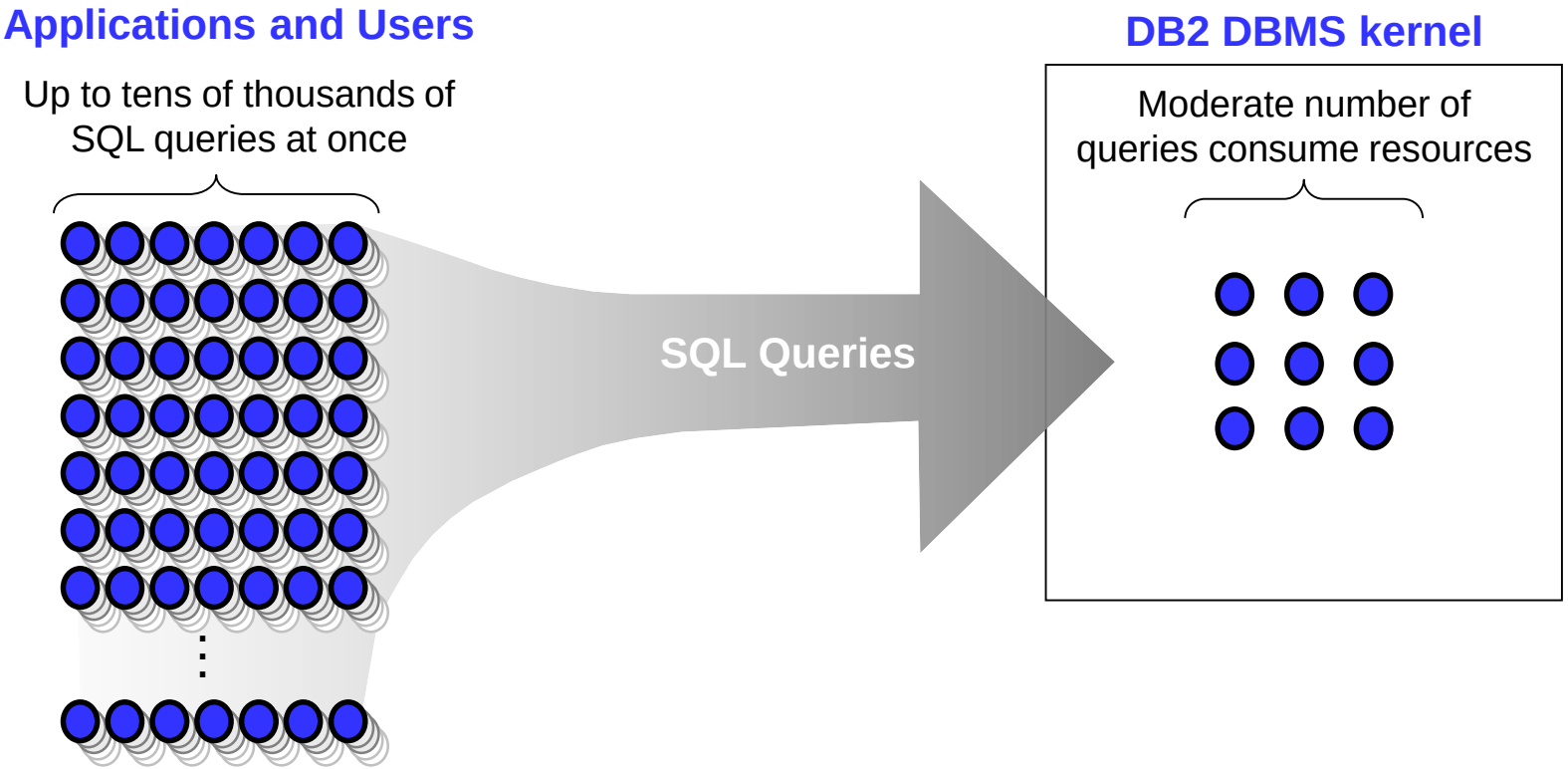
Default Workload Management



Default Workload Management Explained

- Lightweight queries can enter the system freely
  - Minimizes latency for short queries (< 150000 timerons)
  - Ensures small queries aren't queued behind long ones
- Limited number of heavyweight queries allowed to execute at a time
  - Controls impact of heavyweight queries (>= 150000 timerons)
  - Limit calculated based on hardware during database deployment
- Non-DML activities continue to run unmanaged by default
  - Current Default Workload Management is focused specifically on the impacts of large columnar analytic queries
- End result is simple and effective (if not completely optimal) workload management out of the box

Result: Unlimited Concurrency with "Automatic" WLM



Tuning the Default Workload Management

Insight 2015

The Premier Data and Analytics Conference

#ibminsight

Opportunities for Tuning

- Default Workload Management in BLU is a relatively “blunt” instrument
- Goal is to provide system stability and reasonable performance out of the box
- Better than an unmanaged environment but lots of opportunity to optimize behavior with a little bit of tuning if desired
- This section will work through some simple steps you can take to optimize BLU execution for your environment

Querying the Default Workload Management Settings

- Examine the default estimated cost setting

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

```
EXPENSIVE_QUERY_COST
------------------------
  +1.50000000000000E+005
```

- Examine the default concurrency threshold settings

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

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

Tuning the Managed Query Timeron Cost

- Intention behind the default query cost level is to identify higher cost queries that both consume more resources and are less sensitive to response times
- Default value was selected based on a suite of representative workloads
  - Intended as a “best fit” but will not be optimal for all environments
  - Timeron cost is also not a “perfect” metric
- This section will show how the value can be tailored more specifically for your own environment

"How many queries are above / below the cost line?"

```
with smallcost as
(
  select sum(num_coord_exec) as smallcost from
  table(mon_get_pkg_cache_stmt(null,null,null,-2))
  where query_cost_estimate < 150000
),
smalltime as
(
  select sum(num_coord_exec) as smalltime from
  table(mon_get_pkg_cache_stmt(null,null,null,-2))
  where (coord_stmt_exec_time / nullif(num_coord_exec,0)) < 30
),
total as
(
   select sum(num_coord_exec) as total
   from table(mon_get_pkg_cache_stmt(null,null,null,-2))
)
select (smallcost * 100) / total as pctsmallcost,
       (smalltime * 100) / total as pctsmalltime
from smallcost, smalltime, total;
```

Count of queries below timeron threshold

Count of queries that execute for less than 30 seconds ("short" queries)

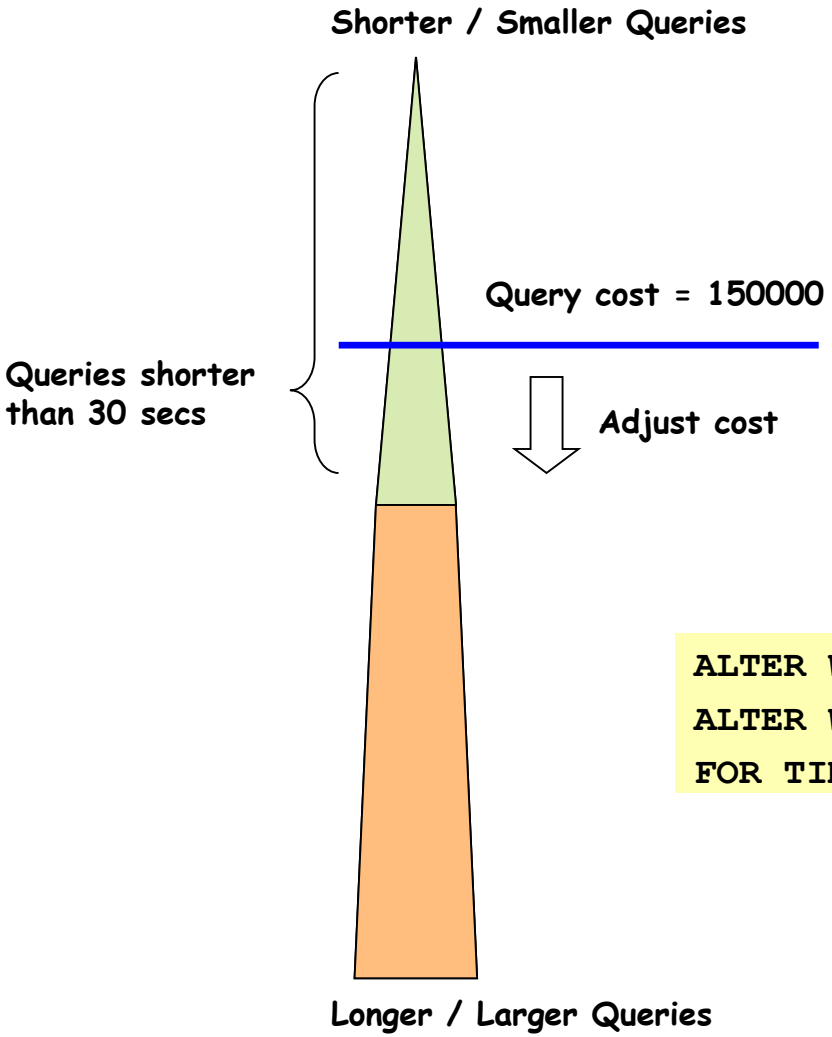
Total number of query executions on the system

PCTSMALLCOST	PCTSMALLTIME
30	50

About 30% of our queries are running "unmanaged" but 50% of our queries are "short running"

#ibminsight

Adjusting the Timeron Cost Threshold



Adjust the cost so that queries that run for less than 30 seconds are not controlled

```
ALTER WORK CLASS SET SYSDEFAULTUSERWCS
ALTER WORK CLASS SYSMANAGEDQUERIES
FOR TIMERONCOST FROM 200000 TO UNBOUNDED
```

Increased from 150000

Tuning the SORTHEAP parameter

- Having an appropriate SORTHEAP setting is critical to getting optimal performance out of your BLU environment.
  - SHEAPTHRES_SHR set based on physical RAM on your system (anywhere from 1/3 to 1/2 is appropriate for BLU)
  - **SORTHEAP on the other hand depends on your workload**
- Default settings are computed based on general assumptions about memory availability and concurrency on your system
  - Reasonable but not always optimal
- With a bit of tuning you can get more bang for your buck
- So how do we go about selecting our SORTHEAP?

“What percentage of my ‘sort’ operations are 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
from table(mon_get_database(-2)))
select sort_ops,
       overflows,
       (overflows * 100) / nullif(sort_ops,0) as pctoverflow
       from ops;
```

Extract total sorts and sort overflows

SORT_OPS	OVERFLOWS	PCTOVERFLOW
1200	300	25

About 25% of our sort operations overflowed and spilled.

```
db2 update db cfg using SORTHEAP 2097152
```

Increasing SORTHEAP will reduce spilling but we can't make this change in a vacuum. Need to consider the impacts on concurrency.

Tuning the Default Concurrency Level

- The default concurrency limit for the managed subclass is computed based on system hardware and with awareness of the default recommended SORTHEAP / SHEAPTHRES_SHR ratio of 1:20.
- If we increase our SORTHEAP we need to also adjust the concurrency level accordingly to avoid overloading SHEAPTHRES_SHR and causing query failures
- This next example will walk through how to compute and perform these adjustments

Tuning the Default Concurrency Level


```
select avg(sort_shrheap_top) as avg_mem
from
table (mon_get_pkg_cache_stmt(null,null,null,-2))
where query_cost_estimate > 150000;
```

```
AVG_MEM
-----------
     132568
```

On average my heavier queries consume 132568 pages (4K) of sort memory

As a rule of thumb start with a concurrency limit of (SHEAPTHRES_SHR / AVG_MEM) * 0.75

To be safe leave a 25% buffer for workload fluctuations

```
ALTER THRESHOLD SYSDEFAULTCONCURRENT
WHEN CONCURRENTDBCOORDACTIVITIES > <N>
CONTINUE
```

A Bit More Fine Tuning

1. Look at the actual overall sort consumption vs. SHEAPTHRES_SHR

```
select sort_shrheap_top from table(mon_get_database(-2))
```

```
SORT_SHRHEAP_TOP
----------------
         6553600
```

Reported in 4K pages = 25GB HWM

2. Compute M = ((SHEAPTHRES_SHR / SORT_SHRHEAP_TOP) – 1)

3. Consider increasing concurrency threshold further if M is large percentage

```
ALTER THRESHOLD SYSDEFAULTCONCURRENT
WHEN CONCURRENTDBCOORDACTIVITIES > <N *(1+(M/2))>
CONTINUE
```

- Final notes:
  - Always leave some room for workload variation
  - Always validate new settings in your test environment first!

Performance Comparison

Insight 2015

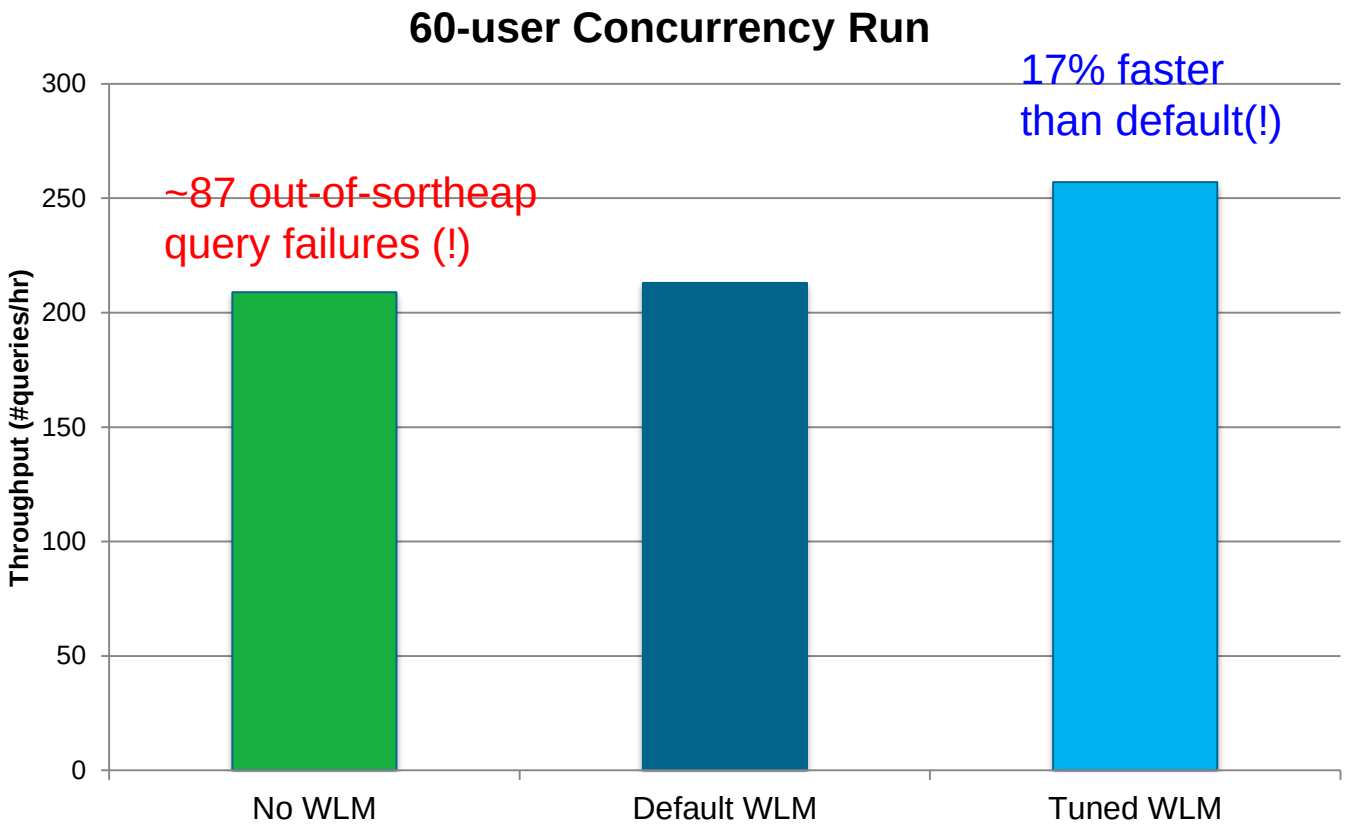
The Premier Data and Analytics Conference

#ibminsight

Test Environment

- **Platform:** Softlayer bare metal system
- **Database size**: 4 TB
- **Workload**: BD Insights
  - IBM developed benchmark which simulates real-world business analytics
  - Consists of 30 queries which can be divided into two categories
    - Queries which represent typical analytic queries used to generate sales reports
    - Queries which represent hand-crafted, deep-dive analytic queries created by sales analysts
  - 60 user-streams

Performance Results



	No WLM	Default WLM	Tuned WLM
CPU utilization	76%	73%	87%

Better CPU Utilization (!)

#ibminsight

Key Takeaways

- BLU Acceleration allows you to achieve an order of magnitude speedup for your analytic workloads
- Controlling admission of heavyweight queries is a must to maintain a stable system in the face of a highly concurrent enterprise scale workload
- We introduced default workload management with BLU to protect your system from being overloaded in the face of high concurrency
- You can use the techniques in this session to fine tune your configuration to squeeze even more performance out of your environment

Notices and Disclaimers

Notices and Disclaimers (con't)

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. IBM EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

- IBM, the IBM logo, ibm.com, Aspera®, Bluemix, Blueworks Live, CICS, Clearcase, Cognos®, DOORS®, Emptoris®, Enterprise Document Management System™, FASP®, FileNet®, Global Business Services ®, Global Technology Services ®, IBM ExperienceOne™, IBM SmartCloud®, IBM Social Business®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, Smarter Commerce®, SoDA, SPSS, Sterling Commerce®, StoredIQ, Tealeaf®, Tivoli®, Trusteer®, Unica®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

Thank You

Insight2015

The Premier Data and Analytics Conference

#ibminsight

Supplementary: Deploying an Optimized Workload Management Configuration

Insight 2015

The Premier Data and Analytics Conference

#ibminsight

Taking things a step further

- We've discussed a number of ways to tune the BLU runtime environment by adjusting the default workload management settings.

- Another path that can be taken is to deploy a more customized workload management environment that can offer finer grained control and hence better performance

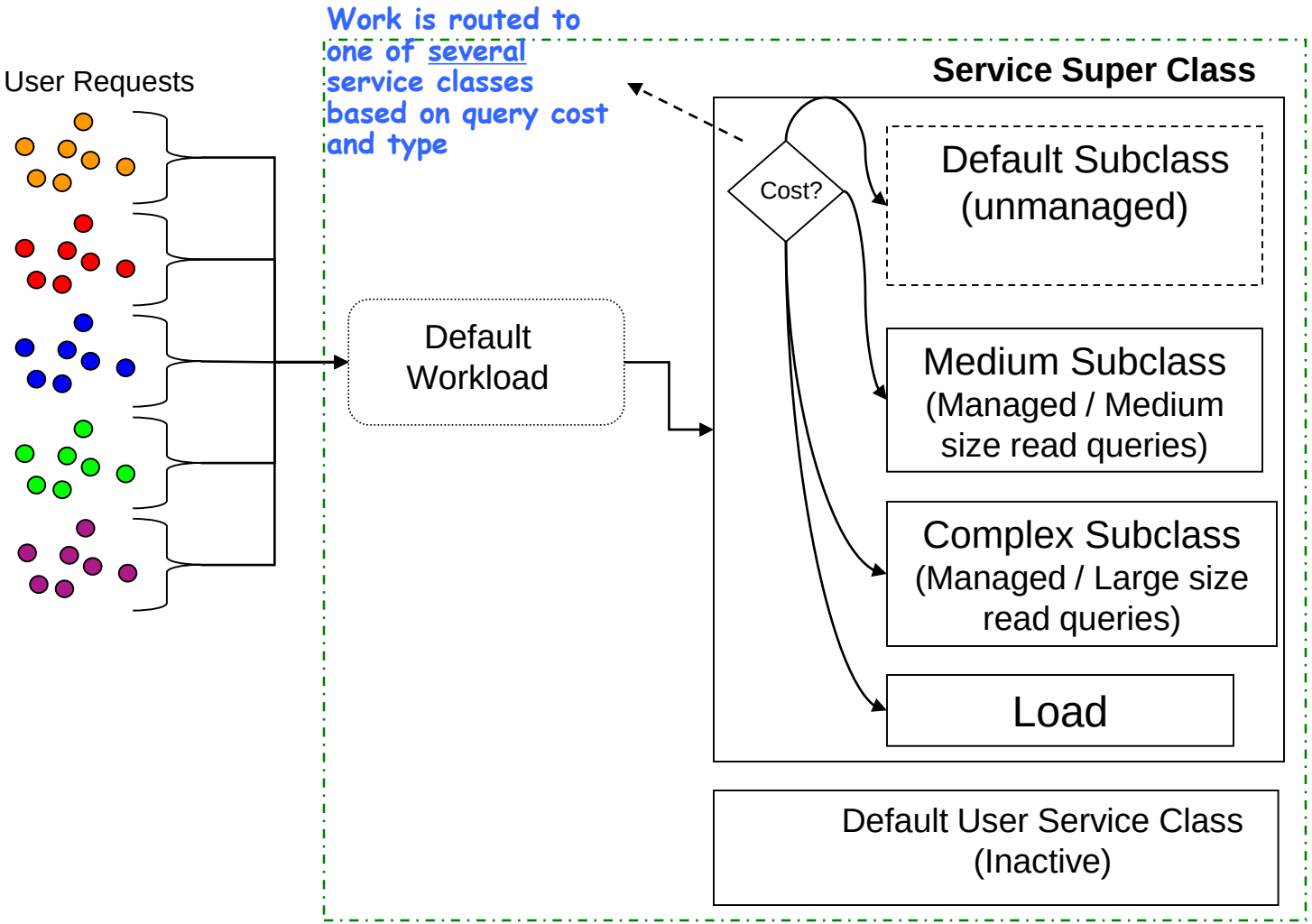
- Goal is to tailor admission and runtime control more specifically to different classes of work and achieve resulting efficiency benefits / avoid inefficiencies
  - For example lighter queries queued behind heavyweight queries

Taking things a step further (cont'd)

- Recommended for environments where
  - Workload is very dynamic with a wide range of query types
  - Query concurrency is relatively high
  - Query workload must coexist with potentially heavy ingest jobs
  - The system needs to meet specific performance goals

- This section will explore the deployment of this type of environment in detail

Optimized WLM Configuration



Key Points

- Core DML workload is divided into several different “lanes” based on query cost
  - Group queries with like response times together and assign them a proportion of system resources
  - Admit queries to each lane based on their allocated resources
  - Ensure reliable throughput rate for different classes of queries via fast / medium / slow lanes
  - Applies to both read + write activities in the system
- Load concurrency explicitly controlled
- DDL and most administrative tasks run unmanaged
- For stored procedures, nested SQL statements are managed individually

Creating the WLM Environment

1. Setting up service classes and thresholds

Create service classes

```
create service class MASTER
create service class COMPLEX under MASTER
create service class MEDIUM under MASTER
create service class LOAD under MASTER

alter workload SYSDEFAULTUSERWORKLOAD service class MASTER
```

Create query concurrency thresholds

We will come back to the query concurrency limits

WLM Best Practices default template limit

```
create threshold MEDIUM_DML_CONCURRENCY
    for service class MEDIUM_DML under MASTER activities enforcement database enable
    when concurrentdbcoordactivities > <X> and queued activities unbounded continue
create threshold COMPLEX_DML_CONCURRENCY
    for service class COMPLEX_DML under MASTER activities enforcement database enable
    when concurrentdbcoordactivities > <Y> and queued activities unbounded continue
create threshold LOAD_CONCURRENCY
    for service class LOAD under MASTER activities enforcement database enable
    when concurrentdbcoordactivities > 4 and queued activities unbounded continue
```

Creating the WLM Environment

2. Setting up the mappings

Work classification

Timeron ranges

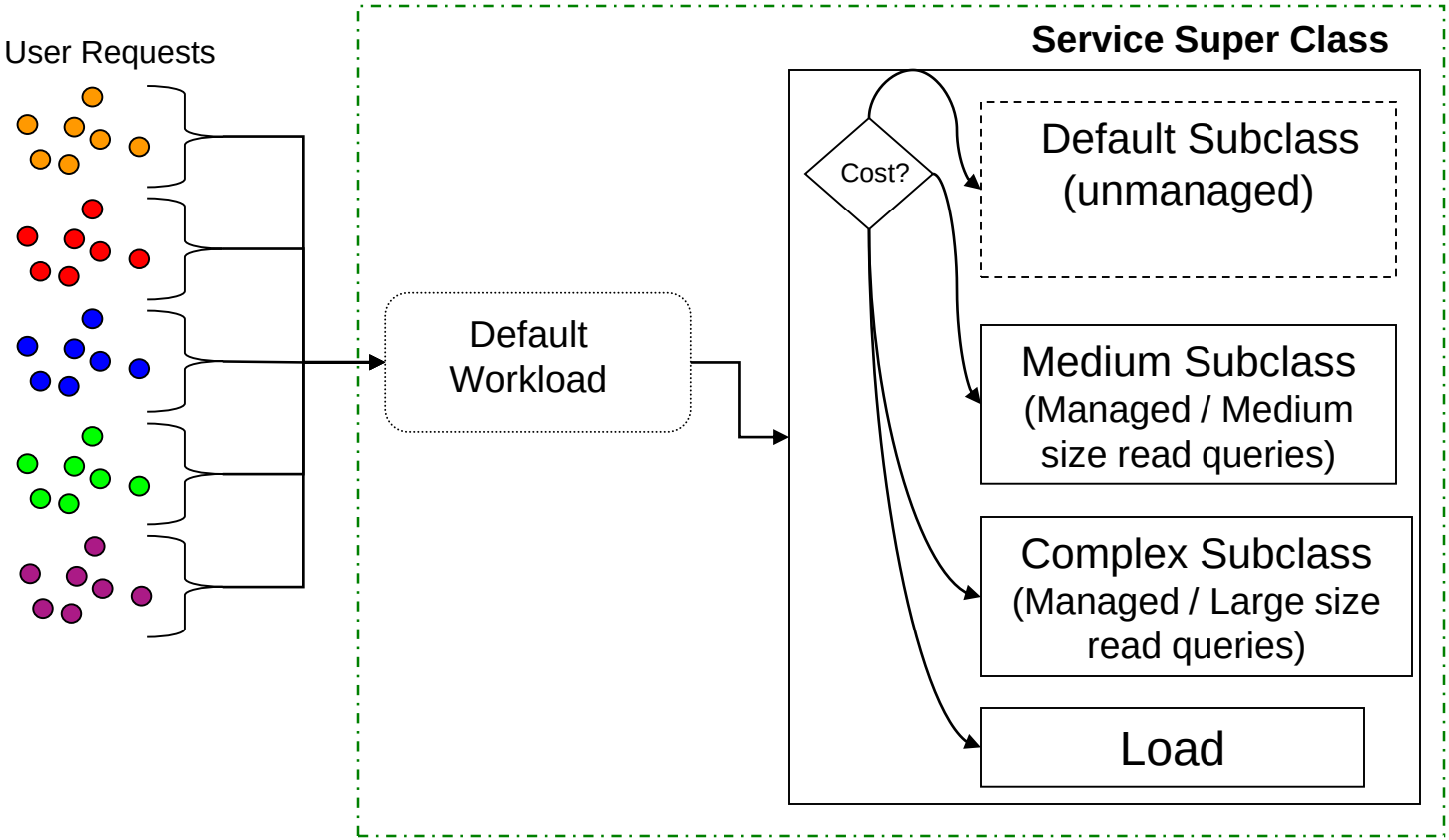
```
create work class set WORK_CLASSES (
    work class medium_cost work type read dml
     for timeroncost from 150000.0 to 5000000.0,
    work class complex_cost work type read dml
     for timeroncost from 5000000.0 to unbounded,
     work class load work type load )
```

Mapping based on classification

```
create work action set WORK_ACTIONS
for service class MASTER using work class set WORK_CLASSES
(
    work action map_medium_cost on work class medium_cost
     map activity without nested to MEDIUM,
    work action map_complex_cost on work class complex_cost
     map activity without nested to COMPLEX,
    work action map_load on work class load
     map activity without nested to LOAD
)
```

Applying Admission Control

1. **Figure out what percentage of SHEAPTHRES_SHR to assign to each service class**



Applying Admission Control

2. Figure out query sort consumption per query per service class

```
select avg(sort_shrheap_top) as avg_mem_medium
from
table (mon_get_pkg_cache_stmt(null,null,null,-2))
where query_cost_estimate > 150000 and
query_cost_estimate < 500000;
```

← Medium queries

```
select avg(sort_shrheap_top) as avg_mem_complex
from
table (mon_get_pkg_cache_stmt(null,null,null,-2))
where query_cost_estimate >= 500000;
```

← Complex queries

3. Compute query concurrency each service class can accommodate within their portion of SHEAPTHRES_SHR

- N_{medium} = (SHEAPTHRES_SHR / AVG_MEM_MEDIUM) * 0.35 * 0.75
- $N_{complex}$ = (SHEAPTHRES_SHR / AVG_MEM_COMPLEX) * 0.55 * 0.75

To be safe leave a 25% buffer for workload fluctuations

Applying Admission Control

4. **Set the concurrency limits for each service class**

```
ALTER THRESHOLD MEDIUM_DML_CONCURRENCY WHEN
CONCURRENTDBCOORDACTIVITIES > <N_medium> CONTINUE
```

```
ALTER THRESHOLD COMPLEX_DML_CONCURRENCY WHEN
CONCURRENTDBCOORDACTIVITIES > <N_complex> CONTINUE
```

5. **Monitor sort consumption in each service class**

```
select service_subclass_name, sort_shrheap_top from
table(mon_get_service_subclass('MASTER',NULL,-2)
```

Applying Admission Control

6. Compute excess capacity for each service class

- M_{medium} = ((SHEAPTHRES_SHR * 0.35 / *<sort_shrheap_top$_{medium}$>*) -1
- $M_{complex}$ = ((SHEAPTHRES_SHR * 0.55 / *<sort_shrheap_top$_{complex}$>*) -1

7. Fine tune concurrency limits

```
ALTER THRESHOLD MEDIUM_DML_CONCURRENCY WHEN
CONCURRENTDBCOORDACTIVITIES > <N_medium * (1+(M_medium/2)> CONTINUE
```

```
ALTER THRESHOLD COMPLEX_DML_CONCURRENCY WHEN
CONCURRENTDBCOORDACTIVITIES > <N_complex * (1+(M_complex/2)> CONTINUE
```

- Remember above that we leave some room for workload variation

Benefits of the Optimized Configuration

- More granular query classes mean more tailored / optimal performance
  - Greater concurrency possible for mid-range queries leading to improved response times
  - Lower concurrency for the most heavyweight queries minimizes their impact on the performance of lightweight and mid-range queries.
  - Less variation in memory consumption within each class means we can apply more efficient limits
- Granularity also allows more detailed workload monitoring
  - Each query class can be monitored and configured separately
- Including management of Load type activities provides more predictable performance when operational updates are expected on a live system