#IDUGDb2

Adaptive Workload Management Overview and Best Practices

David Kalmuk

IBM

Session code: 6048

Platform: Db2 for LUW

Agenda

- Background – Concurrency and Workload Management
- Adaptive Workload Management Technology
- Prioritization using Adaptive Workload Management
- Monitoring + Tuning the Adaptive Workload Manager
- Closing Thoughts

Concurrency and Workload Management

Modern data warehousing systems are expected to handle a wide variety of workloads while remaining responsive

Workload composition is typically highly variable and will include a mix of point queries, interactive reporting, heavy analytics, continuous data ingest

Supporting high concurrency is a key user demand making effective workload management critical in these types of systems

The Goals for Workload Management in a Database System

Ensure System Stability and Responsiveness

Don't overcommit the system but ensure it's well utilized

Schedule jobs to ensure fairness and appropriate responsiveness

Workload Prioritization / Isolation

Allow resources to be subdivided between workloads for prioritization / isolation purposes

Workload Governance and Monitoring

Define rules to govern workloads / detect and abort rogue jobs

Perform workload level monitoring

Db2's "Traditional" Workload Manager

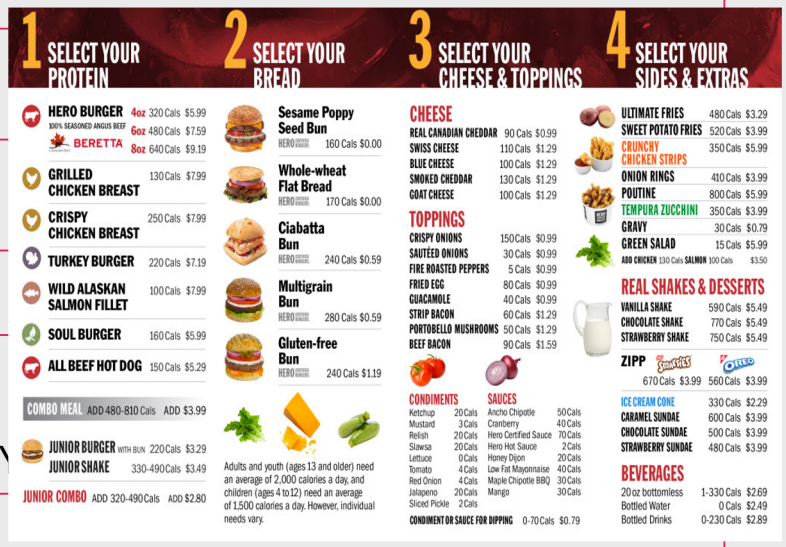
A mature and highly customizable set of capabilities for workload management
- *Classification, mapping, concurrency control, governance thresholds, resource control*

View it as a framework with a comprehensive set of 'tools' for DIY workload management
- *Construct nearly any workload management setup you can imagine*

WLM Best Practices provide a template for building a recommended configurations for managing a warehouse environment
- *Further refinements add scenarios for isolation, prioritization, production shifts*

The Db2 Workload Manager Menu

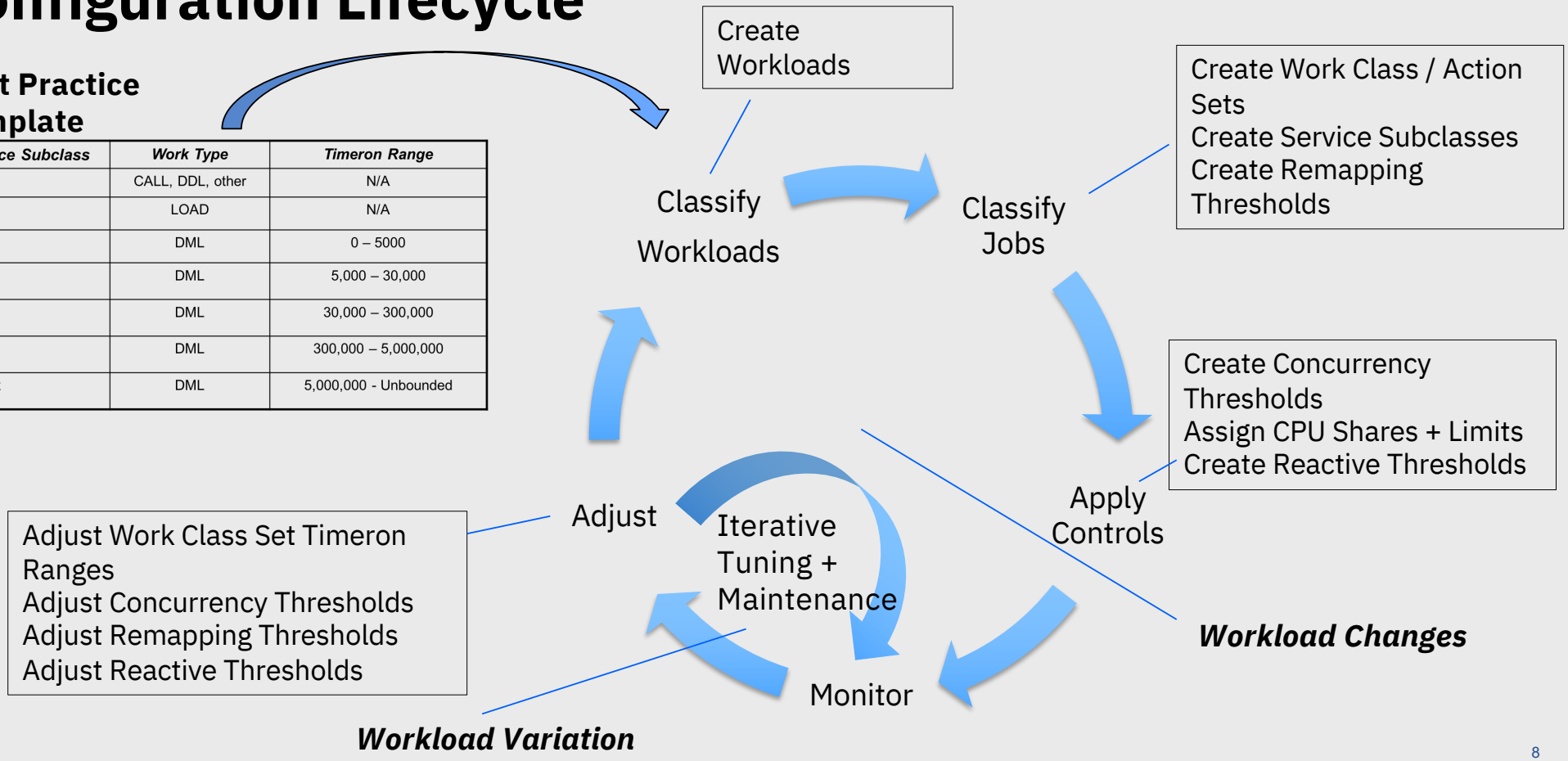
Domain	Options
Workload Classification	WORKLOAD
Workload Prioritization	SERVICE CLASS
Job Classification	WORK CLASS / WORK ACTION SET Remapping THRESHOLD
Job Prioritization	SERVICE SUBCLASS
Admission + Resource Control	Concurrency THRESHOLD CPU LIMIT + SHARE PREFETCH + BUFFERPOOL PRIORITY
Governance	Predictive + Reactive THRESHOLD
Monitoring	SQL Functions (Workload, Service class) Event Monitors (Statistics, Activity)



The Workload Management Configuration Lifecycle

Best Practice Template

Service Subclass	*Work Type*	*Timeron Range*
Default	CALL, DDL, other	N/A
LOAD	LOAD	N/A
Trivial	DML	0 – 5000
Minor	DML	5,000 – 30,000
Simple	DML	30,000 – 300,000
Medium	DML	300,000 – 5,000,000
Complex	DML	5,000,000 - Unbounded



Query Costs and Concurrency Limits

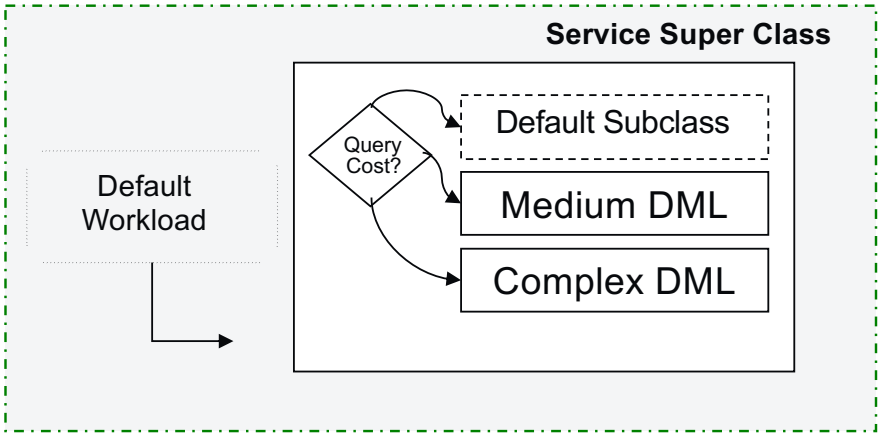
- Maintaining this type of WLM configuration involves manual processes that can be fairly labor intensive
- The underlying reason is that both **query cost ranges** and **concurrency limits** are lower level and **indirect** controls over what we are actually trying to manage
  - **Query cost** = Use estimate of query complexity to differentiate based on **response time**
  - **Concurrency limit** = Control **resource consumption** for jobs in a particular class via fixed limit
- Most database vendors use similar techniques with similar complexities - **why?**
  - Eg. **"Concurrency thresholds", "Throttles", "Slots", "Queues", "Memory limits", etc.**
- The reason is that predicting **response times** and **resource consumption** accurately enough to be actionable is **hard**!
- **Fixed limits** are **much easier** to implement from a technology perspective.

The Challenge of Modern Analytic Workloads

- Diverse range of jobs from miniscule point lookups to massive analytic queries
- Highly dynamic workloads combining high volumes of operational point queries and concurrent complex analytics of varying shapes and sizes

- With in-memory column store technologies fixed resources like memory become the limiting factor vs. CPU
- Much less forgiving if system gets overcommitted; **failure** not **slowdown**
- For these types of workloads *configurations based on fixed limits are necessarily sub-optimal and difficult to tune*

The challenge with query costs and concurrency limits…



0 < Timeron Cost <= ??	Concurrency Limit = ??
?? < Timeron Cost <= ??	Concurrency Limit = ??
?? < Timeron Cost <= ??	Concurrency Limit = ??

For response time < 30 seconds target 30% resources

For response time < 600 seconds target 30% resources

For response time > 600 seconds target 40% resources

Indirect controls; onus is on the user to derive, apply, and adjust to maintain appropriate fixed limits.

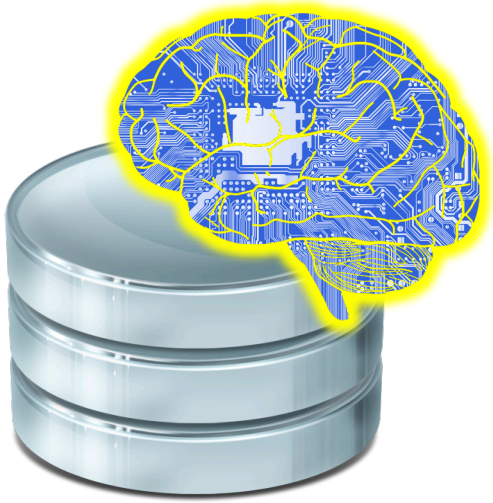
Adaptive Workload Management Technology

Db2's Adaptive Workload Management Technology

Admission management based on query resource requirements instead of fixed limits!

- Adjusts concurrency implicitly based on workload without manual tuning
- Intelligent job scheduling makes more efficient use of system resources
- Resources considered
  - Query sort memory requirements (working memory)
  - Number of parallel agents required for processing
- Available on Db2 Warehouse on Cloud, Db2 Warehouse, IIAS, and Db2 11.5.4
  - Currently limited to DB2_WORKLOAD=ANALYTICS configurations

Adaptive Workload Management Goals



- Deliver true automatic workload management out of the box with zero tuning
- Removes need to configure + tune fixed concurrency limits
- Improved stability and performance
- Enables much simpler and more powerful admission models

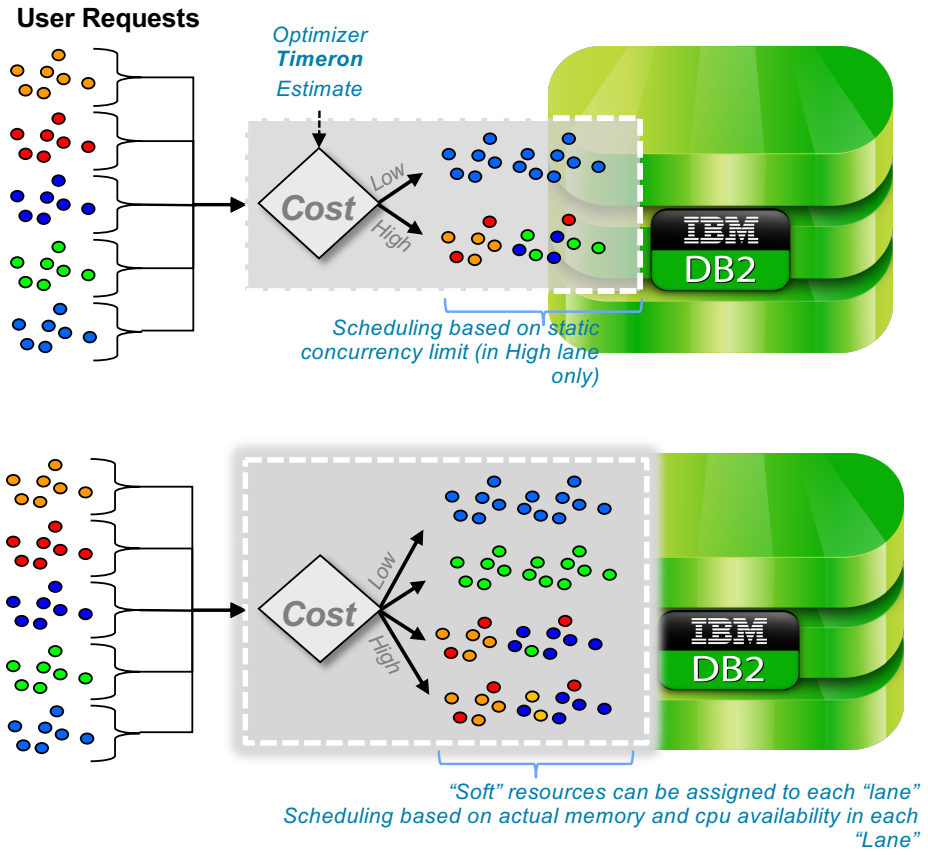
Intelligent Job Scheduling

Traditional WLM

- Cost evaluation includes only “timeron” estimate
- Open ended (no feedback)
- Scheduling based on static concurrency threshold

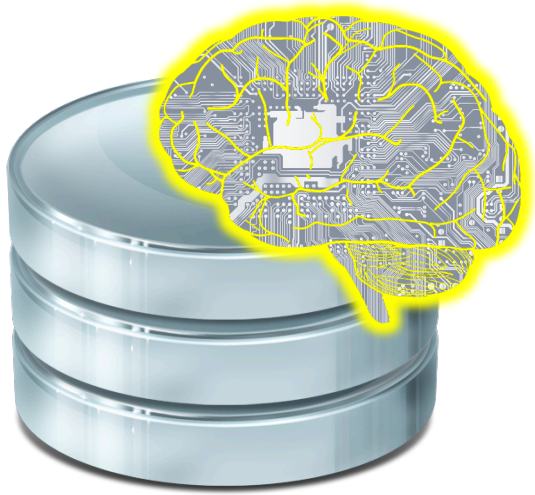
Adaptive WLM

- Cost evaluation includes memory & cpu load & time duration
- Incorporates historical feedback based on past executions
- Scheduling based on dynamic view of resource availability in each “lane”
- Expected benefits
  - ***Improved robustness under high load***
  - ***Improved SLA achievement***
  - ***Improved overall resource efficiency & throughput***



Prioritization using Adaptive Workload Management

Substantially Simplified Workload Management



Create a service superclass pre-configured for one of three defined workload types

- **`INTERACTIVE`** for response sensitive jobs
- **`BATCH`** for longer running jobs
- **`MIXED`** for workloads that run a combination of both

Assign a resource share to the service class

- Specifies the proportion of database resources this service class is entitled to
- Shares can be either `HARD` or `SOFT` for more flexible vs strict resource assignment

The system does the rest!

An Example:

- Divide the database resources into 3 distinct workloads
  - **High priority interactive reports** that require a fast response
  - **ETL jobs** that require sufficient resources to complete within an assigned window of time
  - **Other general purpose tasks** that don't fit into the above categories

Step 1:

Create service classes

- Define the workload type
- Assign resource shares

Step 2:

Create workloads

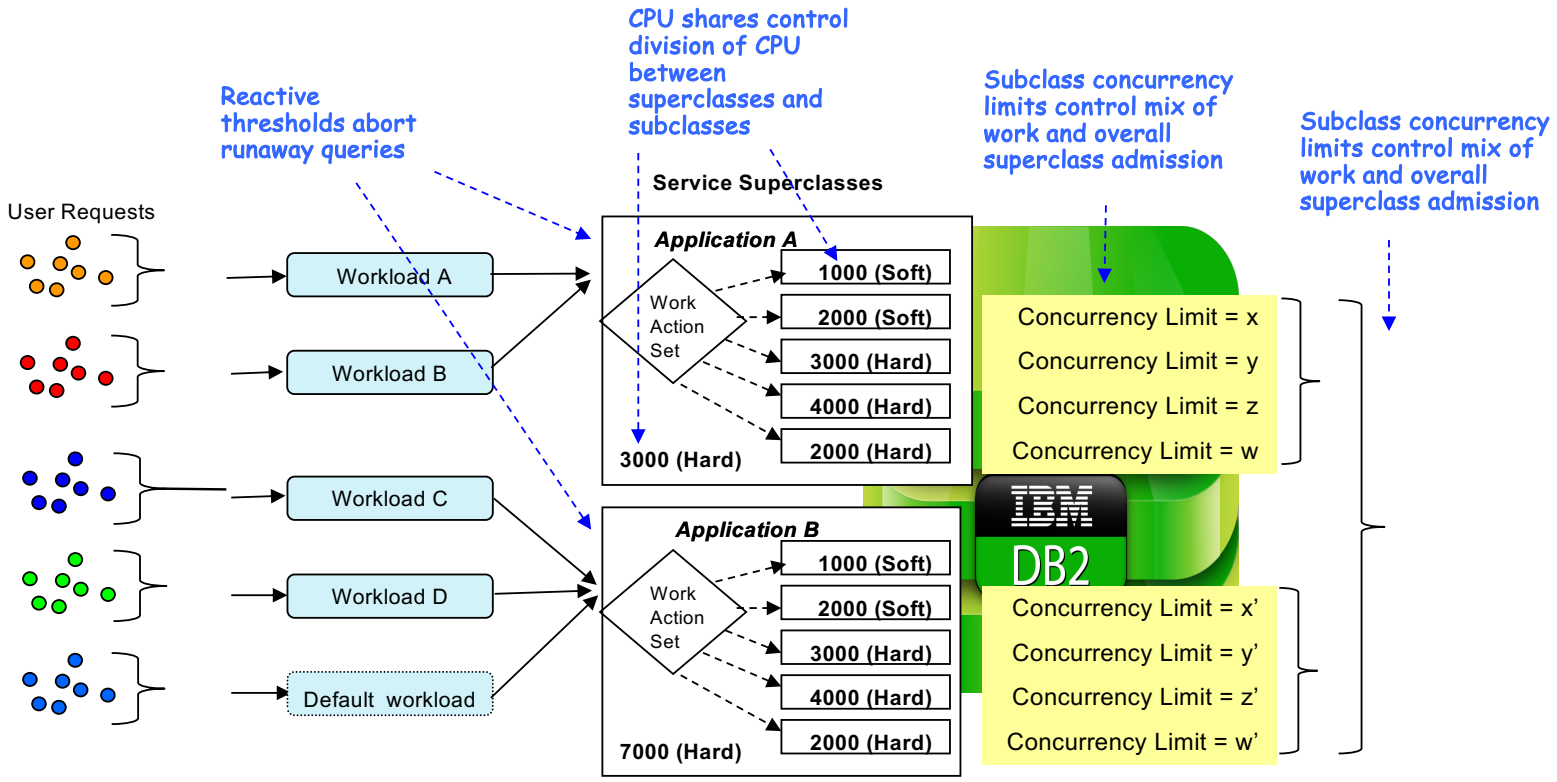
- Define session mapping attributes
- Assign to service class

- **Implementing the steps:**

```
create service class HIPRI soft resource shares 300 for workload type INTERACTIVE
create service class ETL soft resource shares 300 for workload type BATCH
create service class GENERAL soft resource shares 600 for workload type MIXED

create workload REPORTS session_user('EDW_REPORTS') service class HIPRI
create workload ETLJOBS session_user('EDW_ETL_USER') service class ETL
alter workload SYSDEFAULTUSERWORKLOAD service class GENERAL
```

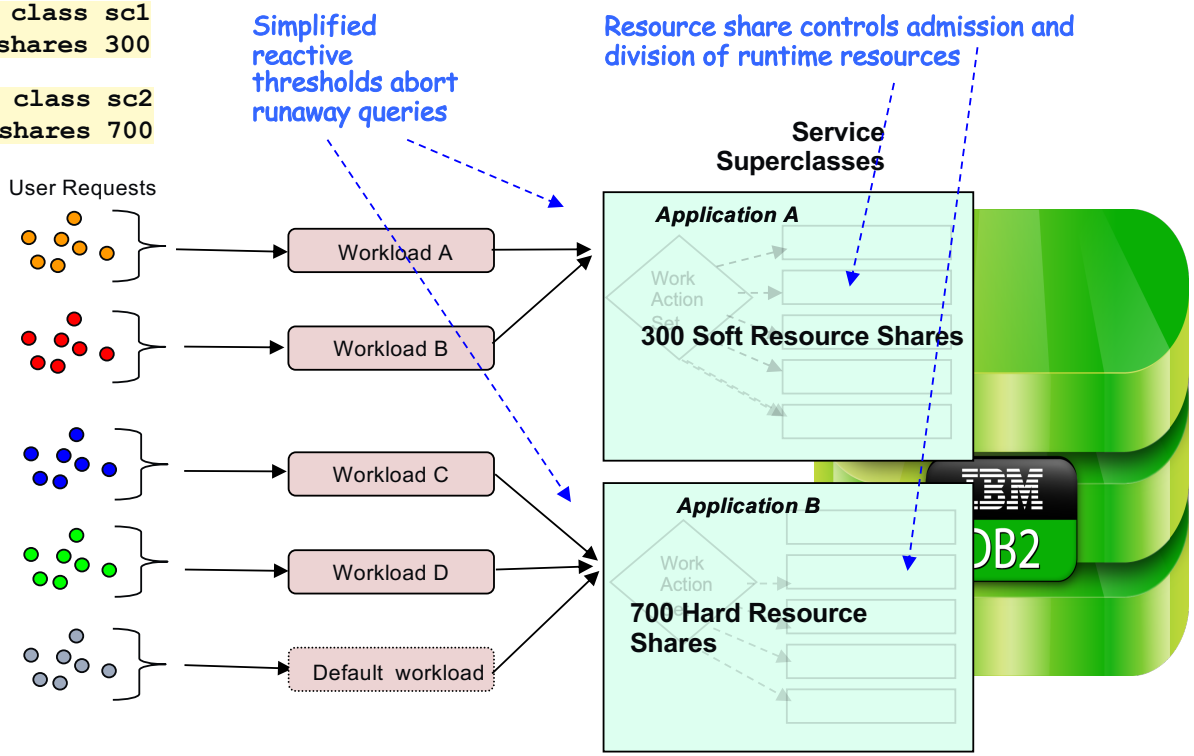
Prioritization with Traditional WLM



Prioritization with Adaptive WLM

```
create service class sc1
soft resource shares 300

create service class sc2
hard resource shares 700
```



Intelligent resource based job scheduling and runtime CPU control ensures fairness + responsiveness and automatically adapts to any workload

User Model Details – Service Class DDL Reference

CREATE/ALTER SERVICE CLASS DDL Clause	Usage
FOR WORKLOAD TYPE <INTERACTIVE | BATCH | MIXED>	Used to pre-configure a service class for a particular query workload.
SOFT RESOURCE SHARES <share>	Used to compute a soft resource entitlement. The soft entitlement is the maximum resources allowed when resources are under contention. A soft entitlement may be exceeded when there is spare resource capacity.
HARD RESOURCE SHARES <share>	Used to compute a hard resource entitlement. The hard entitlement is the maximum amount of resources allowed.
MINIMUM RESOURCE SHARE <value> PERCENT	Used to indicate a percentage of the entitled resources that are held in reserve for the service class (i.e. minimum resource allocation).

More Goodies - Db2 Thresholds

- New **SORTSHRHEAPUTIL** threshold allows you to protect your system from rogue queries with high sort memory demands that might otherwise bottleneck your system.

Example:

- Abort any activity that demands > 25% of the total SHEAPTHRES_SHR

```
CREATE THRESHOLD LARGEACTIVITY FOR DATABASE
WHEN SORTSHRHEAPUTIL > 25 STOP EXECUTION;
```

- Abort any activity that demands > 25% of the total SHEAPTHRES_SHR and blocks other work for > 5 mins

```
CREATE THRESHOLD LARGEACTIVITY FOR DATABASE
WHEN SORTSHRHEAPUTIL > 25 AND BLOCKING ADMISSION FOR MORE THAN 5 MINUTES
STOP EXECUTION;
```

More Goodies - Session Priority

- Set a session **priority** (**HIGH, MEDIUM, LOW, CRITICAL**) which affects how jobs submitted by that session are prioritized for execution within a service superclass

Example:

- Set the session priority to high for a key workload:

```
ALTER WORKLOAD CRITICAL_REPORTS PRIORITY HIGH;
```

- Set the session priority to low for a specific application:

```
CALL SYSPROC.WLM_SET_SESSION_PRIORITY(2361, 'LOW');
```

Short Query Bypass

- Queuing very short queries can have a detrimental impact on performance.
- To avoid adverse impacts, queries with an estimated runtime of under 1 second or an estimate cost < 25000 timerons will bypass admission control with 2 exceptions:
  - Short queries with non-trivial sort memory usage (estimated usage of more than 2% of the configured SHEAPTHRES_SHR) will still go through admission control.
  - Query bypass will be temporarily disabled for any query consuming sort memory when overall database sort memory consumption approaches 100%.
- Note that any queries submitted by a connection associated with the default administration workload (SYSDEFAULTADMWORKLOAD) will unconditionally bypass adaptive workload manager admission control.
  - Can be set from the command line using

```
db2 "set workload to SYSDEFAULTADMWORKLOAD"
```

  Can be set programmatically using the **WLM_SET_CLIENT_INFO** stored procedure
- The **adm_bypassed** monitor element in the **MON_GET_ACTIVITY** interface can be used to identify those queries that bypassed adaptive workload manager admission control.

Other Nuts and Bolts

- The Adaptive Workload Manager simplifies and abstracts lower level constructs but coexists seamlessly with Db2's existing WLM framework
- Subclasses + work-class sets + work action sets are still the underlying mechanisms used for controlling finer grained job scheduling and resource management
- How the Adaptive Workload Manager fits into the Db2 WLM framework
  - *Service superclasses + subclasses*
    - Resource share attribute for admission + runtime control
    - Superclass workload type preconfigures subclasses + work class / work action sets
  - *Work class / work action sets*
    - New mapping type based on query RUNTIME
  - *Thresholds*
    - Simplified syntax + new SORTSHRHEAPUTIL threshold
  - *Session priority concept*
    - New attribute on connections / workloads

Monitoring the Adaptive Workload Manager

Resources that Impact Admission Control

- The adaptive workload manager considers two resources when admitting work into the database; **shared sort memory** and **agents** (threads)
- Incoming queries will queue if the resources they require are unavailable. The most limited resource at any point in time will dictate the observed concurrency.
- **Sort memory** is used by different plan operators in a query (e.g. SORT, GRPBY, HSJN, etc) as well as for column vector working memory.
  - The amount of sort memory used by a query is determined by the number of concurrent sort consuming operators (e.g. SORT, HSJN, etc), the number of tuples processed and the per-operator sort memory limit as defined by the database **sortheap** configuration parameter.
  - The total configured sort memory for the database is determined by the **sheapthres_shr** database configuration parameter.
  - The current sort memory used by a query can be monitored by looking at the **sort_shrheap_allocated** monitor element in the **MON_GET_ACTIVITY** interface
  - The estimated and actual peak sort memory usage for a query can be monitored by looking at the **estimated_sort_shrheap_top** and **sort_shrheap_top** monitor elements respectively in the **MON_GET_ACTIVITY** interface.
  - Adaptive WLM will only admit work up to 95% of the configured sort memory; some memory is held in reserve for queries that bypass WLM.

Resources that Impact Admission Control (cont'd)

- **Agents** are threads that perform work on behalf of queries.
  - The agent requirements for a query are determined by the query degree. The default query degree is controlled by the **dft_degree** database configuration parameter.
  - The estimated number of agents required to execute a query can be examined by looking at the **effective_query_degree** monitor element in the **MON_GET_ACTIVITY** interface.
  - The total number of agents that the adaptive workload manager will admit into the database is determined by the **wlm_agent_load_trgt** database configuration parameter, which specifies the number of agents per CPU core and the number of physical CPU cores. **I.e. max agents admitted = wlm_agent_load_trgt x physical CPU cores**.
  - By default, **dft_degree** is set to ANY; with degree ANY queries will run with a degree equal to the number of physical cores. Under this configuration, the **wlm_agent_load_trgt** can be considered as similar to a concurrency limit (e.g. a load target of 10 would admit approximately 10 queries at a time).

Adaptive WLM configuration

- Out-of-the-box configuration is designed to be largely autonomous + adaptive with no tuning requirements
- One optional tunable that you should be aware of is the `WLM_AGENT_LOAD_TRGT` database configuration parameter
- This parameter controls the maximum thread load per core that the workload manager will allow into the system at a time to avoid degrading processing efficiency.
- The thread load per core on the database is computed as the sum of the `DEGREE` of all the queries executing on the system.
- Example:
  - Running 6 queries with `DEGREE`=12 on a 12-core system results in a thread load per core of 6
  - Running 24 queries with `DEGREE`=1 on a 12-core system results in a thread load per core of 2

Adjusting WLM_AGENT_LOAD_TRGT

- The default `WLM_AGENT_LOAD_TRGT` is computed based on the system hardware and should be optimal for most scenarios
- Consider increasing the `WLM_AGENT_LOAD_TRGT` if:
  - The workload manager is queueing jobs AND
  - There is sufficient sort memory to accommodate more jobs AND
  - None of the system resources are saturated (CPU, I/O, network)
- Consider decreasing the `WLM_AGENT_LOAD_TRGT` if:
  - The system is running a concurrent workload AND
  - The CPU run queues on the system are very heavily loaded and it's degrading system throughput
- Example:

```
UPDATE DB CFG FOR MYDB USING WLM_AGENT_LOAD_TRGT 24
```

Adjusting SORTHEAP and SHEAPTHRES_SHR

- Since Adaptive WLM manages admission based on query resource demands altering the working memory configuration will have a direct impact on job scheduling behavior
- Increasing `SORTHEAP` relative to `SHEAPTHRES_SHR`
  - Allows more memory per operator (and by extension query) reducing execution time, but fewer jobs will be able to run simultaneously
- Decreasing `SORTHEAP` relative to `SHEAPTHRES_SHR`
  - Allows less memory per operator (and by extension query) increasing execution time, but more jobs will be able to run simultaneously
- Increasing `SHEAPTHRES_SHR` by trading off `BUFFERPOOL` memory
  - This strategy can allow increased concurrency without otherwise sacrificing individual query performance
  - Useful in cases where significant large queries result in concurrency bottlenecks

Monitoring Admission Control Queuing Behavior

- Queuing occurs when the resource demands of the current workload exceed the configured resource capacity of the database server causing incoming work to wait until there is capacity for it to execute
- Queuing is expected and should not be viewed as problematic on its own. However, badly behaving applications or queries can cause unexpected queuing resulting in observable delays from a client.
  - For example, consider a query that consumes close to 100% of the resources on the database and blocks other incoming work.
- Monitor elements exposed through SQL functions can be used to understand queuing behaviour on the database as well as to identify the top resource consuming statements.
  - Using this information you can identify the statements responsible for queuing and terminate them if appropriate. For example, if a query that uses 95% of the memory was submitted in error, the application that submitted this query can be terminated by using the FORCE APPLICATION command.

Monitoring Admission Control Queuing Behavior

- **MON_GET_DATABASE** function
  - Provides a summary of overall memory usage and query execution for the database.
  - Elements are reported for each database member. Key monitor elements include:
    - **ACT_COMPLETED_TOTAL** – total number of statements completed
    - **WLM_QUEUE_ASSIGNMENTS_TOTAL** – total number of statements that have been queued by WLM admission control
    - **WLM_QUEUE_TIME_TOTAL** – total amount of queue time incurred by all statements that were queued
    - **SORT_SHRHEAP_ALLOCATED** – current amount of shared sort memory in use
    - **SORT_SHRHEAP_TOP** – peak amount of shared sort memory in used

Monitoring Admission Control Queuing Behavior

- **MON_GET_ACTIVITY** function
  - Provides information about each query currently executing or queued in the database. Elements are reported for each database member. Key monitor elements include:
    - **APPLICATION_HANDLE** – Application that submitted the query
    - **SESSION_AUTH_ID** – Authorization ID of user that submitted the query
    - **SORT_SHRHEAP_ALLOCATED** – Current amount of shared sort memory in use by the query
    - **SORT_SHRHEAP_TOP** – Peak amount of shared sort memory in used by the query
    - **ESTIMATED_SORT_SHRHEAP_TOP** – Estimated peak sort memory usage for the query
    - **EFFECTIVE_QUERY_DEGREE** – Query degree; counted against agent load target
    - **QUERY_COST_ESTIMATED** – Estimated cost (e.g. can use this to help understand bypass behavior)
    - **ADM_RESOURCE_ACTUALS** – Not available until next refresh; indicates whether or not the memory estimate is based on past observed memory consumption
    - **ACTIVITY_STATE** – State of the query; indicates if the query is currently executing, queued or idle (executing, but blocked on a client). Queries in both executing and idle states hold resources
    - **ADM_BYPASSED** – Indicates whether or not the query bypassed admission control
    - **STMT_TEXT** – Query statement text
- Also see **MON_GET_PKG_CACHE_STMT** for a historical view of statement behavior

Monitoring Resource Entitlements and Usage

- The statistics event monitor and statistics table functions (e.g. **MON_GET_SERVICE_SUPERCLASS_STATS**) surface various monitor elements that can be used view resource usage over time. Key monitor elements include:
  - **RESOURCE_ENTITLEMENT** – Percentage of resources that a service class is entitled to based on the configured resource shares for the service class.
  - **AGENT_LOAD_TRGT_UTILIZATION_AVG** – Average utilization of threading resources by work running in the service class, expressed as a percentage of the total threading resources (wlm_agent_load_trgt x number of physical cores)
  - **AGENT_LOAD_TRGT_UTILIZATION_TOP** – Peak utilization of threading resources by work running in the service class, expressed as a percentage of the total threading resources (wlm_agent_load_trgt x number of physical cores)
  - **SORT_SHRHEAP_UTILIZATION_AVG** – Average utilization of shared sort memory by work running in the service class, expressed as a percentage of the configured shared sort memory (sheapthres_shr)
  - **SORT_SHRHEAP_UTILIZATION_TOP** – Peak utilization of shared sort memory by work running in a service class, expressed as a percentage of the configured share sort memory (sheapthres_shr)

Example Monitoring Query #1

- Identify the most constrained resource (agents vs sort)

```
WITH LOADTRGT(LOADTRGT) AS (SELECT MAX(VALUE) FROM SYSIBMADM.DBCFG WHERE NAME =
'wlm_agent_load_trgt'),
     SORTMEM (SHEAPTHRESSHR, SHEAPMEMBER) AS (SELECT VALUE, MEMBER FROM SYSIBMADM.DBCFG
WHERE NAME = 'sheapthres_shr'),
     STMTS(NUMSTMT) AS (SELECT COUNT(*) FROM TABLE(MON_GET_ACTIVITY(NULL,-2)) AS T WHERE
ADM_BYPASSED = 0 AND (ACTIVITY_STATE = 'EXECUTING' OR ACTIVITY_STATE = 'IDLE') AND
MEMBER=COORD_PARTITION_NUM),
     ALLOCMEM(ALLOCMEM, ALLOCMEMBER) AS (SELECT SORT_SHRHEAP_ALLOCATED,MEMBER FROM
TABLE(MON_GET_DATABASE(-2)) AS T)
SELECT MAX(DEC((FLOAT(ALLOCMEM)/FLOAT(SHEAPTHRESSHR))*100, 5,2)) AS PERCENT_SORTMEM_USED,
       MAX(DEC((FLOAT(NUMSTMT)/FLOAT(LOADTRGT))*100,5,2)) AS PERCENT_THREADS_USED
FROM LOADTRGT, SORTMEM, STMTS, ALLOCMEM
WHERE SHEAPMEMBER=ALLOCMEMBER
```

Most constrained resource is sort memory

```
PERCENT_SORTMEM_USED PERCENT_THREADS_USED
-------------------- --------------------
               76.99                11.76
```

Example Monitoring Query #2:

- Currently executing and queued statements with details

```
with sortmem (sheapthresshr, member) as
(select value, member from sysibmadm.dbcfg where name = 'sheapthres_shr')
select b.application_name, b.session_auth_id, a.entry_time, a.local_start_time,
       a.activity_state, a.query_cost_estimate, a.estimated_runtime,
       a.effective_query_degree, a.adm_bypassed,
      (a.estimated_sort_shrheap_top * 100) / c.sheapthresshr as mem_estimate_pct,
      (a.sort_shrheap_top * 100) / c.sheapthresshr as peak_mem_used_pct,
       substr(a.stmt_text, 1, 512) as stmt_text
from table(mon_get_activity(null,-2)) as a,
     table(mon_get_connection(null,-1)) as b,
     sortmem as c
where (a.application_handle = b.application_handle)
order by activity_state;
```

Monitoring Query #2 (cont'd)

Memory estimates used for admission

Very short query admission bypass

Peak memory usage

```
… ACTIVITY_STATE QUERY_COST_ESTIMATE ESTIMATED_RUNTIME EFFECTIVE_QUERY_DEGREE ADM_BYPASSED MEM_ESTIMATE_PCT PEAK_MEM_USED_PCT …
  -------------- ------------------- ----------------- ---------------------- ------------ ---------------- -----------------
  EXECUTING      58                  36733             24                     1            5.14355          4.95233
  EXECUTING      58342               267330            24                     0            3.14355          4.12342
  EXECUTING      58423442            136733            24                     0            11.14355         8.95233
  EXECUTING      182235523           5367333           24                     0            7.14355          9.95233
  QUEUED         679342340083        104336733         24                     0            75.14355         0.00
```

Queued job waiting for admission

Monitoring Resource Entitlement Compliance Example

- Monitor service class resource demand over time relative to entitlement and also display the most heavily contended resource

```
SELECT STATISTICS_TIMESTAMP,
       SUBSTR(SERVICE_SUPERCLASS_NAME, 1, 20) as SUPERCLASS,
       DECIMAL(MAX(RESOURCE_ENTITLEMENT), 5, 2) ENTITLEMENT,
       DECIMAL(MAX(RESOURCE_ENTITLEMENT * (MINRESOURCESHAREPCT) / 100), 5, 2) AS
MINIMUM_ENTITLEMENT,
       CASE WHEN MAX(AGENT_LOAD_TRGT_DEMAND_AVG) > MAX(SORT_SHRHEAP_DEMAND_AVG) THEN
          'THREADS'
       ELSE
          'SORT MEMORY'
       END AS CONSTRAINED_RES,
       DECIMAL(MAX( CASE WHEN AGENT_LOAD_TRGT_DEMAND_AVG > SORT_SHRHEAP_DEMAND_AVG THEN
                      AGENT_LOAD_TRGT_DEMAND_AVG
                    ELSE
                      SORT_SHRHEAP_DEMAND_AVG
                    END ), 5, 2) AS CONSTRAINED_RES_PCT,
       DECIMAL(MAX(AGENT_LOAD_TRGT_DEMAND_AVG), 5, 2) as AGENT_LOAD_DEMAND_AVG,
       DECIMAL(MAX(SORT_SHRHEAP_DEMAND_AVG), 5, 2) AS SORT_SHRHEAP_DEMAND_AVG,
       DECIMAL(MAX(ADM_QUEUED_ACT_LOAD), 5, 2) AS QUEUED_LOAD
FROM SUPERCLASSSTATS_EVMONSTATISTICSU1 A,
     SYSCAT.SERVICECLASSES B
WHERE A.SERVICE_CLASS_ID = B.SERVICECLASSID AND
      A.SERVICE_SUPERCLASS_NAME IN ('S1','S2')
GROUP BY STATISTICS_TIMESTAMP,
         SERVICE_SUPERCLASS_NAME,
         MINRESOURCESHAREPCT
ORDER BY STATISTICS_TIMESTAMP ASC
```

Monitoring Resource Entitlement Compliance Example (Cont'd)

```
STATISTICS_TIMESTAMP       SUPERCLASS ENTITLEMENT MINIMUM_ENTITLEMENT CONSTRAINED_RES CONSTRAINED_RES_PCT AGENT_LOAD_DEMAND_AVG
SORT_SHRHEAP_DEMAND_AVG
-------------------------- ---------- ----------- ------------------ -------------- ------------------- --------------------- --------
---------------
2019-05-01-14.51.31.681770 S1                24.99               0.00 SORT MEMORY                  18.49                 10.43
25.49
2019-05-01-14.51.31.681770 S2                74.99               0.00 SORT MEMORY                  81.02                 58.14
81.02
2019-05-01-14.52.06.424844 S1                24.99               0.00 SORT MEMORY                  40.33                 17.00
40.33
2019-05-01-14.52.06.424844 S2                74.99               0.00 SORT MEMORY                  59.58                 49.98
66.58
…
…
```

The above output shows that sort memory is the most heavily contended resource

Closing Thoughts

Summing Up

Innovative new workload management technology in the Db2 Common SQL Engine that automatically adapts to your workload and vastly simplifies the task of managing your workloads

Leverages intelligent job scheduling for improved stability and performance with zero tuning

Simplified user model allows you to easily divide database resources between different workloads in order to prioritize and meet your performance goals

Technology improvements will continue to roll out across the Hybrid Data Management Platform offerings

What's Next for Adaptive WLM?

- Extending Adaptive WLM to all Db2 configurations
  - Drop DB2_WORKLOAD=ANALYTICS restriction
- Full support for CPU control
  - Add support for integrated CPU shares (current restriction)
- Further incremental efficiency improvements
  - Job scheduling improvements based on field experiences
- Console support
  - Manage Adaptive WLM through the console

Thank You!

Questions?

IDUG

Leading the Db2 User Community since 1988

David Kalmuk
IBM
dckalmuk@ca.ibm.com

Session code: 6048

Please fill out your session evaluation before leaving!

#IDUGDb2