Information Management software

Workload Management with MicroStrategy Software and IBM DB2 9.5

Author:
Scott Cappiello, Senior Director, Program Management, MicroStrategy
Paul Bird, Senior Technical Staff Member, DB2 Development, IBM

A. Introduction and Overview

This paper describes the best practices for configuring MicroStrategy® software to use the workload management (WLM) features of IBM® DB2® 9.5 for Linux®, Unix®, and Windows®.

A critical element of managing an enterprise business intelligence (BI) system is effective workload management (WLM) – applying the right RDBMS resources to the right user requests. DB2 9.5 provides the DB2 Workload Manager, which introduces a new workload management paradigm that makes it easier to monitor and control active work in the system. MicroStrategy software integrates easily with these new WLM features so that insight and control are extended to the BI system.

This document is organized as follows. The first part reviews important concepts of how MicroStrategy software submits work to DB2 9.5 and how DB2 9.5 manages workloads. The second part of the document then describes a detailed example of how MicroStrategy and DB2 9.5 can be configured to achieve workload management objectives typical of a MicroStrategy application.

<u>Target audience for this paper</u>

- DB2 Database Administrators

B. MicroStrategy SQL and Job Execution

There are three categories of SQL requests that MicroStrategy submits to the RDBMS: metadata requests, element requests, and report requests. A *metadata request* is a request to the MicroStrategy object repository, which is stored in an RDBMS. Examples of a metadata request are loading the definition of a report or saving changes to a metric. These requests tend to be "transactional" in nature, generally reading from or writing to one or a small number of rows in a small number of tables.

An *element request* (or element browse) is a request to the RDBMS in order to populate a pick list displayed to the user. Usually an element request accesses a single dimension table and retrieves a reasonable number of rows to display: the application designer usually designs prompts so that a manageable number of elements are displayed for the user to choose from. Element requests, such as the one below, are "simple" in that they consist of a single SELECT statement containing no aggregation, few if any joins, and few if any WHERE conditions.

```
select distinct YEAR_ID  YEAR_ID
from DIM_TIME a11
```

A *report request* is the most interesting kind of request and contains the SQL representing the "question" the user is asking when they execute a report object. Report requests are "analytical" in nature, featuring aggregation functions and GROUP BY's, and also have the potential to be quite complex, including multiple joins, WHERE/HAVING clauses, subqueries, UNIONs, and many other SQL constructs.

Note that a single report request can consist of *multiple SQL statements*, as in the example below. In many cases, MicroStrategy can generate such "multi-pass" requests from a single SQL SELECT statement, using common table expressions, for instance. However, there are some reports that must be executed using temporary tables, for example when calculations performed by MicroStrategy's analytical engine are inserted back into the RDBMS for further processing. Also, customers have the ability on a report-by-report basis to choose whether to use common table expressions or intermediate tables (see the discussion of the VLDB setting the intermediate table type in the MicroStrategy System Administration Guide.) The relevant point for the workload management discussion is that it is possible for MicroStrategy report requests to consist of multiple SQL statements, as illustrated below.

```
create table ZZSP00 (
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)

insert into ZZSP00
select a13.YEAR_ID  YEAR_ID,
         a12.SUBCAT_ID  SUBCAT_ID,
         sum(a11.TOT_UNIT_SALES)  WJXBFS1
from ITEM_MNTH_SLS a11
         join LU_ITEM a12
           on  (a11.ITEM_ID = a12.ITEM_ID)
         join LU_MONTH a13
           on  (a11.MONTH_ID = a13.MONTH_ID)
group by a13.YEAR_ID,
         a12.SUBCAT_ID

create table ZZSP01 (
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)

insert into ZZSP01
select a13.YEAR_ID  YEAR_ID,
         a12.SUBCAT_ID  SUBCAT_ID,
         sum(a11.UNITS_RECEIVED)  WJXBFS1
from INVENTORY_ORDERS a11
         join LU_ITEM a12
           on  (a11.ITEM_ID = a12.ITEM_ID)
         join LU_MONTH a13
           on  (a11.MONTH_ID = a13.MONTH_ID)group by a13.YEAR_ID,
         a12.SUBCAT_ID

select pa1.SUBCAT_ID  SUBCAT_ID,
         a11.SUBCAT_DESC  SUBCAT_DESC,
         pa1.YEAR_ID  YEAR_ID,
         pa1.WJXBFS1  WJXBFS1,
         pa2.WJXBFS1  WJXBFS2
from ZZSP00 pa1
```

```
        join ZZSP01 pa2
          on  (pa1.SUBCAT_ID = pa2.SUBCAT_ID and
        pa1.YEAR_ID = pa2.YEAR_ID)
        join LU_SUBCATEG a11
          on  (pa1.SUBCAT_ID = a11.SUBCAT_ID)

drop table ZZSP00

drop table ZZSP01
```

MicroStrategy generates the SQL for report requests *dynamically* based on the definition of the report object and also on user-specified input. Most MicroStrategy reports make use of prompts. In their simplest form, prompts allow the user to specify parameters that wind up in the WHERE clause, such as a time period for the analysis. But prompts can also be used such that the two executions of the same report result in significantly different SQL, for example, including additional SQL statements, accessing different tables, or performing different calculations.

MicroStrategy has a *connection mapping* feature that determines the database login to use when submitting a request. Typically, users configure the system so that all metadata requests are submitted with the same database login credentials. Some installations also map all report requests to the same database login. But other installations are set up such that each MicroStrategy user executes their report requests as a separate database login. The latter configuration is used when data security rules have been defined in the RDBMS.

MicroStrategy also has the ability to *pool* database connections. Once a session is established with the database, the MicroStrategy Intelligence Server keeps the session open (subject to a user-configurable maximum number of open connections). Any request that can use the same login will use an open session rather than open a new one. Typically, the system is configured to have 2 to 9 sessions open with the object repository to handle metadata requests. An additional 10 to 50 connections can be opened to the reporting RDBMS to handle element and report requests.

All database connections are ODBC connections to the RDBMS. Any SQL syntax that MicroStrategy submits to the RDBMS must be valid over an ODBC connection.

MicroStrategy has the ability to issue *pre- and post-SQL statements* with element requests and report requests. These are user-defined SQL strings that are executed either before or after the SQL for the request itself. Pre- and post-SQL statements can be *parameterized* by which MicroStrategy variables such as date, timestamp, MicroStrategy user login, report name, project name, and job ID are dynamically inserted into the SQL syntax. The SQL snippet below is a contrived example that illustrates how a pre-SQL statement can be configured to call a hypothetical stored procedure "my_sp1" passing in the MicroStrategy user login and the report name as arguments; it also shows how a post-SQL statement can be configured to call a "my_sp2" procedure passing in the project name, job number, and date of execution as arguments. (The complete list of variables available for pre- and post-SQL statements is given later in this document.)

```
call my_sp1('MSTRUser', 'Management Report')

create table ZZSP00 (
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)

insert into ZZSP00
select a13.YEAR_ID  YEAR_ID,
        a12.SUBCAT_ID  SUBCAT_ID,
        sum(a11.TOT_UNIT_SALES)  WJXBFS1

…

drop table ZZSP00

drop table ZZSP01

call my_sp2('Finance Project', 'Job 123', '20070302')
```

In summary, when a user executes a report in MicroStrategy, the following activity can be submitted to the RDBMS:

- One or more metadata requests can be executed to load the definition of the report.
  - These are submitted through dedicated connections to the metadata repository, where multiple users share the same login credentials and, in fact, the same database session.
- If the report contains prompts, some element requests can be submitted to populate pick lists to display to the user.
  - These can be submitted through a new connection to the reporting database or through an already-open connection that uses the same database login as a previous request.
- After prompts have been answered, a report request is submitted.
  - This request, which can consist of multiple SQL statements, is submitted as described above for element requests.

C. Workload Management in DB2 9.5

Comprehensive workload management (WLM) has been integrated into DB2 9.5. With the WLM features of DB2 9.5, you can divide work into classes and tailor the DB2 9.5 server to support a variety of users and applications on the same system.

All work within DB2 9.5 is executed within the context of a specific *service class*. A service class acts as a unique execution environment for a grouping of work. Resources such as CPU priority can be allocated differently to different service classes, so that higher priority work requests receive more resources.

Connections coming into DB2 9.5 are classified into *workloads,* which are then mapped to service classes. A workload definition consists of a series of values for one or more connection attributes, such as the application name or SESSION_USER. Each incoming connection is assigned to a workload by comparing its connection attributes to the definitions of each DB2 9.5 workload; the first match that is found is the workload that is used. For example, a workload can be defined as all requests coming from SESSION_USER jdoe. The following connection attributes can be used to define workloads:

- Application name
- SYSTEM_USER
- SESSION_USER
- Any group of SESSION_USER
- Any role of SESSION_USER
- CLIENT USERID
- CLIENT APPLNAME
- CLIENT WRKSTNNAME
- CLIENT ACCTNG

The last four attributes, known as client information fields, can be changed during an open session through the use of the WLM_SET_CLIENT_INFO stored procedure. The syntax of this procedure is:

```
WLM_SET_CLIENT_INFO (client_userid, client_wrkstnname, client_applname,
client_acctstr, client_workload)
```

where the value of each client information field can be passed in as a string. The *client_workload* argument provides a way to directly assign the connection to the built-in workload SYSDEFAULTADMWORKLOAD. For example, the SQL snippet below sets the user ID, workstation name, application name, accounting string, and workload assignment mode for the client.

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('DB2USER',
'machine.torolab.ibm.com', 'microstrategy', 'accounting department',
'AUTOMATIC')
```

It is not necessary to specify all client information fields when using this stored procedure. The following SQL snippet sets the user ID to DB2USER2 for the client without setting the other client attributes.

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('DB2USER2', , , ,)
```

See http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0053116.html for a complete description of the WLM_SET_CLIENT_INFO procedure.

While workload definitions classify work based on connection attributes, DB2 9.5 also provides ways to treat work activities differently based on their type or some individual characteristic that is independent of connection attributes. *Work class sets* and *work action sets* provide the capability to further discriminate activities based on such characteristics, such as "put DML in a different service class than DDL" or "put all read queries of less than 100 timerons in a different service class than all the other read queries."

D. Configuring MicroStrategy and DB2 for Workload Management

In this section, we describe one way to configure a system running MicroStrategy software and DB2 9.5 to achieve some common workload management objectives. Note that this is not the only possible configuration; some alternative configurations are described in the section "Alternative Workload Management Configurations." Our objectives in this example are as follows:

- **Metadata requests should return promptly.** Application designers want to ensure that metadata requests return promptly. These requests are submitted as a result of browsing through folders, loading report definitions, etc. Metadata requests need to return quickly (sub-second) in order for the application to feel "snappy."
- **Element requests should return promptly.** Application designers want to ensure that element requests return promptly. After executing a report, the user must wait until all pick lists have returned before prompts are displayed. Element requests need to return quickly so that the user doesn't wait very long before even submitting their selections.
- **Prioritized users should get prioritized resource allocation.** Application designers want to prioritize report requests based on variables that are known at the time the report is submitted. For example, report requests from the CEO should be prioritized ahead of other requests. Combinations of variables can also be used for prioritizing resources. For example, users from the Finance department should have priority over non-Finance users when submitting reports in the Finance application, while Marketing users have priority over non-Marketing users when submitting reports in the Marketing application.
- **Report requests should be prioritized based on estimated cost.** Application designers want to ensure that short-running reports are executed quickly, while it might be acceptable for long-running queries to run longer. Furthermore, application designers want to ensure that a group of long-running queries doesn't "starve out" other shorter-running queries. Users expect that their simple requests will come back quickly and not depend on other analysis being conducted at the same time.

MicroStrategy Configuration

In order for DB2 9.5 to classify work submitted by MicroStrategy, the MicroStrategy application must be configured to send identifying information along with each query it submits.

Metadata Requests

We will configure MicroStrategy to submit all metadata requests using a dedicated DB2 login. Then we can identify any request coming from that login as a metadata request.

A database instance is a MicroStrategy object that represents a database or database server. A database instance has an ODBC connection string and a login associated with it. To direct all metadata requests to the same workload in DB2 9.5, configure a separate database instance to represent the metadata repository. Use a dedicated DB2 login such as 'MSTR_MD' as the default login for this database instance. All metadata requests from all MicroStrategy users will be submitted on dedicated connections using this same login.

Prioritized Element Requests and Report Requests

We cannot use the DB2 login to identify report requests and element requests, because MicroStrategy can be configured to submit these requests using a different DB2 login for each user. (MicroStrategy can also be configured to submit report/element requests using a pooled DB2 login. But the relevant point for workload management is that either of these configurations can be used, so we cannot rely on a dedicated DB2 login to identify report or element requests.) Instead, we will configure MicroStrategy to include some additional identifying information in the SQL it submits for each request.

Configure the "report pre-SQL statement" (a VLDB setting at the Project level) to the following string:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=!u', , 'Project=!p',
'Report=!o',)
```

This stored procedure call sets the DB2 9.5 connection attributes CLIENT USERID, CLIENT APPLNAME, CLIENT ACCTNG to hold the values for the MicroStrategy user ID, project name, and report name, respectively. The CLIENT WRKSTNNAME (the second argument in the WLM_SET_CLIENT_INFO procedure) is not set.

The result for report requests will be the following SQL:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=jdoe', ,
'Project=CategoryManagement', 'Report=SalesSummary',)

create table ZZSP00 (
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)

insert into ZZSP00
select a13.YEAR_ID  YEAR_ID,
        a12.SUBCAT_ID  SUBCAT_ID,
…
```

and the following SQL for element requests:

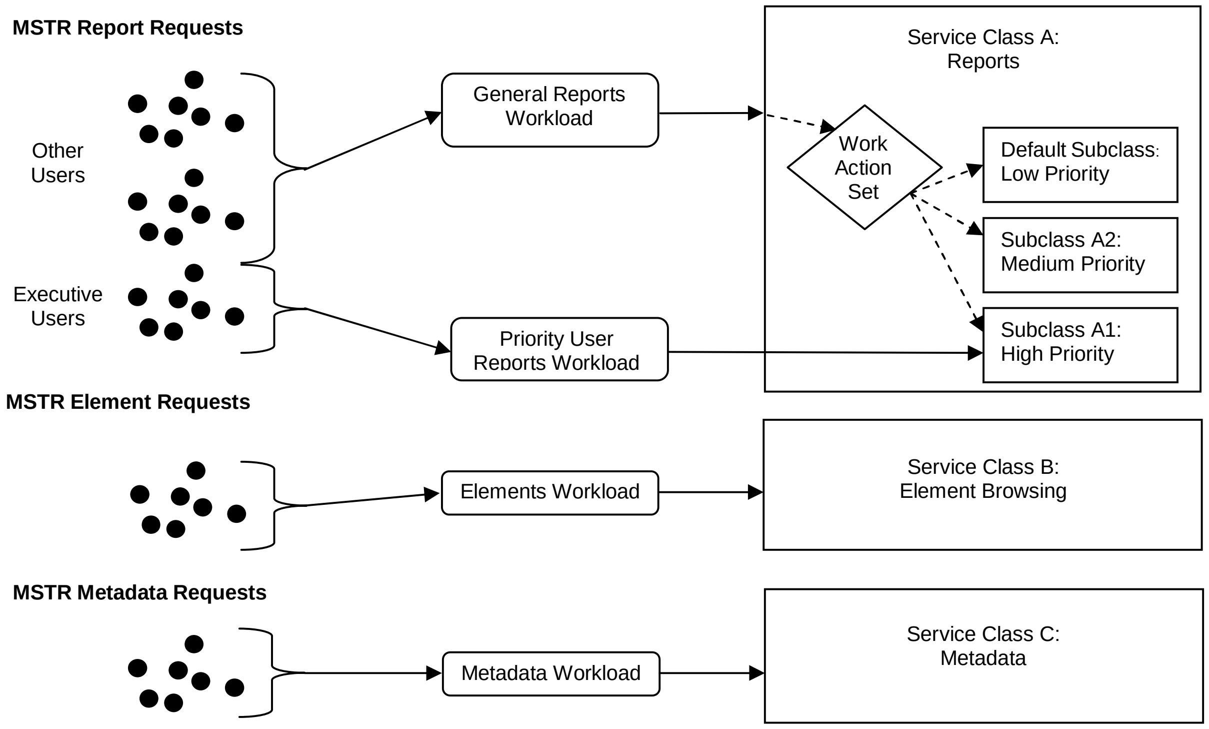
```
CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=jdoe', ,
'Project=CategoryManagement', 'Report=-',)

select  count(distinct a21.QUARTER_ID)  WJXBFS1
from    LU_QUARTER a21

select  a11.QUARTER_ID  QUARTER_ID,
       a11.QUARTER_DESC  QUARTER_DESC0
from   LU_QUARTER a11
order by       1 asc
```

The following MicroStrategy variables can also be submitted to DB2 9.5 and used for WLM:

!d	Date from Intelligence Server
!t	Time from Intelligence Server
!u	MicroStrategy user name
!o	Report name
!r	Report ID (16-digit hex ID)
!p	Project name
!z	Project ID (16-digit hex ID)
!j	Intelligence Server job ID
!i	Intelligence Server job priority (integer from 0 to 999)

DB2 9.5 Configuration

The diagram below illustrates the workloads, service classes, work class sets, and work action sets in DB2 9.5 that we will use to achieve these objectives.



Service Classes

With the DB2 9.5 commands below, we will establish five service classes to handle the work from the MicroStrategy application. We will see later how we map the different types of work to each of these service classes.

- Metadata requests get a dedicated service class (Service Class C in the diagram).
- Element requests get a dedicated service class (Service Class B).

- Report requests are mapped to a service class that is separate from metadata and element requests (Service Class A). Each query associated with a report request can be further mapped to a service subclass based on the estimated cost of the query:
    - Simple (lowest-cost) queries get a prioritized service subclass (Service Class A1).
    - Medium (moderate-cost) report requests get a dedicated service subclass (Service Class A2).
    - Complex (highest-cost) report requests remain in the default subclass of the service super class (Service Class A) separate from the prioritized service subclasses. Note that this default subclass is created automatically by DB2 when the super class is created.
- Report requests from high-priority users are "fast-tracked" to the high-priority service subclass (Service Class A1) regardless of the estimated cost of each query.

```
CREATE SERVICE CLASS scMSTRMetadata;
CREATE SERVICE CLASS scMSTRElementBrowsing;
CREATE SERVICE CLASS scMSTRReports;
CREATE SERVICE CLASS scMSTRMediumPriority UNDER scMSTRReports;
CREATE SERVICE CLASS scMSTRHighPriority UNDER scMSTRReports;
```

In this configuration, you can allocate CPU priority to the "priority" service classes if you want: scMSTRMetadata, scMSTRElementBrowsing, scMSTRMediumPriority, and scMSTRHighPriority. Service class scMSTRReports serves as an "all other" service class that does not receive the same level of priority as the other service classes.

Workloads

Next we will define workloads to classify the requests coming from MicroStrategy based on the identifying information available from the connections:

- wlMSTRMetadata: Any connection with SESSION_USER 'MSTR_MD' will be considered a metadata connection and mapped to the scMSTRMetadata service class.
- wlMSTRElementBrowsing: Any connection not identified as a metadata one, with the application name 'MJMulPrc_32.EXE' and CLIENT ACCTNG string 'Report=-', will be identified as an element request connection. Note that the evaluation order of this workload is positioned after wlMSTRMetadataRequests, so only connections not matching the attributes of that workload will be considered for this workload. MJMulPrc_32.EXE is the MicroStrategy process that submits requests to DB2 9.5 and this is what DB2 9.5 sees as the application name. When MicroStrategy issues an element request, the report name is empty and appears as 'Report=-' in the SQL submitted.
- wlMSTRPriorityReports: Any connection not identified as a metadata or element request connection, with the application name 'MJMulPrc_32.EXE' and CLIENT USERID string 'MSTRUser=ceo' or 'MSTRUser=coo' or 'MSTRUser=cfo', will be identified as a high-priority connection and mapped to the scMSTRHighPriorityReports service class.

- wlMSTRReports: Any connection not identified as one of the previous workloads (enforced in the workload definition by use of the POSITION AFTER specification) with application name ‘MJMulPrc_32.EXE’ will be identified as part of this catch-all workload and mapped to the scMSTRReports service class.

```
CREATE WORKLOAD wlMSTRMetadata
  SESSION_USER(‘MSTR_MD’)
  SERVICE CLASS scMSTRMetadata;
CREATE WORKLOAD wlMSTRElementBrowsing
  APPLNAME(‘MJMulPrc_32.EXE’)
  CURRENT CLIENT_ACCTNG(‘Report=-’)
  SERVICE CLASS scMSTRElementBrowsing
  POSITION AFTER wlMSTRMetadata;
CREATE WORKLOAD wlMSTRPriorityReports
  APPLNAME(‘MJMulPrc_32.EXE’)
  CURRENT CLIENT_USERID(‘MSTRUser=ceo’, ‘MSTRUser=cfo’,
‘MSTRUser=coo’)
  SERVICE CLASS   scMSTRHighPriorityReports
  POSITION AFTER wlMSTRElementBrowsing;
CREATE WORKLOAD wlMSTRReports
  APPLNAME(‘MJMulPrc_32.EXE’)
  SERVICE CLASS scMSTRReports
  POSITION AFTER wlMSTRPriorityReports;
```

With these workload definitions, we have used connection information to identify metadata requests, element requests, and report requests from high-priority users and to map them to their appropriate service classes.

Work Class Sets and Work Action Sets

Now we will use a work class set and a work action set to differentiate among report requests based on their estimated cost so that simpler reports can be prioritized accordingly. Note that if temporary tables are used, then a single MicroStrategy report request can consist of multiple SQL statements sent to DB2 9.5. The cost estimate from DB2 9.5 is expressed in units called timerons and is provided by the SQL Compiler for each individual query within a report not the report as a whole. Some passes might be considered costly while other passes in the same report are considered simple.

An example work class set wcsALLDML (shown below) identifies DML statements based on their estimated cost. Statements that are estimated to cost between 0 and 1000 timerons are identified as “simple” statements; statements between 1000 and 20000 timerons are “medium.”[1]

The work action set wasMSTActions maps the statements as identified by the work classes above to an appropriate service class: scMSTRHighPriority for the simple reports, and scMSTRMediumPriority for the medium reports.

[1] Note that these specific estimate values have been created simply for the purposes of the example and do not reflect any real or expected estimate values for the SQL statements.

```
CREATE WORK CLASS SET wcsALLDML
  (WORK CLASS wcSmallDML WORK TYPE DML
   FOR TIMERONCOST FROM 0 TO 1000
   WORK CLASS wcMediumDML WORK TYPE DML
   FOR TIMERONCOST FROM 1000 TO 20000);

CREATE WORK ACTION SET wasMSTRActions
  FOR SERVICE CLASS scMSTRReports
  USING WORK CLASS SET wcsALLDML
  (WORK ACTION waMSTRHighPriority ON WORK CLASS wcSmallDML
     MAP ACTIVITY TO scMSTRHighPriority
   WORK ACTION waMSTRMediumPriority ON WORK CLASS wcMediumDML
     MAP ACTIVITY TO scMSTRMediumPriority);
```

The result of these definitions is that simple queries will be placed in the high-priority service subclass, and medium queries will be placed in the medium-priority service subclass. Any statement not identified as simple or medium by the work class set, which includes "complex" DML statements estimated to cost more than 20000 timerons as well as DDL statements such as DROP TABLE, will remain in the scMSTRReports service super class and therefore execute in the default service subclass.

Note again that because the cost estimation occurs on a per-statement basis, a single MicroStrategy report consisting of multiple statements could be distributed across multiple service subclasses.

Configuration Summary

The DB2 9.5 and MicroStrategy configurations above will result in the following behavior:

- Metadata requests are submitted through a dedicated DB2 login. These requests are identified as a workload (wlMSTRMetadata) based on this login (SESSION_USER) and mapped to a dedicated service class (scMSTRMetadata).
- Element requests are submitted with some additional information in the CLIENT_USERID, CLIENT_APPLNAME, and CLIENT_ACCTNG registers using the WLM_SET_CLIENT_INFO stored procedure. These requests are identified as a workload (wlMSTRElementBrowsing) based on the CLIENT_ACCTNG string and mapped to a dedicated service class (scMSTRElementBrowsing).
- Report requests are submitted with some additional information in the CLIENT_USERID, CLIENT_APPLNAME, and CLIENT_ACCTNG registers using the WLM_SET_CLIENT_INFO stored procedure. Requests from high-priority users are identified as a workload (wlMSTRPriorityReports) based on the CLIENT_USERID string and mapped to a dedicated service class (scMSTRHighPriorityReports).
  - Note that this workload makes no distinction between work requests based on estimated cost and those based on DML. If the CLIENT_USERID string indicates that this is a high-priority request, then all statements that comprise this report will be submitted to the same service class.

- All other report requests are routed to the scMSTRReports service calls, where they might also be routed to dedicated subclasses based on their estimated cost.
    - INSERT from SELECTs, SELECTs with low cost will go to the scMSTRHighPriorityReports service class.
    - INSERT from SELECTs, SELECTs with medium cost will go to the scMSTRMediumPriorityReports service class.
    - Other INSERT from SELECTs, SELECTs will remain in the scMSTRReports service class.
    - DECLAREs, DROPs, CREATE INDEXs, CALL RUNSTATS, and other DDL or unidentifiable statements will remain in the scMSTRReports service class.

E. Workload Management in DB2 9.5

Monitoring and Control

Once workload management has been configured, DB2 9.5 provides extensive features for monitoring and controlling database activities. Below we discuss some simple queries that can be used to confirm that the workload management configuration described in this document is functioning as expected. See http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.wlm.doc/doc/c0051399.html for a more complete description of monitoring and control capabilities of DB2 9.5 workload management.

Monitoring Queries using SQL

The WLM_GET_SERVICE_SUPERCLASS_STATS procedure returns basic statistics of one or more service super classes. Use this table function to obtain information about the concurrent connection high watermark that was calculated since the last statistics reset.

```
select *
from table(WLM_GET_SERVICE_SUPERCLASS_STATS(cast(null as
VARCHAR(128)), -2)) as scstats
```

The WLM_GET_SERVICE_SUBCLASS_STATS procedure returns basic statistics of one or more service subclasses. Use this table function to obtain summarized statistics such as the number of activities and average execution time calculated since the last statistics reset.

```
select *
from table(WLM_GET_SERVICE_SUBCLASS_STATS(cast(null as
VARCHAR(128)), cast(null as VARCHAR(128)), -2)) as scstats
```

The WLM_GET_WORKLOAD_STATS procedure returns workload statistics for every combination of workload name and database partition number. Use this table function to obtain summarized statistics for one or all workloads and database partitions.

```
select *
from table(WLM_GET_WORKLOAD_STATS(cast(null as VARCHAR(128)), -2))
as wlstats
```

Monitoring Queries using Event Monitors

The WLM_COLLECT_STATS procedure causes statistics for service classes, workloads, work classes, and threshold queues to be gathered and written to the statistics event monitor. The statistics for service classes, workloads, work classes, and threshold queues are also reset at this time. If you activate aggregate activity statistics using the COLLECT AGGREGATE ACTIVITY DATA clause on any work action or service classes, then DB2 9.5 will also collect distribution information about the different activities being processed.

```
call wlm_collect_stats()
```

Alternatively, you can set the WLM_COLLECT_INT database configuration parameter to have DB2 9.5 automatically collect these statistics for you at a regular interval and write them out to the activity statistics event monitor.

If you want information about the individual SQL statements being run by any specific workload, work action, or service class, you can activate capture of activity information using the COLLECT ACTIVITY DATA clause on the area of interest. Doing so will cause a record to be written for each SQL statement executed in that area to the activities event monitor. Different levels of information are available for this clause including ones that provide the statement text, compilation environment, and input data values. In addition, if the system monitor switches are active, any relevant information for each statement is also captured to the same event monitor along with the basic statement information.

Thresholds and CONCURRENTDBCOORDACTIVITIES

A threshold is a control feature that allows you to establish limits over different behaviors of an SQL statement, such as consumption of a specific resource or maximum time for processing. If a threshold is violated, a specified action can be triggered such as stopping execution of the activity that caused the threshold to be violated.

The CONCURRENTDBCOORDACTIVITIES threshold specifies the maximum number of recognized coordinator activities[2] that can run concurrently across all database partitions. This type of threshold, a concurrency control threshold, is a very effective and powerful tool used to sequence and limit the amount of work being asked of the database at any particular time. In some cases where this threshold is used, applications that start more than one concurrent activity can potentially consume all the concurrency available for this threshold, thus creating a *self-deadlock* scenario. However, the MicroStrategy application submits all work for the same report in a serial manner: each statement must complete before the next statement is sent.

[2] In DB2 9.5, a recognized activity is a DML or DDL SQL statement or Load.

As a result, the MicroStrategy application is not susceptible to this self-deadlock scenario and is compatible with the use of the CONCURRENTDBCOORDACTIVITIES threshold.

F. Workload Management in DB2 9.5

Example MicroStrategy Reports

The following reports were executed using the configuration defined above. The SQL syntax below illustrates which queries were directed to which service classes in DB2 9.5.

Report 1a: Common Table Expressions

This report consists of two SQL passes generated by MicroStrategy. Because it is generated using Intermediate Table Type = Common Table Expressions, DB2 9.5 sees the entire report as one READ statement. Because the estimated cost of the query exceeded the 20000 timerons threshold specified above, the request is left in the general MSTR Reports service class.

```
(Executes in scMSTRReports)
with     gopa1 as
 (select          a11.CUSTOMER_ID  CUSTOMER_ID
         from    CUSTOMER_SLS       a11
         group by        a11.CUSTOMER_ID
         having sum(a11.TOT_DOLLAR_SALES) > 1000.0
         )select a13.SUBCAT_ID  SUBCAT_ID,
         max(a14.SUBCAT_DESC)  SUBCAT_DESC,
         sum(a11.QTY_SOLD)  WJXBFS1
from     ORDER_DETAIL       a11
         join    gopa1  pa12
           on    (a11.CUSTOMER_ID = pa12.CUSTOMER_ID)
         join    LU_ITEM       a13
           on    (a11.ITEM_ID = a13.ITEM_ID)
         join    LU_SUBCATEG        a14
           on    (a13.SUBCAT_ID = a14.SUBCAT_ID)
group by         a13.SUBCAT_ID
```

Report 1b: Temporary Tables

This is the same report as in the previous example, except that the intermediate table type is now set to temporary tables. When using temporary tables, DB2 9.5 sees the report as one DDL statement, one WRITE (INSERT) statement, one READ (SELECT) statement, and a second DDL (DROP) statement. In this example, the INSERT statement has low estimated cost, and is routed to the High Priority service subclass. The SELECT statement has higher estimated cost and is left in the general MSTR Reports service class. The DDL statements are always left in the general MSTR Reports service class. This example shows how one MicroStrategy report can be broken down into work that crosses multiple service classes.

```
(Executes in scMSTRReports)
declare global temporary table session.ZZMQ00(
          CUSTOMER_ID          SMALLINT)
partitioning key (CUSTOMER_ID) on commit preserve rows not logged

(Executes in scHighPriorityReports)
insert into session.ZZMQ00
select  a11.CUSTOMER_ID  CUSTOMER_ID
from    CUSTOMER_SLS       a11
group by        a11.CUSTOMER_ID
having sum(a11.TOT_DOLLAR_SALES) > 1000.0

(Executes in scMSTRReports)
select  a13.SUBCAT_ID  SUBCAT_ID,
        max(a14.SUBCAT_DESC)  SUBCAT_DESC,
        sum(a11.QTY_SOLD)  WJXBFS1
from    ORDER_DETAIL        a11
        join    session.ZZMQ00       pa12
          on    (a11.CUSTOMER_ID = pa12.CUSTOMER_ID)
        join    LU_ITEM        a13
          on    (a11.ITEM_ID = a13.ITEM_ID)
        join    LU_SUBCATEG          a14
          on    (a13.SUBCAT_ID = a14.SUBCAT_ID)
group by        a13.SUBCAT_ID

(Executes in scMSTRReports)
drop table session.ZZMQ00
```

Report 2a: Common Table Expressions

This report consists of three SQL passes generated by MicroStrategy. Because the estimated cost of the query is within the 1000 to 20000 timerons threshold specified above, the request is routed to the Medium Priority service subclass.

```
(Executes in scMediumPriorityReports)
with      gopa1 as
 (select         a11.SUBCAT_ID  SUBCAT_ID,
                 sum(a11.TOT_UNIT_SALES)  WJXBFS1
        from    CITY_SUBCATEG_SLS          a11
        group by        a11.SUBCAT_ID
        ),
         gopa2 as
 (select         a12.SUBCAT_ID  SUBCAT_ID,
                 sum(a11.UNITS_RECEIVED)  WJXBFS1
        from    INVENTORY_ORDERS          a11
                join    LU_ITEM        a12
                  on    (a11.ITEM_ID = a12.ITEM_ID)
        group by        a12.SUBCAT_ID
        )select pa11.SUBCAT_ID  SUBCAT_ID,
        a13.SUBCAT_DESC  SUBCAT_DESC,
        pa11.WJXBFS1  WJXBFS1,
        pa12.WJXBFS1  WJXBFS2
```

```
from    gopa1  pa11
        join    gopa2  pa12
          on    (pa11.SUBCAT_ID = pa12.SUBCAT_ID)
        join    LU_SUBCATEG        a13
          on    (pa11.SUBCAT_ID = a13.SUBCAT_ID)
```

Report 2b: Temporary Tables

This is the same report using temporary tables. Again, all the DDL statements are left in the General Reports service class. But in this example, all INSERT and SELECT statements have low cost and are routed to the High Priority service subclass.

```
(Executes in scMSTRReports)
declare global temporary table session.ZZSP00(
        SUBCAT_ID    SMALLINT,
        WJXBFS1      DOUBLE)
partitioning key (SUBCAT_ID) on commit preserve rows not logged

(Executes in scHighPriorityReports)
insert into session.ZZSP00
select  a11.SUBCAT_ID  SUBCAT_ID,
        sum(a11.TOT_UNIT_SALES)  WJXBFS1
from    CITY_SUBCATEG_SLS        a11
group by        a11.SUBCAT_ID

(Executes in scMSTRReports)
declare global temporary table session.ZZSP01(
        SUBCAT_ID    SMALLINT,
        WJXBFS1      DOUBLE)
partitioning key (SUBCAT_ID) on commit preserve rows not logged

(Executes in scHighPriorityReports)
insert into session.ZZSP01
select  a12.SUBCAT_ID  SUBCAT_ID,
        sum(a11.UNITS_RECEIVED)  WJXBFS1
from    INVENTORY_ORDERS         a11
        join    LU_ITEM        a12
          on    (a11.ITEM_ID = a12.ITEM_ID)
group by        a12.SUBCAT_ID

(Executes in scHighPriorityReports)
select  pa11.SUBCAT_ID  SUBCAT_ID,
        a13.SUBCAT_DESC  SUBCAT_DESC,
        pa11.WJXBFS1  WJXBFS1,
        pa12.WJXBFS1  WJXBFS2
from    session.ZZSP00       pa11
        join    session.ZZSP01       pa12
          on    (pa11.SUBCAT_ID = pa12.SUBCAT_ID)
        join    LU_SUBCATEG          a13
          on    (pa11.SUBCAT_ID = a13.SUBCAT_ID)

(Executes in scMSTRReports)
drop table session.ZZSP00
```

```
(Executes in scMSTRReports)
drop table session.ZZSP01
```

G. Alternative Workload Management Configurations

Prioritizing Reports Based on User Is Optional

In the configuration described above, the workload wlMSTRPriorityReports is entirely optional. If you do not want to prioritize report requests based on MSTR user names, simply omit the definition of this workload.

Identifying Workloads Based on MicroStrategy Report Priority

MicroStrategy provides several variables in addition to those described in the above configuration, which can be passed in to DB2 9.5 using the WLM_SET_CLIENT_INFO stored procedure. Workload management could be configured to classify reports based on the MicroStrategy job priority. MicroStrategy job priority is a function of the MicroStrategy user, the project, the request type, and a report-specific "cost" attribute. See MicroStrategy System Administration Guide for more details on setting up job priority in MicroStrategy.[3]

To pass MicroStrategy job priority into DB2 WLM, change the pre- and post-SQL statement for reports as follows:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=!u', , 'Project=!p',
'Priority=!i',)
```

This will result in the following SQL for report requests:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=jdoe', ,
'Project=CategoryManagement', 'Priority=999',)

create table ZZSP00 (
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)

insert into ZZSP00
select a13.YEAR_ID  YEAR_ID,
        a12.SUBCAT_ID  SUBCAT_ID,
…
```

The priority value is 999 for high priority, 666 for medium priority, and 333 for low priority.

The corresponding workloads in DB2 9.5 could be configured as follows:

[3] This will be available with the MicroStrategy 9 release.

```
CREATE WORKLOAD wlMSTRHighPriorityReports
  APPLNAME(‘MJMulPrc_32.EXE’)
  CURRENT CLIENT_ACCTNG(‘Priority=999’) SERVICE CLASS
scMSTRHighPriorityReports
CREATE WORKLOAD wlMSTRMediumPriorityReports
  APPLNAME(‘MJMulPrc_32.EXE’)
  CURRENT CLIENT_ACCTNG(‘Priority=666’) SERVICE CLASS
scMSTRMediumPriorityReports
CREATE WORKLOAD wlMSTRLowPriorityReports
  APPLNAME(‘MJMulPrc_32.EXE’)
  CURRENT CLIENT_ACCTNG(‘Priority=333’) SERVICE CLASS scMSTRReports
```