Best Practices
DB2 Workload Management

Paul Bird
Senior Technical Staff Member
DB2 Technology Development

Nela Krawez, Ph.D.
DB2 Technical Solution Architect

Nikolaj Richers, Ph. D.
DB2 Information Development

Huaxin Zhang, Ph. D.
DB2 Technology Development
Executive Summary

Today, for both strategic and financial reasons, many businesses are consolidating multiple individual data servers onto a single shared data server. As each new data server is merged, it potentially adds a very different type of workload to the mix with different system interactions. The consolidated server must now support a variety of different, concurrent workloads. Good workload management practices are critical to meeting business commitments in such an environment.

When your database system encounters performance degradation due to the different, and sometimes conflicting, resource demands from the work being executed on it, then you need DB2 workload manager to help you prevent, detect, and resolve these conflicts.

This paper introduces you to the current best practices for DB2 workload manager on your DB2 Version 9.5 for Linux, UNIX, and Windows data server which can be used to help you meet your business objectives for the work being executed on DB2 data servers. The best practices presented here are based on IBM field experience in benchmark and proof-of-concept exercises as well as feedback from customer adoptions of DB2 Version 9.5.

There are three sections to this paper covering:

1. Best practices in design
2. Best practices in implementation
3. Best practices in monitoring

There are also appendices in this paper that provide you with more in-depth information and guidance such as:

- A specific scenario that shows you how to help meet your business priorities with DB2 workload manager in a data warehouse environment
- An example script to use with the AIX® Workload Manager to achieve dynamic processor allocation while using hard maximum limits on CPU use
- Guidance on how to upgrade from Query Patroller and DB2 Governor

This paper will be most useful to you if you are already familiar with some of the basic concepts of DB2 workload management. If you find that you would like more background information first, you can refer to “Introduction to DB2 9.5 workload management” or to some of the further reading material suggested at the end of this paper.
Best Practices: Design

The underlying principles of workload management are always the same:

1. Understand the work that needs to run on your data server and how it relates to your business commitments.

2. Determine whether your system has the capacity to run this work without any restrictions; you can only put through as much work at any one time as your system can handle.

3. “Divide and conquer” the work to run on your system as necessary in order to have the work execute based on its priority to the business without exceeding your system’s capabilities.

This section discusses some of the current best practices in how to approach the investigation and design of any workload management implementation.

Understand existing systems

These principles apply as much to a database that is being readied for release into production as they do to one that has already been deployed. The advantage of existing systems is that they can be monitored and mined for the answers to help you understand the work to be run on the target system.

Understanding the work that will run on your database is a critical piece of designing an appropriate workload management implementation. This information tells you what type of contention and interactions you might expect which, in turn, helps you decide what features and capabilities of DB2 workload management interest you. To better understand the types of information available to you, see “Best Practices: Tuning and Monitoring Database System Performance.”

To gather information about work that runs on an existing DB2 system, take advantage of the new monitoring capabilities, such as those described in the monitoring section of this document, to gather information from the default DB2 workload and user service class that are available once DB2 V9.5 is installed.

If you are using Query Patroller (QP) on an existing system, get a head-start on acquiring the answers by using the information stored in the QP control tables. These tables show you the sources of work and the type of work being submitted.

Assigning business priority to database work

Just as important as knowing what work will be executed, is knowing what its business priority is relative to all the other work on the system, what performance objectives (if any) exist for that work, and how the business priorities map to the submitted database requests. Sometimes the business priority, or expectation, is expressed in the form of a formal service level agreement (SLA). Sometimes the priority is simply not known at all...
or is expressed in very informal, relative terms (for example, application A is more important than application B).

**Start by mapping applications to business priorities:** In terms of business priority, it is simplest to start thinking in terms of a simple classification system such as high, medium, and low priority. You can then begin to look at the different business processes that submit work and classify each of them as belonging to one of those priority classes. Some business processes have multiple levels and paths within them, not all of which share the same high level classification given to the primary process. For example, a high priority on-line application might have some ancillary reporting or maintenance paths that do not share the same high priority as the main application. The degree to which applications are uniquely identified with business processes dictates, to a large extent, whether you can use simple mapping of business priority directly to an application or whether you must parse the requests submitted by an application to potentially assign different priority to each of them.

Business priority is best reflected by having a DB2 service class defined for each different class of business priority. A service class is where the database work is actually executed and is the best place to control the importance assigned to that work. The assignment of business priority to incoming work, that is, the assignment of incoming work to different DB2 service classes, is best done using DB2 workload definitions and DB2 work action sets.

In those cases where you can directly apply the business priority to the work submitted by a specific application, map the workload to a DB2 service class that reflects the business priority assigned to the application. To map the workload, create a DB2 workload to represent the connections made by that application and point that workload directly at the DB2 service class which represents the appropriate business priority for that application.

In other cases, such as when dealing with some middleware applications, you might be unable to assign one business priority to a specific application because the application services multiple processes of different priorities at the same time and it is not always possible to distinguish between the different users being serviced. In these cases, look at the individual requests being submitted and apply business priority to each piece using one of the available techniques. The mapping of business priority at the level of individual requests by directing those requests to the appropriate DB2 service class is best implemented using the mapping mechanism available as part of the DB2 work action set mechanism (as it is used within a DB2 service superclass).

**An alternative method for mapping business priorities:** If you do not have an appropriate level of information about the work being submitted to the database, you can choose to use an alternate approach to mapping business priority to database work. Approach the problem of priority assignment from a pure database perspective. To this end, classify the work by the expected impact, or life time, on the server using a short-medium-long classification system in which you treat short requests as being the most important and long ones as least important. In this paradigm, the thinking is that the more short queries that you can push through your system, the better the service
provided to the whole enterprise. In other words, the philosophy here is that it is better for the system as a whole to have long queries run slightly slower than it is to have short queries suffer a large degradation in performance.

However, even in this paradigm, the same issues arise. The degree to which an application submits only one class of request determines whether you can use simple mappings or have to classify requests on an individual basis. Again, the approach to determining the relative impact of the submitted work will be reflected in the workload management mechanisms you choose to use. An application submitting only one type of work can be mapped directly to a service class set aside for that class of work by using the normal DB2 workload to DB2 service class mapping. The sorting of work and mapping of individual requests to the appropriate DB2 service class based on their projected impact is again best implemented using the mapping mechanism of the DB2 work action set.

Obviously, a mix of these two classification systems is possible, as are any number of other ones. The key conclusion to be reached is that, regardless of the classification system used, the extent of the alignment between the individual applications and the business priority classification system being used determines which DB2 workload management mechanism best informs the database of that priority.

**Take advantage of your experience**

If you have an existing system, then you have vital knowledge on how it works and the problems it has experienced in the past. The less you know with certainty about the underlying root causes of these difficulties, then the less aggressive you should be in implementing an initial new workload management approach. In such cases, plan on a good, starting implementation such as suggested in the next section and plan on multiple iterations of monitoring followed by tuning until the system is stable and under control.

Also, use your experience to ensure that all the factors that are active on your system are considered. Look at the system holistically; not every problem should be solved using just the capabilities of DB2 workload management. Consider the full arsenal of DB2 features and capabilities at your disposal. These other features and capabilities of your DB2 data server can and should be leveraged alongside DB2 workload management to ensure your performance objectives are being met.

A predictable, stable data server requires the following things:

- Good logical and physical design.
  - For more information see the following documents:
    - “Best Practices for Physical Database Design” and the Balanced Warehouse documentation
    - “DB2 for Linux, Unix, Windows Performance Tuning and Troubleshooting: Best Practices”
• Queries that are tuned to yield the best performance.
  ○ For more information see “Writing and tuning queries for optimal performance”

• Use of other features that can aid performance, like self-tuning memory and utility throttling.

The knowledge and tools that you have been using for years are not suddenly irrelevant, they just need to be tempered with the new capabilities in DB2 workload management. By combining DB2 workload management with these other features and capabilities, you can create stable, predictable DB2 data servers that can perform reliably even at times of peak demand. Use your experience and judgment to pick the right tool for the job.

**Start simply and change slowly**

As with any new capability, it is usually not a good idea to implement a complex design or to make many different changes at the same time. Complexity and churn often hide the real problem and sometimes introduce new problems.

In the case of the control capabilities of DB2 workload management, start with a simple, straight-forward implementation and evolve it one change at a time. This approach allows you to fully understand and appreciate the impact of each change as well as detect any unintended consequences or interactions.

As well, it is best to keep the scope of any controls such as those provided by DB2 thresholds as narrowly focused as possible, as opposed to applying them globally. Some broad database level thresholds might be desired to govern the global behavior of the system but, in general, keeping the thresholds focused on the problem areas ensures that you see only the effects that you intended. Usually the most effective place to assign a DB2 threshold to work is on a DB2 service subclass.

**Best practices: Implementation**

This section provides the current best practices for implementing each of the different capabilities provided with DB2 workload management introduced in DB2 V9.5.

**DB2 workloads**

A DB2 workload should be defined for each possible source of work (that is, a database connection) that is of interest to you, whether it be for an application, a user, a specific department, or whatever.

There is no performance penalty for introducing new workload definitions, and each workload definition allows you to monitor or control a specific part of the incoming work on your system. Workload definitions also allow you to implement future changes quickly since you will already have the mechanism in place to identify the specific set of connections that you want to affect with the change.
Middleware applications can present a challenge for identification as many of them interact with the database using the same authorization ID and identical connection information. In such cases, the database cannot identify the end-user behind these requests without additional information.

Have the middleware application support the identification of end-user applications by using the modifiable client information fields (for example, CLIENT USERID, CLIENT APPLNAME, CLIENT WRKSTNNAME, and CLIENT ACCTNG). These fields can be set by any application connecting to the DB2 data server by using the various DB2 client connection options or by invoking the WLM_SET_CLIENT_INFO stored procedure.

Most popular middleware applications either provide the ability to set client information fields or a way to inject user provided SQL into strategic locations during processing. These fields provide a way to clearly identify many external aspects of the business process and organization submitting the work to the DB2 data server. The fields can also be used to create unique DB2 workload definitions.

Client information is very useful in many problem determination and monitoring scenarios, so that it is important to establish a standard format and usage expectation for these fields within your business in order to ensure that this information is readily recognized and understood by the people it is intended to help.

Without the ability to create unique workloads for connections of interest, you must rely on the mechanisms such as the work action set to segregate and treat the work being submitted solely based on the actual characteristics of the work such as type (for example, DML or DDL, READ or WRITE, etc.) or cost estimates. This severely limits your future flexibility and your ability to take advantage of the fine-grained monitoring capabilities provided by workload management.

**DB2 service classes**

When you install DB2 V9.5, a default user service class is created automatically and this default user service class is where all work submitted to the database runs (via the mapping of connections to the default workload definition that is also created at installation time). This is your starting point and any additional DB2 workloads that you create automatically point to this default service class unless a new one is explicitly identified.

This section provides recommendations for creating new DB2 service classes and pulling work out of the default service class so that it can be treated differently in terms of business priority. Each DB2 service class that you define gives you a point of control for defining resource priorities used for execution as well as the ability to do lightweight monitoring of executing work. In contrast, the more service classes you define, the more complicated your system becomes to monitor and control as a whole.

You might choose to keep the default service class in use after your implementation is complete as a place to process unmapped or unimportant work. If you choose to do this, determine what business priority should be assigned to such work and properly control
the resources for the default user service class in addition to any others that you might create.

As discussed in the “Assigning business priority to database work” section, the DB2 service class is the best way to reflect business priority within the database. This process is often simplified by implementing a fairly straight-forward classification system where incoming work is assigned to a service class representing either a HIGH, MEDIUM, or LOW business priority or a short, medium or long expected duration.

To implement a HIGH, MEDIUM, or LOW business priority classification system, create a DB2 service superclass and two additional service subclasses within that superclass.

Each of these service classes represents a different business priority with the default subclass for the superclass also being the default home for any unmapped work submitted to the superclass. Depending on how you wish to treat such work, you will most likely assign the default subclass to represent the LOW or MEDIUM priority class of work. The example implementation in this document assigns the LOW classification to the default subclass created for the superclass and the MEDIUM and HIGH classification to the two explicitly created subclasses. See “Figure 1. Recommended service class implementation” for an example.

![Figure 1. Recommended service class implementation](image)

Resources for each of these different service classes can reflect the relative priority by setting the service class attributes to values that reflect the corresponding priority (for example, the LOW business priority service subclass could get LOW prefetch priority and a low relative processor priority setting while HIGH business priority subclass could get a HIGH prefetch priority and a high relative processor priority). In general, start with having the MEDIUM business priority class reflect the original default (that is, unmodified) settings of the superclass as this is the normal setting for execution within DB2 databases.

---

1 Note that the default attribute settings of any service subclass simply reflect the values of the superclass.
Do not set the service class processor priority of any service class higher than the DB2 default system service class, either directly using the processor priority attribute or indirectly through AIX Workload Manager integration.

The default behavior in this approach is such that anything mapped by a DB2 workload to the service superclass A executes in the default subclass and picks up the LOW priority settings. In order to acquire different priority settings, the incoming work needs to be mapped either to the other service subclasses using a DB2 work action set or directly to the desired service subclass. Refer to “Figure 2. Example of assigning work to different service subclasses” for an example.

![Figure 2. Example of assigning work to different service subclasses](image)

The default service subclass is assigned the lowest or medium priority because any work not recognized or not remapped within a DB2 work action set applied to the superclass is automatically assigned the lowest (or medium) priority², as long as the DB2 workload is mapped to the superclass. In general, high priority is reserved for special work rather than the ordinary run-of-the-mill work submitted to the database. You can also directly map incoming work from a workload to a subclass itself in order to assign that work that priority and by doing so, that work bypasses any DB2 work action set on the superclass. Please note that you cannot directly assign a workload to the default subclass, which should be taken into consideration when deciding which subclass will represent what business priority.

You can use this recommended approach as a template for both the monolithic approach to managing work on your database (that is, all work on the database is processed within one superclass) and for more diversified approach where different groups of work are managed using different approaches (for example, one set of work is managed using business priority and another using expected duration).

Once you have set up the DB2 service classes and workloads to reflect how you want work to be processed, validate your implementation by running known workloads and

---

² Recall that all work on DB2 actually executes within a subclass and that any work assigned to the superclass will execute in the default subclass of that superclass.
comparing the various request and activity counts for each workload and service class to ensure that things are flowing as expected. You can also use this time to test the effects of the implementation on the processing patterns. You can validate this by using either the WLM table functions to interrogate DB2 directly or the WLM statistics event monitor to collect the statistics of the different WLM objects to a table for further analysis.

Integration with AIX Workload Manager service classes
If you are running DB2 V9.5 on the AIX operating system, you have the option of integrating the DB2 service classes with complementary AIX Workload Manager (WLM) service classes. Doing this enables you to take advantage of the native control and monitoring capabilities provided by AIX WLM.

AIX WLM offers many different ways to control processors and you must experiment to see which one, if any, is right for you. The key thing to keep in mind is that the AIX WLM processor control mechanisms, with the exception of hard maximum limits, operate on the assumption that lower priority work needs to be restricted only when processor cycles are in high demand. When the processor is not constrained, most AIX WLM control mechanisms have little effect. Conversely, the more that the processor is utilized, then the greater the effect that they have.

Recommendation for controlling processor consumption on AIX systems: When integrating with AIX Workload Manager for processor control purposes, use hard maximum processor use limits unless your system is always heavily processor constrained at those times when you want the control to be effective.
**Hard maximum limits on processor use** strictly control processor consumption regardless of the degree of processor utilization. Low-priority workloads are controlled at all times and thus benefit high-priority work even in environments that are not processor constrained. Our testing and benchmark experiences with AIX WLM have borne this out.

The primary limitation of hard maximum limits is that the settings are not flexible and do not allow for dynamic borrowing of processor time by the low-priority work if no high-priority work is present.

One approach that we have found useful to help ameliorate this limitation and simulate some of the same benefits of the soft maximum limits on processor use is to run a script in the background that dynamically adjusts the hard maximum limit settings depending on the current utilization rate in each service class. As the high-priority work begins to use less processor time, the settings are adjusted to allow the low-priority work to use more, and the reverse happens as high-priority processor use begins to climb. An example of this script is provided in the appendix “Moderating AIX Workload Manager hard maximum processor use limits dynamically.”

Our experiences in some benchmarks have shown that an effective use of AIX WLM service classes is to implement a firm allocation of processor resource between different parts of the enterprise by using hard maximum processor limits. Within each of these AIX WLM service classes, a number of DB2 service classes, all referring to the same AIX service class, are then introduced to further subdivide each firm allocation for different purposes. So far, we have not encountered a scenario requiring more than 2 or 3 AIX WLM service classes to adjust processor use priority.
One advantage of having an AIX WLM service class defined is that you can use operating system statistics for the AIX WLM service class to supplement the statistics gathered from the DB2 data server. In cases where you have a 1:1 mapping between DB2 workload management service classes and AIX WLM service classes, you can directly attribute the AIX statistics to the work running in that DB2 service class.

**DB2 work action sets**

A DB2 work action set is the mechanism provided in DB2 V9.5 to distinguish between different types of work and treat them differently. There are many different reasons for wanting to introduce a work action set including the reasons already mentioned in this paper.

Follow these best practices when defining a work action set (and the prerequisite DB2 work class set):

1. Minimize the number of work class sets; ideally, you should need only one work class set regardless of the number of existing work action sets.

2. Ensure that the individual work class definitions do not overlap with each other to prevent confusion over work classification; if they intentionally overlap, make sure that the naming of overlapping work class definitions indicates their shared or common focus.
3. Ensure that the order of your work action set definition is correct and that work is properly classified and handled.

4. Validate your work action set by running known workloads and comparing the counts for each work class; check the contents of the default work class (represented by the asterisk) for the work action set statistics to see if any work is not being properly classified.

**DB2 thresholds**

During implementation, follow the best practices mentioned for thresholds in the design section of this paper to keep the scope of the thresholds as limited and as well-defined as possible. The primary best practice for the implementation of any DB2 threshold is to ensure that the threshold violations event monitor is created and activated; otherwise, you will not know when and what thresholds are being encountered.

For any threshold that affects activities, always collect detailed activity information if a threshold is going to stop execution of an activity in order to allow an effective follow-up investigation to be performed.

**Using the CONCURRENTDBCOORDACTIVITIES threshold**

The CONCURRENTDBCOORDACTIVITIES threshold specifies the maximum number of recognized coordinator activities\(^3\) that can run concurrently on the database. This threshold, a concurrency control threshold that also allows queuing, is a very effective and powerful tool that can be used to sequence and limit the amount of work being asked of the database at any particular time.

Since it enforces the specified concurrency rate across the database as a whole, the application of this threshold has slightly higher overhead than most other thresholds. In general, only apply this threshold at the level of a service subclass. If it does need to be applied at the database level (for example, to control activities that can be disruptive, such as load operations), then apply it very selectively using a database level work action set to minimize the work affected by the threshold.

Applying this threshold reduces or limits the impact of a group of work on the database system to allow other work to take advantage of the freed resources. If you find that higher priority work is not performing as desired when there is other lower priority work being processed at the same time, consider imposing a concurrency control on lower priority work until the performance objectives of the higher priority work are met.

**How to get started:** Start with the service class containing the lowest priority work and define a CONCURRENTDBCOORDACTIVITIES threshold on it. Depending on the needs of your system, your initial stating value for the threshold might actually not be low enough and you will want to reduce it in order to increase the resources available to higher priority work. You can repeat this iteratively while monitoring the higher priority work until you find the right balance.

\(^3\) In DB2 V9.5, a recognized activity is a DML statement, a DDL SQL statement, or a load operation.
In some cases, the low priority threshold concurrency limit reaches the value of 1 and, assuming that you do not want to actually stop that work completely, the next stage of reducing the amount of lower priority work running on the system is to leave that particular threshold set to 1 and move up to the service class containing the next lowest priority work to repeat the same series of steps.

Always define the low priority threshold with an unbounded queuing limit so that new activities queue rather than fail and the threshold is never violated.

Where possible, try not to define a concurrency threshold on the highest priority work as the extra overhead might do more harm than good, especially if the work tends to be of short duration. In general, applying a concurrency threshold to the heaviest work has far more benefits than applying it to lighter work. In some cases, a threshold on high priority work is also be needed if the sheer amount of work executing on the database system is larger than the system resources can handle.

In those cases, consider an 85%, 10%, and 5% breakdown in the proportion of concurrency rates between the service classes containing HIGH, MEDIUM, and LOW priority work. This emphasizes allowing more of the higher priority work to get through while still permitting lower priority work to make some progress.

The CONCURRENTDBCOORDDACTIVITIES threshold is not the same as the concurrency control used by the Query Patroller query class mechanisms. The CONCURRENTDBCOORDDACTIVITIES threshold controls all direct and indirect activities, where indirect activities are nested activities such as SQL statements run within a stored procedure. The CALL statement is one activity and any nested SQL is another activity.

In some scenarios where the CONCURRENTDBCOORDDACTIVITIES threshold is used, applications that start more than one concurrent activity can potentially consume all the concurrency allowed by this threshold, creating a queue-deadlock scenario.

Two simple examples of such scenarios are:

**Scenario #1** (assume a concurrency threshold of 2)
1. Application A opens a cursor (ticket #1 is consumed)
2. Application B opens a cursor (ticket #2 is consumed)
3. Application A issues update where current of statement (queued)
4. Application B issues update where current of statement (queued)

**Scenario #2** (assume a concurrency threshold of 2)
1. Application A calls a stored procedure STPA (ticket #1 is consumed)
2. Application B opens a cursor (ticket #2 is consumed)
3. STPA opens a cursor (queued)
4. Application B issues update where current of statement (queued)

It is critical to always be aware of what you are queuing in order to ensure that what is being queued is appropriate for this specific concurrency threshold.

When using the CONCURRENTDBCOORDACTIVITIES threshold, always have an accompanying ACTIVITYTOTALTIME activity threshold affecting the same work being queued to ensure that any queue-deadlock scenarios eventually get detected and resolved when the ACTIVITYTOTALTIME threshold expires for the queued activities.

If you are using stored procedures or other SQL routines invoked by the CALL statement and want to impose a concurrency threshold on the nested work but not on the CALL statement itself, consider the following alternative implementation, which might be acceptable in some contexts:

1. Define the standard service class environment as recommended earlier with a superclass consisting of 3 subclasses representing different business priorities.

2. Apply concurrency thresholds to these subclasses as needed.

3. Add an additional subclass to represent CALL statements without any concurrency threshold applied to it.

4. Introduce a work action set which maps all CALL statements to the subclass set aside for CALL statements but use the WITHOUT NESTED clause in the work action definition. This clause dictates that any subsequent SQL statements issued within the called routine must themselves pass through the work action set where they can be mapped to different service classes than their parent.

5. Route all workloads to the shared superclass.

In this implementation, the CALL statements themselves have no concurrency control placed on them but all their children have such controls imposed. See “Figure 5. Example of an alternate CONCURRENTDBCOORDACTIVITIES threshold implementation” for an overview of this alternative.
If you suspect a possible queue-deadlock situation, use the WLM_GET_QUEUE_STATS table function or the db2pd command to query the latest statistics for the concurrency thresholds involved to help confirm or deny your suspicion. If you are gathering WLM statistics automatically, use this information to detect the lack of movement in the concurrency threshold queues.

If a queue-deadlock scenario does arise and you do not have an ACTIVITYTOTALTIME activity threshold defined (or do not want to wait for it to be violated), resolve this scenario in two ways:

- Use the ALTER THRESHOLD statement to raise the concurrency rate.
- Cancel some of the executing or queued activities using the WLM_CANCEL_ACTIVITY procedure until the problem is resolved.

If you want to remove the concurrency threshold completely:

1. Disable the threshold first to prevent new activities from being queued.
2. Alter the threshold to raise the concurrency rate to allow the queued activities to begin executing.
3. Once the queue is empty, drop the threshold.

Finally, in some cases, it might be possible to use the TOTALSCPARTITIONCONNECTIONS threshold as an alternate means of imposing control. This threshold affects new connections at a the same database partition so the compatible scenario would require a single database partition where all connections are established (for example, the administration partition in a balanced warehouse).
configuration) as well as connections that actually disconnect when their work is completed.

Best practices: Monitoring

Monitoring is a critical part of the initial design and ongoing tuning of any successful workload management implementation. This section does not tell you what or how to monitor as those answers depend on the specific scenario, but it does suggest some best practices that cut across almost all scenarios and which can make monitoring much easier overall.

The first best practice for monitoring is to create all of the new workload management event monitors introduced with DB2 V9.5, specifically the activity, statistics, and threshold violations event monitor. Unlike other event monitors, the creation and activation of workload management event monitors does not cause any events to be generated. Instead, event generation is defined on the individual workload management entities such as the service classes and workloads. By creating these event monitors, you do not impose any overhead on the system and you set up the monitoring environment such that you can call the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure or alter the collection attribute of an entity, such as a service class, at any time and can instantly capture the desired information.

It is very important to understand the behaviors and workload of your database system. Turn on the collection of aggregate activity data for all your service classes at the BASIC level, and set the WLM_COLLECT_INT database configuration parameter to have all the workload management statistics collected to the statistics event monitor at regular time intervals that are best suited for your monitoring needs.

What these settings do is to collect all the statistics kept in DB2 memory, at regular intervals, for each of the different workload management objects, write all of the statistics to the workload management statistics event monitor, and then reset the internal statistics for use in the next interval. The overhead of gathering all the statistics and aggregate activity data is negligible, while the value in helping you follow the pattern and behavior of work within your system is immense. Turning on aggregate activity data and collecting the available workload management statistics on a regular basis provides you with an ongoing profile of your database system performance for various time periods, like the day or the week, which can be used to pinpoint problem time periods and understand ongoing trends and patterns. This information is also very valuable for helping you to determine what threshold values, if any, need to be set to control unexpected behaviors within your system. For more information on statistics, see “DB2 workload management histograms, Part 3: Visualizing and deriving statistics from DB2 histograms using SQL,” or consider using the trend analyses Performance Expert can provide. The “Further reading” section might also be helpful in providing additional pointers.
While people often want to capture the complete details of a statement’s execution, it is often unacceptable to the same people to impose any additional overhead on those same statements to collect the statistics.

On systems where the statements are very short or the volume of concurrent statements is high, avoid collecting detailed activity data for all service classes at the same time.

If you want detailed activity data but your workloads or systems cannot accept the overhead of collecting detailed information for all statements, approach your data collection more selectively by collecting individual activity information on selected workloads and by limiting the collection to a specific duration. Think of it as SQL sampling for that workload. Volume and overhead permitting, the same concept can be applied at the service class level if the workload level is too granular for your monitoring needs.

Using this approach, you control which work is affected and for how long, while still capturing the detailed information to the activity event monitor. Feeding this data to the DB2 design advisor or running the historical reports on this data using the wlmhist.pl script provided with DB2 V9.5, will produce meaningful results for both the individual samples or, if you wait until you have sampled everything, the database as a whole.

Prune any event monitor output tables as needed, and archive important information for future reference.

Do not backup the event monitor tables themselves directly during production as this interferes with the writing process and affects system performance or your monitoring capabilities during that time. Take the data out of the event monitor table and put it in a secondary location, then back up that secondary source.

On heavily utilized systems, consider setting up a separate service class for using workload management SQL table functions by your monitoring application or connections so that the queries can be given a higher priority and better response times.

When modifying the workload management settings themselves, it is a good idea to consider always using the SYSDEFAULTADMWORKLOAD workload definition when submitting such requests if you want the requests to bypass any concurrency thresholds that might be in place (but note that you cannot assign this workload to a separate service class).
Conclusion

Workload management is one of the key features needed to create a stable, predictable DB2 data server. You can leverage DB2 workload management provided in DB2 V9.5, along with the other features and capabilities of your DB2 data server, to ensure your business objectives can be met by a data server that performs to expectations. If you do follow the best practices outlined for DB2 workload management here, you should be off to a successful start to your workload management implementation!
Further reading

- Tutorial for workload management -

- IBM DB2 Database for Linux, UNIX, and Windows Information Center -
  http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp

- IBM Data Warehousing and Business Intelligence - http://www-306.ibm.com/software/data/db2bi/

- WLM FAQ in DB2 V9.5 documentation:

- WLM Hands On Tutorial in DB2 V9.5 documentation (partially) and available for download on IBM Developer Works:


- IDUG 2008 Session E04: DB2 9.5 Workload Manager: A Handyman’s Tour - Explores less well-known aspects of DB2 Workload Manager in DB2 9.5

- IOD 2007 Session 1656: Managing Mixed Workloads in the Data Warehouse - Covers enhancements in DWE Design Studio to aid in using DB2 Workload Manager and also shows sample scenarios

- IOD 2007 Session 1480: Workload Management and Monitoring with DB2 Warehouse - Covers enhancements in DB2 Performance Expert to take advantage of some of the features of DB2 Workload Manager

- DB2 for Linux, Unix, and Windows Best Practices –
Contributors

Tim Vincent

Chief Architect DB2 LUW
APPENDICES

Scenario: Best practices in a data warehousing environment

The best practices described earlier apply across many different scenarios. This section describes how you can use DB2 workload management and AIX Workload Manager in a data warehouse environment.

To get started, ask yourself the following questions to help you identify different aspects of the work running in your environment:

- With what level of granularity do you want to control the work? Classify queries into multiple categories based on criteria like department, user ID, or application. Consider what activities, users, and applications should have higher priority compared to other work.

- What queries or activities are undesired or not permissible? For example, you might consider a query too expensive if it takes longer than 1000 seconds to complete.

- For how much work is your environment sized? How many queries can run concurrently without system overload?

- What would you like to know about your work profile? That is, what should you begin monitoring? You might ask the following questions:
  - What applications ran the most expensive queries?
  - What query needed the most CPU?
  - How many queries ran longer than 5 minutes?

Understand the work your data warehouse server performs

If you have not yet gained a good understanding of the work running on your environment, monitor it in order to obtain its work characteristics before you customize DB2 workload management. Examples of characteristics of interest include user names, application names, concurrency levels, and resource consumption. To understand the work being performed on your data server, use histograms that show the distribution of activity estimated cost and runtime.
For example, in a default workload management configuration, the following statements create an event monitor, activate it, and alter the service class to enable extended aggregate activity data statistics collection:

```
CREATE EVENT MONITOR DB2STATISTICS FOR STATISTICS WRITE TO TABLE;
SET EVENT MONITOR DB2STATISTICS STATE 1;
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED;
```

Now run a typical workload or wait until already running activities have completed to collect some statistics. Next, flush the statistical data to the HISTOGRAMBIN_DB2STATISTICS table with the following command, which also resets the statistics data in memory:

```
CALL WLM_COLLECT_STATS;
```

To see the distribution of runtimes, query the histogram table to return a CoordActExecTime histogram that shows the execution times of non-nested activities at the coordinator partition. Use the results primarily to identify activities that run longer than one second; sub-second queries typically do not require workload management in a data warehouse environment:

```
SELECT TOP/1000 AS TIME_seconds,
       SUM(NUMBER_IN_BIN) AS #EXECUTIONS
FROM HISTOGRAMBIN_DB2STATISTICS
WHERE HISTOGRAM_TYPE = 'CoordActExecTime'
GROUP BY TOP/1000 order by TOP/1000
```

The output might look like this:

<table>
<thead>
<tr>
<th>TIME_SECONDS</th>
<th>#EXECUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>385</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>38</td>
<td>6</td>
</tr>
<tr>
<td>59</td>
<td>22</td>
</tr>
<tr>
<td>89</td>
<td>21</td>
</tr>
<tr>
<td>136</td>
<td>12</td>
</tr>
<tr>
<td>208</td>
<td>8</td>
</tr>
<tr>
<td>317</td>
<td>0</td>
</tr>
<tr>
<td>483</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 6. Histogram graph of execution times of non-nested activities at the coordinator partition (CoordActExecTime)

When represented as a graph, the histogram shows that most queries completed in less than one second, some queries needed between several seconds and several minutes to complete, and two queries ran significantly longer. The accumulated execution time for short queries is much lower than the execution time of the long queries. Typically, runtimes correlate with the consumption of resources and if you look at the average accumulated execution time represented by the blue line in the graph, you can see that these long queries consume most of the available resources and should be controlled through workload management.

To help define query classes and thresholds per class that prevent high cost queries from monopolizing resources, look at the distribution of estimated costs for the same workload in histograms.
The following query returns a CoordActEstCost histogram:

```sql
SELECT TOP/1000 AS TIMERONS_in_THOUSAND,
       SUM(NUMBER_IN_BIN) AS #EXECUTIONS
FROM HISTOGRAMBIN_DB2STATISTICS
WHERE HISTOGRAM_TYPE = 'CoordActEstCost'
GROUP BY TOP/1000 order by TOP/1000;
```

For the current connections on the server, some work characteristics can be obtained by querying the SNAPAPPL_INFO administrative view (which imposes the overhead of an application snapshot when it is called). Alternatively, the workload management table functions can be used. Historical perspective can be acquired from a tool such as DB2 Performance Expert or by using the activity event monitor to capture information.

For example, the following query returns the number of current database connections and processor time used by agents for all partitions, grouped by application name and system authorization ID from the SNAPAPPL_INFO view:

```sql
SELECT A.DBPARTITIONNUM, COUNT(*) #CONNECTION,
       INTEGER(SUM(AGENT_USR_CPU_TIME_S+AGENT_SYS_CPU_TIME_S)) CPU_SECOND,
       SUBSTR(APPL_NAME,1,20) APPLNAME,
       SUBSTR(PRIMARY_AUTH_ID,1,16) SYSTEM_USER
FROM SYSIBMADM.SNAPAPPL_INFO A, SYSIBMADM.SNAPAPPL B
WHERE APPL_NAME NOT IN
     ('db2stmm','db2wlmd','db2taskd','db2evmg_DB2DETAILDEA')
AND A.AGENT_ID=B.AGENT_ID
AND A.DBPARTITIONNUM=B.DBPARTITIONNUM
GROUP BY APPL_NAME,PRIMARY_AUTH_ID, A.DBPARTITIONNUM
ORDER BY A.DBPARTITIONNUM,APPL_NAME;
```

This query returns the following output:
To gain a better historical perspective on database activities, use an activity event monitor to capture the information for you. A more detailed workload analysis can also be performed using a performance analysis and tuning tool like DB2 Performance Expert.

**Determine the connection attributes for use in workload definitions**

You can determine the connection attributes required to define workload objects for each application that is connected to the database by using the DB2 workload manager table functions. For example:

```sql
SELECT COUNT(*) COUNT, SUBSTR(APPLICATION_NAME, 1, 10) APPLNAME, SUBSTR(SYSTEM_USER,1,10) SYSTEM_USER , SUBSTR(CLIENT_USER,1,10) CLIENT_USER, SUBSTR(CLIENT_WRKSTNNAME,1,21) CLIENT_WRKSTNNAME , SUBSTR(CLIENT_ACCTNG,1,10) CLIENT_ACCTNG, SUBSTR(CLIENT_APPLNAME,1,10) CLIENT_APPLNAME FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) A GROUP BY APPLICATION_NAME, SYSTEM_USER, SESSION_USER , CLIENT_WRKSTNNAME, CLIENT_ACCTNG, CLIENT_USER, CLIENT_APPLNAME;
```

This query returns output that shows the number of the currently running applications along with the following connection attributes that can be used for DB2 workload definitions APPLNAME (application name), SYSTEM_USER (system authorization ID), SESSION_ID (session authorization ID), CLIENT_USER (client user ID), CLIENT_WRKSTNNAME (client workstation name), CLIENT_ACCTNG (client accounting string), and CLIENT_APPLNAME (client application name):

<table>
<thead>
<tr>
<th>COUNT</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_ID</th>
<th>CLIENT_USER</th>
<th>CLIENT_WRKSTNNAME</th>
<th>CLIENT_ACCTNG</th>
<th>CLIENT_APPLNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>db2batch</td>
<td>DB2INST3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>db2bp</td>
<td>DB2INST3</td>
<td>nela</td>
<td>boss</td>
<td>appl#1.torolab.ibm.c</td>
<td>123-456</td>
<td>-</td>
</tr>
</tbody>
</table>
To complete the list of connection attributes for a specific user, issue a query like the following, which returns the SESSION_USER GROUP (user group) and SESSION_USER ROLE (user role name) of the user DB2INST3:

```sql
SELECT GROUP as "SESSION_USER GROUP" FROM TABLE (SYSPROC.AUTH_LIST_GROUPS_FOR_AUTHID('DB2INST3')) AS T;
SELECT ROLENAME as "SESSION_USER ROLE" FROM TABLE(SYSPROC.AUTH_LIST_ROLES_FOR_AUTHID ('DB2INST3','U')) AS T;
```

Applications can set client information attributes with the WLM_SET_CLIENT_INFO procedure in order to record the identity of the application or end-user currently using the connection at the DB2 data server, which is necessary if no other distinguishing connection attribute can be used.

For example, the following statement sets the application identification information retrieved previously:

```sql
CALL SYSPROC.WLM_SET_CLIENT_INFO('nela',
    'appl#1.torolab.ibm.com', 'boss', '123-456', 'AUTOMATIC');
```

Once you have identified the work running on your data server, classify the applications and users into different groups according to their type and business priority. In a data warehousing environment, groups like the following example groups might apply:

- Daily reporting queries—identified by connection attributes like the system authorization ID
- Ad hoc or complex queries—identified by the client user ID or application name
- ETL jobs for real time data warehouses—identified by the application name.

**Obtain consistent response times with priority settings**

Consistency in response times is critical in a data warehouse environment. Where service level agreements are in effect, response times might be mandated.

Typically, at least two user or application groups exist in a data warehouse environment, a “light” group running simple or, at most, medium-expensive queries that have a short run time and need fewer resources (like daily reports), and a “heavy” or power user group running complex ad hoc queries that need significantly more resources. Some users might also be permitted to submit additional critical queries, which take priority over other work (sometimes called “VIP” or “CEO” queries).

The priority of a service class is compared against that of other service classes in order to determine resource allocation. To ensure consistent response times for short and critical queries, allow the queries to execute in a separate service class with higher priority. Place
complex queries submitted by power users into a separate service subclass with low priority.

To get started, create a service class with two subclasses, and then create two workloads that identify the more and the less important applications or users. Assign these workloads to the two different service subclasses:

<table>
<thead>
<tr>
<th>Service Class</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER</td>
<td>CREATE SERVICE CLASS POWER;</td>
</tr>
<tr>
<td>LOW_PRIO</td>
<td>CREATE SERVICE CLASS LOW_PRIO UNDER POWER;</td>
</tr>
<tr>
<td>HIGH_PRIO</td>
<td>CREATE SERVICE CLASS HIGH_PRIO UNDER POWER;</td>
</tr>
<tr>
<td>High_PRIO</td>
<td>CREATE WORKLOAD HIGH_PRIO APPLNAME('db2bp') SYSTEM_USER ('DB2INST3') SERVICE CLASS HIGH_PRIO UNDER POWER;</td>
</tr>
<tr>
<td>Low_PRIO</td>
<td>CREATE WORKLOAD LOW_PRIO APPLNAME('db2batch') SERVICE CLASS LOW_PRIO UNDER POWER;</td>
</tr>
<tr>
<td>High_PRIO</td>
<td>CREATE WORKLOAD HIGH_PRIO APPLNAME('db2bp') SYSTEM_USER ('DB2INST3') SERVICE CLASS HIGH_PRIO UNDER POWER;</td>
</tr>
<tr>
<td>Low_PRIO</td>
<td>CREATE WORKLOAD LOW_PRIO APPLNAME('db2batch') SERVICE CLASS LOW_PRIO UNDER POWER;</td>
</tr>
</tbody>
</table>

Commit after each statement, or use autocommit. The application-name is case sensitive, and the user name must be specified in upper case.

Next, assign the highest possible priority to applications in the service subclass HIGH_PRIO. You can assign agent priority and prefetch priority.

<table>
<thead>
<tr>
<th>Service Class</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH_PRIO</td>
<td>ALTER SERVICE CLASS HIGH_PRIO UNDER POWER AGENT PRIORITY -20 PREFETCH PRIORITY HIGH;</td>
</tr>
<tr>
<td>LOW_PRIO</td>
<td>ALTER SERVICE CLASS LOW_PRIO UNDER POWER AGENT PRIORITY 20 PREFETCH PRIORITY LOW;</td>
</tr>
</tbody>
</table>

Assign the lowest possible priority to applications in the service subclass LOW_PRIO.

Applications that do not belong into either the LOW_PRIO or HIGH_PRIO workloads are mapped to the default workload and thus, to the default user service class SYSDEFAULTUSERCLASS which has been left at the default priority (PREFETCH PRIORITY MEDIUM and AGENT PRIORITY 0).

To ensure that system work can take precedence over user work, set the agent priority of the system service class to be as high as or higher than the highest agent priority you set for the user service classes:

<table>
<thead>
<tr>
<th>Service Class</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>ALTER SERVICE CLASS SYSDEFAULTSYSTEMCLASS AGENT PRIORITY -20 PREFETCH PRIORITY HIGH;</td>
</tr>
</tbody>
</table>

If you want to see what your current DB2 workload management settings are after creating or altering workload management objects, look at the system catalogs or use the db2look command. For example, the following command creates a wlm.definitions.out output file that contains your current DB2 workload management settings for the database PROD:

```
db2look -d PROD -wlm -o wlm.definitions.out
```

**Define work action sets to distinguish between types of work**

If a user-defined workload sees different types of work, you can use work action sets to distinguish between these types of work and treat them differently, as shown in figures 2 and 3 of the "DB2 service classes" section.

To use work action sets, you assign the workload to a service superclass, not a subclass, with a statement like the following:

```
CREATE WORKLOAD ALL_PRIO APPLNAME('db2bp') SYSTEM_USER ('DB2INST3') SERVICE CLASS POWER;
```

To distinguish short from medium, and medium from long work, create a work class set that defines the work type criteria. For example, for the READ type, use the query cost in timerons estimated by the optimizer:

```
CREATE WORK CLASS SET control_cost
  (WORK CLASS long
   WORK TYPE READ FOR TIMERONCOST FROM 2000001 To UNBOUNDED,
   WORK CLASS medium
   WORK TYPE READ FOR TIMERONCOST FROM 20001 TO 2000000,
   WORK CLASS short
   WORK TYPE READ FOR TIMERONCOST FROM 0 To 20000);
```

Now you can use the work class set to create a work action set to map the work to the HIGH, MEDIUM, and LOW priority subclasses of the POWER superclass.

```
CREATE WORK ACTION SET query_cost FOR SERVICE CLASS POWER
  USING WORK CLASS SET control_cost
  (WORK ACTION MAP_LONG ON WORK CLASS long
   MAP ACTIVITY TO LOW_PRIO,
   WORK ACTION MAP_SHORT ON WORK CLASS short
   MAP ACTIVITY TO HIGH_PRIO);
```

You can also use the same work class set to limit the concurrency of expensive types of work (similar to the Query Patroller approach, and possible only at the database level):

```
CREATE WORK ACTION SET query_concur FOR DATABASE
  USING WORK CLASS SET control_cost
  (WORK ACTION limit_long ON WORK CLASS long
   WHEN CONCURRENTDBCOORDACTIVITIES > 2 CONTINUE,
   WORK ACTION limit_medium ON WORK CLASS medium
   WHEN CONCURRENTDBCOORDACTIVITIES > 2 CONTINUE);
```
WHEN CONCURRENTDBCOORDACTIVITIES > 10 CONTINUE,
WORK ACTION no_limit_short ON WORK CLASS short COUNT ACTIVITY);

**Integrate with AIX Workload Manager**

In this next example, DB2 workload management service classes are mapped directly to AIX Workload Manager (WLM) classes in order to integrate DB2 workload management with AIX WLM. When this is done, the agent priority settings of the DB2 service class are ignored by the DB2 data server. To map the previously created DB2 service classes to the AIX WLM classes, which are defined later, issue the following statements (this step could also have been included as part of the original CREATE statements):

```
ALTER SERVICE CLASS POWER OUTBOUND CORRELATOR 'Power';
ALTER SERVICE CLASS LOW_PRIO UNDER POWER AGENT PRIORITY DEFAULT OUTBOUND CORRELATOR 'Power.LowPrio';
ALTER SERVICE CLASS HIGH_PRIO UNDER POWER AGENT PRIORITY DEFAULT OUTBOUND CORRELATOR 'Power.HighPrio';
ALTER SERVICE CLASS sysdefaultsystemclass agent PRIORITY DEFAULT OUTBOUND CORRELATOR 'DefSystem';
```

For better monitoring, map all default DB2 service classes to AIX WLM classes:

```
ALTER SERVICE CLASS "SYSDEFAULTMAINTENANCECLASS" OUTBOUND CORRELATOR 'DefMaint';
ALTER SERVICE CLASS "SYSDEFAULTUSERCLASS" OUTBOUND CORRELATOR 'DefUser';
```

You must have root-level user privileges to configure and enable AIX WLM.

To begin, activate the correct AIX WLM configuration or create a new AIX WLM configuration. To create a new AIX WLM configuration called db2workload, issue the following command (or use a tool that can do the same for you):

```
 cp -r /etc/wlm/template /etc/wlm/db2workload
```

The current configuration of AIX WLM is the one in the directory pointed to by the symbolic link /etc/wlm/current. To make /etc/wlm/db2workload the current configuration, issue the following command:

```
wlmctrl -d db2workload
```

This command updates the /etc/wlm/current symbolic link to point to /etc/wlm/db2workload and starts AIX WLM.
To match the DB2 workload objects created earlier, an AIX WLM class `db2Power` and two subclasses called `db2HighPrio` and `db2LowPrio` must be created.

These objects will map to the DB2 service subclasses `HIGH_PRIO` and `LOW_PRIO`, respectively:

```plaintext
mkclass db2Power
mkclass db2Power.HighPrio
mkclass db2Power.LowPrio
mkclass db2DefSystem
```

In order for AIX Workload Manager to accept the outbound correlators, define application tags. These application tags ensure the automatic assignment of the DB2 services classes to the related AIX classes. To define the application tags, edit the appropriate rules file for the AIX super- and subclasses and add the new outbound correlators as application tags.

After editing, the rules file that applies to the superclasses will look like this:

```plaintext
/etc/wlm/db2workload vi rules
* IBM_PROLOG_BEGIN_TAG
* This is an automatically generated prolog.
* *
* bos530 src/bos/etc/wlm/rules 1.2
* *
* Licensed Materials - Property of IBM
* *
* (C) COPYRIGHT International Business Machines Corp. 1999,2002
* All Rights Reserved
* *
* US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
* *
* IBM_PROLOG_END_TAG
* class  resvd user group application type tag
   System  -     root  -  -  -       -       -       Power
   db2Power  -  -  -  -  -       -       -       HighPrio
   db2Power  -  -  -  -  -       -       -       LowPrio
   db2DefSystem  -  -  -  -  -       -       -       DefSystem
   db2DefUser  -  -  -  -  -       -       -       DefUser
   db2DefMaint  -  -  -  -  -       -       -       DefMaint
   Default  -  -  -  -  -       -       -       -
```

Always leave the `System` class listed above the user defined classes, so that system processes are mapped to their default service classes. Also, leave the `Default` user class listed below your user defined classes.

The rules file that applies to the subclasses of the `db2Power` superclass should look like this:

```plaintext
```
Use the `wlmcntrl` command to update the definition of the AIX Workload Manager with this configuration:

```
wlmcntrl -ud db2workload
```

Once AIX WLM is started, any DB2 work on your data server is automatically assigned to the appropriate AIX WLM class.

Now that the AIX WLM environment has been set up, set the priorities for the DB2 workload, either by assigning relative values with shares or by assigning absolute amounts with hard limits. Shares provide a more flexible approach as resource assignments can be varied dynamically between classes. If you have strict priority requirements, choose hard limits. For either shares or limits, you can change the AIX WLM configuration dynamically later on.

To set processor priority using shares for each of the AIX classes, issue the following commands:

```
chclass -c shares=90 db2Power.HighPrio
chclass -c shares=10 db2Power.LowPrio
```

Alternatively, to limit the processor usage of the low priority class using a hard limit, issue the following command:

```
chclass -c hardmax=10 db2Power.LowPrio
```

To remove the resource entitlements, issue the following command:

```
chclass -c hardmax=100 db2Power.LowPrio
```

Only processor resource is regulated for DB2 services classes.

After any changes to the DB2 workload management configuration, issue the `wlmcntrl -ud db2workload` command to update the AIX Workload Manager.

You can use the `wlmcntrl -q` command to verify that AIX WLM is started and the `lsclass -fr` command to see the AIX WLM configuration (root privileges are not required).

AIX WLM can be disabled with the `wlmcntrl -o` command.
Please note that when using the partitioned database feature (DPF) with multiple physical machines, the same AIX WLM configuration has to be created on each machine. This differs from DB2 workload manager where any change is propagated to all DB2 nodes on all machines.

To monitor AIX WLM, use the topas command, which reports selected statistics about an activity on the local system, or generate a report in nmon format (with W and S options). You can also use AIX WLM specific monitoring tools like wlmstat or wlmmon and wlmperf commands which provide graphical views.

**Protect your system from being overloaded**

Systems can be bogged down by the sheer number of requests if you do not take the necessary precautions. The number of queries or transactions that can be sustained concurrently depends on many factors (not all under DB2 workload management control) and you should tune your data server against a real load after initial settings have been made.

If you have set up your resource controls and would like to control concurrency as well, consider the following concurrency limit as a starting point:

Permit a maximum of 20 concurrent complex queries:

```sql
CREATE THRESHOLD QUEUE_LOW_PRIO
FOR SERVICE CLASS LOW_PRIO UNDER POWER
ACTIVITIES ENFORCEMENT DATABASE
WHEN CONCURRENTDBCOORDACTIVITIES > 20
CONTINUE;
```

As you gain a better understanding of the work executing on your data warehouse environment, adjust these threshold values dynamically in order to custom tailor your configuration to the actual load.

To remove the threshold, it should be disabled first:

```sql
ALTER THRESHOLD QUEUE_LOW_PRIO DISABLE;
DROP THRESHOLD QUEUE_LOW_PRIO;
```

**Tame the monster query**

When you prevent very expensive rogue or runaway queries from monopolizing system resources, you maintain system stability for other work that is being performed on your data server, which is key to any workload management strategy.

Very expensive queries can have different origins. These queries might be created accidentally by an end user who forgets to code a join predicate appropriately, which in turn creates a very long running query with a very high estimated cost. Or, they might reflect reporting queries that evaluate complex business questions and do need to be
executed, although without negatively impacting all other activities on your data server. Even if a complex reporting query does require an answer as soon as possible, you should not allow it to execute without imposing workload management controls, because of the very significant impact such a query can have on all other queries by consuming excessive amounts of your system resources.

Queries with a very high cost can be controlled either predictively before query evaluation starts, or reactively in response to how the query is behaving during execution.

Either way of controlling queries requires that you create thresholds that are breached when an excessively expensive query enters your system or while it executes.

To manage queries prior to execution with predictive thresholds: Consider what users, groups, or applications should be allowed to execute queries that have a very high cost, and put them into a dedicated service class. Then, determine with the help of a CoordActEstCost histogram what should be considered an estimated cost that is too high for your data server to sustain without impacting other work significantly, and create a threshold for it.

For example, in a warehouse environment, the predicted cost of a query that is considered too high even for power users might be 10 000 000 timerons. To create a predictive threshold for the service class POWER for power users that stops any query exceeding that cost, issue the following statement:

```
CREATE THRESHOLD HIGH_COSTS
FOR SERVICE CLASS LOW_PRIO UNDER POWER ACTIVITIES
ENFORCEMENT DATABASE
WHEN ESTIMATESQLCOST > 50000000
COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
STOP EXECUTION;
```

You can also decide to let such queries run but capture information for possible performance tuning analysis at a later time by using the CONTINUE option in the threshold definition.

To manage queries in response to how they behave during query evaluation with reactive thresholds: Determine what constitutes a condition that should breach your reactive threshold. Conditions that indicate a query has begun to consume too many system resources might include how long the query has been executing, how much temporary table space it is consuming, or how many rows have been returned during query evaluation.

For example, the following threshold uses the amount of time elapsed during query evaluation in the service class POWER for power users as the condition which breaches the thresholds. Queries are permitted to run in their service class for 60 minutes, after which the threshold puts a hard stop to any query that is still running after 60 minutes, which is deemed to be excessively long. Activity data is collected with details, and data...
values, which allows you to gain an understanding of the kinds of activities that are simply taking too long to finish.

```
CREATE THRESHOLD TOO_LONG
FOR SERVICE CLASS POWER ACTIVITIES
ENFORCEMENT DATABASE
WHEN ACTIVITYTOTALTIME > 60 MINUTES
COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
STOP EXECUTION;
```

In situations where the activity is queued by a queuing threshold, the total activity time includes the time spent in the queue awaiting execution.

**Limit the number of concurrent load operations**

Because of memory requirements, it is a good practice to limit the number of concurrent load operations on the system. If you run only one or very few load operations at the same time, control their memory usage with the DATA BUFFER parameter, which sets the total amount of memory available to the load utility and allows it to run more efficiently (if not specified, the load utility determines the DATA BUFFER value from the UTIL_HEAP_SZ parameter and the total number of concurrent load operations). For loading into multidimensional clustering tables significant amount of memory is sometimes required.

Control load operations by the putting the load work type into a separate work class:

```
CREATE WORK CLASS SET LOAD_TYPE
(WORK CLASS LOAD_WC WORK TYPE LOAD);
```

To limit the number of concurrent load operations to one at the database level, create a work action:

```
CREATE WORK ACTION SET CONTROL_LOAD FOR DATABASE
USING WORK CLASS SET LOAD_TYPE
(WORK ACTION LIMIT_LOAD ON WORK CLASS LOAD_WC
WHEN CONCURRENTDBCOORDACTIVITIES > 1
CONTINUE);
```

When several load operations are started in parallel, they now run only sequentially. You can see the state of these load operations in the WORKLOAD_OCCURRENCE_STATE column with the following query:

```
SELECT
  SUBSTR(APPLICATION_NAME,1,10) AS APPL_NAME,
  SUBSTR(CHAR(APPLICATION_HANDLE),1,10) AGENTID,
  SUBSTR(WORKLOAD_NAME,1,22) AS WORKLOAD_NAME,
  SUBSTR(CLIENT_APPLNAME,1,25) AS CLIENT_APPLNAME,
  WORKLOAD_OCCURRENCE_STATE AS WL_STATE FROM
  TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('','',-2))
ORDER BY WORKLOAD_OCCURRENCE_STATE DESC;
```
The output shows that all but one load operations are queued:

<table>
<thead>
<tr>
<th>APPL_NAME</th>
<th>AGENTID</th>
<th>WORKLOAD_NAME</th>
<th>CLIENT_APPLNAME</th>
<th>WL_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2bp</td>
<td>65638</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>load_from_flat3.sql</td>
<td>UOWWAIT</td>
</tr>
<tr>
<td>db2bp</td>
<td>65604</td>
<td>HIGH_PRIO</td>
<td>-</td>
<td>UOWEXEC</td>
</tr>
<tr>
<td>db2bp</td>
<td>65637</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>load_from_flat2.sql</td>
<td>QUEUED</td>
</tr>
<tr>
<td>db2bp</td>
<td>65639</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>load_from_flat1.sql</td>
<td>QUEUED</td>
</tr>
<tr>
<td>db2bp</td>
<td>65640</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>load_from_flat4.sql</td>
<td>QUEUED</td>
</tr>
<tr>
<td>db2bp</td>
<td>65660</td>
<td>HIGH_PRIO</td>
<td>load_from_flat.sql</td>
<td>QUEUED</td>
</tr>
</tbody>
</table>

To remove these limits, disable the WLM objects and drop them:

```
ALTER WORK ACTION SET CONTROL_LOAD DISABLE;
ALTER WORK ACTION LIMIT_LOAD DISABLE;
DROP WORK ACTION SET CONTROL_LOAD;
DROP WORK CLASS SET LOAD_TYPE;
```

**Monitor expensive queries**

Expensive queries can have a disruptive effect on your data server and you should monitor for them to understand what the queries are and to what extent they need to be controlled. You can look for any outliers (and potentially rogue) queries that exist in a light weight manner by using the various histograms available to you. This also lets you build up a historical perspective by using the automated collection of statistics.

The event monitor collects activities according to defined workload monitoring rules. For example: To define a rule that enables monitoring of all queries for the entire database that ran longer than five minutes (five minutes is the minimum you can use in DB2 V9.5) without interrupting the queries, create the following threshold:

```
CREATE THRESHOLD MONITOREVENT
FOR DATABASE ACTIVITIES
ENFORCEMENT DATABASE
WHEN ACTIVITYTOTALTIME > 5 MINUTES
COLLECT ACTIVITY DATA ON ALL WITH DETAILS AND VALUES CONTINUE;
```

To collect information about complex queries, you can also use a threshold triggered by the estimated cost of a query. For example, if any query has an estimated cost greater than 10 000 timerons, the following threshold turns on activity data collection with both details and values collected for the service class HIGH_PRIO:

```
CREATE THRESHOLD ACTIVCOST
FOR SERVICE CLASS HIGH_PRIO ACTIVITIES
ENFORCEMENT DATABASE
```
WHEN ESTIMATEDSQLCOST > 10000
COLLECT ACTIVITY DATA ON ALL WITH DETAILS AND VALUES
CONTINUE;

An event monitor to record this activity data on disk has to be created:

CREATE EVENT MONITOR WLM_EVENT FOR ACTIVITIES WRITE TO TABLE;

The CREATE EVENT MONITOR statement requires access to the default table space that exists across all database partitions (default USERSPACE1, for example). You can also rename the tables required by the event monitor and specify a MONITOR_TS table space you created specifically for monitoring data:

CREATE EVENT MONITOR WLM_EVENT FOR ACTIVITIES WRITE TO TABLE
ACTIVITY (TABLE WLM_EVENT IN MONITOR_TS),
ACTIVITY_STMT (TABLE WLM_EVENT_STMT IN MONITOR_TS),
ACTIVITY_VALS (TABLE WLM_EVENT_VALS IN MONITOR_TS),
CONTROL (TABLE WLM_EVENT_CONTROL IN MONITOR_TS);

Now turn the event monitor on:

SET EVENT MONITOR WLM_EVENT STATE 1;

When multiple thresholds of the same type apply to one activity, only one threshold will be enforced. In our examples, if you define the thresholds on services classes to tame “monster” queries on ACTIVITYTOTALTIME or ESTIMATEDSQLCOST, the thresholds for monitoring with the same type will be ignored.

**Analyze event monitor data**

Once you have run some work on your data server, the event monitor tables can be analyzed. Below are some typical scenarios that might interest you.

You can find the activities (queries) with the longest total running time with the following statement:

```
SELECT
  SUBSTR(APPL_ID,1,26) as APPL_ID,
  SUBSTR(CHAR(ACTIVITY_ID),1,10) AS ACTIVITY_ID,
  SUBSTR(APPL_NAME, 1,10) AS APPL_NAME,
  SUBSTR(ACTIVITY_TYPE,1,10) AS TYPE,
  TIMESTAMPDIFF(2, CHAR(TIME_COMPLETED-TIME_STARTED)) AS TOTALTIME,
  SQLCODE, SUBSTR(SESSION_AUTH_ID,1,8) AS USER,
  SUBSTR(SERVICE_SUBCLASS_NAME,1,20) AS SERVICE_SUBCLASS_NAME
FROM WLM_EVENT AS A
WHERE PARTITION_NUMBER=CURRENT DBPARTITIONNUM
ORDER BY TOTALTIME DESC
FETCH FIRST 10 ROWS ONLY;
```

This query returns the following output:
This output shows that the APPL_ID *N0.db2inst3.080609131850 with ACTIVITY_ID 2 ran the longest query, by a wide margin. To see the SQL statement text of this activity, issue the following statement:

```
SELECT SUBSTR(S_STMT_TEXT, 1, 1000) AS STMT
FROM WLM_EVENT AS A,
     WLM_EVENT_STMT AS S
WHERE A.APPL_ID = S.APPL_ID AND
    A.ACTIVITY_ID = S.ACTIVITY_ID AND
    A.UOW_ID = S.UOW_ID AND
    A.APPL_ID = '*N0.db2inst3.080609131850' AND
    A.ACTIVITY_ID = 2 AND
    A.PARTITION_NUMBER = CURRENT DBPARTITIONNUM;
```

It might also be interesting to find the database partition where the activity spent the most time, which can be done with the following statement:

```
SELECT SUBSTR(APPL_ID, 1, 26) AS APPL_ID,
        PARTITION_NUMBER AS DBPART,
        SUBSTR(ACTIVITY_TYPE, 1, 10) AS TYPE,
        TIMESTAMPDIFF(2, CHAR(TIME_COMPLETED-TIME_STARTED)) AS TOTALTIME
FROM WLM_EVENT WHERE APPL_ID = '*N0.db2inst3.080609131850' AND
                    ACTIVITY_ID = 2
ORDER BY PARTITION_NUMBER;
```

The output shows the time spent across all partitions, which you can compare:
Moderating AIX Workload Manager hard maximum processor use limits

When processors are not constrained by high demand, most AIX Workload Manager (WLM) control mechanisms have little effect. In contrast, the hard maximum processor use limit approach strictly controls processor consumption regardless of the degree of processor utilization, which allows you to control low-priority workloads at all times and thus benefit high-priority work even in environments that are not processor constrained. To regain the dynamic allocation of processor time lost by using hard maximums, you need to run an automated script in the background like the one shown here that dynamically adjusts the hard maximum settings depending on the current utilization rate in each service class.

How moderated hard maximum settings work: AIX WLM provides tools for easy monitoring of system resource usage among AIX service classes. You can monitor system usage every second with this command:

```
$ wlmstat 1
```

Once you have collected this monitoring information, calibrate your AIX WLM configuration to meet the business priority of incoming work. Unlike DB2 workload management, changes to AIX WLM are effective immediately upon running queries. The monitoring information and configuration can then be combined using an automation script.

For example, suppose you have queries coming in from two departments in your company and your goal is to ensure that the queries of department A assigned to the class Power.HighPrio are not affected by concurrent queries from department B in the Power.LowPrio class.

For the sake of simplicity, assume that all queries from department A are coming from user DB2INST3 using db2bp, and that all queries from department B are coming from the db2batch command. This way, you can reuse the CREATE SERVICE CLASS and CREATE WORKLOAD DDL in the section “Obtain consistent response times with priority settings” to set up DB2 workload management. You also need to set up AIX
WLM and bind the DB2 workload to AIX WLM using the steps introduced in the section “Integrate with AIX Workload Manager”.

First, use the wlmstat command to monitor the processor usage of queries from the two departments. The following is the output of wlmstat when neither department runs any activities:

<table>
<thead>
<tr>
<th>CLASS</th>
<th>CPU</th>
<th>MEM</th>
<th>DKIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Unmanaged</td>
<td>0</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>Default</td>
<td>0</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>Shared</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>System</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Power</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Power.HighPrio</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Power.LowPrio</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TOTAL</td>
<td>0</td>
<td>52</td>
<td>0</td>
</tr>
</tbody>
</table>

Once department A runs some activities, the Power.HighPrio entry shows non-zero processor usage which you can monitor with the script. While department A submits queries and hence has non-zero processor usage, use the script to effectively suppress queries from department B by setting a low processor hard maximum for department B:

```bash
> chclass -a inheritance=no -c hardxmax=1 Power.LowPrio
```

Once the queries from department A have finished, the script can remove the hard maximum limit on department B:

```bash
> chclass -a inheritance=no -c hardxmax=100 Power.LowPrio
```

This setting and resetting process is repeated whenever there are new queries from department A. The whole procedure can be automated by running the script `auto_tune.sh` (which relies on `compute.awk`) as root.

**auto_tune.sh:**

```bash
#!/bin/bash
wlmstat l | gawk -f compute.awk | while read NEWCPUA NEWCPUB do
  echo "New processor limits for Departments A and B are $NEWCPUA and $NEWCPUB \n"
  chclass -a inheritance=no -c hardmax=${NEWCPUA} Power.High
  chclass -a inheritance=no -c hardmax=${NEWCPUB} Power.Low
  wlmctrl -ud db2workload
done
```

**compute.awk:**

```bash
BEGIN{prev=0; newa=0;newb=0;}
```
Figures 7 and 8 show the processor usage for all service classes, first using default WLM settings and then using the auto tuning script. The blue service class represents the workload of a single TPCH Q18 query from department B, while the red service class represents an ad hoc query from department A. The ad hoc query pulls data from a different table using a separated buffer pool. Run alone, the ad hoc query finishes in around 120 seconds. The ad hoc query is slowed down to 312 seconds (almost 160%) by Q18 when using default WLM settings. If you compare figure 7 and figure 8, you can see that after setting hard maximum processor use limits dynamically, the query from department A receives much more processor resource and has almost the same lifespan as when run alone (down from 312 seconds to 138 seconds), and the TPCH Q18 query is delayed only by less than 100 seconds (up from 1134 seconds to 1233 seconds).

Figure 7. Processor usage for all service class under default WLM
Figure 8. Processor usage among all service classes under dynamic hard maximum limit

This example is deliberately kept simple to convey the idea of moderated hard maximum settings. In reality, you might have more than two service classes and your goal might be based on overall query throughput rather than on processor usage among all service classes.

If you have more than one AIX physical machine (or LPAR), you will need to tune the hard maximum processor use limits on each machine (or LPAR). This differs from DB2 workload manager where any change is propagated to all DB2 nodes on all machines. Aside from tuning hard maximum processor use limits, you might also wish to resort to setting concurrency limits with DB2 workload management.
Think DB2 workload manager when upgrading from Query Patroller and DB2 Governor

Prior to DB2 V9.5, Query Patroller and DB2 Governor were used together to provide the workload management controls needed to successfully run complex workloads on DB2. As of Version 9.5, DB2 workload manager is the strategic tool for workload management on your DB2 data server and provides a greatly enhanced set of workload management features.

The following chart gives a high-level overview of the essential differences in control and monitoring capabilities between Query Patroller and DB2 workload manager:

<table>
<thead>
<tr>
<th>Query Patroller</th>
<th>DB2 workload manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>acts as a ‘gate keeper’: once work is admitted, it is free to execute as it desires</td>
<td>acts as a ‘hall monitor’: it ensures that work goes to its correct place and follows the relevant rules during execution</td>
</tr>
<tr>
<td>can show current state, SQL and ApplHandle, but no in-flight information about work until execution has completed</td>
<td>can show the current state of the application, SQL, and even agents at any time during execution</td>
</tr>
<tr>
<td>is aware only of the coordinator perspective for submitted work</td>
<td>is aware of the work across all database partitions</td>
</tr>
<tr>
<td>does not provide any mechanism to explicitly control the resources used for execution</td>
<td>provides mechanisms to control and influence resources used during execution</td>
</tr>
<tr>
<td>details on all managed activities written to control tables (disk)</td>
<td>nothing gets written to disk unless requested by a user-created event monitor</td>
</tr>
<tr>
<td>monitoring is fine-grained only (individual activities are collected)</td>
<td>monitoring can be fine-grained (individual activities) or coarse-grained (aggregate statistics)</td>
</tr>
</tbody>
</table>

When you upgrade from Query Patroller to workload manager, plan to control work on your data server according to the best practices outlined here. You can take advantage of the depth of information that you already have about your workload from the Query Patroller control tables to help you get a good initial implementation under workload manager.

With Query Patroller, you categorize an activity into a query class based on its estimated cost and the query class concurrency rate dictates how many activities of each class are allowed to run at the same time. Do not blindly emulate this approach with DB2 Workload Management because concurrency control might not be needed to achieve your goals.
With DB2 Workload Management, you have many more options, and you should explore them. You can separate and isolate competing workloads from each other with DB2 service classes and apply specific resource controls that affect response times by changing the processor and prefetcher priorities each service class receives. If you cannot separate work by its source through a workload, you can use a DB2 work action set to separate types of work based on characteristics such as the estimated cost, which is similar to what Query Patroller provides.

At this point, you can adjust the resources as needed to achieve your performance objectives. If the application of resource controls does not fully achieve the necessary results, you can begin applying other features of DB2 Workload Management. This includes the application of additional DB2 thresholds like concurrency thresholds.

Some of the DB2 Governor reactive thresholds will find a direct functional equivalent in DB2 workload management thresholds, like those controlling maximum execution time, the maximum number of rows returned, or the maximum connection idle time. Others are unique to workload management or to the DB2 Governor and upgrading to DB2 Workload Management will require you to rethink your approach to controlling work on your DB2 data server in current workload management terms. One difference you will find is that changes to DB2 Governor rules can apply to already running queries, whereas changes to WLM thresholds apply only to new queries.

**Why is there no tool to upgrade from Query Patroller automatically?**

There is no automated tool to upgrade from Query Patroller to DB2 Workload Management for the same reason that you should not emulate a Query Patroller approach of managing concurrency without applying other features of DB2 Workload Management first. The types of controls and mechanisms available differ between DB2 Workload Management and Query Patroller, as do their basic control paradigms. Both Query Patroller-DB2 Governor and DB2 Workload Management can co-exist in the same environment, so that your move from one to the other can take place in a controlled, granular manner.

When adopting DB2 workload manager features and capabilities, it is often possible to solve many common workload management scenarios more simply and more effectively by applying resource controls than by perpetuating a Query Patroller approach based on query classes. Use your adoption as an opportunity to review your overall approach to workload management in order to determine what the simplest and best solution might be.

Upgrading from Query Patroller (and DB2 Governor) to DB2 workload manager can take place in stages, which can reduce the impact and risk to your data server environment by only gradually adopting features and capabilities of DB2 workload manager.

On your DB2 data server, user requests that end up being processed in the default user service superclass SYSDEFAULTUSERCLASS are eligible for interception by Query Patroller, regardless of the workload used to map them there. Requests that get mapped
to a user defined service class cannot be intercepted. Similarly, DB2 Governor can apply its controls only to user requests that are mapped to the default user service superclass.

The following illustration shows a DB2 workload manager implementation that coexists with Query Patroller. Some user requests are handled by DB2 workload manager, whereas others are handled by Query Patroller in the default user service class. User requests handled by Query Patroller are mapped from both the default workload and from a user-defined workload:

Figure 9. Example: Coexistence of DB2 workload manager and Query Patroller
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Without limiting the above disclaimers, IBM provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any recommendations or techniques herein is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Anyone attempting to adapt these techniques to their own environment do so at their own risk.

This document and the information contained herein may be used solely in connection with the IBM products discussed in this document.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.
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 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.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE: © Copyright IBM Corporation 2008. All Rights Reserved.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.