



C. Nin Lei

# Workload Management for DB2 Data Warehouse

## Abstract

This IBM™ Redpaper addresses the workload management issues encountered in a data warehouse environment. It delivers a set of guidelines for overcoming these obstacles by utilizing the z/OS® Workload Manager. It also provides tips and techniques of properly implementing a Workload Manager policy.

## Management of large queries

A key objective of workload management is to prevent “monster” queries from monopolizing system resources. At issue here are not the long queries created accidentally by users, such as forgetting to code a join predicate. More often the long queries reflect legitimate business questions. Certainly they need to be executed. However, unless they are of a critical nature and answers are required as soon as possible, they shouldn't pose a significant impact to the short queries by consuming most of the system resources.

One technique is to use a predictive governor to screen all the queries prior to execution. Queries that are labeled as long would be scheduled for execution during off-shift hours. But this approach proves to be labor intensive and drains valuable DBA's time. Besides, the governor can't catch all the long running queries. Occasionally some slip through and take away system resources from the short queries.

Another solution calls for using a runtime governor to monitor resource usage during query execution. Queries that consume more than a threshold of CPU cycles will be cancelled automatically. This isn't a good solution either, because a high threshold would waste many CPU cycles prior to query cancellation. On the other hand, a low threshold would trigger many false alarms, canceling many short or medium queries unnecessarily.

## Workload Manager

The Workload Manager on zSeries® provides the most sophisticated workload management capabilities commercially. It delivers a period aging mechanism where a query travels from one period to another during its execution. Initially a query runs in the first period with the highest priority. As a query consumes a user-defined amount of system resources, it drops to the second period and potentially runs with a lower priority. A lower period can have an equal or lower priority to a higher period, but it can't have a higher priority. As a query consumes more and more system resources, it will drop further to lower periods. Workload Manager supports a total of eight periods. The definition of the query periods and priorities is stored in a policy.

Figure 1 shows a sample policy that's used successfully in many benchmarks at the Teraplex Center. Keep in mind that this policy is tailored for the Teraplex environment and you probably need to modify it for your configuration. Nevertheless, the sample policy gives a good demonstration of how a policy can be set up to manage coexistence of small, medium, and large queries.

Period	Duration	Performance Goal	Importance
1	5,000	Velocity = 80	2
2	50,000	Velocity = 60	2
3	1,000,000	Velocity = 40	3
4	30,000,000	Velocity = 30	4
5		Discretionary	

*Figure 1 Sample policy that's used successfully in many benchmarks at the Teraplex Center*

This policy contains five periods for the queries. The duration of the first period is 5000 service units, which is approximately half a second of CPU time of a z900 engine. It has an importance level of 2, which gives it a relatively high priority. The scale of importance level ranges from 1 to 5, with 1 being the highest priority.

Velocity goals are preferred for data warehouse queries. Since a query can qualify a significantly different number of rows with different predicate values, e.g., state = CA vs. state = VT, it is difficult to predict the response time of a query. Response time goals are generally discouraged except for the canned queries, which use a predictable amount of system resources. In this example, the velocity goal is set to 80, suggesting that a query running in the first period will be given CPU cycles at least 80% of the time that it is capable of running.

An importance level of 2 and a velocity goal of 80 in the first period points to a relatively high priority, since the highest priority possible is an importance level of 1 and a velocity goal of 100. This allows short queries consuming a small amount of resources, in this case

equivalent to half a second of CPU time of a z900 CP, potentially getting all or most of the system resources during their execution. Whether one should bump up the goal even higher depends on the other workloads running concurrently in the system.

With this policy set up, it becomes unnecessary to prescreen queries. If a query consumes more than 5000 service units, Workload Manager then drops it to the second period with equivalent importance level but with a lower velocity goal. A query can stay in this period for consumption of another 50,000 service units (roughly 5 CPU seconds) before it drops to another period again. Although a query still enjoys an importance level of 2 in this period, a velocity goal of 60 indicates it will obtain CPU cycles only 60% of the time. Clearly a query running in the first period can consume a larger amount of CPU cycles in a given amount of time.

A query that doesn't complete execution at the end of the second period is probably no longer a short query. It will drop to lower periods and is either a medium or long query. Extremely complex queries, after consuming approximately 30,000,000 service units (3000 CPU seconds), are pushed to the lowest period in this policy with a discretionary performance goal. This setting indicates a query running in this period will get CPU cycles only if all the other workloads have met their goals.

An advantage of using a discretionary goal is that queries can stay in the system inactive waiting for CPU cycles. If a governor is used to control system resource usage, a complex query would consume 3000 CPU seconds and be canceled upon hitting the resource consumption threshold. As a result, 3000 CPU seconds are wasted because the answer set for the canceled query is not available. With a discretionary goal, however, a complex query can stay in the system indefinitely. When CPU cycles become available, it can wake up and consume whatever it can get. If the system enters a CPU constraint state again, the complex query "goes back to sleep" and waits for the next window of CPU availability. No cycles are ever wasted due to query cancellation.

The sample policy gives a good example of classifying the queries into multiple categories. Trivial queries take a very short time to execute and they run in the first period. Short queries take a bit more CPU cycles and they run in the second period. Medium queries take up to 100 CPU seconds in the third period, while long queries take up to 3000 CPU seconds in the fourth period. Monster queries could take hours of CPU time; they run in the lowest period with a discretionary goal. The beauty of this solution is the avoidance of defining a query to a category prior to execution. In general, the fixed assignment approach doesn't work very well in data warehouse applications. A query that is a short query today (e.g., using predicate state = VT) may become a long query tomorrow when a different input is used for the predicate (e.g., state = CA). It is better letting the Workload Manager to classify a query dynamically based on resource consumption monitoring.

Although five periods are defined in the sample policy, systems support personnel can adjust up and down by adding or deleting periods in their installations. A minimum of three periods is probably required-mapping to short, medium, and long queries. On the other hand, using the maximum of eight periods may be overkill unless the query workload is extremely complex.

Users should also adjust the service unit definitions in each period to tailor to their configurations. For example, a query is considered to be a "monster" if it consumes more than 3000 CPU seconds. Some users may feel this is too generous and would probably reduce the threshold to a smaller number.

## Consistent response time

One could expect consistent response times for the shorter queries with a policy defined in the last section. Since the shorter queries run with the higher priorities in the service class, longer queries would have very little impact on them. These longer-running queries run in a lower period and receive fewer resources in a given period of time. This consistency in response times plays a very important role in users accepting a data warehouse system. The last thing a user would like to see is running a query in 5 seconds on Monday and then taking 5 minutes on Tuesday for the exact same query.

Figure 2 shows a set of measurements that were made to demonstrate the effect of consistent query response times in a highly concurrent environment.

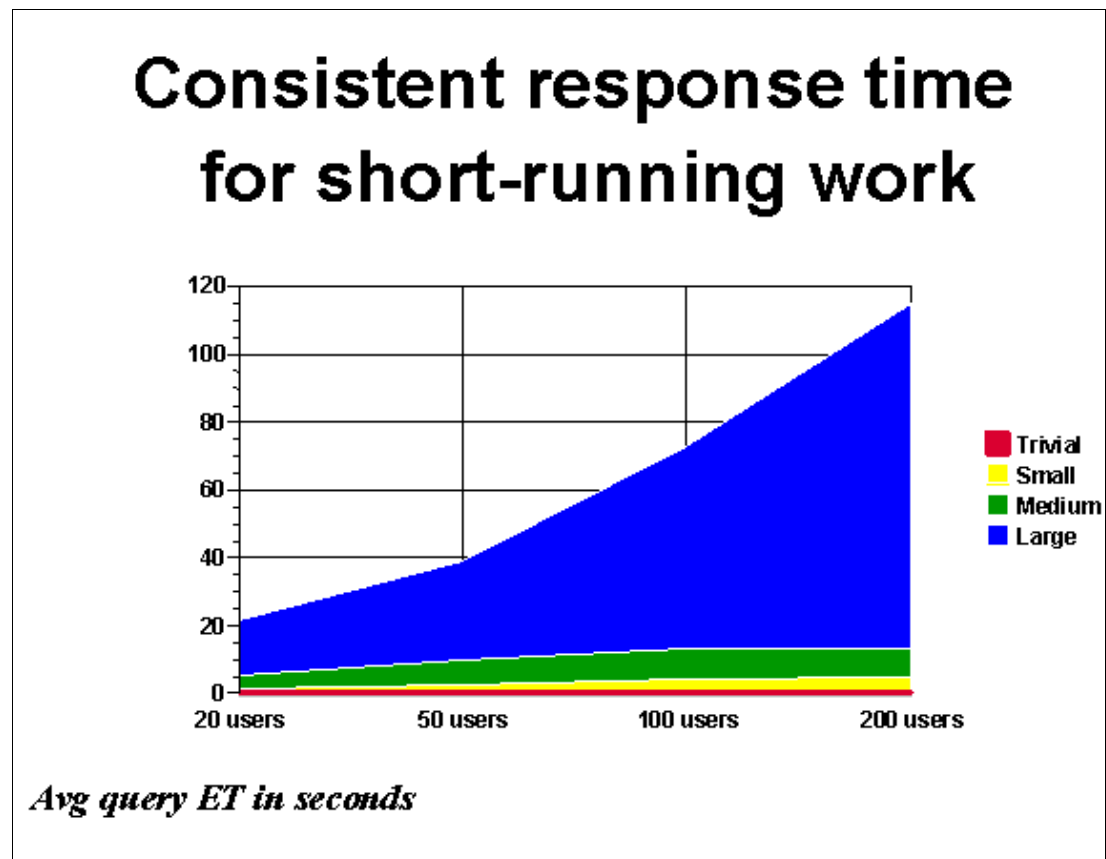


Figure 2 Consistent response time for short-running work

Four data points were taken with different numbers of concurrent users. The system reached 100% CPU busy at the 50-user point. Although there were 2X and 4X more users at the 100 and 200 user points, the response time curves of the trivial, small, and medium queries stayed relatively flat. Since the Workload Manager policy pins the large queries to the lowest priority, cycles are stolen from them to feed the other queries in a CPU constraint environment. This is a generally acceptable approach since it gives end users running short queries the perception of a very responsive system.

An interesting observation is that the use of CPU cycles follows an 80-20 rule: 80% of the cycles are consumed by 20% of the queries. Although the measurements stop at 200 users, it is expected that response times of the trivial, short, and medium queries continue to stay relatively flat with a higher number of concurrent users, say 500 users. This is because even

at the 200 user point, a significant amount of CPU cycles is still consumed by the large queries.

It should be pointed out that it is acceptable to drive a data warehouse system to 100% CPU busy. To maintain good response times, an OLTP system may maintain a CPU utilization of no more than 85 or 90%. Warehouse systems behave very differently, primarily because the workload is read only. Lock contention either doesn't exist or is cut down to the minimum. It may even be acceptable to drive a system to “more than 100% CPU busy” by means of adding more concurrent queries after the system reaches the CPU saturation point. In that case the shorter queries continue to run with good response times at the expense of the longer queries.

## Critical queries

There are many synonyms for this type of query. Some would call them CEO queries, while others would label them as senator queries, in reference to government agencies receiving phone calls from senators' offices demanding reports immediately. We recommend that a separate service class be created to house this category of queries.

There is only one period in this service class, implying that critical queries will never drop to a different period with a lower priority. The importance level is 1, indicating that it is the most important workload in the system. A velocity of 90% is chosen, making some CPU cycles available to the rest of the workload. Certainly a critical query will run even faster if a velocity goal of 100% is used. The downside of that approach is that the execution of a critical query could potentially stop all other workloads.

## Multiple service classes

A single query service class serves very well for many installations. There are times when an installation might want to have more granular control of system resources based on certain criteria: by departments, users, etc. For example, queries submitted by sales department users could have very different characteristics from marketing department queries. Likewise, work submitted by different users might run with different priorities. An installation might want to define multiple service classes for multiple departments or users. It is important to note that priority in one service class is used to compare with another service class to determine resource allocation.

## Active warehousing

It's more and more common that users are interested in refreshing their warehouse on a real-time basis. As opposed to updating a warehouse at nights or on weekends, users would like to have access to the data shortly after the data is available from the operational databases. This update can be achieved in three ways: insert, load resume, and online load resume. Insert and online load resume do not lock up any partitions and can be coexistent with user queries. Load resume does lock up a partition while it is running; however, certain database designs such as partition by time, allows queries and load jobs to access different partitions. In any case, there could be contention at the secondary index level.

To speed up warehouse refreshing, one can set up a different policy to favor utility jobs (or insert jobs) during the refresh cycle. Figure 3 shows an example of how users can switch policies depending upon the importance of the refresh jobs.

WLM POLICY = BASE				WLM POLICY = REFRESH			
Service Class	Period	Importance	Performance Goal	Service Class	Period	Importance	Performance Goal
QUERY	1	2	Velocity = 80	QUERY	1	2	Velocity = 80
	2	3	Velocity = 60		2	3	Velocity = 60
	3	4	Velocity = 40		3	4	Velocity = 40
	4	5	Velocity = 20		4	5	Velocity = 20
			Discretionary				Discretionary
UTIL	1	5	Velocity = 10	UTIL	1	1	Velocity = 80

Figure 3 Users can switch policies depending upon the importance of the refresh jobs

With the base policy, utilities run with a low priority as reflected by an importance level of 5 and a velocity goal of 10. Queries are heavily favored as dictated by the business requirements. As new data becomes available, users can switch to the REFRESH policy to speed up the refreshing process. Figure 4 on page 7 shows several measurements that were run at the Teraplex to illustrate the advantages of having dual policies.

# Active Warehousing

Concurrent Refresh and Queries  
(favor Queries)

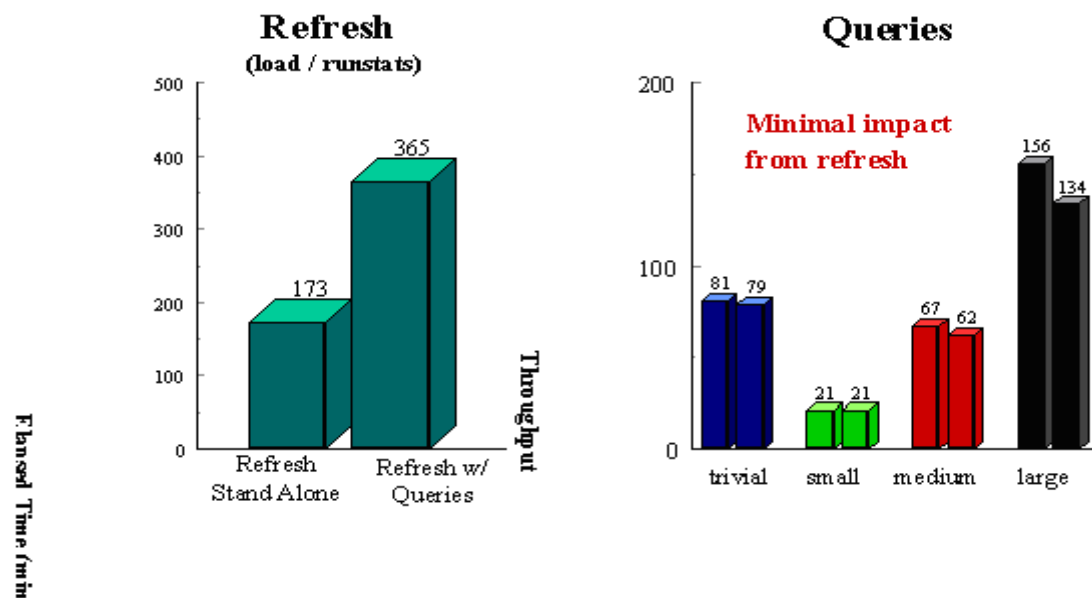


Figure 4 These measurements were run at the Teraplex to illustrate the advantages of having dual policies

The base policy was used for this measurement. Since the refresh jobs were not given a high priority, its elapsed time elongated to 365 seconds from a standalone run of 173 seconds. On the other hand, there's virtually no impact to the queries from the refresh jobs. Query throughput stayed practically the same.

# Active Warehousing

## Concurrent Refresh and Queries (favor Refresh)

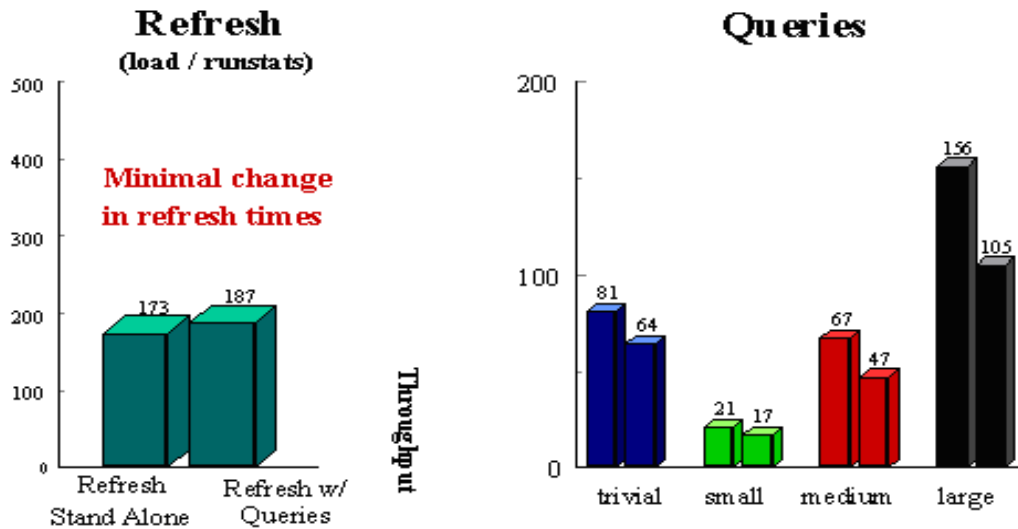


Figure 5 Queries from the refresh jobs

The measurements shown in Figure 5 were run with the REFRESH policy. Very few changes were observed for the refresh jobs as the run-time went up slightly from 173 seconds in the standalone mode to 187 seconds in a concurrent run with queries. But there's a noticeable impact to the queries from the refresh jobs. The most dramatic decrease in throughput took place in the large queries because they ran with the lowest priority.

## Miscellaneous implementation tips

1. In general, use velocity goals for BI applications. Response time goals are more suitable for OLTP applications.
2. If it is desirable to use response time goals for BI applications, use them in early periods and assign velocity goals for later periods. Also, consider using response time goals for canned queries.
3. Utilize dynamic policy changes for different workload requirements. It's not common to have a prime shift policy to favor queries and an off-shift policy to favor the ETL process.
4. Define report classes to assist capacity planning.
5. Monitor RMF™ Monitor I workload activity reports to determine query response times and CPU consumption.
6. Assign important work to an importance level of 1 or 2.
7. Set achievable goals. Overly aggressive goals for one service class could impact other service classes significantly.



8. Choose velocity goals far apart to be meaningful. For example, defining a velocity goal of 70% for the first period and 65% for the second period doesn't cause a noticeable differentiation between the two periods. A query running in the second period can consume resources almost as fast as a query in the first period.

## Author

**C. Nin Lei** is recognized as an expert in DB2 performance and database design. One of his main interests is in Very Large Data Bases (VLDB), covering areas of database and application design and performance analysis. He speaks regularly at database and data warehouse conferences. He is currently an architect at the IBM zSeries Benchmark Center conducting customer studies. He drives performance analysis work for all projects, and he provides architectural guidance to customers for designing their applications to meet high performance and scalability requirements. He also advises customers on new technology exploitation issues.



# 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 give 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.

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.

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.

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:

This information contains sample application programs in source language, which illustrates 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. 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 IBM's application programming interfaces.

Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:  
[ibm.com/redbooks](http://ibm.com/redbooks)
- ▶ Send your comments in an email to:  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to:


IBM Corporation, International Technical Support Organization  
Dept. HYJ Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400 U.S.A.



## Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM®

Redbooks (logo) ™

RMF™

z/OS®

zSeries®

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