Db2 LUW Performance Demystified: An Introduction to Wait Time Analysis

By C Raghavendra and Toby Haynes

# **Introduction**

Performance tuning in Db2 LUW can be challenging, especially when database responsiveness degrades without an obvious root cause. This article aims to provide an overview to diagnosing and resolving performance issues by focusing on one of the most critical aspects of database performance: wait times.

As the first installment in a series on understanding and analyzing Db2 LUW wait time metrics, this post explains why wait times matter, reviews many types of database waits, and examines how to measure them effectively using modern tools (such as **db2mon** leveraging the **MON\_GET\_\*** monitoring functions).

Whether you're investigating a sudden performance drop or proactively fine-tuning your environment, this series will help you uncover where time is being spent and how to improve performance.

## **Why Wait Times Matter**

## In Db2 LUW performance analysis, understanding where time is spent is fundamental to effective tuning. Unlike higher-level metrics (for example, commit rate) which may only indicate that a performance problem exists, wait time metrics provide direct insight into delays that impact throughput and responsiveness. Wait time metrics are generally low-level measurements, closer to the system and database fundamentals such as storage performance or locking. They help pinpoint the layer where performance is hindered.

## **Wait time types**

This table shows the many wait time metrics found in the [MON\_GET\_DATABASE table function](https://www.ibm.com/docs/en/db2/12.1.0?topic=routines-mon-get-database-get-database-metrics). This table has been ordered into groups by the area of the database engine that is being measured.

|  |  |
| --- | --- |
| Wait Time Monitoring Metric | Description |
| TOTAL\_ACT\_WAIT\_TIME | Total time spent waiting within the database server, while processing an activity |
| TOTAL\_WAIT\_TIME | Total time spent waiting within the database server |
| *Communicating between client and server* | |
| AGENT\_WAIT\_TIME | Application queued to wait for an agent under concentrator configurations |
| CLIENT\_IDLE\_WAIT\_TIME | Waiting for the client to send its next request |
| TCPIP\_RECV\_WAIT\_TIME | Waiting for an incoming client request over TCP/IP excluding idle time |
| TCPIP\_SEND\_WAIT\_TIME | Blocking on a TCP/IP send to the client |
| IPC\_RECV\_WAIT\_TIME | Agent waiting to receive an incoming client request using the IPC communications protocol |
| IPC\_SEND\_WAIT\_TIME | Agenet blocking on an IPC send to the client |
| COMM\_EXIT\_WAIT\_TIME | Waiting for the return from a communication exit library API function |
| *Internal database engine operations* | |
| LOCK\_WAIT\_TIME | Waiting for locks |
| LOG\_BUFFER\_WAIT\_TIME | Agent waiting for space in the log buffer |
| LOG\_DISK\_WAIT\_TIME | Agent waiting for log records to be flushed to disk |
| PREFETCH\_WAIT\_TIME | Waiting for an I/O server (prefetcher) to finish loading pages into the buffer pool |
| LOB\_PREFETCH\_WAIT\_TIME | Waiting for an I/O server (prefetcher) to read or write a LOB |
| TOTAL\_EXTENDED\_LATCH\_WAIT\_TIME | Waiting for a latch where the requester yielded CPU |
| *Auditing and Diagnostics* | |
| AUDIT\_FILE\_WRITE\_WAIT\_TIME | Waiting to write an audit record |
| AUDIT\_SUBSYSTEM\_WAIT\_TIME | Waiting for space in audit buffer |
| DIAGLOG\_WRITE\_WAIT\_TIME | Agent waiting on a write to the db2diag log file |
| EVMON\_WAIT\_TIME | Agent waiting for an event monitor record to become available |

|  |  |
| --- | --- |
| *External data sources* | |
| EXT\_TABLE\_RECV\_WAIT\_TIME | Agent waiting for the external table readers to read & process data from external tables |
| EXT\_TABLE\_SEND\_WAIT\_TIME | Agent waiting for the sent data to be processed and written by the external table writers |
| FED\_WAIT\_TIME | Waiting for Federation server to run a statement and get results from a remote data source |
| *Communication between agents* | |
| FCM\_MESSAGE\_RECV\_WAIT\_TIME | Agent waiting for an FCM reply message containing the results of a previously sent FCM request message |
| FCM\_MESSAGE\_SEND\_WAIT\_TIME | Agent blocking on an FCM message send |
| FCM\_RECV\_WAIT\_TIME | Waiting to receive data through FCM |
| FCM\_SEND\_WAIT\_TIME | Blocking on an FCM send operation |
| FCM\_TQ\_RECV\_WAIT\_TIME | Waiting to receive the next buffer from a table queue |
| FCM\_TQ\_SEND\_WAIT\_TIME | Waiting to send the next buffer through a table queue |
| IDA\_RECV\_WAIT\_TIME | Waiting to receive data from an in-database analytics process |
| IDA\_SEND\_WAIT\_TIME | Waiting to send data to an in-database analytics process |
| *Db2 pureScale-specific* | |
| CF\_WAIT\_TIME | Agent communicating with the cluster caching facility |
| DATA\_SHARING\_REMOTE\_LOCKWAIT\_TIME | Remote applications waiting while the table transitions out of the NOT\_SHARED data sharing state |
| LOCK\_WAIT\_TIME\_GLOBAL | Waiting on global lock waits |
| RECLAIM\_WAIT\_TIME | Agent waiting on page locks, where the lock request caused a page to be reclaimed |
| SPACEMAPPAGE\_RECLAIM\_WAIT\_ TIME | Agent waiting on page locks for pages related to internally maintained object space management where the lock request caused a reclaim from another member |

All of these monitoring counters are of BIGINT data type, and all corresponding columns report values in milliseconds.

## ***Table 1:*** *Wait Time Metrics available through MON\_GET\_DATABASE*

## **Tools for Monitoring Performance**

Tools like db2mon use the MON\_GET\_\* table functions to offer visibility into many database metrics, enabling experienced database administrators to examine time spent actively processing database work and time waiting for one or more database and/or system resources. The MON\_GET\_\* table functions provide broad coverage of many diverse database metrics, building on the high performance, low impact monitoring capabilities originally delivered in Db2 v9.7 and continuously extended through the current v12.1 release. The use of the old SNAPSHOT metrics is strongly discouraged, especially on databases driving substantial CPU workloads on highly parallel systems, where the performance impact of data collection can be severe.

# **Strategies for performance analysis**

## Getting the most impact

By focusing on areas of higher wait times, database teams can prioritize efforts where they will have the greatest impact, avoiding chasing metrics with complex dependencies on lower-level performance. This approach is applicable to top-down performance analysis, but it can also be applied at the SQL statement level for specific insights.

## Defining a Goal

When addressing performance issues in Db2 LUW, it is essential to begin with a clear objective. Whether the goal is to restore previous performance levels, reduce CPU load, or identify the most critical bottleneck—such as disk, network, or CPU—understanding where time is being spent is the key to effective tuning.

A diagram of a problem

AI-generated content may be incorrect.

**Figure 1**: Methodical Approach to Performance Analysis

Db2 provides a rich set of monitoring tools and interfaces that allow administrators to drill down into performance data and isolate the root causes of inefficiencies.

Monitoring table functions

At the core of Db2’s performance monitoring capabilities are the MON\_GET\_\* table functions. These functions provide detailed metrics on various aspects of database activity, including time spent and the number of occurrences of specific events. These metrics are invaluable for identifying whether performance issues stem from excessive processing or from time spent waiting on resources. The table below summarize a few of the many.

|  |  |
| --- | --- |
| MON\_GET\* Function Name | Description |
| mon\_get\_database | Offers a high-level overview of database activity, including I/O, logging, and wait times. |
| mon\_get\_workload | Provides metrics at the workload level, useful for analyzing application behavior and resource usage. |
| mon\_get\_connection | Connection-level metrics, helpful in establishing whether all the connections in a workload class or application are performing and executing equivalently. |
| mon\_get\_pkg\_cache\_stmt | Detailed statistics on SQL statements found in the SQL package cache, covering row metrics and wait times. |
| mon\_get\_bufferpools | Reports on buffer pool metrics, useful for calculating hit ratios and measuring read/write activity. |
| mon\_get\_tablespace | Delivers detailed statistics on tablespace usage, I/O performance, and wait times. |
| mon\_get\_transaction\_log | Provides insights into transaction log activity, including log write times and I/O statistics. |

***Table 2:*** *Commonly used MON\_GET\* table functions.*

Handling cumulative monitoring data

The MON\_GET\_\* table functions are a detailed source of performance data in Db2 LUW. However, because these functions report cumulative metrics from the time the database was activated, the counters cannot be reset. Most DBAs are interested in the current activity, so we need to find the difference between two data collections taken over a defined interval to accurately assess activity during a specific period.

The technique used in db2mon involves capturing a baseline dataset using a global temporary table for each of the table functions, waiting for a short duration (typically under 300 seconds), and then subtracting the baseline from a new data collection. This approach is often referred to as “Delta values”, enables precise tracking of performance trends and helps isolate short-lived events that might otherwise go unnoticed in the cumulative data.

*Note*: The MON\_GET\_\* table functions accumulate counts and time spent in BIGINT data types. For the busiest monitors, these values can exceed the maximum BIGINT value and will overflow back to a small value again. Collecting MON\_GET\_\* table functions at longer intervals can expose more of these overflow events, leading to “odd” or negative values in the db2mon report. It is unlikely that a 30 second db2mon collection will include an overflow, but a 3000 second db2mon collection is at much higher risk.

## Other performance utilities

An older but sometimes useful tool for quick performance analysis is **monreport. dbsummary()**. This stored procedure provides a high-level summary of some of the database performance metrics.  By executing CALLmonreport.dbsummary(60) users can generate a report that highlights some critical metrics over a 60-second interval. This includes values like CLIENT\_IDLE\_WAIT\_TIME and TOTAL\_RQST\_TIME, which help determine whether delays are occurring inside Db2 or in external layers such as the application or network. However, monreport.dbsummary() does not include all wait categories, some of which are essential for deep-dive performance analysis.  
The **db2mon** utility provides a complete breakdown of wait times under the DB#WAITT section (Database-wide scope), SQL#TOPWAIT (SQL statement scope). To explore how db2mon works and how to get started, refer to the article [*An Introduction to db2mon*](https://www.idug.org/news/an-introduction-to-db2mon).

**Working with Waits**

Where to put the effort?

Before diving into internal Db2 server diagnostics, it is crucial to determine whether performance delays originate mostly within Db2 or more from external sources such as the application implementation, Db2 client, or network transport. By comparing the CLIENT\_IDLE\_WAIT\_TIME (time Db2 spends waiting for the next request from the client after the previous request has been sent) with the TOTAL\_RQST\_TIME (time Db2 spends processing those requests on the server), we can assess where the majority of time seen by the end user is spent. If the idle wait time is more than five times the server-side request processing time, performance tuning will have the most impact concentrated outside the Db2 server (e.g. on network congestion or application proximity to the database server). In such cases, tuning efforts should shift toward optimizing the application logic, improving client-server communication, or addressing network latency. This distinction ensures that performance tuning is focused in the area that will most improve the performance from the user’s perspective.

A screenshot of a web page

AI-generated content may be incorrect.**Figure 2**: Wait time outside of Db2

# Digging into time spent on the database server

If the majority of time spent waiting is within the database server, the next step is to analyze how time is spent within the Db2 engine. The key question becomes:

* Is the system spending more time actively processing, or
* Is it predominantly waiting on internal resources?

The answer will determine the direction of further investigation. Fortunately, stored procedures like monreport.dbsummary() can provide a quick and accessible breakdown of time spent across various wait categories.

A blue rectangular object with white text

AI-generated content may be incorrect.

**Figure 3**: Wait within Db2 itself

If the report reveals that wait times account for a significant portion of the total request time, it strongly indicates that internal Db2 waits are the primary performance bottleneck. In such cases, a deeper dive into specific wait categories is warranted, and it is worth collecting db2mon reports.

*Note*: If the database is spending most of its time processing requests and the host is running above 70-80% CPU utilization, it may be worth investigating the SQL statements that are consuming the most CPU or provisioning more CPU for this host.

## Introduction to Database Wait Categories

If the database request is spending a significant percentage waiting, the next step is to analyze the various wait categories that contribute to overall request latency.

Each wait category (e.g. disk I/O, locks, latches, log write, etc.) offers unique insights into system behavior and potential areas for improvement. By examining these categories individually, we can identify which category of wait is consuming the most time, assess what other information needs to be studied, and determine the appropriate corrective actions.

The following table provides a structured overview of key wait categories, how to measure them, what elevated values might indicate, and recommended steps to mitigate their impact.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Wait Time Category**** | ****What the Wait Time Represents**** | ****How to Measure It**** | ****What High Values Might Indicate**** | ****Recommended Actions**** |
| ****Pool Read Time**** | Time spent reading pages into the buffer pool. | Use MON\_GET\_TABLESPACE to check average read time. | Indicates storage performance issues or excessive I/O. | Optimize high-I/O statements; increase buffer pool size. |
| ****Log Disk Wait Time**** | Time spent writing log records to disk. | Divide log\_write\_time by num\_log\_write\_io from MON\_GET\_TRANSACTION\_LOG. | Suggests slow log device performance. | Ensure log write time is under 2–4 ms; optimize log device throughput. |
| ****HADR Wait Time**** | Time spent waiting for HADR log transaction synchronization. | Compare average log disk wait time (MON\_GET\_DATABASE) with HADR wait time (MON\_GET\_HADR). | Indicates network latency or standby log performance issues. | Improve network performance; check standby log I/O. |
| ****Direct Read/Write Time**** | Time spent on direct I/O operations for LOBs. | Use MON\_GET\_TABLESPACE to check direct\_read\_requests and direct\_write\_requests. | Indicates heavy LOB usage or slow disk I/O. | Optimize LOB access patterns; evaluate disk subsystem perf. |
| ****Prefetch Wait Time**** | Time spent waiting for prefetch operations to complete. | Calculate prefetch size and average time using MON\_GET\_TABLESPACE. | May indicate poor prefetch configuration or slow storage. | Tune prefetch settings (extent size, prefetchers); assess storage performance. |
| ****Latch Wait Time**** | Time spent waiting for latches controlling access to shared resources. | Use MON\_GET\_EXTENDED\_LATCH\_WAIT to check latch\_name and total\_extended\_latch\_ wait\_time. | Indicates contention on shared memory structures like buffer pool pages. | Identify and partition hot tables/indexes; reduce concurrency on shared resources. |

**Table 3:** Wait time categories

## Basic Best Practices for Wait Time Analysis

1. **Top-Down**: Begin with database-level metrics, then narrow your focus to the most impacted area (which might be bufferpools, tablespaces, individual SQL statements or so forth).
2. **Use the Right Tools**: Choose the tool that best fits the depth and scope of your analysis: db2mon, monreport.dbsummary() or MON\_GET\* table functions
3. **Prioritize Common Waits:** Focus on lock waits, I/O waits, and latch contention before exploring less common issues.
4. **Always use Delta Metrics:** Avoid relying on cumulative metrics. Use temporary tables and time-based snapshots to calculate deltas for accurate insights.

# **Conclusion**

Db2 LUW performance tuning is not just about identifying slow queries—it is about understanding where time is being spent and why. By focusing on wait times and leveraging modern monitoring tools like db2mon and MON\_GET\_\* table functions, you can follow a data-driven diagnostic strategy.

The concepts introduced in this post are designed to help you start to pinpoint performance root cause. Distinguishing between application and database server delays allows effort to be spent in the right area.

Ultimately, performance tuning is an iterative process and may have many stages of change, testing and validation to reach the chosen goal. With the right tools, methodology, and mindset, you can turn performance measurements into actionable insights that drive real improvements in system responsiveness and user satisfaction.

# **About the Authors**

**C Ragahvendra** is a Senior Staff Software Engineer and DB2 LUW Product Engineer in IBM ISL Labs Bangalore. Raghavendra has over 13 years of experience in IBM and brings tons of intelligent ideas/suggestions/best practices for solving complex Db2 Performance issues. He is also an automation expert in Korn shell and ansible playbooks for various automation activities. For his tremendous efforts, he also received a prestigious “Hall of fame award” ,"IBM Rockstar Award" and "Outstanding Technical Achievement Award" for various automation activities for Michelin Client. Raghavendra is instrumental in identifying design issues in the db2 list utilities and the db2 list utility details command where Reorg is not displayed and raised a Db2 aha idea for it. Raghavendra is always committed and working towards the improvements of Db2 that could help all our Enterprise clients. He can be reached at [C.Raghavendra@ibm.com](mailto:C.Raghavendra@ibm.com)

**Toby Haynes:** With a background in Observational Astronomy at the University of Cambridge, England, Toby joined IBM Canada in 1999 as a software developer in the Db2 Runtime engine. After eight years of development experience, he moved over to the Db2 Performance team to start work on what would become Db2 pureScale v9.8, working under Steve Rees. In the late summer of 2018, he stepped up to the role of Technical Manager for Db2 pureScale, looking after an active team of developers and support analysts. Now a Senior Software Developer at IBM, Toby is examining many areas of Db2 for robustness, usability and performance excellence.