developerWorks®

IBM i wait accounting

Thread-level run-wait analysis

Dawn May

August 22, 2013

IBM i wait accounting is a technology built into the operating system that can identify what every thread or task is doing on the system when it is not using the processor. Wait accounting is a very powerful tool for performance analysis and problem determination. This article describes wait accounting and explains how you can use it to troubleshoot performance problems or to improve the performance of your applications.

IBM i wait accounting

IBM i wait accounting is a technology built into the IBM i operating system that provides the ability to identify what any thread or task is doing when it is not using the CPU. Because threads and tasks wait for a wide variety of reasons, wait accounting can be a very powerful capability to aid in understanding the wait conditions and possibly eliminating or reducing wait time, which can have a significant effect on performance.

This article describes what wait accounting is and explains why threads wait and how you can use wait accounting to troubleshoot performance problems or to improve the performance of your applications.

Let's start by reviewing some terminology.

- A **job** is the construct through which all work is done on the system. Every job has at least one thread and can have multiple threads.
- A **thread** is a unit of execution within a job; all jobs have at least one thread and can have many threads. Threads have structures that support their use by the operating system and applications.
- A Licensed Internal Code (LIC) **task** is a unit of execution that does not have the IBM i job or thread-level constructs. Every thread is represented by a task, but tasks also exist independently without the IBM i thread-level constructs. LIC tasks are generally not visible externally except through the IBM i performance tools or service tools. The only operating system command that can display LIC tasks is Work with System Activity (WRKSYSACT).

Wait accounting concepts apply to both threads and tasks. The terms thread and task are used when talking about an executable unit of work.

A thread or task can be in one of the following two states.

1. Running on the processor; this is the *running* state.
2. Waiting to run on the processor.
   When a thread or task is waiting to run on the processor, there are three different types of waits:

- **Ready to run, waiting for the processor:** This is a special wait state and is generally referred to as *CPU Queuing*; the thread or task is queued, waiting to run on the CPU. There are a few different reasons for CPU queuing to occur. The most common is when the CPU is over-capacity and is busy processing the higher priority work. As a result, the lower priority tasks and threads have to wait on a queue for the CPU to become available.
- **Idle waits**: An idle wait is a normal and expected wait condition. Idle waits occur when the thread is waiting for external input. The input might come from a user, the network, or another application. Until the input is received, there is no work that can be done.
- **Blocked waits:** Blocked waits are generally a result of serialization mechanisms to synchronize access to shared resources. Blocked waits might be normal and expected – for example, serialized access to updating a row in a table, disk I/O operations, or communication I/O operations. But, if there are too many *normal* wait conditions due to serialization, their analysis might yield insights for improving the application design for improved parallelism. Blocked waits might also not be normal; wait accounting can be used to identify unexpected block points which can then be analyzed to understand why the wait is occurring and how to possibly reduce or even eliminate the wait condition.

The identification and tracking of wait information is done automatically by the operating system for all threads and tasks all the time. Therefore, the information that you need to do the wait analysis is always available.

Wait buckets

Nearly all of the wait conditions in the IBM i operating system have been identified and enumerated – that is, each unique wait point is assigned a numerical value. This is possible because IBM has complete control over both the LIC and the operating system. In addition, the architecture of the operating system is highly componentized and well-defined interfaces exist for various serialization mechanisms. As of the IBM i 6.1 release, there are 268 unique wait conditions! Keeping track of over 250 unique wait conditions for every thread and task would consume too much storage, and therefore, a grouping approach has been used. Each unique wait condition is assigned to one of the 32 groups or 'buckets'. As threads or tasks go into and out of wait conditions, the task dispatcher maps the wait condition to the appropriate group.

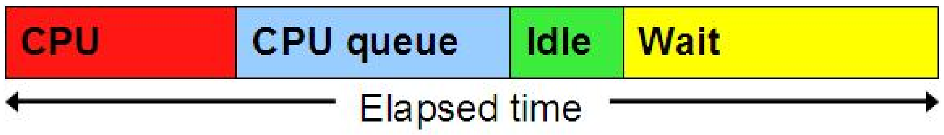
Having an understanding of wait accounting can help you better understand the performance charts you can display with the Performance Data Investigator tool and the way to interpret the data you are viewing.

The Wait bucket descriptions section reviews the 32 wait buckets that exist in the IBM i 6.1 and 7.1 releases.

Run-wait time signature

You can think of the life span of a thread or a task in a graphical manner, breaking out the time spent running or waiting. This graphical depiction is called the **run-wait time signature**. At a high level, this signature looks as follows:

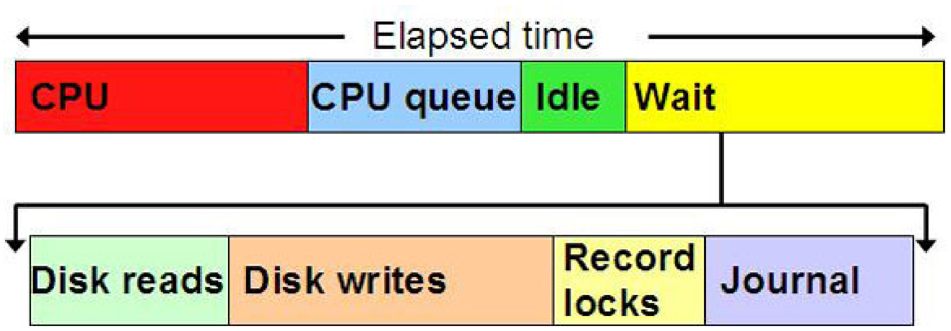
Figure 1: Run-wait time signature



Traditionally, the focus for improving the performance of an application was to have it use the processor as efficiently as possible, or to have more and faster processors. On IBM i with wait accounting, we can examine the time spent on waiting and understand what contributed to that wait time. If there are elements of waiting that can be reduced or eliminated, then the overall performance can also be improved.

If we take the run-wait time signature, using wait accounting, we can now identify the components that make up the time the thread or task was waiting.

Figure 2: Run-wait time signature components



For example, if the thread's wait time was due to reading and writing data to disk, locking records for serialized access and journaling the data, we could see the waits as broken out above. When you understand the types of waits that are involved, you can start to ask yourself some questions. For the example above, the following questions can be asked:

- Are disk reads causing page faults? If so, are my memory pool sizes appropriate?
- What programs are causing the disk reads and writes? Is there unnecessary I/O that can be reduced or eliminated? Or, can the I/O operations be done in a manner (for example, asynchronously) so that they would not affect the runtime of the thread or task?
- Is my record-locking strategy optimal? Or, am I locking records unnecessarily?
- What files are being journaled? Are all the journals required and optimally configured?

With the IBM i 6.1 release, there were 32 wait groups or *buckets* that have been defined. The definition of the wait groups varies from release to release and might change in the future. At the very end of this article, you can find a list of the 32 wait groups along with a high-level description of each.

Many of these wait groups are visible when you do the wait analysis on your application. Understanding what your application is doing and why it is waiting in those situations can possibly help you reduce or eliminate unnecessary waits.

Holders and waiters

Not only does IBM i keep track of the resource that a thread or task is waiting on, but it also keeps track of the thread or task that has the resource allocated to it. This is a very powerful feature. A **holder** is the thread or task that is using the serialized resource. A **waiter** is the thread or task that wants access to that serialized resource.

Call stacks

IBM i also manages call stacks for every thread or task. This is independent of the wait accounting information. The call stack shows the programs and procedures that have been invoked and can be very useful in understanding the logical flow that leads to getting into a wait condition. The combination of holders and waiters along with call stacks provide a very powerful capability to analyze wait conditions.

No other operating system provides such a rich function!

Collecting job wait data

Collection Services and **Job Watcher** are the two commonly used performance data collection mechanisms on IBM i that collect the wait accounting information. Collection Services is on by default and always collects the job wait data.

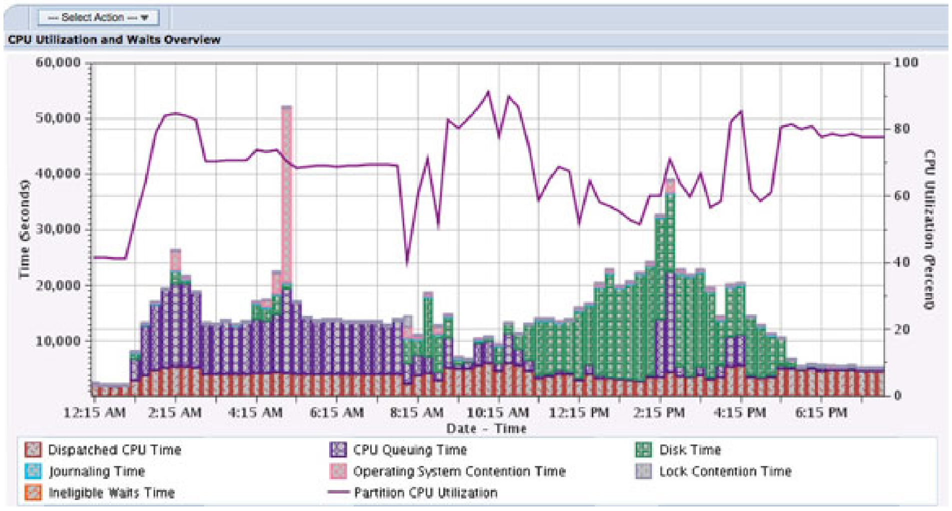
Job Watcher must be started when you need to collect job watcher performance data and it too will automatically collect the job wait data. Job Watcher also collects holder and waiter information, and (optionally) call stacks and SQL query related data. If you want to use Job Watcher for wait analysis, you need to ensure that you are collecting the call stack data. This is accomplished by having the Additional Data Categories (**ADLDTACGY**) parameter on the Job Watcher definition (**ADDJWDFN**) to specify that call stacks (***CALLSTACK**) are to be collected .

Analyzing job wait data

After the performance data has been collected, you can graphically analyze the data. Starting with the IBM i 6.1 release, the IBM Navigator for i web console has the **Investigate Data** feature to graphically view performance data through a browser interface. The Performance Data Investigator article reviews the capabilities of this analysis interface.

A good starting point for viewing job wait data is the CPU utilization and waits overview perspective. This chart shows the CPU utilization of the partition along with the wait information for the collection. The example shown below is from Collection Services data, but a similar chart is available with Job Watcher data.

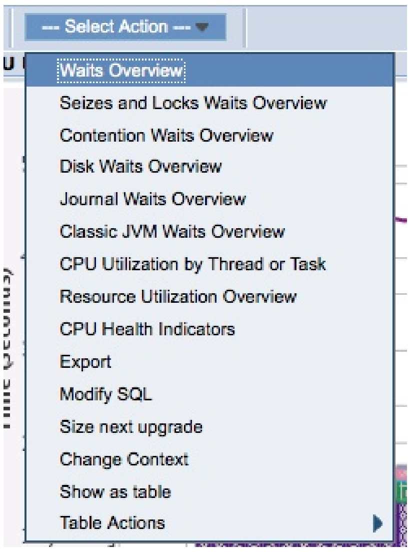
Figure 3: CPU utilization and waits overview example



As you can see from the figure, the wait information is displayed as stacked bars in the graph. In this particular example, you can see dispatched CPU time, CPU queuing time, disk time, and operating system contention time all show up as areas of interest. Lock contention time exists, but it is a small amount throughout most of the intervals. In this particular example, we need to better understand what is causing the single spike in operating system contention time (although collection services data might not have sufficient details to do so; Job Watcher data will most likely be needed to get to root-case resolution) and what is causing the disk time. The disk time could very well be a normal behavior.

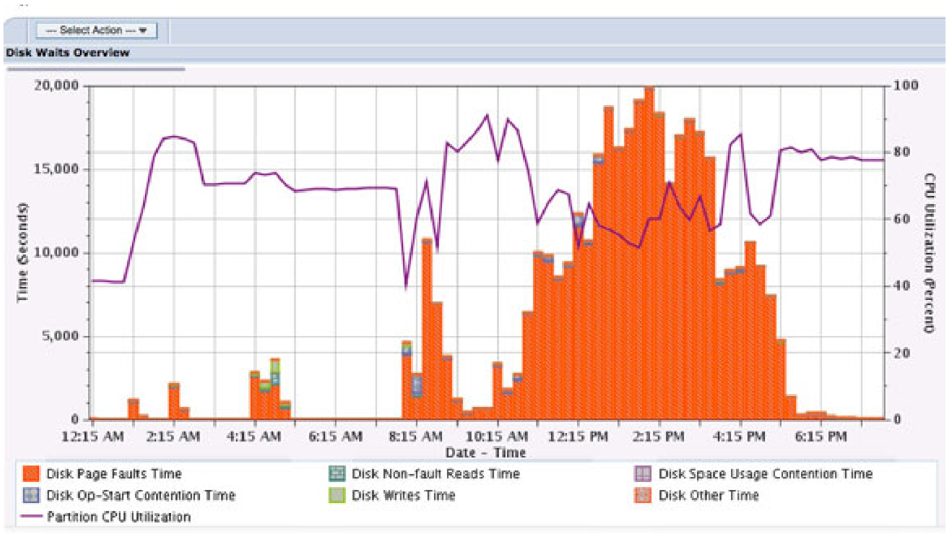
Depending upon the type of wait data you see, you can select the appropriate drill-down option to get more information about the waits, which can include the jobs that are in that wait state. The following figure shows the drill-down options that are available at this point with Collection Services data.

Figure 4: Drill-down options provide additional options to view wait data



If we take the disk waits overview drill-down option, we can see that the vast majority of the disk time was due to page faulting. Further analysis with the Collection Services data can help us identify the jobs and threads associated with that disk fault time.

Figure 5: Disk Waits show page faulting



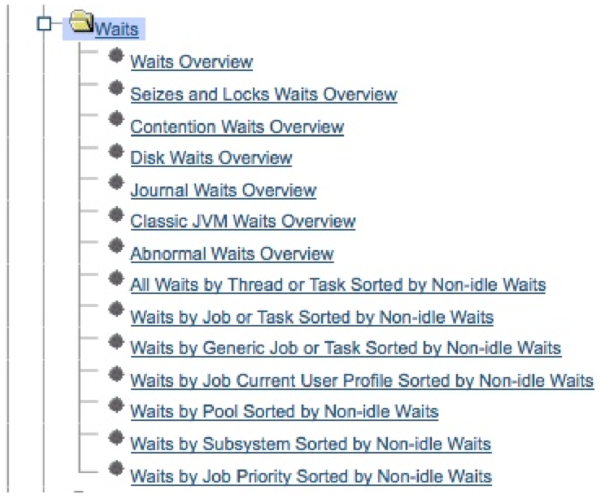
With the Performance Data Investigator tool, you can view wait information in many different charts. Not all charts display the wait data in the same way. With Collection Services data, for example, the **CPU Utilization and Waits Overview** chart shows a subset of the wait buckets. There are seven categories and some buckets are grouped. **Waits Overview** displays 15 of the selected waits buckets. **All Waits by Thread or Task** provides all the buckets, but should be used across narrowed-down intervals for better response time.

Using the Performance Data Investigator tool, you can also analyze Job Watcher data in a manner very similar to how you analyze Collection Services data. Job Watcher is a powerful performance tool because it allows you to collect and examine call stacks to determine the function that is being run, which can be used to help you figure out why that function is experiencing contention. In addition to Job Watcher, you can see who is holding the resource. Assuming that call stacks are collected, you can also see the program logic flow for how the thread got into the particular wait state.

Regardless of the types of waits you can see, you also need to understand what is causing the contention to make any changes to alleviate the condition. In general, if you see any kinds of contention, you should look at the call stack to determine the function that is being run, and then try to discern why that function is experiencing contention by examining the holders and waiters for the resource being waited on.

Analyzing Job Watcher data has more charts and tables associated with wait data and additional drill-down options that are not available when analyzing Collection Services data. The following figure shows the list of supplied wait graphs that you can view with Job Watcher data.

Figure 6: Wait graphs using Job Watcher data


Waits
- Waits Overview
- Seizes and Locks Waits Overview
- Contention Waits Overview
- Disk Waits Overview
- Journal Waits Overview
- Classic JVM Waits Overview
- Abnormal Waits Overview
- All Waits by Thread or Task Sorted by Non-idle Waits
- Waits by Job or Task Sorted by Non-idle Waits
- Waits by Generic Job or Task Sorted by Non-idle Waits
- Waits by Job Current User Profile Sorted by Non-idle Waits
- Waits by Pool Sorted by Non-idle Waits
- Waits by Subsystem Sorted by Non-idle Waits
- Waits by Job Priority Sorted by Non-idle Waits

Wait bucket descriptions

Wait bucket descriptions

The following list shows the wait buckets in the IBM i 6.1 and IBM i 7.1 releases along with a high-level description of each.

1. **Time dispatched on a CPU**
   Time dispatched on a CPU is a special bucket and contains the time the thread or task was dispatched on a CPU. This is the task dispatcher's view of when the thread or task was dispatched to the virtual processor, which is not the same as the CPU time that is tracked for the thread or task.
2. **CPU queuing**
   CPU queuing is a wait state where the thread or task is queued, waiting to run on the processor. There are a few different reasons that CPU queuing can occur. The most common is if the partition is over-capacity and there is more work than what the partition processing resources can accommodate. In such a case, the work is queued to wait for the CPU. Workload capping latency is another wait that might contribute to CPU queuing.
3. **Reserved**
   The *reserved* wait bucket is not used.
4. **Other waits**
   Other waits is a catch-all grouping where wait points that do not fit well into other groups are placed, and as such, there is no general description for this bucket. There are a variety of wait conditions that are grouped into this bucket, including general purpose waits that are not further uniquely identified.
5. **Disk page faults**
   Disk page faults are those wait points that are related to faulting; page faulting on IBM i is often normal and expected. Page faults occur when data from a disk must be brought into the main storage but it was not explicitly read in. The disk page faults wait bucket can help you identify whether your faulting might be excessive and what is causing it, which can then

lead you to assess if you need to take action to reduce the faulting rate. For example, you might need to adjust memory pool sizes to keep frequently used data in memory. There are a variety of techniques that can be used to optimize bringing data from disk into memory, which are beyond the scope of this article.

6. **Disk non fault reads**
   Disk non fault reads are explicit synchronous reads that are performed to bring data into the main store from the disk. When the synchronous read operation is performed, the application will wait for the read operation to complete. The *disk non fault reads* wait bucket can show you how much time your application has spent on reading data from the disk and can help you assess whether that time is significant enough to consider application changes, such as asynchronous reads.
7. **Disk space usage contention**
   When an object is created or extended, free disk space has to be located to meet the request and there is some level of serialization that is associated with that. Typically, you should see very little of this type of wait. If it is present in significant percentages, it usually means that your application is performing a high rate of object create/extend/truncate/delete operations. (Note that opening a DB2 file causes a create activity). The size of the disk space requests is not important; it is the rate of the requests that is important.
8. **Disk op-start contention**
   Disk op-start contention can occur when a disk operation start is delayed due to a very high rate of concurrent disk operations in progress at the moment it is requested. If you see this wait type, you might need to look at the overall disk operations occurring to see if there are significant disk I/O inefficiencies that should be eliminated.
9. **Disk writes**
   Disk writes are explicit synchronous writes that are performed to store data from the main store to the disk. When the synchronous write operation is performed, the application waits for the write operation to complete. The *disk writes* wait bucket can show you how much time your application spends on writing data to the disk and can help you assess whether that time is significant enough to consider application changes, such as asynchronous writes. However, this wait bucket also includes waits that are related to waiting for asynchronous disk writes to complete.
10. **Disk other**
    Disk other waits is the catch-all grouping for all the other reasons that the system may wait for disk operations.
11. **Journaling**
    As the bucket name implies, *journaling* waits are those waits that are associated with IBM® DB2® journaling; the disk write operations to write to the journal. These write operations are separate from the general disk write operations and allow you to understand journal writes separately. If you see high rates of journal waits, you might want to review the journal design and application's use of journaling.
12. **Semaphore contention**
    Semaphores are a serialization mechanism that is generally used in UNIX/POSIX applications. *Semaphore contention* identifies the waits that occur when using semaphores. Semaphores can be used by applications, licensed program products, as well as the operating system.

13. **Mutex contention**
    Mutexes are a serialization mechanism that is generally used in UNIX/POSIX applications. *Mutext contention* identifies the waits that occur when using Mutexes. Mutexes can be used by applications, licensed program products, and the operating system.
14. **Machine level gate serialization**
    Machine level gate serialization is an internal serialization mechanism used within the licensed internal code to serialize logic and access to data structures. If there is a locking conflict on a gate, it is shown as machine-level gate serialization. While gates are an internal locking mechanism, actions that you take on certain functions (for example, journals) can result in gates being used to complete those functions.
15. **Seize contention**
    Seize contention is an internal serialization mechanism used within the licensed internal code to serialize access to objects and data structures; seizes are sort of an internal version of locks. Seize contention occurs when multiple threads flow through the same seize/release logic. If you see seize contention, you should look at the call stack to determine the function that is being run, and then try to discern why that function is experiencing contention.
16. **Database record lock contention**
    Database record lock contention occurs when your application has locking conflicts on database records. If you see this lock contention, you should look at your application's DB record locking logic.
17. **Object lock contention**
    Object lock contention occurs for a variety of reasons – your application can explicitly lock objects using the ALCOBJ command. The operating system might implicitly lock objects when certain actions (such as creating or deleting an object, moving it, or changing the ownership) are performed on the objects.
18. **Ineligible wait**
    This bucket is the amount of time that a thread has been in an *ineligible wait* state. Ineligible waits generally occur if the maximum active (MAXACT) parameter of the memory pool is too small.
19. **Main storage pool overcommitment**
    Main storage pool overcommitment occurs when a main storage pool is overcommitted. Disk reads or page faults are delayed in order to locate free main storage page frames.
20. **Classic Java user including locks**
    See the "Classic Java Virtual Machine (JVM)" description.
21. **Classic Java Virtual Machine (JVM)**
    The wait buckets associated with Classic JVM. However, during the IBM i 7.1 release, Classic JVM is no longer supported, and therefore, these wait buckets are essentially reserved. Note that with the IBM Technology for Java, which is the only supported JVM for IBM i 7.1, the waits are actually accumulated in the Portable Application Solutions Environment (PASE) bucket, because this JVM is implemented in PASE.
22. **Classic Java other**
    See the "Classic Java Virtual Machine (JVM)" description.
23. **Socket accepts (idle)**
    Socket accepts (idle) is a normal idle wait that occurs when waiting for a request to arrive on a socket.

24. **Socket transmits**
    Socket transmits are the waits associated with sending data through the sockets' application programming interfaces (APIs). Note that many TCP/IP application servers use the sockets APIs.
25. **Socket receives**
    Socket receives are the waits associated with receiving data through the sockets' APIs. Note that many TCP/IP application servers use the sockets' APIs.
26. **Socket other**
    Socket other waits are due to I/O completion waits or use of the select-socket API.
27. **IFS waits**
    IFS waits are those waits that are due to the integrated file system (IFS) pipe operations.
28. **PASE**
    PASE waits are those waits that occur in the Portable Application Solutions Environment (PASE). PASE is an IBM AIX® runtime environment that allows AIX applications to run on IBM i. The IBM Technology for Java JVM also runs in PASE. There may be a large variety of applications that run in PASE and no further detail on the waiting behavior within those applications is available.
29. **Data queue receives**
    Data queue receives are simply those waits on data queue objects.
30. **Idle / waiting for work**
    Idle / waiting for work is the bucket that contains the normal wait conditions that occur when an application is in an expected wait state. As was described earlier in this article, idle waits are normal and expected, and as such, these wait types are grouped into this one wait bucket. In general, the analysis of normal wait conditions is uninteresting.
31. **Synchronization token contention**
    Synchronization tokens are a serialization mechanism that is generally used in UNIX/POSIX applications. *Synchronization token contention* identifies the waits that occur when using Synchronization tokens. Synchronization tokens can be used by applications, licensed program products, as well as the operating system.
32. **Abnormal contention**
    Abnormal contention is another catch-all bucket and generally contains unexpected or rare wait conditions that you should generally not see.

Conclusion

IBM i wait accounting is a very powerful capability for an improved understanding of the lifespan of a thread or a task and can be used for both performance troubleshooting and application optimization.

Resources

- i Can Tell You Why You are Waiting provides a high-level overview of IBM i wait accounting.
- Job Waits White Paper describes wait accounting from a much more technical view and contains additional details about wait accounting that are beyond the scope of this article. It also contains description about wait buckets.

- IBM i Performance Data Investigator is a web-based graphical analysis tool that can be used to analyze wait data.
- IBM iDoctor is a client-based graphical analysis tool that can be used to analyze wait data.
- IBM i Performance Tools on the developerWorks site provides a lot of useful information and references on the IBM i performance tools topics.

Discuss

Participate in the following IBM i forums:

- IBM i
- IBM i Performance Tools