IBM iDoctor for IBM i
Collection Services Wait Buckets (V5R3/V5R4)

IBM iDoctor for IBM i Development Team

11 January 2009

Abstract

This document provides a discussion of the Wait Buckets used by Collection Services and shown within iDoctor Collection Services Investigator.

Changes

11 Jan 2009 – Updated for website
2 Nov 2006 – Initial Creation

Introduction

This document is a discussion of the i5/OS "Wait Buckets" as seen in Collection Services performance data for OS versions V5R3 and V5R4. This data is found in performance data files QAPMJOBWTD (bucket descriptions) and QAPMJOBWT (wait bucket data for each job, task or thread).

Note: In IBM i 6.1 both Collection Services and Job Watcher share the same 32 bucket mapping (not discussed in this document).

Wait Buckets

Wait buckets do not really account for resources used; they are used to account for **elapsed time**. The wait buckets for a job, thread or task quantify how long it has:

1) Been "active" (dispatched on a partition's processing unit) or
2) Been "ready to run" or "CPU queued" or
3) Waited, differentiated into a variety of wait reasons (buckets)

In general, the sum of all these times should match elapsed ("wall-clock") time, with two potentially large exceptions:

- The current, "in progress" wait at the time Collection Services takes its interval sample
- Entire intervals in which a thread or task has not used CPU (Collection Services' "zero CPU" exclusion)

Through V5R4, Collections Services does not include the current, in progress wait, into the wait buckets data. It is not included until the wait ***completes***. See **Current Wait** and **Effects of Long Waits** below, after the Wait Buckets descriptions for more details.

The Wait Buckets (WBs), and information associated with them, are maintained within the system on a per thread and per Licensed Internal Code (LIC) task basis. Each individually dispatchable unit of work (a thread of a Job or a LIC task) has its own set of WBs. Neither the system nor Collections Services aggregate together the WBs in multi-threaded jobs. Collection Services reports WBs for all threads and tasks that used CPU during a sample interval. Whenever a record is written to QAPMJOBMI file representing resources used by thread or task during the interval, a matching record is written to QAPMJOBWT file, representing WBs of this thread or task.

There are two metrics associated with each WB: count and time

Count: the number of times the thread or LIC task has experienced a state covered by the specific WB (more precisely, how many times it has transitioned out of that state)
Time: the elapsed time (wall-clock time) the thread or LIC task has spent in a state covered by the specific WB

Collection Services uses a default layout of WBs - approximately 200 individual wait points are mapped to 16 wait buckets.
However, it should be noted that there is another tool that harvests WBs - Job Watcher. Job Watcher uses the same individual wait points, but they are mapped differently - to 32 WBs instead of 16.
Job Watcher WBs are described in detail in a Job Watcher document, different versions of which are linked from here https://www-912.ibm.com/i_dir/idoctor.nsf.
Job Watcher WBs are reported in a different set of files and should not be normally a concern for Collection Services users.
However, running Job Watcher on a system will have an impact on how Collection Services reports WBs.
1) On V5R3 system, Job Watcher will completely redefine the WBs used by the system.
The WBs reported by Collection Services, will be 32 WBS defined by Job Watcher. See Job Watcher documentation for the details about that set of WBs.
2) On V5R4 system, Collection Services will always report 16 default WBS. In addition, if Job Watcher is active, Collection Services will report Job Watcher buckets 17 to 32. It is important to understand that buckets 1 to 16 always account for entire lifetime of a thread or task. Buckets 17 to 32 are meant to provide additional detail. In other words - time reported in buckets 17 to 32 is also reported in some buckets 1 to 16.

When in doubt, look at the contents of QAPMJOBWTD file. That file reports the descriptions of all the buckets currently in effect.

The rest of this document describes the default layout that consists of 16 WBs.

OK, what are wait "mappings"?

In an i5/OS partition, ALL "waiting" is done-at, or controlled-by the LIC. This is very advantageous, from a performance-monitoring point of view in that it allows ALL wait points to be instrumented with some type of Wait Accounting.

"ALL" truly means all wait points. These run the gamut from:

- Waiting for a disk read operation due to a page fault
- Waiting for internal serialization(s) during LIC access to an object
- Waiting for a data base record lock
- Waiting for a thread to start or end in Java
- Waiting on a Socket Accept or Receive for a piece on incoming data

- Waiting for a user to "press Enter" in a 5250 interactive application
- Waiting induced by HLDJOB (hold job), or by waiting for an Activity Level slot to become available

In all, there are approx. 200 such wait points, or wait reasons.

It would be good if all 200 individual wait points had their own pair of (count, time) accounting. However, this would consume too much system resource, specifically main memory. Instead, the system maps the 200 wait points into either 16 or 32 "buckets".

The default 16 Wait Buckets

The titles of the 16 default WBs, as seen in the Collection Services file named QAPMJOBWTD are:

```
 1   Time dispatched on a CPU
 2   CPU queuing
 3   Reserved
 4   Other waits
 5   DASD (page faults)
 6   DASD (other)
 7   Socket
 8   Idle / waiting for work
 9   Seize
10   Object lock
11   Record lock
12   Gate
13   Java
14   Journal
15   Mutex and Semaphore
16   Reserved
```

WB 1: "Time dispatched on a CPU". This is really not a WAIT bucket! This is the only one of the 16 that represents any type of "active" time. The official definition of WB1 is "time dispatched to a virtual processor within the LPAR". This accumulates the amount of wall-clock time a thread or task has been “dispatched” to a virtual processor. “Dispatched to a virtual processor” means the thread or task has been assigned a virtual processor” - not necessarily that it has been actually using CPU resource.

The dispatched time value can differ from the “CPU Time” as measured by other means (DSPJOB, WRKACTJOB, WRKSYSACT, job accounting, the JBCPU field itself in Collection Service's QAPMJOBMI file. True CPU time represents the **Physical Processor** time consumed by a thread or task. The difference between WB1 and CPU time can be large. The main factors that cause these discrepancies are:

- Both the Processor Hardware Multi-Threading (HMT) and the Simultaneous Multi Threading (SMT) features. This can cause bucket 1's time to be larger than the actual CPU time. HMT is when more than one thread or task can be simultaneously assigned to the same physical processor. In that scenario, they share the processor's cycles, mainly during long "off chip" operations, like memory fetches. Bucket 1 will record the elapsed time a thread or task has been dispatched. The "real" CPU value will only include the exact number of cycles used by the thread or task while it was dispatched.

- Background assisting tasks, like those used in the DB Multitasking Feature. Background assisting tasks, which promote (add) their CPU usage back into the client job/thread, will cause the client thread's bucket 1 value to be smaller than the measured CPU time.

- LPAR shared/partial processors. This is where the tricky concept of Virtual Processors (VPs) comes into play. Bucket 1 actually records the elapsed time a thread or task is dispatched to a *Virtual Processor*, not (necessarily) a Physical Processor. A VP is known only within the LPAR it is configured. The hypervisor assigns Physical Processors to VPs. There can be a one-to-one mapping, or there can be more VPs than there are Physicals. The Physicals can be shared, wholly or partially, across LPARS. Due to this high degree of VP <-> Physical configuration options, it's possible that when a thread or task is dispatched to a VP, the bucket 1 time will be greater than the CPU time. This is because WB1 will include time the thread/task is dispatched, but is "waiting for it's turn" at a physical processor.

WB 2: "CPU Queuing". This is simply the amount of time a thread/task has waited… ready to run… for a virtual processor to become available *within the LPAR*. Note, in some ways, this bucket is not much different than the "LPAR shared/partial processor" aspect of bucket 1. It is just that the time waiting for a physical processor is accounted for differently.

Summary and example of WB1 and WB2:

This graphic shows the relationship between WB1, WB2 and "real" CPU time:

```
-------------------------------------------------------------------------------------------> time
 --Wait--><-----"ready to run"--------><-------on a VIRTUAL Processor------->
          <----------WB2------------->
                                      <-----------------WB1---------------------->
                                       XXXXcccXXXXXXXXXcccXXXXccc
```

c = Real CPU time (msecs in QAPMJOBMI for example).
X = Could be a combo of competing for a physical processor and/or accounting considerations of Processor Multithreading

A thread is ready to execute instructions, because its current wait has ended. For example, a Socket Receive was satisfied. Call this time A. At time A, the i5/OS tries to dispatch the thread to a virtual processor. Suppose none are available until time B. The delta of B-A is added to bucket 2. At time B, this also begins the timing of what will eventually be added into WB 1. Suppose the virtual processor is shared or partial and a physical processor is not yet available. During this period, time is still "accumulating" in what will be added to WB 1. Eventually, a physical processor becomes available and the virtual processor is assigned to a physical processor at time C. Suppose the thread uses a little CPU and then blocks on another Socket Receive at time D. Delta D-C represents the maximum amount of "real" CPU time that would be "charged" to the thread. Usually, it is roughly 1/2 of D-C because the processor is running in Multi-Threading mode, and 2 threads are sharing the cycles of a physical processor. Delta D-B is added into WB1. This example is the simplest in some ways, depending on how much real CPU a thread needs before it blocks again, there could be several cycles of this sequence occurring between blocks.

Note: there may always be minuscule amounts of time reported in WB2; even if there are virtual processors are readily available. It's an artifact of the fact that SOME tiny, finite amount of time transpires between when a thread becomes "ready to run" and when it is dispatched to a processor.

WB 3: "Reserved". No wait points assigned. There may be nonzero values in the (count, time) for this bucket, but they have no meaning.

WB 4: "Other Waits". In V5R4, there are 27 individual wait points (block points) mapped into WB4. This is the unfortunate, but needed "catch all" bucket. All wait points not otherwise differentiated (via assignment to buckets 4-16) are included here. As with many of the other buckets described here, if WB4 is a large percentage of a thread's wait time, it might be necessary to dig deeper. See **How to Dig Deeper** below.

There is a particular type of wait that is mapped into the default WB4 worth mentioning. These are the IFS (Integrated File System) "Pipe" waits. If an application is known to use IFS Pipe operations, then it's possible that these waits...in WB4... could be "normal".

How to Dig Deeper

This applies to all WBs.

There are several options available to gather more information about ambiguous WBs:

1) Good Run vs. Bad Run approach. Look at Collection Services data from several time periods. Include those that had good response time and/or throughput, and those that had poor. You might learn, for example, that during "good" runs, WB4 and WB7 have high percentages for particular threads. Without digging into those further, you could draw (the reasonably sound) conclusion that those waits are "OK". Then, when a run comes by with a radically different breakdown of waits, you could conclude that the run is either radically better or more likely, radically worse. In that case, spend the time to dig into the newly high WBs, not the "normal" ones (yes, the "normal ones" could get bigger.... it's not an easy "science"!).

2) Exploit Collection Services "Current Wait" information. Current Wait info, in either Collection Services or Job Watcher, can give clues as to which specific wait/block point might be involved of all those mapped to any given WB. See the description of "Current Wait" after the bucket descriptions.

3) Use Job Watcher to harvest Current Wait, Current Wait Object, the Holding thread or task (if applicable) and the Call Stack of the waiting thread or task. (Of course, as mentioned above, Job Watcher uses a different waits->buckets mapping, so be careful).

Further discussion on this is beyond the scope of this document.

WB 5: "DASD (page faults)". In V5R4, there are 6 individual wait points (block points) mapped into WB5. These represent the various ways a thread or task can wait for a page fault read from disc. As with all the WBs, some simple division of the time/count can yield some interesting data. For WB5, this would be the average time page faults reads had taken for the thread or task.

WB 6: "DASD (other)". In V5R4, there are 51 individual wait points (block points) mapped into WB6. These represent almost all (see Journal below) other waits for disc operations, other than page faults. It includes:

- Synchronous, non-fault reads
- Waiting for asynchronous reads to complete
- Synchronous writes
- Waiting for asynchronous writes to complete

WB 7: "Socket". In V5R4, there are 14 individual wait points (block points) mapped into WB7. These are all the various ways a thread can wait/block on a Socket. For example:

- Receive
- Accept
- Send

WB 8: "Idle/Waiting-for-work". In V5R4, there are 5 individual wait points (block points) mapped into WB8. There are so few, they might as well be itemized here:

- Classic Java Virtual Machine (JVM) "user" waits
- APPN, 5250 display and 5250 printer communications waits
- Data Queue waits

Note: this bucket, like the rest, is far from perfect. The preceding one, WB7-Socket, in many applications would also represent waiting-for-work. So could an IFS Pipe wait from WB4.

WB 9: "Seize". In V5R4, there are 17 individual wait points (block points) mapped into WB9. Think of seizes as the Licensed Internal Code's (LIC's) equivalent of Locks. A seize almost always occurs on/against an MI object (DB2 Physical File Member, DB2 Access Path, Data Queue, Library, User Profile…). Seizes can conflict with Locks and can cause Lock waits. There are a large variety of seizes: shared, exclusive, "fair", and "intent-exclusive". It's beyond the scope of this document to explain all there is to know about seizes. They are, after all, internal LIC primitives that are subject to change at any time. If seizes are a significant percentage of a Run/Wait Signature, consider following **How to Dig Deeper** above. Job Watcher is especially useful for seizes in that the object name involved and the Holding thread/task are frequently very useful pieces of information to have.

WB 10: "Object Lock". In V5R4, there are 22 individual wait points (block points) mapped into WB10. Object locks are used by i5OS, LPPs and applications (ALCOBJ command) to serialize usage of an object. The same How **to Dig Deeper** considerations apply as per WB9-Seizes.

WB 11: "Record Lock". In V5R4, there are 6 individual wait points (block points) mapped into WB11. This should be self-explanatory. Job Watcher will resolve the DB2 file/member name AND the relative record involved.

WB 12: "Gate". In V5R4, there are 2 individual wait points (block points) mapped into WB12. Gates are a low level, high performance serialization mechanism used by LIC. There are many (far more than the 2 mapped to this WB) types of waits that have a gate as the actual wait mechanism. Fortunately, many of these are more granularly identified, and are included in other WBs. WB12 is the "catch all"... for the gate waits that have not otherwise been "individualized".

Threads vs. LIC Tasks: WB12 is a good place to briefly talk about the differences in what is considered "normal" or "abnormal" waits... from the perspective of them occurring in job threads or LIC tasks. In general, high WB12 times in job Threads is "bad", and warrants further investigation. High WB12 times in LIC tasks are more likely "normal".

WB 13: "Java". In V5R4, there are 20 individual wait points (block points) mapped into WB13. These waits are all the Java waits that are not considered "idle" (idle ones are included in WB8). Note: this WB only applies to Classic Java Virtual Machine (JVM) (verses IBM® Technology for Java™ Virtual Machine).

WB 14: "Journal". In V5R4, there are 4 individual wait points (block points) mapped into WB14. These waits represent how long a thread has waited for Journal related activities. Principally, waiting for the bundled Journal disk write(s) to complete.

Note: there is an incorrect mapping in V5R4. There are 2 non-Journal block points mapped to WB14:

- Mutex Waits
- Semaphore Waits

Both are ILE Posix style synchronization mechanisms used by LPPs and application programs. Because of this, they will not be mapped into WB15, as they should be.

WB 15: "Mutex/Semaphore". In V5R4, there are only 2 individual wait points (block points) mapped into WB15. However these blocks points are essentially never used in V5R4. In V5R3 these are used and do represent Mutex and Semaphore waits. See the description of WB14 above... in V5R4 that is where Mutex and Semaphore waits will be reported.

The following is a brief description of the four types of wait mechanisms that are mapped to WB15. These are used in ILE applications and LPPs. Such as Domino, MQ Series and others.

Mutex: A mutex is a mutual exclusion synchronization mechanism that can be used to control access to shared data areas, where access to the same data is needed by multiple threads in a process and/or by threads in multiple processes. The locking mechanism of a mutex ensures that while one thread has a mutex locked, no other threads can successfully lock the same mutex at the same time, thereby controlling access to the data the mutex protects.

Semaphore: In general, a counting semaphore, or semaphore for short, is a synchronization device which permits serialization of a finite number of resources. A semaphore consists of two pieces of state information: a count and max_count. The range of the count varies in the industry, but in no case can the count exceed the max_count. A semaphore is in its signaled state if count > 0, and it is nonsignaled otherwise. The two basic operations performed on a semaphore are wait and post. A wait executed against a semaphore with a count > 0 causes the count to be decremented by 1. A wait executed against a semaphore with a count <= 0 causes the calling thread to block until the count is increased (or effectively increased if there are waiters currently blocked on the semaphore) by a post operation. A post operation increases (or effectively increases) the count by a positive amount, n, up to the max_count of the semaphore. A post operation may unblock up to n threads, in priority order.

WB 16: "Reserved". In V5R4, there are 23 individual wait points (block points) mapped into WB16. Despite of the name, this WB is not "reserved". The wait points mapped to it represent "bad" waits. The include:

- Multiple retries on low-level serializations
- Excessively "busy" wait objects, like queues
- Running low of free frames in a mainstore pool
- Contention on low level LIC Storage Management constructs
- Excessive number of outstanding disc I/O operations

Current Wait

In addition to the WB data, Collection Services also reports a so-called "Current Wait".
Current wait is reported when thread or task was in the wait state at the moment of sampling for the data.

Collection Services reports for the current wait:
- Duration of a current wait - since the wait condition started
- The WB that will be updated when the wait will be over
- The individual block point that has a detail reason for the wait.

Effects of long waits.

System updates the WB only after the wait completes.
Remember, that Collection Services does not report thread or task unless it used CPU in the current sampling interval.
Imagine that thread was in a long wait - the wait that spans one or more sampling intervals.
Collection Services will not report anything in the intervals where thread was blocked.
It will only be reported in the interval when wait will be over and thread will be able to use at least some CPU.
At this time, the entire duration of the wait will be added to the appropriate WB.
As a result, the time value in WB can be significantly larger than an interval duration.

Trademarks and Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

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

A8/400	e-business on demand	I5/OS
AS/400e	IBM	OS/400
eServer	IBM (logo)	System i5
	iSeries	DB2

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, other countries, or both.
Intel, Intel Logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks 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.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.