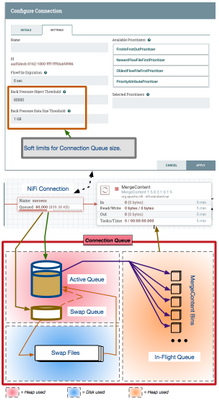
**Nifi Connection: Heap Usage**

A NiFi connection is where FlowFiles are temporarily held between two connected NiFi Processor components. Each connection that contains queued NiFi FlowFiles will have a footprint in the JVM heap. This article will breakdown a connection to show how NiFi manages the FlowFiles that are queued in that connection and that affects heap and performance.

Article:

First let me share the 10,000 foot view and then I will discuss each aspect of the following image:



\*\*\* NiFi FlowFiles consist of FlowFile Content and FlowFile Attributes/metadata. FlowFile content is never held in a connections heap space. Only the FlowFile Attributes/metadata is placed in heap by a connection.

The "Connection Queue":

The connection queue is where all FlowFiles queued in the connection are held. To understand how these queued FlowFiles affect performance and heap usage, lets start by focusing on the "Connection Queue" dissection at the bottom of the above image.

The overall size of a connection is controlled by the configured "back Pressure Object Threshold" and "Back Pressure Data Size threshold" settings the user defines per connection.

Back Pressure Object Threshold and Back Pressure Data Size Threshold:

The "Back Pressure Object Threshold" default setting here is 10000.

The "Back Pressure Data Size Threshold" defaults to 1 GB.

Both of these settings are soft limits. This means that they can be exceeded. As an example, lets assume default settings above and a connection that already contains 9,500 FlowFiles. Since the connection has not reached or exceeded object threshold yet, the processor feeding that connection will be allowed to run. If that feeding processor should produce 2,000 FlowFiles when it executes, the connection would grow to 11,500 queued FlowFiles. The preceding processor would then not be allowed to execute until the queue dropped below the configured threshold once again.

The same hold true for the Data Size threshold. Data Size is based on the cumulative reported size of the content associated to each queued FlowFile.

Now that we know how the overall size of "connection queue" is controlled, lets break it down it to its parts:

1. ACTIVE queue: FlowFiles enter a connection will initially begin to placed in the active queue. FlowFiles will continue to placed in to this queue until that queue has reached the global configured nifi swap threshold. All FlowFiles in the active queue are held in heap memory. The processor consuming Flowfiles from this connection will always pull FlowFiles from the active queue. The size of the active queue per connection is controlled by the following property in the nifi.properties file:

nifi.queue.swap.threshold=20000

Increasing the swap threshold increase the potential heap footprint of every single connection in your dataflow(s).

2. SWAP queue: Based on above default setting, once a connection reaches 20,000 FlowFiles, new FlowFiles entering the connection are placed in the swap queue. The swap queue is also held in heap and is hard coded to 10,000 FlowFiles max. If space is freed in the active queue and no swap files exist, FlowFiles in the swap queue will be moved directly to the active queue.

3. SWAP FILES: Each time the swap queue reaches 10,000 FlowFiles, a swap file is written to disk that contains those FlowFiles. At that point new FlowFiles are again written to the swap queue. Many Swap files can be created. Using image above where connection contains 80,000 FlowFiles, there would be 30,000 FlowFiles in heap and 5 swap files. As the active queue has freed 10,000 FlowFiles, the oldest swap file are moved to the active queue until all swap files are gone. The fact that swap files must be written to and read from disk, having a lot of swap files being produced across your dataflow will affect throughput performance of your dataflow(s).

4. IN-FLIGHT queue: Unlike the above 3, the in-flight queue only exists when the processor consuming from this connection is running. The consuming processor will only pull FlowFiles from the active queue and place them in the in-flight queue until processing has successfully completed and those FlowFiles have been committed to an outbound connection from the consuming processor. This in-flight queue is also held in heap. Some processors work on 1 FlowFile at a time, others work on batches of FlowFile, and some have the potential of working on every single FlowFile on an incoming connection queue. In the last case, this could mean high heap usage while those FlowFiles are being processed.

The example above is one of those potential case using the MergeContent processor. The MergeContent processor places FlowFiles from the active queue in virtual bins. How many bins an what makes a bin eligible for merge is governed by the processor configuration. What is important to understand that is is possible that every FlowFile in the connection could make its way into the "in-flight queue". In image example, if the MergeContent were running, all 80,000 queued Flowfiles would likely be pulled in to heap via the in-flight queue.

----

Take away from this article:

1. Control heap usage by limiting size of connection queue when possible. (Of course if your intention is to merge 40,000 FlowFiles, there must be 40,000 Flowfiles in the incoming connection. However, you could have two mergeContent processors in series each merging smaller bundles with same end result with less overall heap usage.)

2. With default back pressure object threshold settings, there will be no swap files produced on most connections (remember soft limits) which will result in better throughput performance.

3. The default configured swap threshold of 20,000 is a good balance in most cases of active queue size and performance. For smaller flows you may be able to push this higher and for extremely large flows you may want to set this lower. Just understand it is a trade-off of heap usage for performance. But if your run out of heap, there will be zero performance.