# Metrics



**Revision history**

STATUS: DRAFT

30 Sep 2023 Improve readability of the metrics table

29 Dec 2021 1st draft of Nutch Metric documentation completed and open for feedback (lewismc)

## Introduction

This page provides a narrative on Nutch application metrics. It details which metrics are captured for which Nutch Job's within which Tasks.

Metrics are important because they tell you vital information about any given Nutch (and subsequently MapReduce) process. They provide accurate measurements about how the process is functioning and provide basis to suggest improvements.

Metrics provide a data-driven mechanism for intelligence gathering within Nutch operations and administration.

## Audience

The page is intended for

- users who wish to learn about how Nutch Jobs and Tasks are performing, and
- developers who would wish to further extend/customize Nutch metrics

## Related Development Work

- **NUTCH-2909** - Getting issue details... STATUS
- Several rows in the **metrics table** below reference JIRA issues.

## Building Metrics on MapReduce Context's

As Nutch is a native MapReduce application, the Mapper and Reducer functions of each NutchTool implementation i.e. `CommonCrawlDataDumper`, `CrawlDb`, `DeduplicationJob`, `Fetcher`, `Generator`, `IndexingJob`, `Injector`, `LinkDb`, `ParseSegment` utilize ***MapContext***'s and ***ReduceContext***'s. These Context's are passed to the Mapper and Reducer initially during setup but also used throughout each Mapper or Reducer task lifecycle.

**Hadoop documentation**

The canonical Hadoop documentation for Mapper and Reducer provides much more detail about the involvement of Context's in each task lifecycle.

This is relevant because these Context's inherit certain methods from the interface org.apache.hadoop.mapreduce.TaskAttemptContext. Specifically, the ***getCounter(...)*** methods facilitate access to Hadoop **Counter**'s which we discuss below.

# Hadoop Counter's

A **Counter** is simply a record comprising a name and value. As one would expect, Counter's can be incremented in order to count for example how many total records were processed within a task completion.

The following example shows how Counter's are used within the Nutch Injector to count the total number of URLs filtered during the Map phase of this job.

**Use of Counters in the Nutch Injector**

```
@Override
public void map(Text key, Writable value, Context context)
    throws IOException, InterruptedException {
  if (value instanceof Text) {
    // if its a url from the seed list
    String url = key.toString().trim();

    // remove empty string or string starting with '#'
    if (url.length() == 0 || url.startsWith("#"))
      return;

    url = filterNormalize(url);
    if (url == null) {
      context.getCounter("injector", "urls_filtered").increment(1);
```

The code on **Line 14** demonstrates the ***urls_filtered*** counter for ***injector*** counter group being incremented by 1.

The end result is that we generate useful, insightful metrics for each mapper and reducer task for any given Nutch Job.

See below for details on each Nutch metric available.

# Metrics Table

The table below provides a canonical, comprehensive collection of Nutch metrics.

**Table Ordering Logic**

The table is arranged

1. by **Tool** column; alphabetically
2. by the **Hadoop Metric Group;** alphabetically for the given tool
3. by **Hadoop Metric Name**; alphabetically for the given metric group

| Tool /Object | Hadoop Metric Group | Hadoop Metric Name | Sink Metrics Key | Sink Metric Type | Description | Usage and Comments |
|---|---|---|---|---|---|---|
| CleaningJob | CleaningJob Status | Deleted documents | | | The total count of *DB_GONE* (HTTP 404) and/or *DB_DUPLICATE* documents ultimately deleted from the indexer(s). | This metric is useful for determining whether filtering or duplicate detection needs to happen further upstream prior to indexing. Ideally *DB_GONE* and *DB_DUPLICATE* documents would not make it into production indices in the first place. |
| CrawlDbFilter | CrawlDB filter | Gone records removed | | | The total count of *DB_GONE* (HTTP 404) records deleted from the CrawlDB during an update. | See NUTCH-1101 - Getting issue details... STATUS for more details. |
| | CrawlDB filter | Orphan records removed | | | The total count of *orphaned* pages e. g. a page which have no more other pages linking to it, deleted from the CrawlDB during an update. | See NUTCH-1932 - Getting issue details... STATUS for more details. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | CrawlDB filter | URLs filtered | | | The total count of filtered pages e.g. pages which didn't pass one or more URLFIlter implementation(s), deleted from the CrawlDB during an update. | This metric is generally useful for determining the overall effectiveness of URLFilter plugins over time.<br><br>**POSSIBLE IMPROVEMENT**: This metric could be improved if an association could be made between the URL was filtered and the URLFilter which filtered it. This would facilitate aggregating URLFiltering results by URLFilter. |
| CrawlDbReducer | CrawlDB status | *CrawlDatum.getStatusName(CrawlDatum().getStatus())* | | | With each URL able to have only one state at any given point in time, this metric facilitates aggregated counts of the different types of CrawlDatum states for a given CrawlDB. | The state of any given URL will change as the URL transitions through a crawl cycle. Available URL states are defined in the CrawlDatum e.g., *STATUS_DB_UNFETCHED*, *STATUS_DB_FETCHED*, *STATUS_FETCH_SUCCESS*, etc. Practically, CrawlDatum status' are defined using byte signatures but accessed programmatically using static final constants.<br><br>This metric can be used to identify the presence of undesired URL CrawlDatum status' for given URL's e.g., *STATUS_DB_GONE*. Such an event could then trigger a cleaning/pruning operation. |
| DeduplicationJob | Deduplication JobStatus | Documents marked as duplicate | | | The total number of duplicate documents in the CrawlDB. | The process of identifying (near) duplicate documents is of vital importance within the context of a search engine. The precision of any given information retrieval system can be negatively impacted if (near) duplicates are not identified and handled correctly. This does not always mean removing them, for example maybe (near) duplicates are important for versioning purposes. In most cases however it is preferred to identify and remove (near) duplicate records.<br><br>The Deduplication algorithm in Nutch groups fetched URLs with the same digest and marks all of them as duplicate except the one with the highest score (based on the score in the crawldb, which is not necessarily the same as the score indexed). If two (or more) documents have the same score, then the document with the latest timestamp is kept. If the documents have the same timestamp then the one with the shortest URL is kept.<br><br>A duplicate record will have a CrawlDatum status of *CrawlDatum.STATUS_DB_DUPLICATE*. |
| DomainStatistics | N/A | MyCounter.EMPTY_RESULT | | | The total count of empty (probably problematic) URL records for a given host, domain, suffix or top-level domain. | It is possible that the DomainStatistics tool may identify an empty record for a given URL. This may happen regardless of whether the tool is invoked to retrieve host, domain, suffix or top-level domain statistics. When this discovery event occurs, it it is likely that some investigation would take place to understand why. For example, the CrawlDbReader could be invoked with the ***-url*** command line argument to further debug/detail what CrawlDatum data exists. |
| | N/A | MyCounter.FETCHED | | | The total count of fetched URL records for a given host, domain, suffix or top-level domain. | This metric is particularly useful for quickly drilling down through large datasets to determine, for example, how much 'coverage' *has been achieved* for a given host, domain, suffix or top-level domain. This figure can be compared to a website administrators total. |
| | N/A | MyCounter.NOT_FETCHED | | | The total count of unfetched URL records for a given host, domain, suffix or top-level domain. | This metric is particularly useful for quickly drilling down through large datasets to determine, for example, how much 'coverage' *still has to be achieved* for a given host, domain, suffix or top-level domain. When combined with the *fetched* figure and compared to a website administrators total it can provide useful insight. |
| Fetcher | FetcherStatus | bytes_downloaded | | | The total bytes of fetched data acquired across the Fetcher Mapper task(s). | Over time, this can be used to benchmark how much data movement is occurring over the Nutch crawl network.<br><br>**POSSIBLE IMPROVEMENT:** This metric could be improved if a correlation could be made between the volume of data and the source it came from whether that be a given host, domain, suffix or top-level domain. |

<table>
<tr>
<td>FetcherStatus</td>
<td>hitByThrougputThreshold</td>
<td></td>
<td></td>
<td>A total count of the URLs dropped across all fetch queues due to throughput dropping below the threshold too many times.</td>
<td>This aspect of the Nutch Fetcher configuration is designed to prevent slow fetch queues from stalling the overall fetcher throughput. However it usually has the impact of increasing the latency/timeliness of URLs actually being fetched if they are essentially dropped because of low throughput. This means that they are shelved until a future fetch operation.<br><br>The specific configuration settings is<br><br>fetcher.throughput.threshold.pages<br><br><pre>
<property>
  <name>fetcher.throughput.threshold.pages</name>
  <value>-1</value>
  <description>The threshold of minimum pages per
second. If the fetcher downloads less
  pages per second than the configured threshold,
the fetcher stops, preventing slow queue's
  from stalling the throughput. This threshold
must be an integer. This can be useful when
  fetcher.timelimit.mins is hard to determine. The
default value of -1 disables this check.
  </description>
</property>
</pre><br>A more thorough understanding of Fetcher configuration relating to (slow) throughput requires an understanding of the following configuration settings as well<br><br>Additional fetcher throughput configuration<br><br><pre>
<property>
  <name>fetcher.throughput.threshold.retries</name>
  <value>5</value>
  <description>The number of times the fetcher.
throughput.threshold.pages is allowed to be
exceeded.
  This settings prevents accidental slow downs
from immediately killing the fetcher thread.
  </description>
</property>

<property>
<name>fetcher.throughput.threshold.check.after<
/name>
  <value>5</value>
  <description>The number of minutes after which
the throughput check is enabled.</description>
</property>
</pre><br>POSSIBLE IMPROVEMENT: It would be advantageous to understand which URLs from which hosts in the queue(s) were resulting in slow throughput. This would facilitate investigation into why this was happening.</td>
</tr>
</table>

| | | | | | | |
|---|---|---|---|---|---|---|
| | FetcherStatus | hitByTimeLimit | | | A total count of the URLs dropped across all fetch queues due to the fetcher execution time limit being exceeded. | This metric is valuable for quantifying the number of URLs which have been effectively *timebombed* e.g. shelved for future fetching due to overall fetcher runtime exceeding a predefined timeout.<br><br>Although by default the Fetcher ***never times out*** e.g. the configuration is set to **-1**, if a timeout is preferred then the following configuration property can be edited.<br><br>**fetcher.timelimit.mins**<br><br>`<property>`<br>`  <name>fetcher.timelimit.mins</name>`<br>`  <value>-1</value>`<br>`  <description>This is the number of minutes allocated to the fetching.`<br>`  Once this value is reached, any remaining entry from the input URL list is skipped`<br>`  and all active queues are emptied. The default value of -1 deactivates the time limit.`<br>`  </description>`<br>`</property>`<br><br>**POSSIBLE IMPROVEMENT:** It could be useful to record the fact that a URL was staged due to it being hit by the timeout limit. This could possibly be stored in the CrawlDatum metadata.<br><br>Also see NUTCH-2910 - Getting issue details... STATUS |
| FetcherThread | FetcherStatus | AboveExceptionThresholdInQueue | | | The total count of URLs purged across all fetch queues as a result of the maximum number of protocol-level exceptions (e.g. timeouts) per queue being exceeded. | ***!! Also see the same metric below generated by the QueueFeeder. !!***<br><br>This metric is useful for quantifying the number of URLs shelved for future fetching due to anomalies occurring during fetcher execution exceeding a predefined ceiling.<br><br>Although by default the Fetcher ***never enforces this behaviour*** e.g. the configuration is set to **-1**, if this is changed then this total count will become a useful metric to track. Further information on the configuration parameter can be seen below<br><br>**fetcher.max.exceptions.per.queue**<br><br>`<property>`<br>`  <name>fetcher.max.exceptions.per.queue</name>`<br>`  <value>-1</value>`<br>`  <description>The maximum number of protocol-level exceptions (e.g. timeouts) per`<br>`  host (or IP) queue. Once this value is reached, any remaining entries from this`<br>`  queue are purged, effectively stopping the fetching from this host/IP. The default`<br>`  value of -1 deactivates this limit.`<br>`  </description>`<br>`</property>`<br><br>**POSSIBLE IMPROVEMENT:** It could be useful to record the fact that a URL was staged due to it being hit by the exception limit. Additionally, it could be useful to write metadata to all URLs which contributed towards the limit being met. This could possibly be stored in the CrawlDatum metadata. |
| | FetcherStatus | FetchItem.notCreated.redirect | | | A total count of URLs across all fetch queues for which following redirect(s) resulted in no result. | Essentially, each FetcherThread attempts to understand and eliminate all redirect options e.g. duplicate redirect URL, before giving up on a redirect URL entirely. In the case of a redirect URL for which no logical fetch outcome can be produced e.g. that the FetchItem is *null*, redirecting is simply deactivated as it is impossible to continue. |
| | FetcherStatus | outlinks_detected | | | A total count of detected outlinks for all fetch items (URLs) across all fetch queues. | This metric can be used to estimate the number in URLs to be fetched in the next fetch phase. If for example, resources were being allocated within the client at configuration/compile time (rather than dynamically at runtime) this could be used to inform resource reservations, utilization and data partitioning logic. |
| | FetcherStatus | outlinks_following | | | From ***outlinks_detected*** (see directly above), this is a total count of URLs which will be followed. | This metric value *may* be the same as ***outlinks_detected*** or it may be less. This ultimately depends on a few things i.e.,<br><br>• whether the fetcher is configured to follow external outlinks<br>• whether a given URL is already followed<br>• whether a given URL is already fetched |
| | FetcherStatus | *ProtocolStatus.getName()* | | | Total counts of all the different fetched finished status' for all URLs. | For a comprehensive collection of the various fetched finish status' to expect, check out the ***private static final HashMap<Integer, String> codeToName*** defined within ProtocolStatus. This metric is useful for understanding, from across your CrawlDb, the status of certain URLs. Via different tools, you can begin to investigate further. |

| | | | | | |
|---|---|---|---|---|---|
| FetcherStatus | redirect_count_exceeded | | | Total count of all URLs which have exceeded the maximum configured number of redirects. | See the following configuration property in nutch-default.xml<br>**http.redirect.max**<br>`<property>`<br>`  <name>http.redirect.max</name>`<br>`  <value>0</value>`<br>`  <description>The maximum number of redirects the fetcher will follow when`<br>`  trying to fetch a page. If set to negative or 0, fetcher won't immediately`<br>`  follow redirected URLs, instead it will record them for later fetching.`<br>`  </description>`<br>`</property>`<br>This metric is useful for understanding how many URLs are possibly skipped due to a large number of redirects.<br>Also see the following property<br>**http.redirect.max.exceeded.skip**<br>`<property>`<br>`  <name>http.redirect.max.exceeded.skip</name>`<br>`  <value>false</value>`<br>`  <description>`<br>`  Whether to skip the last URL in a redirect chain when when redirects`<br>`  are followed (http.redirect.max > 0) and the maximum number of redirects`<br>`  in a chain is exceeded (redirect_count > http.redirect.max).`<br>`  If not skipped the redirect target URLs are stored as `linked`<br>`  and fetched in one of the following cycles. See also NUTCH-2748.`<br>`  </description>`<br>`</property>` |
| FetcherStatus | redirect_deduplicated | | | Total count of duplicate (and hence not fetched) redirected URLs. | No fetching takes place for this class of redirect URLs as they are duplicates of other redirect URLs already fetched. |
| FetcherStatus | robots_denied | | | Total count of all URLs not fetched due to being denied by robots.txt rules. | By default Nutch is configured to respect and comply with robots.txt rules for any given site. It is useful to know how many URLs may not be fetched from a given site due to robots.txt compliance. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | FetcherStatus | robots_denied_maxcrawldelay | | | Total count of URLs which are skipped due to the robots.txt crawl delay being above a configured maximum. | The following configuration property must be consulted for a detailed explanation behind this metric.<br>**fetcher.max.crawl.delay**<br>`<property>`<br>`<name>fetcher.max.crawl.delay</name>`<br>`<value>30</value>`<br>`<description>`<br>`If the Crawl-Delay in robots.txt is set to greater than this value (in seconds) then the fetcher will skip this page, generating an error report.`<br>`If set to -1 the fetcher will never skip such pages and will wait the amount of time retrieved from robots.txt Crawl-Delay, however long that might be.`<br>`</description>`<br>`</property>`<br>Essentially, a delay of 5 seconds is used for fetching requests **to the same host** unless a crawl delay is specified within the robots.txt. Also see<br>**fetcher.server.delay**<br>`<property>`<br>`<name>fetcher.server.delay</name>`<br>`<value>5.0</value>`<br>`<description>The number of seconds the fetcher will delay between successive requests to the same server. Note that this might get overridden by a Crawl-Delay from a robots.txt and is used ONLY if fetcher.threads.per.queue is set to 1.`<br>`</description>`<br>`</property>` |
| | FetcherStatus | robots_defer_visits_dropped | | | Total count of URLs skipped after all trials to fetch the robots.txt failed with HTTP 5xx. | See NUTCH-2573 and property `http.robots.503.defer.visits` |
| | ParserStatus | *ParseStatus.majorCodes[p.getData().getStatus().getMajorCode()]* | | | Total count of major codes (see right) from parsing URLs. | ParseStatus defines three major categories for the result of an URL parse operation. i.e., ***notparsed***, ***success*** and ***failed***. This metric is useful for debugging how many parse operations failed for a given crawl cycle. Subsequent parse attempts can then be made or the URL record can be handled appropriately. |
| Generator | Generator | EXPR_REJECTED | | | Total count of URLs rejected by Jexl expression(s) during the generate phase. | This metric is useful for determining the impact that that Jexl expressions (provided via the Generator CLI) have on filtering URLs. The expressions are evaluated during the Generator Map phase.<br>All of the configuration which drives this metric is read or set from Java code and not explicitly defined in nutch-default.xml. |

| | | | | | |
|---|---|---|---|---|---|
| Generator | HOSTS_AFFECTED_PER_HOST_OVERFLOW | | | Total count of host(s) or domain(s) affected by the number of URLs exceeding a configured fetchlist size threshold. | This configuration property is essentially turned off by default e.g. there is no defined maximum number of URLs per fetchlist. However, if a maximum is defined, then it is useful to know, how many hosts or domains included in fetchlists have more URLs than allowed. In these cases additional URLs won't be included in the fetchlist but bumped on to future crawling cycles.<br><br>The configuration property below will directly drive this metric.<br><br>**generate.max.count**<br><br>`<property>`<br>`  <name>generate.max.count</name>`<br>`  <value>-1</value>`<br>`  <description>The maximum number of URLs in a single`<br>`  fetchlist. -1 if unlimited. The URLs are counted according`<br>`  to the value of the parameter generate.count.mode.`<br>`  </description>`<br>`</property>` |
| Generator | INTERVAL_REJECTED | | | Total count of records rejected due to retry or fetch interval being above a configured thershold. | This configuration property is essentially turned off by default e.g. there is no minimum defined retry interval. This metric is useful for understanding the impact that changing that has on URL filtering.<br><br>The configuration property below drives this metric.<br><br>**generate.min.interval**<br><br>`<property>`<br>`  <name>generate.min.interval</name>`<br>`  <value>-1</value>`<br>`  <description>Select only entries with a retry interval lower than`<br>`  generate.min.interval. A value of -1 disables this check.</description>`<br>`</property>` |
| Generator | MALFORMED_URL | | | Total count of malformed URLs filtered. | In the Generator, malformed URLs are either discovered by<br><br>1. an URL normalizer implementation. This is turned on by default but can be toggled on or off within the Generator CLI or programmatically<br>2. Attempting to extract either the URL host or domain (depending on which one is configured). See the following property<br><br>**generate.count.mode**<br><br>`<property>`<br>`  <name>generate.count.mode</name>`<br>`  <value>host</value>`<br>`  <description>Determines how the URLs are counted for generate.max.count.`<br>`  Default value is 'host' but can be 'domain'. Note that we do not count`<br>`  per IP in the new version of the Generator.`<br>`  </description>`<br>`</property>` |
| Generator | SCHEDULE_REJECTED | | | Total count of URLs not suitable for selection (in a given crawl cycle due to the fetch time being higher than the current time. | The metric description covers the default fetch schedule case but this can change depending on the actual implementation. Of specific interest is FetchSchedule#shouldFetch(...) which explains further. This metric can be useful for comparing implementations of FetchSchedule. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Generator | SCORE_TOO_LOW | | | Total count of filtered URL entries with a score lower than a configured threshold. | The configuration parameter which drives this metric is<br>**generate.min.score**<br>`<property>`<br>`  <name>generate.min.score</name>`<br>`  <value>0</value>`<br>`  <description>Select only entries with a score larger than`<br>`  generate.min.score.</description>`<br>`</property>`<br>The default value for this configuration property means that all entries should be selected.<br>The metric can be useful to determine if a configured minimum value is too high and filters too many URLs from being included in fetchlists. |
| | Generator | STATUS_REJECTED | | | Total count of URLs filtered by a CrawlDatum status filter. | The following configuration property is used to straight filter URLs depending on their CrawlDatum status<br>**generate.restrict.status**<br>`<property>`<br>`  <name>generate.restrict.status</name>`<br>`  <value></value>`<br>`  <description>Select only entries of this status, see`<br>`  https://issues.apache.org/jira/browse/NUTCH-1248</description>`<br>`</property>`<br>As an indication of the status keys which can be used, see CrawlDatum.statNames.<br>This metric is useful to simply see how effective the status filters are. |
| | Generator | URLS_SKIPPED_PER_HOST_OVERFLOW | | | Total count of URLs skipped by the number of URLs exceeding a configured fetchlist size threshold. | This configuration property is essentially turned off by default e.g. there is no defined maximum number of URLs per fetchlist. However, if a maximum is defined, then it is useful to know, how many URLs are skipped. In these cases additional URLs won't be included in the fetchlist but bumped on to future crawling cycles.<br>The configuration property below will directly drive this metric.<br>**generate.max.count**<br>`<property>`<br>`  <name>generate.max.count</name>`<br>`  <value>-1</value>`<br>`  <description>The maximum number of URLs in a single`<br>`  fetchlist. -1 if unlimited. The URLs are counted according`<br>`  to the value of the parameter generate.count.mode.`<br>`  </description>`<br>`</property>` |
| IndexerMapReduce | IndexerStatus | deleted (duplicates) | | | Total count of duplicate records deleted from the indexing backend. | This configuration property is deactivated by default however it can be activated by providing the ***-deleteGone*** flag to the IndexerJob CLI.<br>The duplication algorithm is documented well in the DeduplicationJob Javadoc.<br>**POSSIBLE IMPROVEMENT:** See the commentary on whether a distinction should be made between HTTP 404/gone, temp/perm redirects and duplicates. This may ultimately result in these functions being broken out into individual flags and/or configuration properties in nutch-default.xml at which stage this documentation will be updated. |

| | | | | | |
|---|---|---|---|---|---|
| IndexerStatus | deleted (IndexingFilter) | | | Total count of deleted records as a result of being skipped/filtered by indexing filters. | This configuration property is deactivated by default but it can be activated either by providing the ***-params indexer.delete.skipped.by.indexingfilter=true*** flag to the IndexerJob CLI or editing it in nutch-default.xml as shown below<br><br>**indexer.delete.skipped.by.indexingfilter**<br><br>`<property>`<br>`  <name>indexer.delete.skipped.by.indexingfilter</name>`<br>`  <value>false</value>`<br>`  <description>Whether the indexer will delete documents that were skipped by indexing filters`<br>`  </description>`<br>`</property>`<br><br>Additional information on the original implementation can be found at https://issues.apache.org/jira/browse/NUTCH-1449 |
| IndexerStatus | deleted (gone) | | | Total count of HTTP 404/gone records deleted from the indexing backend. | If a Nutch deployment uses the crawl script then this configuration property is activated by default otherwise it is deactivated. However it can be activated by providing the ***-deleteGone*** flag to the IndexerJob CLI (as is done in the crawl script).<br><br>**POSSIBLE IMPROVEMENT:** See the commentary on whether a distinction should be made between HTTP 404/gone, temp/perm redirects and duplicates. This may ultimately result in these functions being broken out into individual flags and/or configuration properties in nutch-default.xml at which stage this documentation will be updated. |
| IndexerStatus | deleted (redirects) | | | Total count of temporary and permanently redirected records deleted from the indexing backend. | This configuration property is deactivated by default however it can be activated by providing the ***-deleteGone*** flag to the IndexerJob CLI.<br><br>**POSSIBLE IMPROVEMENT:** See the commentary on whether a distinction should be made between HTTP 404/gone, temp/perm redirects and duplicates. This may ultimately result in these functions being broken out into individual flags and/or configuration properties in nutch-default.xml at which stage this documentation will be updated. |
| IndexerStatus | deleted (robots=noindex) | | | Total count of deleted records which contain ***meta robots="noindex"*** | This configuration property is deactivated by default but it can be activated either by providing the ***-params indexer.delete.robots.noindex=true*** flag to the IndexerJob CLI or editing it in nutch-default.xml as shown below<br><br>**indexer.delete.robots.noindex**<br><br>`<property>`<br>`  <name>indexer.delete.robots.noindex</name>`<br>`  <value>false</value>`<br>`  <description>Whether the indexer will delete documents marked by robots=noindex`<br>`  </description>`<br>`</property>`<br><br>See Google's Block Search indexing with noindex documentation for more information.<br><br>Additional information on the original implementation can be found at https://issues.apache.org/jira/browse/NUTCH-1434 |
| IndexerStatus | errors (IndexingFilter) | | | Total count of errors which occurred during execution of the indexing filtering and normalization chain. | Both filtering and normalization are deactivated by default in IndexingJob but can be activated by providing the ***-filter*** and ***-normalize*** flags respectively to the IndexerJob CLI.<br><br>This metric is useful for determining, for example, the impacts that changes to an indexing filtering /normalization chain are having on URLs ultimately being indexed. A higher count means that something is wrong.<br><br>**POSSIBLE IMPROVEMENT:** It could be useful to record the fact that URL filtering and/or normalization processes resulted in an error for a given record. This could possibly be stored in the relevant datum/metadata container.<br><br>**POSSIBLE IMPROVEMENT:** Additionally, although URL filtering and normalization happens in both Map and Reduce phases of a job, this metric is only implemented in the Reduce phase. We need to determine if we want to extend this metric to also cover the Map phase of IndexingJob.<br><br>**POSSIBLE IMPROVEMENT:** Finally, we need to determine whether this is a general metric we wish to capture for any job where filtering and normalization takes place. |

| | | | | | |
|---|---|---|---|---|---|
| IndexerStatus | errors (ScoringFilter) | | | Total count of errors which occurred during execution of the scoring filtering chain. | This metric is useful for determining, for example, the impacts that changes to an indexing scoring chain are having on URLs ultimately being indexed. A higher count means that something is wrong.<br><br>Scoring is always executed during the indexing reduce phase. Scoring implementations are initially configured in the ***plugin.includes*** configuration property of nutch-default.xml however their order of execution is based on the following configuration property<br><br>**scoring.filter.order**<br><br>`<property>`<br>`  <name>scoring.filter.order</name>`<br>`  <value></value>`<br>`  <description>The order in which scoring filters are applied.  This`<br>`  may be left empty (in which case all available scoring filters will`<br>`  be applied in system defined order), or a space separated list of`<br>`  implementation classes.`<br>`  </description>`<br>`</property>`<br><br>Some scoing plugins have specific settings in nutch-default.xml which should be consulted prior and during execution and use in production deployments.<br><br>**POSSIBLE IMPROVEMENT:** Finally, we need to determine whether this is a general metric we wish to capture for any job where scoring takes place. |
| IndexerStatus | indexed (add /update) | | | Total count of records successfully indexed or updated by the indexing backend. | This metric is literally the last thing which is counted within the IndexingJob reduce phase. When combined with other metrics it can be used to provide an overall understanding of indexing results. |
| IndexerStatus | skipped (IndexingFilter) | | | Total count of records skipped /filtered by indexing filters. | Note the similarity to *deleted (IndexingFilter)* above however in this case the documents are not deleted from the indexing backend. |
| IndexerStatus | skipped (not modified) | | | Total count of records skipped with a db_notmodified status i.e., that the page was successfully fetched and found not modified. | This configuration property is deactivated by default but it can be activated either by providing the ***-params indexer.skip.notmodified=true*** flag to the IndexerJob CLI or editing it in nutch-default.xml as shown below<br><br>**indexer.skip.notmodified**<br><br>`<property>`<br>`  <name>indexer.skip.notmodified</name>`<br>`  <value>false</value>`<br>`  <description>Whether the indexer will skip records with a db_notmodified status.`<br>`  </description>`<br>`</property>` |

<table>
<tr><td rowspan="4">Injector</td><td>injector</td><td>urls_filtered</td><td></td><td></td><td>Total count of new seed URLs filtered by the Injector.</td><td>URL normalization and then filtering operations are executed within the Injector Map task(s). They are both turned off by default in nutch-default.xml however if these values are not interpreted the Injector turns normalization and filtering operations on by default.<br><br>This metric is useful to determine the impact that normalization and filtering have on the injection of seeds contained within seed lists. For more information on configuration see<br><br>Normalization and Filtering<br><br><pre><property><br>    <name>crawldb.url.normalizers</name><br>    <value>false</value><br>    <description><br>        !Temporary, can be overwritten with the<br>command line!<br>        Normalize URLs when updating crawldb<br>    </description><br></property><br><br><property><br>    <name>crawldb.url.filters</name><br>    <value>false</value><br>    <description><br>        !Temporary, can be overwritten with the<br>command line!<br>        Filter URLS when updating crawldb<br>    </description><br></property></pre></td></tr>
<tr><td>injector</td><td>urls_injected</td><td></td><td></td><td>Total count of seed URLs injected by the Injector.</td><td>The total number of URLs in the seed file or directory passed as seed input to the Injector.</td></tr>
<tr><td>injector</td><td>urls_injected_unique</td><td></td><td></td><td>Total count of unique seeds URLs injected by the Injector.</td><td>The total number of unique URLs in the seed file or directory passed as seed input to the Injector. If the seed files include duplicates, the value of this counter differs from that of urls_injected.<br><br>The difference between urls_injected_unique and urls_merged is the number of how many URLs the CrawlDb has grown by the end of an Injector invocation.</td></tr>
<tr><td>injector</td><td>urls_merged</td><td></td><td></td><td>Total count of unique seed URLs merged with an existing CrawlDatum record.</td><td>This metric is useful for seeing how many existing URL records were affected by a given seed list within the Injector reduce task(s). Several configuration settings are used to determine what those affects are...<br><br>URLs merging in Injector<br><br><pre><property><br>  <name>db.injector.overwrite</name><br>  <value>false</value><br>  <description>Whether existing records in the<br>CrawlDB will be overwritten<br>  by injected records.<br>  </description><br></property><br><br><property><br>  <name>db.injector.update</name><br>  <value>false</value><br>  <description>If true existing records in the<br>CrawlDB will be updated with<br>  injected records. Old meta data is preserved.<br>The db.injector.overwrite<br>  parameter has precedence.<br>  </description><br></property></pre><br>You should also consult the Injector.InjectorReducer#reduce documentation as it adequately describes the Injector merging algorithm.</td></tr>
</table>

| | | | | | | |
|---|---|---|---|---|---|---|
| | injector | urls_purged_404 | | | Total count of deleted/purged URLs due to an existing CrawlDatum.STATUS_DB_GONE | If turned on, 404 purging occurs during the Injector Map task(s). It relies on the following configuration property<br>**db.update.purge.404**<br>`<property>`<br>`  <name>db.update.purge.404</name>`<br>`  <value>false</value>`<br>`  <description>If true, updatedb will add purge records with status DB_GONE`<br>`  from the CrawlDB.`<br>`  </description>`<br>`</property>`<br>The metric is useful for understanding how many seed URLs are already known to be essentially dead links.<br>**POSSIBLE IMPROVEMENT:** The property description above needs to be updated to accommodate usage of this property in Injector. |
| | injector | urls_purged_filter | | | Total count of existing CrawlDatum records filtered by one or more filters and/or normalizers during the Injector execution. | Not to be confused with similar metric ***urls_filtered*** *above.* This configuration setting can only be activated in the Injector Map phase by passing the ***-filterNormalizeAll*** flag on the command-line. |
| ParseSegment | ParserStatus | *ParseStatus.majorCodes [parseStatus.getMajorCode()]* | | | Total individual record counts of major parse status i.e., ***notparsed***, ***success*** *and* ***failed***. | This metric is really useful for determining total counts of generic parse status' particularly as it provides insight into parsing failures. For a given media type within a given dataset or crawl cycle we can better understand what kind of input data results in parse failures and then begin to work to debug why and provide fixes.<br>This metric is generated during the Map phase of the ParseSegment tool.<br>**POSSIBLE IMPROVEMENT:** Create a composite key which includes both the major parse status and minor parse status. An example could be ***failed:FAILED_TRUNCATED*** or similar. Minor parse status' are documented in ParseStatus.java#L50-L84. |
| QueueFeeder | FetcherStatus | filtered | | | Total count of URLs which were filtered or failed to normalize within Fetcher queue feeding thread(s). | The QueueFeeder feeds queues with input items, and re-fills them as items are consumed by FetcherThread-s. This metric is useful for quantifying how many records were filtered or normalized during that process particularly if changes are made to the URL filter(s) or normalizer(s) chain.<br>By default filtering and normalization are deactivated during Fetcher execution however they can be activated by editing the following configuration properties.<br>**Normalizer and Filtering URLs in Fetcher**<br>`<property>`<br>`  <name>fetcher.filter.urls</name>`<br>`  <value>false</value>`<br>`  <description>Whether fetcher will filter URLs (with the configured URL filters).</description>`<br>`</property>`<br>`<property>`<br>`  <name>fetcher.normalize.urls</name>`<br>`  <value>false</value>`<br>`  <description>Whether fetcher will normalize URLs (with the configured URL normalizers).<`<br>`/description>`<br>`</property>` |

<table>
<tr><td>(also QueueFeeder)</td><td>FetcherStatus</td><td>AboveExceptionThresholdInQueue</td><td></td><td></td><td>The total count of URLs purged across all fetch queues as a result of the maximum number of protocol-level exceptions (e.g. timeouts) per queue being exceeded.</td><td>!! Also see the same metric below generated by the FetcherThread. !!<br><br>This metric is useful for quantifying the number of URLs shelved for future fetching due to anomalies occurring during fetcher execution exceeding a predefined ceiling.<br><br>Although by default the Fetcher never enforces this behaviour e.g. the configuration is set to -1, if this is changed then this total count will become a useful metric to track. Further information on the configuration parameter can be seen below<br><br>fetcher.max.exceptions.per.queue<br><br><property><br>  <name>fetcher.max.exceptions.per.queue</name><br>  <value>-1</value><br>  <description>The maximum number of protocol-level exceptions (e.g. timeouts) per<br>  host (or IP) queue. Once this value is reached, any remaining entries from this<br>  queue are purged, effectively stopping the fetching from this host/IP. The default<br>  value of -1 deactivates this limit.<br>  </description><br></property><br><br>POSSIBLE IMPROVEMENT: It could be useful to record the fact that a URL was staged due to it being hit by the exception limit. Additionally, it could be useful to write metadata to all URLs which contributed towards the limit being met. This could possibly be stored in the CrawlDatum metadata.</td></tr>
<tr><td>ResolverThread</td><td>UpdateHostDb</td><td>checked_hosts</td><td></td><td></td><td>Total count of hosts which have essentially been checked and exist /resolve.</td><td>This metric is useful for determining hosts which are able to be resolved through a call to InetAddress.getByName(host). Hosts which do not resolve will through an Exception and are then handled appropriately and covered in one of the accompanying HostDb metrics below.<br><br>This metric value should be a combined total of all other ResolverThread metric counts.<br><br>Also see UpdateHostDbReducer#shouldCheck(HostDatum datum) for logic regarding whether a host should be checked or not.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>existing_known_host</td><td></td><td></td><td>Total count of existing known hosts which exist/resolve.</td><td>Simply a total count of host which were previously contained within the HostDb.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>existing_unknown_host</td><td></td><td></td><td>The total count of existing hosts which should been forgotten.</td><td>This metric can be used to inform downstream purging operations. A host is marked with this counter if either the configuration property value below is less than the number of DNS attempt failures or of the configuration property value is deactivated (set to a value of -1).<br><br>hostdb.purge.failed.hosts.threshold<br><br><property><br>  <name>hostdb.purge.failed.hosts.threshold</name><br>  <value>3</value><br>  <description><br>  If hosts have more failed DNS lookups than this threshold, they are<br>  removed from the HostDB. Hosts can, of course, return if they are still<br>  present in the CrawlDB.<br>  </description><br></property><br><br>As stated in the configuration above it may also be prudent to consult the CrawlDb if the desired outcome is for this Host to NOT enter back into the HostDb.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>new_known_host</td><td></td><td></td><td>Total count of brand new hosts for which no HostDatum exists.</td><td>This metric is useful for understanding how the HostDb is growing over time.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>new_unknown_host</td><td></td><td></td><td>Total count of new unknown hosts which do not exist/resolve.</td><td>If somehow the host does not exist/resolve and we've never encountered it before e.g. it is a new host and the HostDatum is empty, then it is initialized with date = ${today's date and time} and the DNS failure field of the records HostDatum is incremented by 1.<br><br>This is useful for tracking anbd informing downstream conditional logic based on new hosts which do not resolve upon discovery.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>purged_unknown_host</td><td></td><td></td><td>The total count of existing hosts which should been forgotten and ultimately purged.</td><td>This metric is is useful for tracking previously undiscovered hosts which ultimately don't resolve/exist.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>rediscovered_host</td><td></td><td></td><td>Total count of existing hosts which have experienced a DNS failure(s).</td><td>The metric is useful for quantifying how many hosts are having DNS issues. This might be useful if you wish to normalize and/or filter these hosts. Alternatively you may also for example wish to contact the host webmasters indicating that their DNS issues have been detected.</td></tr>
<tr><td></td><td>UpdateHostDb</td><td>Long.toString(datum.numFailures()) + "_times_failed"</td><td></td><td></td><td>Count groupings of DNS and connection failures for each Host in the HostDb.</td><td>This metric is usful for determining whether there are some hosts which have been failing a significant number of times for action to be taken. For example, if a Host DNS lookup has failed 6 times would you wish to keep trying or simply purge it and ensure it is filtered in the future as early as possible in the crawl cycle.</td></tr>
</table>

<table>
<tr><td rowspan="3">Sitemap Processor</td><td>Sitemap</td><td>existing_sitemap_entries</td><td></td><td></td><td>Total count of sitemap entries already present in the CrawlDb that have been overwritten with new sitemap data.</td><td>This metric is used to quantify, for records already present in crawldb, new CrawlDatum data. This happens if the same URL is obtained from a new sitemap, and we have activated overwriting of sitemaps. The information from the new sitemap overwrites the original datum essentially emitting the original crawl datum.<br><br>The overwriting feature is deactivated by default but can be activatd in nutch-site.xml as below.<br><br>sitemap.url.overwrite.existing<br><br><pre><property><br>  <name>sitemap.url.overwrite.existing</name><br>  <value>false</value><br>  <description><br>    If true, the record's existing modified time, interval and score are<br>    overwritten by the information in the sitemap. WARNING: overwriting<br>    these values may have unexpected effects on what is crawled. Use this<br>    only if you can trust the sitemap and if the values in the sitemap do<br>    fit with your crawler configuration.<br>   </description><br></property></pre></td></tr>
<tr><td>Sitemap</td><td>failed_fetches</td><td></td><td></td><td>Total count of sitemaps for which fetching failed.</td><td>If there were any problems fetching the sitemap, we log the error and let it go. This metric may fluxuate depending on the following configuration property value in nutch-default.xml<br><br>sitemap.redir.max<br><br><pre><property><br>  <name>sitemap.redir.max</name><br>  <value>3</value><br>  <description><br>    Maximum number of redirects to follow.<br>   </description><br></property></pre><br>POSSIBLE IMPROVEMENT: As of writing we are not sure how often sitemaps are redirected. For example do more complex redirect scenarios exis than simple http >> https? In the future we might have to handle redirects differently.</td></tr>
<tr><td>Sitemap</td><td>filtered_records</td><td></td><td></td><td>Total count of filtered and/or normalized sitemaps.</td><td>Filter and normalizing are both activated within the SitemapProcessor by default however either can be deactivated by using the -noFilteri and -noNormalize flags on the SitemapProcessor CLI or editing the configuration property values in nutch-site.xml as below<br><br>Sitemap filtering and normalizing<br><br><pre><property><br>  <name>sitemap.url.filter</name><br>  <value>true</value><br>  <description><br>    Filter URLs from sitemaps.<br>  </description><br></property><br><br><property><br>  <name>sitemap.url.normalize</name><br>  <value>true</value><br>  <description><br>    Normalize URLs from sitemaps.<br>   </description><br></property></pre></td></tr>
</table>

| | | | | | | |
|---|---|---|---|---|---|---|
| | Sitemap | filtered_sitemaps_from_hostname | | | Total count of filtered and/or normalized sitemaps generated from hosts in the HostDb. | On some occasions robots.txt will not define a sitemap.xml location but that doesn't mean one doesn't exist. Sometimes a sitemap can be inferred from only the host which is one of the compelling reasons as to why a HostDb is extremely useful. For more details see SitemapProcessor#generateSitemapsFromHostname(String host, Context context).<br><br>Filter and normalizing are both activated within the SitemapProcessor by default however either can be deactivated by using the ***-noFilter***i and ***-noNormalize*** flags on the SitemapProcessor CLI or editing the configuration property values in nutch-site.xml as below<br><br>**Sitemap filtering and normalizing**<br><br>`<property>`<br>`  <name>sitemap.url.filter</name>`<br>`  <value>true</value>`<br>`  <description>`<br>`  Filter URLs from sitemaps.`<br>`  </description>`<br>`</property>`<br><br>`<property>`<br>`  <name>sitemap.url.normalize</name>`<br>`  <value>true</value>`<br>`  <description>`<br>`  Normalize URLs from sitemaps.`<br>`  </description>`<br>`</property>` |
| | Sitemap | new_sitemap_entries | | | Total count of brand new sitemap entries added to the CrawlDb. | For the newly discovered URLs we acquire via sitemap, the CrawlDatum status is set as unfetched and the record is emitted to the CrawlDb. |
| | Sitemap | sitemaps_from_hostname | | | Total count of sitemaps generated from hosts contained in the HostDb. | In direct oppose to ***filtered_sitemaps_from_hostname*** this metric is useful for quantifying how many sitemaps were actually inferred from only the host. This metric could be used, for example, to inform webmasters that they should add a sitemap.xml location to their robots.txt. |
| | Sitemap | sitemap_seeds | | | Total count of sitemaps which were injected as seeds. | This metric is useful for calculating the difference between sitemaps which were injected vs sitemaps inferred from hosts in the HostDb. If sitemaps were injected via passing the ***-sitemapUrls*** flag into the SitemapProcessor CLI then they will be processed from an existing *CrawlDatum* as opposed to being inferred from a recored present in the HostDb. |
| UpdateHostDbMapper | UpdateHostDb | filtered_records | | | Total count of records filtered within an update to the HostDb. | Useful for determining the impact that filtering and normalization plugins and associated rules have on the resulting HostDb. See the following configuration properties<br><br>**hostdb.url.filter and hostdb.url.normalize**<br><br>`<property>`<br>`  <name>hostdb.url.filter</name>`<br>`  <value>false</value>`<br>`  <description>`<br>`  Whether the records are to be passed through configured filters.`<br>`  </description>`<br>`</property>`<br><br>`<property>`<br>`  <name>hostdb.url.normalize</name>`<br>`  <value>false</value>`<br>`  <description>`<br>`  Whether the records are to be passed through configured normalizers.`<br>`  </description>`<br>`</property>`<br><br>This value can also be overridden on the command line with the ***-filter*** and ***-normalize*** flags. |
| UpdateHostDbReducer | UpdateHostDb | total_hosts | | | Total count of all hosts processed in one HostDb update. | Useful when considered alongside other metrics to determine the overall impact that configuration, such as filtering or skipping, has on processing of hosts contained within the HostDb. |

| | | | | | | |
|---|---|---|---|---|---|---|
| (also UpdateHostDbReducer) | UpdateHostDb | skipped_not_eligible | | | Total count of records skipped within an update to the HostDb. | Records are skipped if the ***-checkNew*** flag is used whilst invoking the tool AND it is a new Host (meaning that the HostDatum is empty). Also see the following configuration property<br>**hostdb.check.new**<br>`<property>`<br>`  <name>hostdb.check.new</name>`<br>`  <value>true</value>`<br>`  <description>`<br>`  True if newly discovered hosts eligible for DNS lookup check. If false,`<br>`  hosts that are just added to the HostDB are not eligible for DNS lookup.`<br>`  </description>`<br>`</property>` |
| Web Graph | WebGraph.outlinks | added links | | | Total count of links which will ultimately be used in WebGraph scoring. | Useful metric to indicate how many links are essentially used within the scoring process. |
| (also WebGraph) | WebGraph.outlinks | removed links | | | Total count of GONE pages not used in Webgraph scoring. | Useful metric to indicate presence of gone/404 records which are not used within the scoring process. |
| WARCExporter | WARCExporter | exception | | | Total count of IOException and/or IllegalStateException's caught during tool execution. | These Exception's *can* occur in the WARCExporter Reduce task(s) and if they do it is convenient to count them. They can arise when attempting to write records with a WARCWritable value. |
| | WARCExporter | invalid URI | | | Total count of invalid entries with an invalid ***WARC-Target-URI*** | These Exception's *can* occur in the WARCExporter Reduce task(s) and if they do it is convenient to count them. This metric is essentially indicating that an Exception was raised and caught as a result of an invalid ***WARC-Target-URI****i* for some given record. |
| | WARCExporter | missing content | | | Total count of records with missing (null) content. | These Exception's *can* occur in the WARCExporter Reduce task(s) and if they do it is convenient to count them.<br><br>**POSSIBLE IMPROVEMENT:** Providing traceability to the individual record would enable debugging of why content is null. |
| | WARCExporter | missing metadata | | | Total count of records with missing (null) CrawlDatum metadata. | These Exception's *can* occur in the WARCExporter Reduce task(s) and if they do it is convenient to count them.<br><br>**POSSIBLE IMPROVEMENT:** Providing traceability to the individual record would enable debugging of why CrawlDatum metadata is null. |
| | WARCExporter | omitted empty response | | | Total count of records not written because they have empty responses. | This setting can be turned on via the command line with the ***-onlySuccessfulResponses***i flag. If it is turned on it means that records with the CrawlDatum status' not STATUS_FETCH_SUCCESS or STATUS_FETCH_NOTMODIFIED will not be written as they are irregular responses. |
| | WARCExporter | records generated | | | Total count of WARC records successfully exported written. | A useful metric when combined with other counts as it provides a more comprehensive understanding of data quality i.e., missing and or incomplete data. |

# Conclusion

This document aims to provide a detailed account of Nutch application metrics such that a data-driven approach can be adopted to better manage Nutch operations. Several cells in the **USAGE AND COMMENTS** column in the above table offer areas for **POSSIBLE IMPROVEMENT**. These suggestions are targeted towards Nutch crawler administrators and developers interested in evolving/improving Nutch metrics.