HCP Tuning Guide 1

Component Tuning Variables

Date of Publish: 2019-04-09

https://docs.hortonworks.com

Contents

Component Tuning Variables Overview

There are a number of services that you can use to tune the performance of your HCP cluster. These services include Kafka, Storm, and HDFS. Within these services, you can modify parsers, enrichment, and indexing (Elasticsearch or Solr).

When you consider tuning your HCP architecture, it is important to note where you can modify settings. For example, Storm gives you the ability to independently set tasks in executors for parser topologies. This is important if you want to set the number of tasks higher than the number of executors to accommodate for future performance tuning and rebalancing without the need to bring down your topologies. However, for enrichment and indexing topologies, HCP uses Flux, and there is no method for specifying the number of tasks from the number of executors in Flux. By default, the number of tasks equals the number of executors.

The following lists the major properties for each service that you can modify to tune your cluster:

- Kafka partitions
- Storm
  - Kafka spout
    - Polling frequency
    - Polling timeouts
    - Offset commit period
    - Max uncommitted offsets
  - Storm workers (OS processes)
  - Storm executors (threads in a process)
  - Storm ackers
  - Max spout pending
  - Spout and bolt parallelism
- HDFS
  - Batch size
  - Replication factor
- Indexing
  - Elasticsearch templates
  - Solr

Kafka Partitions

The main lever you can adjust to tune Kafka throughput is the number of partitions.

When you calculate the number of Kafka partions, it is important to remember that it is the unit of parallelism. A topic with more partitions has higher throughput and higher latency than a topic with less. It is important to determine the correct number of partitions because too many will lead to unnecessarily higher latency while too few will not meet throughput expectations. This parameter is not directly configured via Metron but rather when manually creating the associated Kafka topics.

Storm Tuning

There are several Storm properties you can adjust to tune your Storm topologies. Achieving the desired performance can be iterative and will take some trial and error.

Hortonworks recommends you start your tuning with the Storm topology defaults and smaller numbers in terms of parallelism. Then you can iteratively increase the values until you achieve your desired level of performance. Use the offset lag tool to verify your settings.

The following sections assume log type messages. However, if your data consists of emails which are much larger in size, then you should adjust your values accordingly.

Kafka Spouts

The Kafka spouts value is the number of threads in Storm that will read from a Kafka topic. It is important to match the number of spouts with the number of partitions. If there are less consumer threads than Kafka spouts, the Storm topology may not be able to keep up with incoming events. If there are more consumer threads than required, they will often stay idle while still consuming resources. This is important because Kafka has certain ordering guarantees for message delivery per partition that would not be possible if more than one consumer in a given consumer group is able to read from that partition.

You can modify the following spout settings in the spout-config.json file. However, if the spout default settings work for your system, you can omit these settings from the file. These default settings are based on recommendations from Storm and are provided in the Kafka spout itself.

```
{
    ...
    "spout.pollTimeoutMs" : 200,
    "spout.maxUncommittedOffsets" : 10000000,
    "spout.offsetCommitPeriodMs" : 30000
}
```

Storm Topology Parallelism

To provide a uniform distribution to each machine and jvm process, you can modify the values for the number of tasks, executors, and workers properties. Start with small values and iteratively increase the values so you don't overwhelm you CPU with too many processes.

The following table lists the variables you can set to adjust the parallelism in a Storm topology and provides recommendations for their values:

Storm Topology Variables	**Description**	**Value**
num tasks	Tasks are instances of a given spout or bolt.	Set the number of tasks as a multiple of the number of executors.
num executors	Executors are threads in a process.	Set the number of executors as a multiple of the number of workers.
num workers	Storm workers are OS processes on Storm nodes.	The number of workers should relate to the number of dedicated storm nodes. It is generally good practice in a Storm topology to allocate one worker per node where each worker has approximately the same number of executors (spouts, parsers, ackers, and error writers). However, it may not be efficient to do this for certain low volume parsing topologies. In this case, it may be better to have the combined workers of multiple low volume topologies match the number of nodes. For example, 4 workers for 3 low volume tpologies on a cluster with 12 nodes.

Storm Workers

Storm workers are OS processes on Storm nodes. The number of Storm workers should relate to the number of dedicated storm nodes. It is generally good practice in a Storm topology to allocate 1 worker per node where each worker has approximately the same number of executors (spouts, parsers, ackers and error writers). However, it may not be efficient to do this for certain low volume parsing topologies. In this case, it may be better to have the

combined workers of multiple low volume topologies match the number of nodes. For example, 4 workers each for 3 low volume topologies on a cluster with 12 nodes.

You can change the number of workers in the Storm property topology.workers.

Storm Executors

Storm executors refers to the threads within a worker which process events. In Metron (which uses Flux), the number of tasks will always equal the number of executors. As a result, the number of executors maps directly to the number of bolt/spout instances in a topology.

Usually your number of tasks is equal to the number of executors, which is the default in Storm. Flux does not provide a method to independently set the number of tasks, so for enrichments and indexing, which use Flux, num tasks are always equal to num executors.

Topology Ackers

Storm Ackers are responsible for tracking completed events within a topology. After an event is processed, a checksum is sent to the acker, which when processed, will mark the event as processed. This ensures that no events are lost and they all are processed. The number of ackers can be initially set to either 1 per Storm worker or 1 per Kafka partition. One of these will generally be more than necessary and after tuning for other components is complete, the number of Ackers can be lowered based on the capacity seen during testing.

The topology.ackers.executors setting specifies how many threads are dedicated to tuple acking. Set this setting to equal the number of partitions in your inbound Kafka topic.

```
topology.ackers.executors
```

Max Spout Pending

This parameter limits the number of unacked tuples allowed in a topology. Setting this prevents the topology from becoming overloaded and causing tuples to time out and fail. Setting this too low can result in low throughput with executors not reaching capacity. Setting this too high can lead to increased latency within topologies and possibly failures when there is a spike in events ingested.

You set this property as a form of back pressure to ensure that you don't flood your topology.

```
topology.max.spout.pending
```

Spout Recommended Defaults

As a general rule, it is optimal to set spout parallelism equal to the number of partitions used in your Kafka topic. Any greater parallelism will leave you with idle consumers because Kafka limits the maximum number of consumers to the number of partitions. This is important because Kafka has certain ordering guarantees for message delivery per partition that would not be possible if more than one consumer in a given consumer group is able to read from that partition.

You can modify the following spout settings in the spout-config.json file. However, if the spout default settings work for your system, you can omit these settings from the file. These default settings are based on recommendations from Storm and are provided in the Kafka spout itself.

```
{
    ...
    "spout.pollTimeoutMs" : 200,
    "spout.maxUncommittedOffsets" : 10000000,
    "spout.offsetCommitPeriodMs" : 30000
}
```

Enrichment Tuning

Because all of the data is coming together in enrichments, you will probably need larger enrichments settings than your parallelism settings. Enrichment settings focus more on the compute workload than on the mapping workload in parsers or the IO driven workload in indexing. Enrichment makes significant use of caching for performance.

You can modify many performance tuning properties for enrichment using Ambari or Storm Flux. Modifying properties using Ambari is simple and can be performed by any user. However, you should have knowledge of Storm Flux usage and formatting before attempting to modify any Flux files.

The enrichment properties materialize as follows:

```
Ambari UI -> properties file -> Flux -> Storm
```

Index Tuning

Indexing is primarily IO driven. Tuning indexing tends to focus on the search index (Solr or Elasticsearch). Problems with indexing running too slow will often manifest as Kafka not commiting in time. This results from the indexing backing up so that it fails batches and the poll interval in Kafka is exceeded. The issue is actually with the index rather than Kafka.

You can modify many performance tuning properties for indexing using Ambari or Storm Flux. Modifying properties using Ambari is simple and can be performed by any user. However, you should have knowledge of Storm Flux usage and formatting before attempting to modify any Flux files.

The indexing properties materialize as follows:

```
Ambari UI -> properties file -> Flux -> Storm
```