Cloudera Runtime 7.2.12

Tuning Cloudera Search

Date published: 2020-01-27
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Solr server tuning categories

Solr performance tuning is a complex task. Learn about available tuning options that youcan perform either during deployment or at a later stage.

The following tuning categories can be completed either during deployment or at a later stage. It is less important to implement these changes before taking your system into use.

Table 1: Tuning steps

Tuning option	Description
Configure the Java heap size	Set the Java heap size for the Solr Server to at least 16 GB for production environments. For more information on memory requirements, see Deployment Planning for Cloudera Search.
Enable multi-threaded faceting	Enabling multi-threaded faceting can provide better performance for field faceting. It has no effect on query faceting.
Enable garbage collector (GC) logging	To help identify any garbage collector (GC) issues, enable GC logging in production. The overhead is low.
Configure garbage collection	Select the garbage collection option that offers best performance in your environment.
Configure index caching	Cloudera Search enables Solr to store indexes in an HDFS filesystem. To maintain performance, an HDFS block cache has been implemented using Least Recently Used (LRU) semantics. This enables Solr to cache HDFS index files on read and write, storing the portions of the file in JVM direct memory (off heap) by default, or optionally in the JVM heap.
Tune commit values	Changing commit values may improve performance in certain situations. These changes result in tradeoffs and may not be beneficial in all cases. • For hard commit values, the default value of 60000 (60 seconds) is typically effective, though changing this value to 120 seconds may improve performance in some cases. Note, that setting this value to higher values, such as 600 seconds may result in undesirable performance tradeoffs. • Consider increasing the auto-soft-commit value from 15000 (15 seconds) to 120000 (120 seconds). You may increase this to the largest value that still meets your requirements.
Tune sharding	In some cases, oversharding can help improve performance including intake speed. If your environment includes massively parallel hardware and you want to use these available resources, consider oversharding. You might increase the number of replicas per host from 1 to 2 or 3. Making such changes creates complex interactions, so you should continue to monitor your system's performance to ensure that the benefits of oversharding outweigh the costs.
Minimize swappiness	For better performance, Cloudera recommends setting the Linux swap space on all Solr server hosts as shown below: `sudo sysctl vm.swappiness=1`
Consider collection aliasing to deal with massive amounts of timestamped data in streaming-style applications	If you need to index and near real time query huge amounts of timestamped data in Solr, such as logs or IoT sensor data, you may consider aliasing as a massively scalable solution. This approach allows for indefinite indexing of data without degradation of performance otherwise experienced due to the continuous growth of a single index.

Additional tuning resources

Practical tuning tips outside the Cloudera Search documentation:

- General information on Solr caching is available under Query Settings in SolrConfig in the Apache Solr Reference Guide.
- Information on issues that influence performance is available on the SolrPerformanceFactors page on the Solr Wiki.
- Resource Management describes how to use Cloudera Manager to manage resources, for example with Linux cgroups.

- For information on improving querying performance, see How to make searching faster.
- For information on improving indexing performance, see How to make indexing faster.
- For information on aliasing, see Collection Aliasing: Near Real-Time Search for Really Big Data on Cloudera Blog and Time Routed Aliases in the Apache Solr Reference Guide.

Related Concepts

Tuning garbage collection
Solr and HDFS - the block cache
Tuning replication

Related Tasks

Enable multi-threaded faceting
Enable garbage collector logging

Related Information

Apache Solr Memory Tuning for Production
Solr Memory Tuning for Production (part 2)
Deployment Planning for Cloudera Search
Resource Management

Setting Java system properties for Solr

Several tuning steps require adding or modifying Java system properties. This is how you do it in Cloudera Manager.

Procedure

1. In Cloudera Manager, select the Solr service.
2. Click the Configuration tab.
3. In the Search box, type Java Configuration Options for Solr Server.
4. Add the property to Java Configuration Options for Solr Server using the format -D<property_name>=<*VALUE*>.

   Garbage collection options, such as -XX:+PrintGCTimeStamps, can also be set here. Use spaces to separate multiple parameters.
5. Click Save Changes.
6. Restart the Solr service (Solr service Actions Restart).

Enable multi-threaded faceting

Enabling multi-threaded faceting can provide better performance for field faceting.

About this task

- When multi-threaded faceting is enabled, field faceting tasks are completed in parallel with a thread working on every field faceting task simultaneously. Performance improvements do not occur in all cases, but improvements are likely when all of the following are true:
  - The system uses highly concurrent hardware.
  - Faceting operations apply to large data sets over multiple fields.
  - There is not an unusually high number of queries occurring simultaneously on the system. Systems that are lightly loaded or that are mainly engaged with ingestion and indexing may be helped by multi-threaded faceting; for example, a system ingesting articles and being queried by a researcher. Systems heavily loaded

by user queries are less likely to be helped by multi-threaded faceting; for example, an e-commerce site with heavy user-traffic.

Note: Multi-threaded faceting only applies to field faceting and not to query faceting.

- Field faceting identifies the number of unique entries for a field. For example, multi-threaded faceting could be used to simultaneously facet for the number of unique entries for the fields, "color" and "size". In such a case, there would be two threads, and each thread would work on faceting one of the two fields.
- Query faceting identifies the number of unique entries that match a query for a field. For example, query faceting could be used to find the number of unique entries in the "size" field that are between 1 and 5. Multi-threaded faceting does not apply to these operations.

Procedure

To enable multi-threaded faceting, add facet-threads to queries.

If facet-threads is omitted or set to 0, faceting is single-threaded. If facet-threads is set to a negative value, such as -1, multi-threaded faceting will use as many threads as there are fields to facet up to the maximum number of threads possible on the system.

For example, to use up to 1000 threads, use a similar query:

```
http://localhost:8983/solr/collection1/select?q=*:*&facet=true&fl=id&facet.field=f0_ws&facet.threads=1000
```

Tuning garbage collection

Choose different garbage collection options for best performance in different environments.

Some garbage collection options typically chosen include:

- Concurrent low pause collector: Use this collector in most cases. This collector attempts to minimize "Stop the World" events. Avoiding these events can reduce connection timeouts, such as with ZooKeeper, and may improve user experience. This collector is enabled using the Java system property -XX:+UseConcMarkSweepGC.
- Throughput collector: Consider this collector if raw throughput is more important than user experience. This collector typically uses more "Stop the World" events so this may negatively affect user experience and connection timeouts such as ZooKeeper heartbeats. This collector is enabled using the Java system property -XX:+UseParallelGC. If UseParallelGC "Stop the World" events create problems, such as ZooKeeper timeouts, consider using the UseParNewGC collector as an alternative collector with similar throughput benefits.

You can also affect garbage collection behavior by increasing the Eden space to accommodate new objects. With additional Eden space, garbage collection does not need to run as frequently on new objects.

Related Tasks

Setting Java system properties for Solr

Enable garbage collector logging

To help identify any garbage collector (GC) issues, enable GC logging in production.

Procedure

1. In Cloudera Manager, select the Solr service.
2. Click the Configuration tab.

3. In the Search box, type Java Configuration Options for Solr Server.
4. Add arguments controlling GC logging behavior.
   - The minimum recommended GC logging flags are: -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintGCDetails.
   - To rotate the GC logs: -Xloggc: -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles= -XX:GCLogFileSize=.

Solr and HDFS - the block cache

Cloudera Search enables Solr to store indexes in an HDFS filesystem. To maintain performance, an HDFS block cache has been implemented using Least Recently Used (LRU) semantics. This enables Solr to cache HDFS index files on read and write, storing the portions of the file in JVM direct memory (off heap) by default, or optionally in the JVM heap.

Warning: Do not enable the Solr HDFS write cache, because it can lead to index corruption.

Cloudera Search enables Solr to store indexes in an HDFS filesystem. To maintain performance, an HDFS block cache has been implemented using Least Recently Used (LRU) semantics. This enables Solr to cache HDFS index files on read and write, storing the portions of the file in JVM direct memory (off heap) by default, or optionally in the JVM heap.

Batch jobs typically do not use the cache, while Solr servers (when serving queries or indexing documents) should. When running indexing using MapReduce (MR), the MR jobs themselves do not use the block cache. Block write caching is turned off by default and should be left disabled.

Tuning of this cache is complex and best practices are continually being refined. In general, allocate a cache that is about 10-20% of the amount of memory available on the system. For example, when running HDFS and Solr on a host with 96 GB of memory, allocate 10-20 GB of memory using solr.hdfs.blockcache.slab.count. As index sizes grow you may need to tune this parameter to maintain optimal performance.

Note: Block cache metrics are currently unavailable.

Configure Index Caching

The following parameters control caching. They can be configured at the Solr process level by setting the respective Java system property or by editing solrconfig.xml directly.

If the parameters are set at the collection level (using solrconfig.xml), the first collection loaded by the Solr server takes precedence, and block cache settings in all other collections are ignored. Because you cannot control the order in which collections are loaded, you must make sure to set identical block cache settings in every collection solrconfig.xml. Block cache parameters set at the collection level in solrconfig.xml also take precedence over parameters at the process level.

Parameter	Cloudera Manager Setting	Default	Description
solr.hdfs.blockcache.enabled	HDFS Block Cache	true	Enable the block cache.

Parameter	Cloudera Manager Setting	Default	Description
solr.hdfs.blockcache.read.enabled	Not directly configurable. If the block cache is enabled, Cloudera Manager automatically enables the read cache. To override this setting, you must use the Solr Service Environment Advanced Configuration Snippet (Safety Valve).	true	Enable the read cache.
solr.hdfs.blockcache.blocksperbank	HDFS Block Cache Blocks per Slab	16384	Number of blocks per cache slab. The size of the cache is 8 KB (the block size) times the number of blocks per slab times the number of slabs.
solr.hdfs.blockcache.slab.count	HDFS Block Cache Number of Slabs	1	Number of slabs per block cache. The size of the cache is 8 KB (the block size) times the number of blocks per slab times the number of slabs.

To configure index caching using Cloudera Manager:

1. Go to the Solr service.
2. Click the Configuration tab.
3. In the Search box, type
   HDFS Block Cache

   to toggle the value of solr.hdfs.blockcache.enabled and enable or disable the block cache.

   HDFS Block Cache Blocks per Slab

   to configure solr.hdfs.blockcache.blocksperbank and set the number of blocks per cache slab.

   HDFS Block Cache Number of Slabs

   to configure solr.hdfs.blockcache.slab.count and set the number of slabs per block cache.
4. Set the new parameter value.
5. Restart Solr servers after editing the parameter.

Note:

Increasing the direct memory cache size may make it necessary to increase the maximum direct memory size (MaxDirectMemorySize) allowed by the JVM. Each Solr slab allocates memory, which is 128 MB by default, as well as allocating some additional direct memory overhead. Therefore, ensure that the MaxDirectMemorySize is set comfortably above the value expected for slabs alone. The amount of additional memory required varies according to multiple factors, but for most cases, setting MaxDirectMemorySize to at least 10-20% more than the total memory configured for slabs is sufficient. Setting MaxDirectMemorySize to the number of slabs multiplied by the slab size does not provide enough memory.

For example, for a direct memory of 20 GB the slab count should be no more than 140 with the default slab size, so there are 7 * 128 MB to store per GB of memory.

To increase the maximum direct memory size (MaxDirectMemorySize) using Cloudera Manager

1. Go to the Solr service.
2. Click the Configuration tab.
3. In the Search box, type Java Direct Memory Size of Solr Server in Bytes.
4. Set the new direct memory value.
5. Restart Solr servers after editing the parameter.

Solr HDFS optimizes caching when performing NRT indexing using Lucene's NRTCachingDirectory.

Lucene caches a newly created segment if both of the following conditions are true:

- The segment is the result of a flush or a merge and the estimated size of the merged segment is <= solr.hdfs.nrtcachingdirectory.maxmergesizemb.
- The total cached bytes is <= solr.hdfs.nrtcachingdirectory.maxcachedmb.

The following parameters control NRT caching behavior:

Parameter	Default	Description
solr.hdfs.nrtcachingdirectory.enable	true	Whether to enable the NRTCachingDirectory.
solr.hdfs.nrtcachingdirectory.maxcachedmb	192	Size of the cache in megabytes.
solr.hdfs.nrtcachingdirectory.maxmergesizemb	16	Maximum segment size to cache.

This is an example solrconfig.xml file with defaults:

```
 <directoryFactory name="DirectoryFactory">
    <bool name="solr.hdfs.blockcache.enabled">${solr.hdfs.blockcache.enabled
:true}</bool>
    <int name="solr.hdfs.blockcache.slab.count">${solr.hdfs.blockcache.sla
b.count:1}</int>
    <bool name="solr.hdfs.blockcache.direct.memory.allocation">${solr.hdfs.
blockcache.direct.memory.allocation:true}</bool>
    <int name="solr.hdfs.blockcache.blocksperbank">${solr.hdfs.blockcache.b
locksperbank:16384}</int>
    <bool name="solr.hdfs.blockcache.read.enabled">${solr.hdfs.blockcache.
read.enabled:true}</bool>
    <bool name="solr.hdfs.nrtcachingdirectory.enable">${solr.hdfs.nrtcachi
ngdirectory.enable:true}</bool>
    <int name="solr.hdfs.nrtcachingdirectory.maxmergesizemb">${solr.hdfs.nrt
cachingdirectory.maxmergesizemb:16}</int>
    <int name="solr.hdfs.nrtcachingdirectory.maxcachedmb">${solr.hdfs.nrtc
achingdirectory.maxcachedmb:192}</int>
</directoryFactory>
```

Related Tasks

Setting Java system properties for Solr

Related Information

Query Settings in SolrConfig - Apache Solr Reference Guide

Tuning replication

If you have sufficient additional hardware, you may add more replicas for a linear boost of query throughput and to prevent data loss.

Note: Do not adjust HDFS replication settings for Solr in most cases.

Note, that adding replicas may slow write performance on the first replica, but otherwise this should have minimal negative consequences.

Transaction log replication

Cloudera Search supports configurable transaction log replication levels for replication logs stored in HDFS. Cloudera recommends leaving the value unchanged at 3 or, barring that, setting it to at least 2.

Configure the transaction log replication factor for a collection by modifying the tlogDfsReplication setting in solr config.xml. The tlogDfsReplication is a setting in the updateLog settings area. An excerpt of the solrconfig.xml file where the transaction log replication factor is set is as follows:

```
 <updateHandler class="solr.DirectUpdateHandler2">

    <!-- Enables a transaction log, used for real-time get, durability, and
         solr cloud replica recovery.  The log can grow as big as
         uncommitted changes to the index, so use of a hard autoCommit
         is recommended (see below).
         "dir" - the target directory for transaction logs, defaults to the
                solr data directory.  -->
    <updateLog>
      <str name="dir">${solr.ulog.dir:}</str>
      <int name="tlogDfsReplication">${solr.ulog.tlogDfsReplication:3}</int>
      <int name="numVersionBuckets">${solr.ulog.numVersionBuckets:65536}</
int>
    </updateLog>
```

The default replication level is 3. For clusters with fewer than three DataNodes (such as proof-of-concept clusters), reduce this number to the amount of DataNodes in the cluster. Changing the replication level only applies to new transaction logs.

Initial testing shows no significant performance regression for common use cases.

Related Tasks

Adjust the Solr replication factor for index files stored in HDFS

Adjust the Solr replication factor for index files stored in HDFS

You can adjust the degree to which different data is replicated.

Procedure

1. Go to Solr service Configuration Category Advanced .
2. Click the plus sign next to Solr Service Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml to add a new property with the following values:

   Name: dfs.replication

   Value: 2
3. Click Save Changes.
4. Restart the Solr service (Solr service Actions Restart).