Cloudera Runtime 7.2.12

Indexing Data Using Morphlines

Date published: 2019-11-19
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Indexing data using Morphlines

There are generally two approaches to indexing data using Cloudera Search:

1. Near real time (NRT) indexing
2. Batch indexing

Near real time indexing is generally used when new data needs to be returned in query results in time frames measured in seconds, whereas batch indexing is useful for situations where large amounts of data is indexed at regular intervals, or for indexing a new dataset for the first time.

Near real time indexing generally uses a framework such as Apache Kafka to continuously ingest and index data. The Lily HBase Indexer can also be used for NRT indexing on Apache HBase tables.

Batch indexing usually relies on MapReduce/YARN jobs to periodically index large datasets. The Lily HBase Indexer can also be used for batch indexing HBase tables.

Related Information

Lily HBase near real time indexing using Morphlines

Batch indexing using Morphlines

Lily HBase near real time indexing using Morphlines

You can use the Lily HBase Indexer for near real time (NRT) indexing updates to HBase tables.

The Lily HBase NRT Indexer service is a flexible, scalable, fault-tolerant, transactional, NRT system for processing a continuous stream of HBase cell updates into live search indexes. Typically it takes seconds for data ingested into HBase to appear in search results; this duration is tunable. The Lily HBase Indexer uses SolrCloud to index data stored in HBase. As HBase applies inserts, updates, and deletes to HBase table cells, the indexer keeps Solr consistent with the HBase table contents, using standard HBase replication. The indexer supports flexible custom application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain columnFamily:qua lifier links back to the data stored in HBase. This way, applications can use the Search result set to directly access matching raw HBase cells. Indexing and searching do not affect operational stability or write throughput of HBase because the indexing and searching processes are separate and asynchronous to HBase.

To accommodate the HBase ingest load, you can run as many Lily HBase Indexer services on different hosts as required. Because the indexing work is shared by all indexers, you can scale the service by adding more indexers. The recommended number of indexer is 1 for each HBase RegionServer but in a High Availability environment five worker nodes is the minimum for acceptable performance and reliability. You can co-locate Lily HBase Indexer services with Solr servers on the same set of hosts. RegionServers can also be co-locate with Lily HBase Indexer on the same host to improve performance.

Note: Specific workloads and usage patterns might require additional fine-tuning beyond these general recommendations.

The Lily HBase NRT Indexer service must be deployed in an environment with a running HBase cluster, a running SolrCloud cluster (the Solr service in Cloudera Manager), and at least one ZooKeeper quorum.

Enable cluster-wide HBase replication

The Lily HBase Indexer is implemented using HBase replication, presenting indexers as RegionServers of the worker cluster. This requires HBase replication on the HBase cluster, as well as the individual tables to be indexed.

About this task

To enable replication:

Procedure

1. Go to HBase service Configuration Category Backup
2. Select the Enable Replication checkbox.
3. Set Replication Source Ratio to 1.
4. Set Replication Batch Size to 1000.
5. Click Save Changes.
6. Restart the HBase service (HBase service Actions Restart).

Adding the Lily HBase indexer service

In Cloudera Manager, the Lily HBase Indexer service is called Key-Value Store Indexer, and the service role is called Lily HBase Indexer.

Starting the Lily HBase NRT indexer service

Use Cloudera Manager to start the Lily HBase indexer service.

You can use Cloudera Manager to start the Lily HBase Indexer Service (Key-Value Store Indexer service Actions Start).

Once the service is running, you can create and manage indexers.

Using the Lily HBase NRT indexer service

To index for column families of tables in an HBase cluster:

- Enable replication on HBase column families
- Create collections and configurations
- Register a Lily HBase Indexer configuration with the Lily HBase Indexer Service
- Verify that indexing is working

Enable replication on HBase column families

About this task

Ensure that cluster-wide HBase replication is enabled, as described in Lily HBase near real time indexing using Morphlines on page 4. Use the HBase shell to define column-family replication settings.

Procedure

1. For every existing table, set the REPLICATION_SCOPE on every column family that you want to index:

```
hbase shell
hbase shell> disable 'sample_table'
hbase shell> alter 'sample_table', {NAME => 'columnfamily1', REPLICATION
_SCOPE => 1}
hbase shell> enable 'sample_table'
```

2. For every new table, set the REPLICATION_SCOPE on every column family that you want to index using a command such as the following:

```
hbase shell
hbase shell> create 'test_table', {NAME => 'testcolumnfamily', REPLICATI
ON_SCOPE => 1}
```

Create a Collection in Cloudera Search

About this task

A collection in Search used for HBase indexing must have a Solr schema that accommodates the types of HBase column families and qualifiers that are being indexed. To begin, consider adding the all-inclusive data field to a default schema.

Procedure

Once you decide on a schema, create a collection using commands similar to the following:

```
solrctl instancedir --generate $HOME/hbase_collection_config
## Edit $HOME/hbase_collection_config/conf/schema.xml as needed ##
solrctl config --upload hbase_collection_config $HOME/hbase_collection_confi
g
solrctl collection --create hbase_collection -s <numShards> -c hbase_colle
ction_config
```

Creating a Lily HBase Indexer Configuration File

About this task

Configure individual Lily HBase Indexers using the hbase-indexer command-line utility. Typically, there is one Lily HBase Indexer configuration file for each HBase table, but there can be as many Lily HBase Indexer configuration files as there are tables, column families, and corresponding collections in Search. Each Lily HBase Indexer configuration is defined in an XML file, such as morphline-hbase-mapper.xml.

An indexer configuration XML file must refer to the MorphlineResultToSolrMapper implementation and point to the location of a Morphline configuration file, as shown in the following morphline-hbase-mapper.xml indexer configuration file.

Procedure

- Set morphlineFile to the relative path morphlines.conf. Make sure the file is readable by the HBase system user (hbase by default).

```
$ cat $HOME/morphline-hbase-mapper.xml

<?xml version="1.0"?>
<indexer table="sample_table"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">
   <!-- The relative path on the local file system to the
   morphline configuration file. -->

   <param name="morphlineFile" value="morphlines.conf"/>

   <!-- The optional morphlineId identifies a morphline if there are multi
ple
   morphlines in morphlines.conf -->
   <!-- <param name="morphlineId" value="morphline1"/> -->
```

```
</indexer>
```

The Lily HBase Indexer configuration file also supports the standard attributes of any HBase Lily Indexer on the top-level <indexer> element. It does not support the <field> element and <extract> elements.

Creating a Morphline Configuration File

About this task

After creating an indexer configuration XML file, you can configure morphline ETL transformation commands in a morphlines.conf configuration file. The morphlines.conf configuration file can contain any number of morphline commands. Typically, an extractHBaseCells command is the first command. The readAvroContainer or readAvro morphline commands are often used to extract Avro data from the HBase byte array. This configuration file can be shared among different applications that use morphlines.

Note: To function properly, the morphline must not contain a loadSolr command. The Lily HBase Indexer must load documents into Solr, instead of the morphline itself.

Procedure

You can edit the morphlines.conf file within Cloudera Manager (Key-Value Store Indexer service Configuration Category Morphlines Morphlines File).

Understanding the extractHBaseCells Morphline Command

The extractHBaseCells morphline command extracts cells from an HBase result and transforms the values into a Solr InputDocument. The command consists of an array of zero or more mapping specifications.

Each mapping has:

- The inputColumn parameter, which specifies the data from HBase for populating a field in Solr. It has the form of a column family name and qualifier, separated by a colon. The qualifier portion can end in an asterisk, which is interpreted as a wildcard. In this case, all matching column-family and qualifier expressions are used. The following are examples of valid inputColumn values:
  - mycolumnfamily:myqualifier
  - mycolumnfamily:my*
  - mycolumnfamily:*
- The outputField parameter specifies the morphline record field to which to add output values. The morphline record field is also known as the Solr document field. Example: first_name.
- Dynamic output fields are enabled by the outputField parameter ending with a wildcard (*). For example:

  ```
  inputColumn : "mycolumnfamily:*"
  outputField : "belongs_to_*"
  ```

  In this case, if you make these puts in HBase:

  ```
  put 'table_name' , 'row1' , 'mycolumnfamily:1' , 'foo'
  put 'table_name' , 'row1' , 'mycolumnfamily:9' , 'bar'
  ```

  Then the fields of the Solr document are as follows:

  ```
  belongs_to_1 : foo
  belongs_to_9 : bar
  ```

- The type parameter defines the data type of the content in HBase. All input data is stored in HBase as byte arrays, but all content in Solr is indexed as text, so a method for converting byte arrays to the actual data type is required. The type parameter can be the name of a type that is supported by org.apache.hadoop.hbase.util.Bytes.to* (which

currently includes byte[], int, long, string, boolean, float, double, short, and bigdecimal). Use type byte[] to pass the byte array through to the morphline without conversion.

- type:byte[] copies the byte array unmodified into the record output field
- type:int converts with org.apache.hadoop.hbase.util.Bytes.toInt
- type:long converts with org.apache.hadoop.hbase.util.Bytes.toLong
- type:string converts with org.apache.hadoop.hbase.util.Bytes.toString
- type:boolean converts with org.apache.hadoop.hbase.util.Bytes.toBoolean
- type:float converts with org.apache.hadoop.hbase.util.Bytes.toFloat
- type:double converts with org.apache.hadoop.hbase.util.Bytes.toDouble
- type:short converts with org.apache.hadoop.hbase.util.Bytes.toShort
- type:bigdecimal converts with org.apache.hadoop.hbase.util.Bytes.toBigDecimal

Alternatively, the type parameter can be the name of a Java class that implements the com.ngdata.hbaseindexer.parse.ByteArrayValueMapper interface.

HBase data formatting does not always match what is specified by org.apache.hadoop.hbase.util.Bytes.*. For example, this can occur with data of type float or double. You can enable indexing of such HBase data by converting the data. There are various ways to do so, including:

- Using Java morphline command to parse input data, converting it to the expected output. For example:

```
{
 imports : "import java.util.*;" code: """ // manipulate the contents of
 a record field
 String stringAmount = (String) record.getFirstValue("amount");
 Double dbl = Double.parseDouble(stringAmount); record.replaceValues("
amount",dbl);
 return child.process(record); // pass record to next command in chain
 """
}
```

- Creating table fields with binary format and then using types such as double or float in a morphline.conf. You could create a table in HBase for storing doubles using commands similar to:

```
CREATE TABLE sample_lily_hbase ( id string, amount double, ts timestamp
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,ti:amount#b,ti:
ts,')
TBLPROPERTIES ('hbase.table.name' = 'sample_lily');
```

- The source parameter determines which portion of an HBase KeyValue is used as indexing input. Valid choices are value or qualifier. When value is specified, the HBase cell value is used as input for indexing. When qualifier is specified, then the HBase column qualifier is used as input for indexing. The default is value.

Registering a Lily HBase Indexer Configuration with the Lily HBase Indexer Service

About this task

When the content of the Lily HBase Indexer configuration XML file is satisfactory, register it with the Lily HBase Indexer Service. Register the Lily HBase Indexer configuration file by uploading the Lily HBase Indexer configuration XML file to ZooKeeper. For example:

Procedure

1. If your cluster has security enabled, create a Java Authentication and Authorization Service (JAAS) configuration file named jaas.conf in your home directory with the following contents:

```
Client {
```

```
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true
    principal="jdoe@EXAMPLE.COM";
};
```

Replace jdoe@EXAMPLE.COM with your user principal. Your user account must have WRITE permission to create an indexer. For more information, see Configuring Lily HBase Indexer Security on page 11.

2. If your cluster has security enabled, authenticate with the user principal specified in your jaas.conf file:

```
kinit jdoe@EXAMPLE.COM
```

3. Run the following command to add the JAAS configuration to the system properties:

```
export HBASE_INDEXER_OPTS=-Djava.security.auth.login.config=jaas.conf
```

4. Run the following command to register your indexer configuration file with the indexer service:

```
hbase-indexer add-indexer \
--name myIndexer \
--indexer-conf $HOME/morphline-hbase-mapper.xml \
--connection-param solr.zk=zk01.example.com,zk02.example.com,zk03.examp
le.com/solr \
--connection-param solr.collection=hbase_collection \
--zookeeper zk01.example.com:2181,zk02.example.com:2181,zk03.example.com
:2181
```

5. Verify that the indexer was successfully created as follows:

```
hbase-indexer list-indexers -zookeeper zk01.example.com:2181,zk02.exampl
e.com:2181,zk03.example.com:2181
Number of indexes: 1

myIndexer
  + Lifecycle state: ACTIVE
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: Indexer_myIndexer
  + SEP subscription timestamp: 2013-06-12T11:23:35.635-07:00
  + Connection type: solr
  + Connection params:
    + solr.collection = hbase-collection1
    + solr.zk = localhost/solr
  + Indexer config:
      110 bytes, use -dump to see content
  + Batch index config:
      (none)
  + Default batch index config:
      (none)
  + Processes
    + 1 running processes
    + 0 failed processes
```

Use the update-indexer and delete-indexer command-line options of the hbase-indexer utility to manipulate existing Lily HBase Indexers.

For more help, use the following commands:

```
hbase-indexer add-indexer --help
hbase-indexer list-indexers --help
hbase-indexer update-indexer --help
```

```
hbase-indexer delete-indexer --help
```

Morphline configuration files can be changed without re-creating the indexer itself, but you must restart the Lily HBase Indexer service for the changes to take effect.

Verifying that Indexing Works

Procedure

1. Add rows to the indexed HBase table. For example:

```
hbase shell
hbase(main):001:0> put 'sample_table', 'row1', 'data', 'value'
hbase(main):002:0> put 'sample_table', 'row2', 'data', 'value2'
```

2. If the put operation succeeds, wait a few seconds, go to the SolrCloud UI query page, and query the data. Note the updated rows in Solr.
3. To print diagnostic information, such as the content of records as they pass through the morphline commands, enable the TRACE log level:
   a) Go to Key-Value Store Indexer service Configuration Category Advanced .
   b) Find the Lily HBase Indexer Logging Advanced Configuration Snippet (Safety Valve) property or search for it by typing its name in the Search box.
   c) Add the following to the text box:

```
log4j.logger.org.kitesdk.morphline=TRACE
log4j.logger.com.ngdata=TRACE
```

   d) Click Save Changes.
   e) Restart the service (Key-Value Store Indexer service Actions Restart).
4. Examine the log files in /var/log/hbase-solr/lily-hbase-indexer-* for details.

Using the indexer HTTP interface

Lily HBase Indexer includes an HTTP interface for the list-indexers, create-indexer, update-indexer, and delete-indexer commands.

This interface can be secured with Kerberos for authentication and Apache Ranger for authorization. For information on configuring security for the Lily HBase Indexer service, see Configuring Lily HBase Indexer Security on page 11.

By default, the hbase-indexer command line client does not use the HTTP interface. Use the HTTP interface to take advantage of the features it provides, such as Kerberos authentication and Ranger integration. The hbase-indexer command supports two additional parameters to the list-indexers, create-indexer, delete-indexer, and update-indexer commands:

- --http: An HTTP URI for the HTTP interface. By default, this URI is of the form http://lily01.example.com:11060/indexer/. If this parameter is specified, the Lily HBase Indexer uses the HTTP API. If this parameter is not specified, the indexer communicates directly with ZooKeeper.
- --jaas: Specifies a Java Authentication and Authorization Service (JAAS) configuration file. This is only necessary for Kerberos-enabled deployments.

Note: Make sure that you use fully qualified domain names (FQDN) when specifying hostnames for both the Lily HBase Indexer host and the ZooKeeper hosts. Using FQDNs helps ensure proper Kerberos realm mapping.

For example:

```
hbase-indexer list-indexers --http http://lily01.example.com:11060/indexer/
\
```

```
--jaas $HOME/jaas.conf --zookeeper zk01.example.com:2181,zk02.example.com:
2181,zk03.example.com:2181
```

Related Information

Configuring Lily HBase Indexer Security

Configuring Lily HBase Indexer Security

The Lily HBase Indexer includes an HTTP interface for the list-indexers, create-indexer, update-indexer, and delete-indexer commands. This interface can be secured with Kerberos for authentication and Apache Ranger for authorization.

Configure Lily HBase Indexer to use TLS/SSL

Although Cloudera recommends using AutoTLS, you also have the option to set up TLS manually for the Lily HBase Indexer.

About this task

To configure and enable Hadoop TLS/SSL for the Lily HBase Indexer (Key-Value Store Indexer) perform the following steps.

Procedure

1. Open the Cloudera Manager Admin Console and go to the Key-Value Store Indexer.
2. Click the Configuration tab.
3. Select Scope All .
4. Select Category All .
5. In the Search field, type TLS/SSL to show the Solr TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration.

Note: These values must be the same for all hosts running the Key-Value Store Indexer role.

Table 1: Key-Value Store TLS/SSL Properties

Property	Description
HBase Indexer TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that HBase Indexer might connect to. This is used when HBase Indexer is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
HBase Indexer TLS/SSL Certificate Trust Store Password (Optional)	The password for the HBase Indexer TLS/SSL Certificate Trust Store File. This password is not required to access the truststore: this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

7. Restart the service.

Configure Lily HBase Indexer Service to use Kerberos authentication

To enable Kerberos authentication for the Lily HBase Indexer service, perform the following steps.

Procedure

1. In the Cloudera Manager admin console, go to Key-Value Store Indexer service Configuration Category Security .

2. Select the kerberos option for HBase Indexer Secure Authentication.
3. Click Save Changes.
4. Go to Administration Security Kerberos Credentials .
5. Click Generate Missing Credentials.
6. Restart the indexer service (Key-Value Store Indexer service Actions Restart).

Batch indexing using Morphlines

Batch indexing usually relies on MapReduce/YARN or Spark jobs to periodically index large datasets, or to index new datasets for the first time. The Lily HBase indexer, also called HBaseMapReduceIndexerTool, can be used for batch indexing HBase tables.

Spark indexing using morphlines

If you are using Apache Spark, you can batch index data using the CrunchIndexerTool.

CrunchIndexerTool is a Spark or MapReduce ETL batch job that pipes data from HDFS files into Apache Solr through a morphline for extraction and transformation. The program is designed for flexible, scalable, fault-tolerant batch ETL pipeline jobs. It is implemented as an Apache Crunch pipeline, allowing it to run on MapReduce or Spark execution engines.

CrunchIndexerTool requires a working MapReduce or Spark cluster, such as one installed using Cloudera Manager.

Note: This command requires a morphline file, which must include a SOLR_LOCATOR directive. The snippet that includes the SOLR_LOCATOR might appear as follows:

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection_name

  # ZooKeeper ensemble
  zkHost :
 "zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/
solr"
}

morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
    commands : [
      { generateUUID { field : id } }

      { # Remove record fields that are unknown to Solr schema.xml.
        # Recall that Solr throws an exception on any attempt to load a
 document that
        # contains a field that isn't specified in schema.xml.
        sanitizeUnknownSolrFields {
          solrLocator : ${SOLR_LOCATOR} # Location from which to fetch
 Solr schema
        }
      }

      { logDebug { format : "output record: {}", args : ["@{}"] } }

      {
        loadSolr {
          solrLocator : ${SOLR_LOCATOR}
        }
      }
    ]
  }
]
```

You can see the usage syntax CrunchIndexerTool by running the job with the -help argument. Unlike other Search indexing tools, the CrunchIndexerTool jar does not contain all dependencies. If you try to run the job without addressing this, you get an error such as the following:

```
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-crunch.j
ar org.apache.solr.crunch.CrunchIndexerTool -help
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/crun
ch/types/PType
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:348)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:214)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
Caused by: java.lang.ClassNotFoundException: org.apache.crunch.types.PType
        at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        ... 4 more
```

To see the command usage (or to run the job), you must first add the dependencies to the classpath:

```
export HADOOP_CLASSPATH="/opt/cloudera/parcels/CDH/lib/search/lib/search-cru
nch/*"
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-crunc
h.jar org.apache.solr.crunch.CrunchIndexerTool -help
```

For reference, here is the command usage syntax:

Important: The command usage help incorrectly notes the following:

NOTE: MapReduce does not require extra steps for communicating with kerberos-enabled Solr

To run the MapReduce job on a Kerberos-enabled cluster, you must create and specify a jaas.conf file.

For example:

```
HADOOP_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" \
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-cru
nch.jar \
org.apache.solr.crunch.CrunchIndexerTool [...]
```

```
MapReduceUsage: export HADOOP_CLASSPATH=$myDependencyJarPaths; hadoop jar $m
yDriverJar
org.apache.solr.crunch.CrunchIndexerTool --libjars $myDependencyJarFiles
[MapReduceGenericOptions]...
        [--input-file-list URI] [--input-file-format FQCN]
        [--input-file-projection-schema FILE]
        [--input-file-reader-schema FILE] --morphline-file FILE
        [--morphline-id STRING] [--pipeline-type STRING] [--xhelp]
        [--mappers INTEGER] [--parallel-morphline-inits INTEGER]
        [--dry-run] [--log4j FILE] [--chatty] [HDFS_URI [HDFS_URI ...]]

SparkUsage: spark-submit [SparkGenericOptions]... --master local|yarn --d
eploy-mode client|cluster
--jars $myDependencyJarFiles --class org.apache.solr.crunch.CrunchIndexerT
ool $myDriverJar
        [--input-file-list URI] [--input-file-format FQCN]
        [--input-file-projection-schema FILE]
        [--input-file-reader-schema FILE] --morphline-file FILE
        [--morphline-id STRING] [--pipeline-type STRING] [--xhelp]
        [--mappers INTEGER] [--parallel-morphline-inits INTEGER]
        [--dry-run] [--log4j FILE] [--chatty] [HDFS_URI [HDFS_URI ...]]

Spark or MapReduce ETL batch job that pipes data from (splittable or non-
splittable) HDFS files into Apache Solr, and along the way runs the data
through a Morphline for extraction and transformation. The program is
designed for flexible, scalable and fault-tolerant batch ETL pipeline
jobs. It is implemented as an Apache Crunch pipeline and as such can run
on either the Apache Hadoop MapReduce or Apache Spark execution engine.

The program proceeds in several consecutive phases, as follows:
1) Randomization phase: This (parallel) phase randomizes the list of HDFS
input files in order to spread ingestion load more evenly among the mapper
tasks of the subsequent phase. This phase is only executed for non-
splittables files, and skipped otherwise.

2) Extraction phase: This (parallel) phase emits a series of HDFS file
input streams (for non-splittable files) or a series of input data records
(for splittable files).
3) Morphline phase: This (parallel) phase receives the items of the
previous phase, and uses a Morphline to extract the relevant content,
transform it and load zero or more documents into Solr. The ETL
```

```
functionality is flexible and customizable using chains of arbitrary
morphline commands that pipe records from one transformation command to
another. Commands to parse and transform a set of standard data formats
such as Avro, Parquet, CSV, Text, HTML, XML, PDF, MS-Office, etc. are
provided out of the box, and additional custom commands and parsers for
additional file or data formats can be added as custom morphline commands.
Any kind of data format can be processed and any kind output format can be
generated by any custom Morphline ETL logic. Also, this phase can be used
to send data directly to a live SolrCloud cluster (via the loadSolr
morphline command).

The program is implemented as a Crunch pipeline and as such Crunch
optimizes the logical phases mentioned above into an efficient physical
execution plan that runs a single mapper-only job, or as the corresponding
Spark equivalent.
Fault Tolerance: Task attempts are retried on failure per the standard
MapReduce or Spark semantics. If the whole job fails you can retry simply
by rerunning the program again using the same arguments.

Comparison with MapReduceIndexerTool:

1) CrunchIndexerTool can also run on the Spark execution engine, not just
on MapReduce.
2) CrunchIndexerTool enables interactive low latency prototyping, in
particular in Spark 'local' mode.
3) CrunchIndexerTool supports updates (and deletes) of existing documents
in Solr, not just inserts.
4) CrunchIndexerTool can exploit data locality for splittable Hadoop files
(text, avro, avroParquet).
We recommend MapReduceIndexerTool for large scale batch ingestion use
cases where updates (or deletes) of existing documents in Solr are not
required, and we recommend CrunchIndexerTool for all other use cases.

CrunchIndexerOptions:
  HDFS_URI               HDFS URI of file or directory tree to ingest.
                         (default: [])
  --input-file-list URI, --input-list URI
                         Local URI or HDFS URI of a UTF-8 encoded file
                         containing a list of HDFS URIs to ingest, one URI
                         per line in the file. If '-' is specified, URIs
                         are read from the standard input. Multiple --
                         input-file-list arguments can be specified.
  --input-file-format FQCN
                         The Hadoop FileInputFormat to use for extracting
                         data from splittable HDFS files. Can be a fully
                         qualified Java class name or one of ['text',
                         'avro', 'avroParquet']. If this option is present
                         the extraction phase will emit a series of input
                         data records rather than a series of HDFS file
                         input streams.
  --input-file-projection-schema FILE
                         Relative or absolute path to an Avro schema file
                         on the local file system. This will be used as
                         the projection schema for Parquet input files.
  --input-file-reader-schema FILE
                         Relative or absolute path to an Avro schema file
                         on the local file system. This will be used as
                         the reader schema for Avro or Parquet input
                         files. Example: src/test/resources/test-
                         documents/strings.avsc
  --morphline-file FILE  Relative or absolute path to a local config file
                         that contains one or more morphlines. The file
                         must be UTF-8 encoded. It will be uploaded to
                         each remote task. Example: /path/to/morphline.conf
```

```
  --morphline-id STRING  The identifier of the morphline that shall be
                         executed within the morphline config file
                         specified by --morphline-file. If the --morphline-
                         id option is omitted the first (i.e. top-most)
                         morphline within the config file is used.
                         Example: morphline1
  --pipeline-type STRING
                         The engine to use for executing the job. Can be
                         'mapreduce' or 'spark'. (default: mapreduce)
  --xhelp, --help, -help
                         Show this help message and exit
  --mappers INTEGER      Tuning knob that indicates the maximum number of
                         MR mapper tasks to use. -1 indicates use all map
                         slots available on the cluster. This parameter
                         only applies to non-splittable input files
                         (default: -1)
  --parallel-morphline-inits INTEGER
                         Tuning knob that indicates the maximum number of
                         morphline instances to initialize at the same
                         time. This kind of rate limiting on rampup can be
                         useful to avoid overload conditions such as
                         ZooKeeper connection limits or DNS lookup limits
                         when using many parallel mapper tasks because
                         each such task contains one morphline. 1
                         indicates initialize each morphline separately.
                         This feature is implemented with a distributed
                         semaphore. The default is to use no rate limiting
                         (default: 2147483647)
  --dry-run              Run the pipeline but print documents to stdout
                         instead of loading them into Solr. This can be
                         used for quicker turnaround during early trial &
                         debug sessions. (default: false)
  --log4j FILE           Relative or absolute path to a log4j.properties
                         config file on the local file system. This file
                         will be uploaded to each remote task. Example:
                         /path/to/log4j.properties
  --chatty               Turn on verbose output. (default: false)

SparkGenericOptions:     To print all options run 'spark-submit --help'

MapReduceGenericOptions: Generic options supported are
  --conf <configuration file>
                         specify an application configuration file
  -D <property=value>    use value for given property
  --fs <local|namenode:port>
                         specify a namenode
  --jt <local|resourcemanager:port>
                         specify a ResourceManager
  --files <comma separated list of files>
                         specify comma separated files to be copied to the
                         map reduce cluster
  --libjars <comma separated list of jars>
                         specify comma separated jar files to include in
                         the classpath.
  --archives <comma separated list of archives>
                         specify comma separated archives to be unarchived
                         on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]


Examples:
```

```
# Prepare - Copy input files into HDFS:
export myResourcesDir=src/test/resources # for build from git
export myResourcesDir=/opt/cloudera/parcels/CDH/share/doc/search-*/search-c
runch # for CDH with parcels
export myResourcesDir=/usr/share/doc/search-*/search-crunch # for CDH with
packages
hadoop fs -copyFromLocal $myResourcesDir/test-documents/hello1.txt hdfs:/us
er/systest/input/

# Prepare variables for convenient reuse:
export myDriverJarDir=target # for build from git
export myDriverJarDir=/opt/cloudera/parcels/CDH/lib/solr/contrib/crunch #
 for CDH with parcels
export myDriverJarDir=/usr/lib/solr/contrib/crunch # for CDH with packages
export myDependencyJarDir=target/lib # for build from git
export myDependencyJarDir=/opt/cloudera/parcels/CDH/lib/search/lib/search-
crunch # for CDH with parcels
export myDependencyJarDir=/usr/lib/search/lib/search-crunch # for CDH with
 packages
export myDriverJar=$(find $myDriverJarDir -maxdepth 1 -name 'search-crunch-
*.jar' ! -name '*-job.jar' ! -name '*-sources.jar')
export myDependencyJarFiles=$(find $myDependencyJarDir -name '*.jar' | sort
 | tr '\n' ',' | head -c -1)
export myDependencyJarPaths=$(find $myDependencyJarDir -name '*.jar' | sort
 | tr '\n' ':' | head -c -1)
export myJVMOptions="-DmaxConnectionsPerHost=10000 -DmaxConnections=10000 -
Djava.io.tmpdir=/my/tmp/dir/" # connection settings for solrj, also custom
tmp dir
# MapReduce on Yarn - Ingest text file line by line into Solr:
export HADOOP_CLIENT_OPTS="$myJVMOptions"; export HADOOP_CLASSPATH=$myDepend
encyJarPaths; hadoop \
  --config /etc/hadoop/conf.cloudera.YARN-1 \
  jar $myDriverJar org.apache.solr.crunch.CrunchIndexerTool \
  --libjars $myDependencyJarFiles \
  -D mapreduce.map.java.opts="-Xmx500m $myJVMOptions" \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --files $myResourcesDir/test-documents/string.avsc \
  --morphline-file $myResourcesDir/test-morphlines/loadSolrLine.conf \
  --pipeline-type mapreduce \
  --chatty \
  --log4j $myResourcesDir/log4j.properties \
  /user/systest/input/hello1.txt

# Spark in Local Mode (for rapid prototyping) - Ingest into Solr:
spark-submit \
  --master local \
  --deploy-mode client \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  # --driver-library-path /opt/cloudera/parcels/CDH/lib/hadoop/lib/native #
for Snappy on CDH with parcels\
  # --driver-library-path /usr/lib/hadoop/lib/native # for Snappy on CDH wit
h packages \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  $myDriverJar \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --morphline-file $myResourcesDir/test-morphlines/loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j $myResourcesDir/log4j.properties \
  /user/systest/input/hello1.txt
```

```
# Spark on Yarn in Client Mode (for testing) - Ingest into Solr:
Same as above, except replace '--master local' with '--master yarn'
# View the yarn executor log files (there is no GUI yet):
yarn logs --applicationId $application_XYZ

# Spark on Yarn in Cluster Mode (for production) - Ingest into Solr:
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  --files $(ls $myResourcesDir/log4j.properties),$(ls $myResourcesDir/test-
morphlines/loadSolrLine.conf)\
  $myDriverJar \
  -D hadoop.tmp.dir=/tmp \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --morphline-file loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j log4j.properties \
  /user/systest/input/hello1.txt

# Spark on Yarn in Cluster Mode (for production) - Ingest into Secure (Ke
rberos-enabled) Solr:
# Spark requires two additional steps compared to non-secure solr:
# (NOTE: MapReduce does not require extra steps for communicating with kerb
eros-enabled Solr)
# 1) Create a delegation token file
#    a) kinit as the user who will make solr requests
#    b) request a delegation token from solr and save it to a file:
#       e.g. using curl:
#       "curl --negotiate -u: http://solr-host:port/solr/admin?op=GETDELEG
ATIONTOKEN > tokenFile.txt"
# 2) Pass the delegation token file to spark-submit:
#    a) add the delegation token file via --files
#    b) pass the file name via -D tokenFile
#       spark places this file in the cwd of the executor, so only list the
 file name, no path
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  --files $(ls $myResourcesDir/log4j.properties),$(ls $myResourcesDir/test-
morphlines/loadSolrLine.conf),tokenFile.txt\
  $myDriverJar \
  -D hadoop.tmp.dir=/tmp \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  -DtokenFile=tokenFile.txt \
  --morphline-file loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j log4j.properties \
  /user/systest/input/hello1.txt
```

MapReduce indexing

Cloudera Search provides the ability to batch index documents using MapReduce jobs.

Running an example indexing job

For examples of running a MapReduce job to index documents, see Cloudera Search Tutorial

MapReduceIndexerTool

MapReduceIndexerTool (MRIT) is a MapReduce batch job driver that takes a morphline and creates a set of Solr index shards from a set of input files and writes the indexes into HDFS in a flexible, scalable, and fault-tolerant manner. MRIT also supports merging the output shards into a set of live customer-facing Solr servers, typically a SolrCloud.

Important: Merging output shards into live customer-facing Solr servers can only be completed if all replicas are online.

The indexer creates an offline index on HDFS in the output directory specified by the --output-dir parameter. If the --go-live parameter is specified, Solr merges the resulting offline index into the live running service. Thus, the Solr service must have read access to the contents of the output directory to complete the go-live step. In an environment with restrictive permissions, such as one with an HDFS umask of 077, the Solr user may not be able to read the contents of the newly created directory. To address this issue, the indexer automatically applies the HDFS ACLs to enable Solr to read the output directory contents. These ACLs are only applied if HDFS ACLs are enabled on the HDFS NameNode.

The indexer only makes ACL updates to the output directory and its contents. If the output directory's parent directories do not include the run permission, the Solr service is not be able to access the output directory. Solr must have run permissions from standard permissions or ACLs on the parent directories of the output directory.

Important: MRIT does not work if Cloudera Search is deployed with local file system.

Note: Using --libjars parameter in dry-run mode does not work. Instead, specify the JAR files using the HADOOP_CLASSPATH environmental variable.

Related Information

Extracting, transforming, and loading data with Cloudera Morphlines

Using morphlines to index Avro

Using morphlines with syslog

HDFS ACLs

MapReduceIndexerTool input splits

Different from some other indexing tools, the MapReduceIndexerTool does not operate on HDFS blocks as input splits. This means that when indexing a smaller number of large files, fewer hosts may be involved. For example, indexing two files that are each one GB results in two hosts acting as mappers. If these files were stored on a system with a 128 MB block size, other mappers might divide the work on the two files among 16 mappers, corresponding to the 16 HDFS blocks that store the two files.

This intentional design choice aligns with MapReduceIndexerTool supporting indexing non-splittable file formats such as JSON, XML, jpg, or log4j.

In theory, this could result in inefficient use of resources when a single host indexes a large file while many other hosts sit idle. In reality, this indexing strategy typically results in satisfactory performance in production environments because in most cases the number of files is large enough that work is spread throughout the cluster.

While dividing tasks by input splits does not present problems in most cases, users may still want to divide indexing tasks along HDFS splits. In that case, use the CrunchIndexerTool, which can work with Hadoop input splits using the input-file-format option.

MapReduceIndexerTool metadata

The MapReduceIndexerTool generates metadata fields for each input file when indexing. These fields can be used in morphline commands. These fields can also be stored in Solr, by adding definitions like the following to your Solr schema.xml file. After the MapReduce indexing process completes, the fields are searchable through Solr.

```
<!-- file metadata -->
<field name="file_download_url" type="string" indexed="false" stored="tru
e" />
<field name="file_upload_url" type="string" indexed="false" stored="true" />
<field name="file_scheme" type="string" indexed="true" stored="true" />
<field name="file_host" type="string" indexed="true" stored="true" />
<field name="file_port" type="int" indexed="true" stored="true" />
<field name="file_path" type="string" indexed="true" stored="true" />
<field name="file_name" type="string" indexed="true" stored="true" />
<field name="file_length" type="tlong" indexed="true" stored="true" />
<field name="file_last_modified" type="tlong" indexed="true" stored="true" /
>
<field name="file_owner" type="string" indexed="true" stored="true" />
<field name="file_group" type="string" indexed="true" stored="true" />
<field name="file_permissions_user" type="string" indexed="true" stored="
true" />
<field name="file_permissions_group" type="string" indexed="true" stored="t
rue" />
<field name="file_permissions_other" type="string" indexed="true" stored="tr
ue" />
<field name="file_permissions_stickybit" type="boolean" indexed="true" st
ored="true" />
```

Example output:

```
"file_upload_url":"foo/test-documents/sample-statuses-20120906-141433.avro",
"file_download_url":"hdfs://host1.mycompany.com:8020/user/foo/ test-documen
ts/sample-statuses-20120906-141433.avro",
"file_scheme":"hdfs",
"file_host":"host1.mycompany.com",
"file_port":8020,
"file_name":"sample-statuses-20120906-141433.avro",
"file_path":"/user/foo/test-documents/sample-statuses-20120906-141433.avro",
"file_last_modified":1357193447106,
"file_length":1512,
"file_owner":"foo",
"file_group":"foo",
"file_permissions_user":"rw-",
"file_permissions_group":"r--",
"file_permissions_other":"r--",
"file_permissions_stickybit":false,
```

MapReduceIndexerTool usage syntax

Learn about the use of the MapReduceIndexer command line tool.

Important: You must run the indexer tool with the following command-line argument:

```
-D 'mapreduce.job.user.classpath.first=true'
```

Running the tool without this argument triggers the following error:

```
ERROR [main] org.apache.hadoop.mapred.YarnChild: Error running child :
 java.lang.NoSuchMethodError:
com.codahale.metrics.MetricRegistry.meter(Ljava/lang/String;Lcom/cod
ahale/metrics/MetricRegistry$MetricSupplier;)Lcom/codahale/metrics/M
eter;
```

To view the usage syntax in a default parcel-based deployment, run:

```
hadoop jar /opt/cloudera/parcels/CDH/jars/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool --help
```

```
usage: hadoop [GenericOptions]... jar search-mr-*-job.jar org.apache.solr.ha
doop.MapReduceIndexerTool
       [--help] --output-dir HDFS_URI [--input-list URI]
       --morphline-file FILE [--morphline-id STRING] [--solr-home-dir DIR]
       [--update-conflict-resolver FQCN] [--mappers INTEGER]
       [--reducers INTEGER] [--max-segments INTEGER]
       [--fair-scheduler-pool STRING] [--dry-run] [--log4j FILE]
       [--verbose] [--show-non-solr-cloud] [--zk-host STRING] [--go-live]
       [--collection STRING] [--go-live-min-replication-factor INTEGER]
       [--go-live-threads INTEGER] [HDFS_URI [HDFS_URI ...]]
MapReduce batch job driver that takes a morphline and creates a set of
Solr index shards from a set of input files and writes the indexes into
HDFS, in a flexible, scalable and fault-tolerant manner. It also supports
merging the output shards into a set of live customer facing Solr servers,
typically a SolrCloud. The program proceeds in several consecutive
MapReduce based phases, as follows:

1) Randomization phase: This (parallel) phase randomizes the list of input
files in order to spread indexing load more evenly among the mappers of
the subsequent phase.
2) Mapper phase: This (parallel) phase takes the input files, extracts the
relevant content, transforms it and hands SolrInputDocuments to a set of
reducers. The ETL functionality is flexible and customizable using chains
of arbitrary morphline commands that pipe records from one transformation
command to another. Commands to parse and transform a set of standard data
formats such as Avro, CSV, Text, HTML, XML, PDF, Word, Excel, etc. are
provided out of the box, and additional custom commands and parsers for
additional file or data formats can be added as morphline plugins. This is
done by implementing a simple Java interface that consumes a record (e.g.
a file in the form of an InputStream plus some headers plus contextual
metadata) and generates as output zero or more records. Any kind of data
format can be indexed and any Solr documents for any kind of Solr schema
can be generated, and any custom ETL logic can be registered and executed.
Record fields, including MIME types, can also explicitly be passed by
force from the CLI to the morphline, for example: hadoop ... -D
morphlineField._attachment_mimetype=text/csv

3) Reducer phase: This (parallel) phase loads the mapper's
SolrInputDocuments into one EmbeddedSolrServer per reducer. Each such
reducer and Solr server can be seen as a (micro) shard. The Solr servers
store their data in HDFS.
4) Mapper-only merge phase: This (parallel) phase merges the set of
reducer shards into the number of solr shards expected by the user, using
a mapper-only job. This phase is omitted if the number of shards is
already equal to the number of shards expected by the user.
```

```
5) Go-live phase: This optional (parallel) phase merges the output shards
of the previous phase into a set of live customer facing Solr servers,
typically a SolrCloud. If this phase is omitted you can explicitly point
each Solr server to one of the HDFS output shard directories.
Fault Tolerance: Mapper and reducer task attempts are retried on failure
per the standard MapReduce semantics. On program startup all data in the --
output-dir is deleted if that output directory already exists. If the
whole job fails you can retry simply by rerunning the program again using
the same arguments.

positional arguments:
  HDFS_URI              HDFS URI of file or directory tree to index.
                        (default: [])

optional arguments:
  --help, -help, -h     Show this help message and exit
  --input-list URI      Local URI or HDFS URI of a UTF-8 encoded file
                        containing a list of HDFS URIs to index, one URI
                        per line in the file. If '-' is specified, URIs
                        are read from the standard input. Multiple --
                        input-list arguments can be specified.
  --morphline-id STRING The identifier of the morphline that shall be
                        executed within the morphline config file
                        specified by --morphline-file. If the --morphline-
                        id option is ommitted the first (i.e. top-most)
                        morphline within the config file is used.
                        Example: morphline1
  --solr-home-dir DIR   Optional relative or absolute path to a local dir
                        containing Solr conf/ dir and in particular
                        conf/solrconfig.xml and optionally also lib/ dir.
                        This directory will be uploaded to each MR task.
                        Example: src/test/resources/solr/minimr
  --update-conflict-resolver FQCN
                        Fully qualified class name of a Java class that
                        implements the UpdateConflictResolver interface.
                        This enables deduplication and ordering of a
                        series of document updates for the same unique
                        document key. For example, a MapReduce batch job
                        might index multiple files in the same job where
                        some of the files contain old and new versions of
                        the very same document, using the same unique
                        document key.
                        Typically, implementations of this interface
                        forbid collisions by throwing an exception, or
                        ignore all but the most recent document version,
                        or, in the general case, order colliding updates
                        ascending from least recent to most recent
                        (partial) update. The caller of this interface (i.
                        e. the Hadoop Reducer) will then apply the
                        updates to Solr in the order returned by the
                        orderUpdates() method.
                        The                                    default
                        RetainMostRecentUpdateConflictResolver
                        implementation ignores all but the most recent
                        document version, based on a configurable numeric
                        Solr field, which defaults to the
                        file_last_modified timestamp (default: org.apache.
                        solr.hadoop.dedup.
                        RetainMostRecentUpdateConflictResolver)
  --mappers INTEGER     Tuning knob that indicates the maximum number of
                        MR mapper tasks to use. -1 indicates use all map
                        slots available on the cluster. (default: -1)
  --reducers INTEGER    Tuning knob that indicates the number of reducers
```

```
                          to index into.  0  is  reserved  for a mapper-only
                          feature that may  ship  in  a  future  release. -1
                          indicates use all  reduce  slots  available on the
                          cluster. -2 indicates use  one  reducer per output
                          shard,  which   disables   the   mtree   merge MR
                          algorithm. The mtree  merge  MR algorithm improves
                          scalability by spreading  load  (in particular CPU
                          load) among a  number  of  parallel  reducers that
                          can be much larger than  the number of solr shards
                          expected by  the  user.  It  can  be  seen  as  an
                          extension of concurrent  lucene  merges and tiered
                          lucene  merges   to   the   clustered   case.  The
                          subsequent mapper-only phase merges  the output of
                          said large number  of  reducers  to  the number of
                          shards expected by  the  user,  again by utilizing
                          more  available   parallelism   on   the  cluster.
                          (default: -1)
  --max-segments INTEGER
                          Tuning knob that indicates  the  maximum number of
                          segments to be contained  on  output  in the index
                          of each reducer shard.  After  a reducer has built
                          its output index  it  applies  a  merge  policy to
                          merge segments  until  there  are  <=  maxSegments
                          lucene  segments  left  in   this  index.  Merging
                          segments involves reading  and  rewriting all data
                          in all these  segment  files, potentially multiple
                          times,  which  is  very  I/O  intensive  and  time
                          consuming. However, an  index  with fewer segments
                          can later be merged  faster,  and  it can later be
                          queried  faster  once  deployed  to  a  live  Solr
                          serving shard. Set  maxSegments  to  1 to optimize
                          the index for low query  latency. Set  maxSegments
                          to 0 or -1 to skip the entire optimize phase. In a
                          nutshell,   a   small   maxSegments   value  trades

                          indexing  latency  for subsequently improved query
                          latency. This can be  a reasonable  trade-off  for
                          batch  indexing systems. (default: 1)
  --dry-run               Run in local mode  and  print  documents to stdout
                          instead of loading them  into  Solr. This executes
                          the  morphline  in  the  client  process  (without
                          submitting a job  to  MR)  for  quicker turnaround
                          during early  trial  &  debug  sessions. (default:
                          false)
  --log4j FILE            Relative or absolute  path  to  a log4j.properties
                          config file on the  local  file  system. This file
                          will  be  uploaded  to   each  MR  task.  Example:
                          /path/to/log4j.properties
  --verbose, -v           Turn on verbose output. (default: false)
  --show-non-solr-cloud   Also show options for  Non-SolrCloud  mode as part
                          of --help. (default: false)

Required arguments:
  --output-dir HDFS_URI   HDFS directory to  write  Solr  indexes to. Inside
                          there one  output  directory  per  shard  will  be
                          generated.    Example:    hdfs://c2202.mycompany.
                          com/user/$USER/test
  --morphline-file FILE   Relative or absolute path  to  a local config file
                          that contains one  or  more  morphlines.  The file
                          must     be     UTF-8     encoded.     Example:
                          /path/to/morphline.conf
Cluster arguments:
  Arguments that provide information about your Solr cluster.
```

```
  --zk-host STRING       The address of a ZooKeeper  ensemble being used by
                         a SolrCloud cluster. This  ZooKeeper ensemble will
                         be examined  to  determine  the  number  of output
                         shards to create  as  well  as  the  Solr  URLs to
                         merge the output shards into  when using the --go-
                         live option. Requires that  you  also  pass the --
                         collection to merge the shards into.

                         The  --zk-host  option  implements  the  same
                         partitioning semantics as  the  standard SolrCloud
                         Near-Real-Time (NRT)  API.  This  enables  to  mix
                         batch  updates  from  MapReduce  ingestion  with
                         updates from standard  Solr  NRT  ingestion on the
                         same SolrCloud  cluster,  using  identical  unique
                         document keys.

                         Format is: a  list  of  comma  separated host:port
                         pairs,  each  corresponding  to  a  zk  server.
                         Example: '127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:
                         2183' If the optional  chroot  suffix  is used the
                         example  would  look  like:  '127.0.0.1:2181/solr,
                         127.0.0.1:2182/solr,127.0.0.1:2183/solr'     where
                         the client would  be  rooted  at  '/solr'  and all
                         paths would  be  relative  to  this  root  -  i.e.
                         getting/setting/etc... '/foo/bar' would  result in
                         operations being run on  '/solr/foo/bar' (from the
                         server perspective).

                         If --solr-home-dir  is  not  specified,  the  Solr
                         home  directory   for   the   collection   may  be
                         downloaded from this ZooKeeper ensemble.
Go live arguments:
  Arguments for  merging  the  shards  that  are  built  into  a  live Solr
  cluster. Also see the Cluster arguments.

  --go-live              Allows you to  optionally  merge  the  final index
                         shards into a  live  Solr  cluster  after they are
                         built. You can pass the  ZooKeeper address with --
                         zk-host and the relevant  cluster information will
                         be auto detected.  (default: false)
  --collection STRING    The SolrCloud  collection  to  merge  shards  into
                         when  using  --go-live   and  --zk-host.  Example:
                         collection1
  --go-live-min-replication-factor INTEGER
                         The  minimum  number  of   SolrCloud replicas  to
                         successfully merge  any  final  index  shard into.
                         The go-live  job  phase  attempts  to  merge final
                         index shards into all  SolrCloud replicas. Some of
                         these merge operations  may  fail,  for example if
                         some  SolrCloud  servers  are  down.  This  option
                         enables indexing  jobs  to  succeed  even  if some
                         such   merge   operations    fail   on   SolrCloud
                         followers. Successful  merge  operations  into all
                         leaders  are  always  required  for  job  success,
                         regardless   of   the   value   of   --go-live-min-
                         replication-factor.    -1     indicates    require
                         successful merge operations  into  all replicas. 1
                         indicates  require  successful   merge  operations
                         only into leader replicas. (default: -1)
  --go-live-threads INTEGER
                         Tuning knob that indicates  the  maximum number of
                         live merges  to  run  in  parallel  at  one  time.
                         (default: 1000)
Generic options supported are:
```

```
  --conf <configuration file>
                          specify an application configuration file
  -D <property=value>     define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use,
 overrides 'fs.defaultFS' property from configurations.
  --jt <local|resourcemanager:port>
                          specify a ResourceManager
  --files <file1,...>     specify a  comma-separated  list  of  files  to be
                          copied to the map reduce cluster
  --libjars <jar1,...>    specify a comma-separated list of  jar files to be
                          included in the classpath
  --archives <archive1,...>
                          specify a comma-separated list  of  archives to be
                          unarchived on the compute machines
The general command line syntax is:
command [genericOptions] [commandOptions]

Examples:

# (Re)index an Avro based Twitter tweet file:
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file ../search-core/src/test/resources/test-morphlines/tutori
alReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro

# (Re)index all files that match all of the following conditions:
# 1) File is contained in dir tree hdfs:///user/$USER/solrloadtest/twitter/
tweets
# 2) file name matches the glob pattern 'sample-statuses*.gz'
# 3) file was last modified less than 100000 minutes ago
# 4) file size is between 1 MB and 1 GB
# Also include extra library jar file containing JSON tweet Java parser:
hadoop fs \
  -find hdfs:///user/$USER/solrloadtest/twitter/tweets \
  -type f \
  -name 'sample-statuses*.gz' \
  -mmin -1000000 \
  -size -100000000c \
  -size +1000000c \
| sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerT
ool \
  --libjars /path/to/kite-morphlines-twitter-0.10.0.jar \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file ../search-core/src/test/resources/test-morphlines/tutori
alReadJsonTestTweets.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 100 \
  --input-list -

# Go live by merging resulting index shards into a live Solr cluster
# (explicitly specify Solr URLs - for a SolrCloud cluster see next example):
sudo -u hdfs hadoop \
```

```
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file ../search-core/src/test/resources/test-morphlines/tutoria
lReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shard-url http://solr001.mycompany.com:8983/solr/collection1 \
  --shard-url http://solr002.mycompany.com:8983/solr/collection1 \
  --go-live \
  hdfs:///user/foo/indir

# Go live by merging resulting index shards into a live SolrCloud cluster
# (discover shards and Solr URLs through ZooKeeper):
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file ../search-core/src/test/resources/test-morphlines/tutor
ialReadAvroContainer.conf \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --zk-host zk01.mycompany.com:2181/solr \
  --collection collection1 \
  --go-live \
  hdfs:///user/foo/indir

# MapReduce on Yarn - Pass custom JVM arguments (including a custom tmp dire
ctory)
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000 -
Djava.io.tmpdir=/my/tmp/dir/'; \
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 \
  -D 'mapreduce.map.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnection
s=10000' \
  -D 'mapreduce.reduce.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConne
ctions=10000' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file ../search-core/src/test/resources/test-morphlines/tutoria
lReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro
```

Indexing data with MapReduceIndexerTool in Solr backup format

MapReduceIndexerTool (MRIT) is capable of batch indexing a dataset and provide the output in the format of Solr backups, using morphlines. This backup can then be ingested into Solr using a backup opration.

About this task

The MapReduceIndexerTool (MRIT) backup format feature addresses the dilemma of ingesting indexes produced by MRIT jobs into Solr:

- Near-real-time (NRT) ingestion using the --go-live option is resource-intensive and involves merging indexes.
- Batch indexing requires shutting down the Solr server.

MRIT backup format takes the best of both worlds: by creating the index in the Solr backup format, it can be ingested into Solr as a restore operation, using the solrctl command line utility. This method is significantly less resource intensive on the part of Solr compared to NRT with --go-live. Restoring the backup results in a new collection which can be queried directly or put behind an alias.

Procedure

1. To perform a batch indexing job on MRIT with the output in Solr backup format, run the following command:

```
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/mr/
search-mr-*-job.jar  org.apache.solr.hadoop.MapReduceIndexe
rTool --morphline-file [***MORPHLINE_FILE***]  --output-dir
 "[***ABSOLUTE/PATH/TO/OUTPUT/DIRECTORY***]" --use-backup-format
 --backup-name [***USER_SPECIFIED_NAME_FOR_THE_BACKUP***] --zk-ho
st [***HOSTNAME***]:2181/solr --collection [***COLLECTION_NAME***]
 "[***ABSOLUTE/PATH/TO/INPUT/FILE***]"
```

   Replace *[***MORPHLINE_FILE***]*, *[***ABSOLUTE/PATH/TO/OUTPUT/DIRECTORY***]*, *[***USER_SPECIFIED_NAME_FOR_THE_BACKUP***]*, *[***HOSTNAME***]*, *[***COLLECTION_NAME***]*, and *[***ABSOLUTE/PATH/TO/INPUT/FILE***]* with values applicable in your environment.

   For example:

   To parse the contents of hdfs://ns1:8020/tmp/inputfile using the morphline file morphlines.conf and write the resulting index to hdfs://ns1:8020/tmp/output/results/backupName:

```
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job
.jar  org.apache.solr.hadoop.MapReduceIndexerTool --morphline-file morph
lines.conf  --output-dir "hdfs://ns1:8020/tmp/output" --use-backup-forma
t --backup-name backupName --zk-host zk-server:2181/solr --collection co
llection "hdfs://ns1:8020/tmp/inputfile"
```

2. To create a new collection with the contents of the backup:

```
solrctl collection --restore [***USER_DEFINED_COLLECTION_NAME***] -
b [***NAME_OF_THE_INDEX_IN_BACKUP_FORMAT***] -l [***ABSOLUTE/PATH/TO/
RESTORE/TARGET/DIRECTORY***] -i [***REQUEST_ID***]
```

   Make sure that you use a unique <*REQUESTID*> each time you run this command.

   Note:

   Statuses of historic job runs are stored in ZooKeeper and can be retrieved using the solrctl collection --request-status *[***REQUEST_ID***]* command. The number of async call responses stored in a cluster is limited to 10,000.

   Status information can be removed from ZooKeeper using the DELETESTATUS API call.

   Replace *[***USER_DEFINED_COLLECTION_NAME***]*, *[***NAME_OF_THE_INDEX_IN_BACKUP_FORMAT***]*, *[***ABSOLUTE/PATH/TO/RESTORE/TARGET/DIRECTORY***]* with values applicable in your environment.

   For example:

   To create the collection finalcollectionName from the backup backupName to the directory hdfs://ns1:8020/tmp/output/results with the request ID 1234:

```
solrctl collection --restore finalcollectionName -b backupName -l hdfs://
ns1:8020/tmp/output/results -i 1234
```

3. To monitor the status of the restore step, run the following command:

```
solrctl collection --request-status [***REQUEST_ID***]
```

Replace *[***REQUEST_ID***]* with the ID of the task you want to monitor.

For example:

```
solrctl collection --request-status 1234
```

Related Information

Collection aliasing

Asynchronous calls

Lily HBase batch indexing for Cloudera Search

You can batch index HBase tables using the Lily HBase batch indexer MapReduce job (HBaseMapReduceIndexerTool). This batch indexing does not require HBase replication or the Lily HBase Indexer Service. Subsequently you do not need to register a Lily HBase Indexer configuration with the Lily HBase Indexer Service.

The indexer supports flexible, custom, application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain columnFamily:qualifier links back to the data stored in HBase. This way, applications can use the search result set to directly access matching raw HBase cells.

The following procedures demonstrate creating a small HBase table and using the HBaseMapReduceIndexerTool to index the table into a collection:

Important: Do not use the Lily HBase Batch Indexer during a rolling upgrade. The indexer requires all replicas be hosted on the same HBase version. If an indexing job is running during a rolling upgrade, different nodes may be running pre- and post-upgrade versions of HBase.

Populating an HBase Table

Procedure

After configuring and starting your system, create an HBase table and add rows to it. For example:

```
hbase shell
hbase(main):002:0> create 'sample_table', {NAME => 'data'}
hbase(main):002:0> put 'sample_table', 'row1', 'data', 'value'
hbase(main):001:0> put 'sample_table', 'row2', 'data', 'value2'
```

Create a Collection in Cloudera Search

About this task

A collection in Search used for HBase indexing must have a Solr schema that accommodates the types of HBase column families and qualifiers that are being indexed. To begin, consider adding the all-inclusive data field to a default schema.

Procedure

Once you decide on a schema, create a collection using commands similar to the following:

```
solrctl instancedir --generate $HOME/hbase_collection_config
## Edit $HOME/hbase_collection_config/conf/schema.xml as needed ##
solrctl config --upload hbase_collection_config $HOME/hbase_collection_confi
g
```

```
solrctl collection --create hbase_collection -s <numShards> -c hbase_colle
ction_config
```

Creating a Lily HBase Indexer Configuration File

About this task

Configure individual Lily HBase Indexers using the hbase-indexer command-line utility. Typically, there is one Lily HBase Indexer configuration file for each HBase table, but there can be as many Lily HBase Indexer configuration files as there are tables, column families, and corresponding collections in Search. Each Lily HBase Indexer configuration is defined in an XML file, such as morphline-hbase-mapper.xml.

An indexer configuration XML file must refer to the MorphlineResultToSolrMapper implementation and point to the location of a Morphline configuration file, as shown in the following morphline-hbase-mapper.xml indexer configuration file.

Procedure

- Set morphlineFile to the relative path morphlines.conf. Make sure the file is readable by the HBase system user (hbase by default).

```
$ cat $HOME/morphline-hbase-mapper.xml

<?xml version="1.0"?>
<indexer table="sample_table"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">
   <!-- The relative path on the local file system to the
   morphline configuration file. -->

   <param name="morphlineFile" value="morphlines.conf"/>

   <!-- The optional morphlineId identifies a morphline if there are multi
ple
   morphlines in morphlines.conf -->
   <!-- <param name="morphlineId" value="morphline1"/> -->
</indexer>
```

  The Lily HBase Indexer configuration file also supports the standard attributes of any HBase Lily Indexer on the top-level <indexer> element. It does not support the <field> element and <extract> elements.

Creating a Morphline Configuration File

About this task

After creating an indexer configuration XML file, you can configure morphline ETL transformation commands in a morphlines.conf configuration file. The morphlines.conf configuration file can contain any number of morphline commands. Typically, an extractHBaseCells command is the first command. The readAvroContainer or readAvro morphline commands are often used to extract Avro data from the HBase byte array. This configuration file can be shared among different applications that use morphlines.

Note: To function properly, the morphline must not contain a loadSolr command. The Lily HBase Indexer must load documents into Solr, instead of the morphline itself.

Procedure

You can edit the morphlines.conf file within Cloudera Manager (Key-Value Store Indexer service Configuration Category Morphlines Morphlines File).

Understanding the extractHBaseCells Morphline Command

The extractHBaseCells morphline command extracts cells from an HBase result and transforms the values into a Solr InputDocument. The command consists of an array of zero or more mapping specifications.

Each mapping has:

- The inputColumn parameter, which specifies the data from HBase for populating a field in Solr. It has the form of a column family name and qualifier, separated by a colon. The qualifier portion can end in an asterisk, which is interpreted as a wildcard. In this case, all matching column-family and qualifier expressions are used. The following are examples of valid inputColumn values:
  - mycolumnfamily:myqualifier
  - mycolumnfamily:my*
  - mycolumnfamily:*
- The outputField parameter specifies the morphline record field to which to add output values. The morphline record field is also known as the Solr document field. Example: first_name.
- Dynamic output fields are enabled by the outputField parameter ending with a wildcard (*). For example:

  ```
  inputColumn : "mycolumnfamily:*"
  outputField : "belongs_to_*"
  ```

  In this case, if you make these puts in HBase:

  ```
  put 'table_name' , 'row1' , 'mycolumnfamily:1' , 'foo'
  put 'table_name' , 'row1' , 'mycolumnfamily:9' , 'bar'
  ```

  Then the fields of the Solr document are as follows:

  ```
  belongs_to_1 : foo
  belongs_to_9 : bar
  ```

- The type parameter defines the data type of the content in HBase. All input data is stored in HBase as byte arrays, but all content in Solr is indexed as text, so a method for converting byte arrays to the actual data type is required. The type parameter can be the name of a type that is supported by org.apache.hadoop.hbase.util.Bytes.to* (which currently includes byte[], int, long, string, boolean, float, double, short, and bigdecimal). Use type byte[] to pass the byte array through to the morphline without conversion.
  - type:byte[] copies the byte array unmodified into the record output field
  - type:int converts with org.apache.hadoop.hbase.util.Bytes.toInt
  - type:long converts with org.apache.hadoop.hbase.util.Bytes.toLong
  - type:string converts with org.apache.hadoop.hbase.util.Bytes.toString
  - type:boolean converts with org.apache.hadoop.hbase.util.Bytes.toBoolean
  - type:float converts with org.apache.hadoop.hbase.util.Bytes.toFloat
  - type:double converts with org.apache.hadoop.hbase.util.Bytes.toDouble
  - type:short converts with org.apache.hadoop.hbase.util.Bytes.toShort
  - type:bigdecimal converts with org.apache.hadoop.hbase.util.Bytes.toBigDecimal

  Alternatively, the type parameter can be the name of a Java class that implements the com.ngdata.hbaseindexer.parse.ByteArrayValueMapper interface.

  HBase data formatting does not always match what is specified by org.apache.hadoop.hbase.util.Bytes.*. For example, this can occur with data of type float or double. You can enable indexing of such HBase data by converting the data. There are various ways to do so, including:

  - Using Java morphline command to parse input data, converting it to the expected output. For example:

    ```
    {
     imports : "import java.util.*;" code: """ // manipulate the contents of
     a record field
     String stringAmount = (String) record.getFirstValue("amount");
    ```

```
 Double dbl = Double.parseDouble(stringAmount); record.replaceValues("
amount",dbl);
 return child.process(record); // pass record to next command in chain
 """
}
```

- Creating table fields with binary format and then using types such as double or float in a morphline.conf. You could create a table in HBase for storing doubles using commands similar to:

```
CREATE TABLE sample_lily_hbase ( id string, amount double, ts timestamp
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,ti:amount#b,ti:
ts,')
TBLPROPERTIES ('hbase.table.name' = 'sample_lily');
```

- The source parameter determines which portion of an HBase KeyValue is used as indexing input. Valid choices are value or qualifier. When value is specified, the HBase cell value is used as input for indexing. When qualifier is specified, then the HBase column qualifier is used as input for indexing. The default is value.

Running the HBaseMapReduceIndexerTool

HBaseMapReduceIndexerTool is a MapReduce batch job driver that takes input data from an HBase table, creates Solr index shards, and writes the indexes to HDFS in a flexible, scalable, and fault-tolerant manner. It also supports merging the output shards into a set of live customer-facing Solr servers in SolrCloud.

About this task

Important: Merging output shards into live customer-facing Solr servers can only be completed if all replicas are online.

Before you begin

Important: You must run the indexer tool with the following command-line argument:

```
-D 'mapreduce.job.user.classpath.first=true'
```

Running the tool without this argument triggers the following error:

```
ERROR [main] org.apache.hadoop.mapred.YarnChild: Error running child :
 java.lang.NoSuchMethodError:
com.codahale.metrics.MetricRegistry.meter(Ljava/lang/String;Lcom/cod
ahale/metrics/MetricRegistry$MetricSupplier;)Lcom/codahale/metrics/M
eter;
```

Procedure

- Run the command as follows:

```
hadoop --config /etc/hadoop/conf \
jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-*-job
.jar \
--conf /etc/hbase/conf/hbase-site.xml -D 'mapreduce.job.user.classpath.fi
rst=true' \
-Dmapreduce.map.java.opts="-Xmx512m" -Dmapreduce.reduce.java.opts="-Xmx5
12m" \
--hbase-indexer-file $HOME/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr --collection hbase-collection1 \
```

```
--go-live --log4j src/test/resources/log4j.properties
```

Note: For development purposes, use the --dry-run option to run in local mode and print documents to stdout, instead of loading them to Solr. Using this option causes the morphline to run in the client process without submitting a job to MapReduce. Running in the client process provides quicker results during early trial and debug sessions.

To print diagnostic information, such as the content of records as they pass through morphline commands, enable TRACE log level diagnostics by adding the following to your log4j.properties file:

```
log4j.logger.org.kitesdk.morphline=TRACE
log4j.logger.com.ngdata=TRACE
```

The log4j.properties file can be passed using the --log4j command-line option.

To invoke the command-line help, use:

```
hadoop jar /opt/cloudera/parcels/CDH/jars/hbase-indexer-mr-*-job.jar --h
elp
```

Related Information

HBaseMapReduceIndexerTool command line reference

HBaseMapReduceIndexerTool command line reference

Command line syntax, examples and list of parameters.

The general command line syntax is:

```
command [genericOptions] [commandOptions]
```

usage:

```
hadoop [GenericOptions]... jar hbase-indexer-mr-*-job.jar
       [--hbase-indexer-zk STRING] [--hbase-indexer-name STRING]
       [--hbase-indexer-file FILE]
       [--hbase-indexer-component-factory STRING]
       [--hbase-table-name STRING] [--hbase-start-row BINARYSTRING]
       [--hbase-end-row BINARYSTRING] [--hbase-start-time STRING]
       [--hbase-end-time STRING] [--hbase-timestamp-format STRING]
       [--zk-host STRING] [--go-live] [--collection STRING]
       [--go-live-min-replication-factor INTEGER]
       [--go-live-threads INTEGER] [--help] [--output-dir HDFS_URI]
       [--overwrite-output-dir] [--morphline-file FILE]
       [--morphline-id STRING] [--solr-home-dir DIR]
       [--update-conflict-resolver FQCN] [--reducers INTEGER]
       [--max-segments INTEGER] [--fair-scheduler-pool STRING] [--dry-run]
       [--log4j FILE] [--verbose] [--clear-index] [--show-non-solr-cloud]
```

Examples:

(Re)index a table in GoLive mode based on a local indexer config file:

```
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.job.user.classpath.first=true' \
  -Dmapreduce.map.java.opts="-Xmx512m" \
  -Dmapreduce.reduce.java.opts="-Xmx512m" \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
```

```
  --go-live \
  --log4j src/test/resources/log4j.properties
```

(Re)index a table in GoLive mode using a local morphline-based indexer config file. Also include extra library jar file containing JSON tweet Java parser:

```
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  --libjars /path/to/kite-morphlines-twitter-0.10.0.jar \
  -D 'mapreduce.job.user.classpath.first=true' \
  -Dmapreduce.map.java.opts="-Xmx512m" \
  -Dmapreduce.reduce.java.opts="-Xmx512m" \
  --hbase-indexer-file src/test/resources/morphline_indexer_without_zk.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --go-live \
  --morphline-file src/test/resources/morphlines.conf \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --overwrite-output-dir \
  --log4j src/test/resources/log4j.properties
```

(Re)index a table in GoLive mode:

```
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.job.user.classpath.first=true' \
  -Dmapreduce.map.java.opts="-Xmx512m" \
  -Dmapreduce.reduce.java.opts="-Xmx512m" \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --go-live \
  --log4j src/test/resources/log4j.properties
```

(Re)index a table with direct writes to SolrCloud:

```
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.job.user.classpath.first=true' \
  -Dmapreduce.map.java.opts="-Xmx512m" \
  -Dmapreduce.reduce.java.opts="-Xmx512m" \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --reducers 0 \
  --log4j src/test/resources/log4j.properties
```

(Re)index a table based on a indexer config stored in ZK:

```
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.job.user.classpath.first=true' \
  -Dmapreduce.map.java.opts="-Xmx512m" \
  -Dmapreduce.reduce.java.opts="-Xmx512m" \
  --hbase-indexer-zk zk01 \
  --hbase-indexer-name docindexer \
  --go-live \
```

```
   --log4j src/test/resources/log4j.properties
```

MapReduce on Yarn - Pass custom JVM arguments:

```
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'; \
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.map.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnection
s=10000' \
  -D 'mapreduce.reduce.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConne
ctions=10000' \
  --hbase-indexer-zk zk01 \
  --hbase-indexer-name docindexer \
  --go-live \
  --log4j src/test/resources/log4j.properties
```

HBase Indexer Parameters

Parameters for specifying the HBase indexer definition and/or where it should be loaded from.

Table 2: HBase Indexer parameters

Parameter	Type	Description	Example
--hbase-indexer-zk	STRING	The address of the ZooKeeper ensemble from which to fetch the indexer definition named --hbase-indexer-name. Format is: a list of comma separated host:port pairs, each corresponding to a zk server.	'127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183'
--hbase-indexer-name	STRING	The name of the indexer configuration to fetch from the ZooKeeper ensemble specified with --hbase-indexer-zk.	myIndexer
--hbase-indexer-file	FILE	Relative or absolute path to a local HBase indexer XML configuration file. If supplied, this overrides --hbase-indexer-zk and --hbase-indexer-name.	/path/to/morphline-hbase-mapper.xml
--hbase-indexer-component-factory	STRING	Classname of the hbase indexer component factory.	

HBase Scan Parameters

Parameters for specifying what data is included while reading from HBase.

Table 3: HBase scan parameters

Parameter	Type	Description	Example
--hbase-table-name	STRING	Optional name of the HBase table containing the records to be indexed. If supplied, this overrides the value from the --hbase-indexer-* options.	myTable

Parameter	Type	Description	Example
--hbase-start-row	BINARYSTRING	Binary string representation of start row from which to start indexing (inclusive). The format of the supplied row key should use two-digit hex values prefixed by \x for non-ascii characters (e.g. 'row\x00'). The semantics of this argument are the same as those for the HBase Scan#setStartRow method. The default is to include the first row of the table.	AAAA
--hbase-end-row	BINARYSTRING	Binary string representation of end row prefix at which to stop indexing (exclusive). See the description of --hbase-start-row for more information. The default is to include the last row of the table.	CCCC
--hbase-start-time	STRING	Earliest timestamp (inclusive) in time range of HBase cells to be included for indexing. The default is to include all cells.	0
--hbase-end-time	STRING	Latest timestamp (exclusive) of HBase cells to be included for indexing. The default is to include all cells.	123456789
--hbase-timestamp-format	STRING	Timestamp format to be used to interpret --hbase- start-time and --hbase-end-time. This is a java .text.SimpleDateFormat compliant format. If this parameter is omitted then the timestamps are interpreted as number of milliseconds since the standard epoch (Unix time).	yyyy-MM-dd'T'HH:mm:ss.SSSZ

Solr Cluster Arguments

Arguments that provide information about your Solr cluster.

Table 4: Solr cluster arguments

Argument	Type	Description	Example
--zk-host	STRING	The address of a ZooKeeper ensemble being used by a SolrCloud cluster. This ZooKeeper ensemble will be examined to determine the number of output shards to create as well as the Solr URLs to merge the output shards into when using the --go-live option. Requires that you also pass the --collection to merge the shards into. The format is a list of comma separated host:port pairs, each corresponding to a zk server. The --zk-host option implements the same partitioning semantics as the standard SolrCloud Near-Real-Time (NRT) API. This enables to mix batch updates from MapReduce ingestion with updates from standard Solr NRT ingestion on the same SolrCloud cluster, using identical unique document keys. If --solr-home-dir is not specified, the Solr home directory for the collection will be downloaded from this ZooKeeper ensemble.	'127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183' If the optional chroot suffix is used the example would look like: '127.0.0.1:2181/solr127.0.0.1:2182/solr127.0.0.1:2183/solr' where the client would be rooted at '/solr' and all paths would be relative to this root - i.e.: getting/setting/etc... '/foo/bar' would result in operations being run on '/solr/foo/bar' (from the server perspective).
--solr-client-socket-timeout	INTEGER	Solr socket timeout in milliseconds This optional argument overwrites the default 10 minute socket timeout in HBase indexer for the direct writing mode (when the value of the --reducers optional argument is set to 0 and mappers directly send the data to the live Solr). Default value: 10000	

Go Live Arguments

Arguments for merging the shards that are built into a live Solr cluster. Also see the Cluster arguments.

Table 5: Go live arguments

Argument	Type	Description	Example
--go-live		Allows you to optionally merge the final index shards into a live Solr cluster after they are built. You can pass the ZooKeeper address with --zk-host and the relevant cluster information will be auto detected. (default: false)	
--collection	STRING	The SolrCloud collection to merge shards into when using --go-live and --zk-host.	collection1

Argument	Type	Description	Example
--go-live-min-replication-factor	INTEGER	The minimum number of SolrCloud replicas to successfully merge any final index shard into. The go-live job phase attempts to merge final index shards into all SolrCloud replicas. Some of these merge operations may fail, for example if some SolrCloud servers are down. This option enables indexing jobs to succeed even if some such merge operations fail on SolrCloud followers. Successful merge operations into all leaders are always required for job success, regardless of the value of --go -live-min- replication-factor. -1 indicates require successful merge operations into all replicas. 1 indicates require successful merge operations only into leader replicas. (default: -1)	
--go-live-threads	INTEGER	Tuning knob that indicates the maximum number of live merges to run in parallel at one time. (default: 1000)	

Table 6: Optional arguments

Argument	Type	Description	Example
--help -help -h		Show the help message and exit	
--output-dir	HDFS_URI	HDFS directory to write Solr indexes to. Inside there one output directory per shard will be generated.	hdfs://c2202.mycompany. com/ user/$USER/test
--overwrite-output-dir		Overwrite the directory specified by --output-dir if it already exists. Using this parameter will result in the output directory being recursively deleted at job startup. (default: false)	
--morphline-file	FILE	Relative or absolute path to a local config file that contains one or more morphlines. The file must be UTF-8 encoded. The file will be uploaded to each MR task. If supplied, this overrides the value from the --hbase-indexer-* options.	/path/to/morphlines.conf

Argument	Type	Description	Example
--morphline-id	STRING	The identifier of the morphline that shall be executed within the morphline config file, e.g. specified by --morphline-file. If the --morphline- id option is ommitted the first (i.e. top-most) morphline within the config file is used. If supplied, this overrides the value from the --hbase-inde xer-* options.	morphline1
--solr-home-dir	DIR	Optional relative or absolute path to a local dir containing Solr conf/ dir and in particular conf /solrconfig.xml and optionally also lib/ dir. This directory will be uploaded to each MR task.	src/test/resources/solr/minimr
--update-conflict-resolver	FQCN	Fully qualified class name of a Java class that implements the UpdateConflictResolver interface. This enables deduplication and ordering of a series of document updates for the same unique document key. For example, a MapReduce batch job might index multiple files in the same job where some of the files contain old and new versions of the very same document, using the same unique document key. Typically, implementations of this interface forbid collisions by throwing an exception, or ignore all but the most recent document version, or, in the general case, order colliding updates ascending from least recent to most recent (partial) update. The caller of this interface (i. e. the Hadoop Reducer) will then apply the updates to Solr in the order returned by the orderUpdates() method. The default RetainMostRe centUpdateConflictResolver implementation ignores all but the most recent document version, based on a configurable numeric Solr field, which defaults to the file_last_modified timestamp. (default: org.apache. solr.hadoop. dedup. RetainMostRecentU pdateConflictResolver)	

Argument	Type	Description	Example
--reducers	INTEGER	Tuning knob that indicates the number of reducers to index into. • 0 indicates that no reducers should be used, and documents should be sent directly from the mapper tasks to live Solr servers. • -1 indicates use all reduce slots available on the cluster. • -2 indicates use one reducer per output shard, which disables the mtree merge MR algorithm. The mtree merge MR algorithm improves scalability by spreading load (in particular CPU load) among a number of parallel reducers that can be much larger than the number of solr shards expected by the user. It can be seen as an extension of concurrent lucene merges and tiered lucene merges to the clustered case. The subsequent mapper-only phase merges the output of said large number of reducers to the number of shards expected by the user, again by utilizing more available parallelism on the cluster. (default: -1)	
--max-segments	INTEGER	Tuning knob that indicates the maximum number of segments to be contained on output in the index of each reducer shard. After a reducer has built its output index it applies a merge policy to merge segments until there are <= maxSegments lucene segments left in this index. Merging segments involves reading and rewriting all data in all these segment files, potentially multiple times, which is very I/O intensive and time consuming. However, an index with fewer segments can later be merged faster, and it can later be queried faster once deployed to a live Solr serving shard. Set maxSegments to 1 to optimize the index for low query latency. In a nutshell, a small maxSegments value trades indexing latency for subsequently improved query latency. This can be a reasonable trade-off for batch indexing systems. (default: 1)	

Argument	Type	Description	Example
--dry-run		Run in local mode and print documents to stdout instead of loading them into Solr. This executes the morphline in the client process (without submitting a job to MR) for quicker turnaround during early trial and debug sessions. (default: false)	
--log4j	FILE	Relative or absolute path to a log4j.properties config file on the local file system. This file will be uploaded to each MR task.	/path/to/log4j.properties
--verbose -v		Turn on verbose output. (default: false)	
--clear-index		Will attempt to delete all entries in a solr index before starting batch build. This is not transactional so if the build fails the index will be empty. (default: false)	
--show-non-solr-cloud		Also show options for Non-SolrCloud mode as part of --help. (default: false)	

Supported Generic Options

The following generic options are supported:

Table 7: Supported generic options

Option	Description
--conf <configuration file>	Specify an application configuration file.
-D <property=value>	Define a value for a given property.
-fs <file:///|hdfs://namenode:port>	Specify default filesystem URL to use, overrides the fs.defaultFS property from configurations.
--jt <local|resourcemanager:port>	Specify a ResourceManager.
--files <file1,...>	Specify a comma-separated list of files to be copied to the map reduce cluster.
--libjars <jar1,...>	Specify a comma-separated list of jar files to be included in the classpath.
--archives <archive1,...>	Specify a comma-separated list of archives to be unarchived on the compute machines.

Related Information

Java SimpeDateFormat

Using --go-live with SSL or Kerberos

Establish trust between the indexer client and Solr server(s).

About this task

The go-live phase of the indexer jobs sends a MERGEINDEXES request from the indexer client (the node from which the MR job was submitted) to the live Solr servers. If the Solr server has SSL enabled, you need to ensure that the indexer client trusts the certificate presented by the Solr server(s), otherwise you get an SSLPeerUnverifiedExc eption.

Procedure

1. Specify the location of the trust store by setting the following HADOOP_OPTS variable before launching the indexer job:

```
HADOOP_OPTS="-Djavax.net.ssl.trustStore=/etc/cdep-ssl-conf/CA_STANDARD/t
ruststore.jks "
```

2. If the Solr servers have Kerberos authentication enabled, you need to ensure that the indexer client can authenticate via Kerberos to the Solr servers. For this, you need to create a Java Authentication and Authorization Service configuration (JAAS) file locally on the node where the indexing job is launched:
   - If you are authenticating using kinit to obtain credentials, you can configure the client to use your credential cache by creating a jaas.conf file with the following contents:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 principal="<USER>@EXAMPLE.COM";
 };
```

     Replace *<USER>* with your username, and *EXAMPLE.COM* with your Kerberos realm.
   - If you want the client application to authenticate using a keytab, modify jaas-client.conf as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/PATH/TO/USER.KEYTAB"
 storeKey=true
 useTicketCache=false
 principal="<USER>@EXAMPLE.COM";
};
```

     Replace */PATH/TO/USER.KEYTAB* with the keytab file you want to use and *<USER>@EXAMPLE.COM* with the principal in the keytab. If you are using a service principal that includes the hostname, make sure that it is included in the jaas.conf file (for example, solr/solr01.example.com@EXAMPLE.COM).
3. If you are using a ticket cache, you need to do a kinit to acquire a ticket for the configured principal before launching the indexer.
4. Specify the authentication configuration in the HADOOP_OPTS environment variable:

```
HADOOP_OPTS="-Djava.security.auth.login.config=jaas.conf -Djavax.net.ssl
.trustStore=/etc/cdep-ssl-conf/CA_STANDARD/truststore.jks" \
hadoop --config /etc/hadoop/conf \
jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-*-jo
b.jar \
--conf /etc/hbase/conf/hbase-site.xml -Dmapreduce.map.java.opts="-Xmx512m"
 -Dmapreduce.reduce.java.opts="-Xmx512m" \
--hbase-indexer-file /home/systest/hbasetest/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr \
--collection hbase-collection1 \
```

```
--go-live --log4j src/test/resources/log4j.properties
```

Note:

Communication to Solr servers is only occurring in the go-live phase, not from the MapReduce jobs. Therefore it is enough to place the jaas.conf and the SSL trust store on the node from which the indexer client is started as it will be the one that communicates to Solr.

Understanding --go-live and HDFS ACLs

When run with a reduce phase, as opposed to as a mapper-only job, the indexer creates an offline index on HDFS in the output directory specified by the --output-dir parameter. If the --go-live parameter is specified, Solr merges the resulting offline index into the live running service. Thus, the Solr service must have read access to the contents of the output directory in order to complete the --go-live step. If --overwrite-output-dir is specified, the indexer deletes and recreates any existing output directory; in an environment with restrictive permissions, such as one with an HDFS umask of 077, the Solr user may not be able to read the contents of the newly created directory. To address this issue, the indexer automatically applies the HDFS ACLs to enable Solr to read the output directory contents. These ACLs are only applied if HDFS ACLs are enabled on the HDFS NameNode.

The indexer only makes ACL updates to the output directory and its contents. If the output directory's parent directories do not include the execute permission, the Solr service cannot access the output directory. Solr must have execute permissions from standard permissions or ACLs on the parent directories of the output directory.