CCP Tuning Guide 2.0.1

Use Case Specific Tuning Suggestions

Date of publish: 2017-11-06

https://docs.cloudera.com/

Legal Notice

Contents

Use Case Specific Tuning Suggestions

The following discussion outlines a specific tuning exercise we went through for driving 1 Gbps of traffic through a Metron cluster running with 4 Kafka brokers and 4 Storm Supervisors.

General machine specs:

- 10 GB network cards
- 256 GB memory
- 12 disks
- 32 cores

Performance Monitoring Tools

Before we get to tuning our cluster, it helps to describe what we might actually want to monitor as well as any potential pain points.

Prior to switching over to the new Storm Kafka client, which leverages the new Kafka consumer API under the hood, offsets were stored in ZooKeeper. While the broker hosts are still stored in ZooKeeper, this is no longer true for the offsets which are now stored in Kafka itself. This is a configurable option, and you may switch back to ZooKeeper if you choose, but Metron is currently using the new defaults. With this in mind, there are some useful tools that come with Storm and Kafka that we can use to monitor our topologies.

Kafka

- Consumer group offset lag viewer
- There is a GUI tool to make creating, modifying, and generally managing your Kafka topics a bit easier - see kafka-manager
- Console consumer - useful for quickly verifying topic contents

Storm

For more information on the Storm user interface, see Reading and Understanding the Storm UI.

View Kafka Offset Lags Example

You can use the Kafka consumer group offset lag viewer to monitor the delta calculations between the current and end offset for a partition.

Procedure

1. Set up some environment variables.

```
export BROKERLIST  your broker comma-delimated list of host:ports>
export ZOOKEEPER  your zookeeper comma-delimated list of host:ports>
export KAFKA_HOME  kafka home dir>
export METRON_HOME  your metron home>
export HDP_HOME  your HDP home>
```

2. If you have Kerberos enabled, set up the security protocol.

```
$ cat /tmp/consumergroup.config
security.protocol=SASL_PLAINTEXT
```

3. Enter the following command to display a table containing offsets for all partitions and consumers associated with a running topology's consumer group:

   ${KAFKA_HOME}/bin/kafka-consumer-groups.sh \ --command-config=/tmp/consumergroup.config \ --describe \ --group enrichments \ --bootstrap-server $BROKERLIST \ --new-consumer

The command displays the following table:

```
GROUP                          TOPIC              PARTITION  CURRENT-
OFFSET  LOG-END-OFFSET  LAG             OWNER
enrichments                    enrichments        9          29746066
   29746067        1               consumer-2_/xxx.xxx.xxx.xxx
enrichments                    enrichments        3          29754325
   29754326        1               consumer-1_/xxx.xxx.xxx.xxx
enrichments                    enrichments        43         29754331
   29754332        1               consumer-6_/xxx.xxx.xxx.xxx
...
```

Note: Output displays only when the topology is running because the consumer groups only exist while consumers in the spouts are up and running.

The LAG column lists the current delta calculation between the current and end offset for the partition. The column value indicates how close you are to keeping up with incoming data. It also indicates whether there are any problems with specific consumers getting stuck.

4. To watch the offsets and lags change over time, add a watch command and set the refresh rate to 10 seconds:

```
watch -n 10 -d ${KAFKA_HOME}/bin/kafka-consumer-groups.sh \
    --command-config=/tmp/consumergroup.config \
    --describe \
    --group enrichments \
    --bootstrap-server $BROKERLIST \
    --new-consumer
```

The watch command runs every 10 seconds and refreshes the screen with new information. The command also highlights the differences from the current output and the last output screens.

Parser Tuning Example

We'll be using the Bro sensor in this parser tuning example.

We started with a single partition for the inbound Kafka topics and eventually worked our way up to 48 partitions. And we're using the following pending value, as shown below. The default is 'null' which would result in no limit.

storm-bro.config

```
{
    ...
    "topology.max.spout.pending" : 2000
    ...
}
```

And the following default spout settings. Again, this can be omitted entirely since we are using the defaults.

spout-bro.config

```
{
    ...

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

And we ran our Bro parser topology with the following options. We did not need to fully match the number of Kafka partitions with our parallelism in this case, though you could certainly do so if necessary. Notice that we only needed 1 worker.

```
/$METRON_HOME/bin/start_parser_topology.sh -k $BROKERLIST -z $ZOOKEEPER -s
bro -ksp SASL_PLAINTEXT
    -ot enrichments
    -e ~metron/.storm/storm-bro.config \
    -esc ~/.storm/spout-bro.config \
    -sp 24 \
    -snt 24 \
    -nw 1 \
    -pnt 24 \
    -pp 24 \
```

From the usage docs, here are the options we've used.

```
 -e,--extra_topology_options (JSON_FILE)       Extra options in the form
                                               of a JSON file with a map
                                               for content.
 -esc,--extra_kafka_spout_config (JSON_FILE)   Extra spout config options
                                               in the form of a JSON file
                                               with a map for content.
                                               Possible keys are:
                                               retryDelayMaxMs,retryDelay

 Multiplier,retryInitialDelayMs,stateUpdateIntervalMs,

 bufferSizeBytes,fetchMaxWait,fetchSizeBytes,maxOffswt

 Behind,metricsTimeBucketSizeInSecs,socketTimeoutMs
-sp,--spout_p (SPOUT_PARALLELISM_HINT)         Spout Parallelism Hint
-snt,--spout_num_tasks (NUM_TASKS)             Spout Num Tasks
-nw,--num_workers (NUM_WORKERS)                Number of Workers
-pnt,--parser_num_tasks (NUM_TASKS)            Parser Num Tasks
-pp,--parser_p (PARALLELISM_HINT)              Parser Parallelism Hint
```

Enrichment Tuning Example

We landed on the same number of partitions for enrichment and indexing as we did for bro - 48.

For configuring Storm, there is a flux file and properties file that we modified. Here are the settings we changed for Bro in Flux. +Note that the main Metron-specific option we've changed to accommodate the desired rate of data throughput is max cache size in the join bolts.

More information on Flux can be found here - https://storm.apache.org/releases/1.1.0/flux.html

general storm settings

```
topology.workers: 8
topology.acker.executors: 48
topology.max.spout.pending: 2000
```

Spout and Bolt Settings

```
kafkaSpout
```

```
     parallelism=48
     session.timeout.ms=29999
     enable.auto.commit=false
     setPollTimeoutMs=200
     setMaxUncommittedOffsets=10000000
     setOffsetCommitPeriodMs=30000
 enrichmentSplitBolt
     parallelism=4
 enrichmentJoinBolt
     parallelism=8
     withMaxCacheSize=200000
     withMaxTimeRetain=10
 threatIntelSplitBolt
     parallelism=4
 threatIntelJoinBolt
     parallelism=4
     withMaxCacheSize=200000
     withMaxTimeRetain=10
 outputBolt
     parallelism=48
```

Indexing (HDFS) Tuning

There are 48 partitions set for the indexing partition, per the previous enrichment exercise.

These are the batch size settings for the Bro index.

```
cat ${METRON_HOME}/config/zookeeper/indexing/bro.json
{
"hdfs" : {
"index": "bro",
    "batchSize": 50,
    "enabled" : true
  }...
}
```

And here are the settings we used for the indexing topology

General storm settings

```
topology.workers: 4
topology.acker.executors: 24
topology.max.spout.pending: 2000
```

Spout and Bolt Settings

```
hdfsSyncPolicy
    org.apache.storm.hdfs.bolt.sync.CountSyncPolicy
    constructor arg=100000
hdfsRotationPolicy
    bolt.hdfs.rotation.policy.units=DAYS
    bolt.hdfs.rotation.policy.count=1
kafkaSpout
    parallelism: 24
    session.timeout.ms=29999
    enable.auto.commit=false
    setPollTimeoutMs=200
    setMaxUncommittedOffsets=10000000
```

```
    setOffsetCommitPeriodMs=30000
hdfsIndexingBolt
    parallelism: 24
```

PCAP Tuning Example

PCAP is a specialized topology that is a Spout-only topology. Both Kafka topic consumption and HDFS writing is done within a spout to avoid the additional network hop required if using an additional bolt.

General Storm topology properties

```
topology.workers=16
topology.ackers.executors: 0

__Spout and Bolt properties__

kafkaSpout
    parallelism: 128
    poll.timeout.ms=100
    offset.commit.period.ms=30000
    session.timeout.ms=39000
    max.uncommitted.offsets=200000000
    max.poll.interval.ms=10
    max.poll.records=200000
    receive.buffer.bytes=431072
    max.partition.fetch.bytes=10000000
    enable.auto.commit=false
    setMaxUncommittedOffsets=20000000
    setOffsetCommitPeriodMs=30000

writerConfig
    withNumPackets=1265625
    withMaxTimeMS=0
    withReplicationFactor=1
    withSyncEvery=80000
    withHDFSConfig
        io.file.buffer.size=1000000
        dfs.blocksize=1073741824
```

Issues

You can run into issues when you tune your system.

```
__Error__

org.apache.kafka.clients.consumer.CommitFailedException: Commit cannot be
 completed since the group has already rebalanced and assigned
the partitions to another member. This means that the time
 between subsequent calls to poll() was longer than the configured
 session.timeout.ms,
which typically implies that the poll loop is spending too much time message
 processing. You can address this either by increasing the
session timeout or by reducing the maximum size of batches returned in
 poll() with max.poll.records
```

Suggestions

This implies that the spout hasn't been given enough time between polls before committing the offsets. In other words, the amount of time taken to process the messages is greater than the timeout window. In

order to fix this, you can improve message throughput by modifying the options outlined above, increasing the poll timeout, or both.