Cloudera Streaming Analytics

Configuration

Date published: 2019-12-17
Date modified: 2019-12-17

https://docs.cloudera.com/

Legal Notice

Contents

How to configure Apache Flink?

Cloudera Streaming Analytics includes Flink with configuration that works out of the box. It is not mandatory to configure Flink to production, but you can use the available configurations to optimize the application behavior in production. Cloudera Manager includes all the necessary configurations for Flink that can also be accessed from the flink-conf.yaml file.

Setting max parallelism

The max parallelism is the most essential part of resource configuration for Flink applications as it defines the maximum jobs that are executed at the same time in parallel instances. However, you can optimize max parallelism in case your production goals differ from the default settings.

In a Flink application, the different tasks are split into several parallel instances for execution. The number of parallel instances for a task is called parallelism. Parallelism can be defined at the operator, client, execution environment and system level. Cloudera recommends setting parallelism to a lower value at the initially and increasing it over time, if the job cannot keep up with the input rate.

To configure the max parallelism, setMaxParallelism is called as it controls the number of key-groups created by the state backends. A key-group is a partition of an operator state. The number of key-groups determines how data is going to be distributed among the parallel operators. If the key-groups are not distributed evenly, the data distribution is also uneven.

Consider the following aspects when setting the max parallelism:

- The number should be large enough to accommodate expected future load increases as this setting cannot be changed without starting from an empty state.
- If P is the selected parallelism for the job, the max parallelism should be divisible by P to get even state distribution.
- Please note that larger max parallelism settings have greater cost on the state backend side, for large scale production jobs benchmarking the size of the state based on the maximum parallelism is useful before changing this parameter.

Based on these criteria, Cloudera recommends setting the max paralellism to factorials or other numbers with a large number of divisors (120, 180, 240, 360, 720, 840, 1260), which will make parallelism tuning easier.

Table 1: Reference values

Stateless	In-memory state	RocksDB state
1 million record / sec / core	100 000 records / sec / core	10 000 records / sec / core

Configuring Flink application resources

Generally, Flink automatically identifies the resources for an application. You can configure the Taskmanager, buffer timeout and high availability for a Flink application to maintain resources in a more efficient way.

YARN automatically kills application containers that use more memory than their allocated limit. To avoid Flink TaskManagers getting killed by YARN use the following calculation to set the size: TM total size = TM Heap + Heap-cutoff. The default heap cutoff is 25% and it is also configurable.

Furthermore, use a YARN queue with preemption disabled to avoid long running jobs being affected when the cluster reaches its capacity limit.

Flink uses network buffers to transfer data from one operator to another. These buffers are filled up with data during the specified time for the timeout. In case of high data rates, the set time is usually never reached. For cases when the data rate is high the throughput can be further increased with setting the buffer timeout to an intentionally higher value due to the characteristics of the TCP channel. However this in turn increases the latency of the pipeline.

Flink on YARN jobs are configured to tolerate a maximum number of failed containers before they terminate. Configure the YARN maximum failed containers setting in proportion to the total parallelism and the expected lifetime of the job.

High Availability is enabled by default in CSA. This eliminates the Job Manager as a single point of failure. You can also tune the application resilience by setting the YARN maximum application attempts, which determines how many times the application will retry in case of failures.

Table 2: Reference values

Configuration	Parameter	Recommended value
TM container memory	-ytm / taskmanager.heap.size	*TM Heap + Heap-cutoff*
Max parallelism	env.setMaxParallelism(num)	*120,720,1260,5040*
Container heap cutoff	containerized.heap-cutoff-ratio	*0.25-0.75*
Buffer timeout	env.setBufferTimeout(millis)	*1-100*
YARN queue	-yqu	*A queue with no preemption*
YARN max failed containers	yarn.maximum-failed-containers	*3*num_containers*
YARN max AM failures	yarn.application-attempts	*3-5*

Configuring RocksDB state backend

You can use RocksDB as a state backend when your Flink streaming application requires a larger state that doesn't fit easily in memory. The RocksDB state backend uses a combination of fast in-memory cache and optimized disk based lookups to manage state.

You can configure the state backend for your streaming application by using the state.backend parameter directly or in Cloudera Manager under the Configuration tab:

You can adjust how much memory RocksDB should use as a cache to increase lookup performance by setting the memory managed fraction of the TaskManagers in Cloudera Manager under the Configuration tab:

The default fraction value is 0.4, but with larger cache requirements you need to increase this value together with the total memory size.

Related Information

Stateful Tutorial: Configure the application for production