Feature 1289 Data Caching Mechanism

- Status: Design
- Author: Jiyong Huang
- Discussion: https://github.com/lf-edge/ekuiper/issues/1289

Motivation

Running on the edge side, it is common to encounter network connection failure. For rules which sink to external systems, especially remote external systems, it is important to cache the data during failures such as network disconnection and resend once reconnected.

Previously, eKuiper has some degree of support for sink caching. It provides a global configuration to switch cache; system/rule level configuration for memory cache serialization interval. However, the cache is only in memory and a copy to the db(mirror of the memory) and do not define clear resend strategy. It is actually part of the failure recovery mechanism (qos). In this proposal, the cache will be saved in both memory and disk so that the cache capacity becomes much bigger; it will also continuously detect failure recovery status and resend without restarting the rule.

Flow

The cache happens only in sink because that is the only place to send data outside eKuiper. Each sink can have its own cache mechanism configured. The flows in each sink is alike. If cache is enabled, all sink events will go through a two phase process: firstly save all to cache; then delete the cache once received an ack.

1. Error detection: after sending failure, the sink should identify recoverable failure (network and so on) by returning a specific error type which will return a failure ack so that the cache can be preserved. For successful sink or unrecoverable errors, a success ack will be sent to delete the cache.
2. Cache mechanism: The cache will be saved in the memory firstly. If it exceeds the memory threshold, the later cache will be saved into the disk. Once the disk cache pass the disk storage threshold, the cache will start to rotate: the earliest cache in the memory will be dropped and replaced by loading the earliest cache in the disk.
3. Resend strategy: If an ack is pending, waiting a success ack to send the cache data one by one. Otherwise, when a new data come, send the first data in cache to detect network conditions. If successful, send all caches in order (mem + disk) with a defined interval in chain, which means send the next data when receiving a successful ack.

Configuration

There wil be two levels of sink cache configuration. A global configuration in `etc/kuiper.yaml` to define the default behavior for all rules. And a rule sink level definition to override the default behaviors.

The configuration items:

- enableCache: whether to enable sink cache. The cache storage configuration follows the metadata storage defined in `etc/kuiper.yaml`
- memoryCacheThreshold: the number of messages to be cached in the memory. For performance reason, the earliest cached messages are stored in the memory in order to resend immediately when the failures are restored. The data here will be lost due to failures like power off.
- maxDiskCache: the maximum number of messages to be cached in the disk. The disk cache is FIFO. If the disk cache is full, the earliest page of messages will be loaded into the memory cache to replace the old one.
- bufferPageSize: Buffer page is the unit to read/write to disk batchly to prevent frequent IO. If the page is not full and eKuiper crashed by hardware or software errors, the last page which have not been written to the disk will be lost.
- resendInterval: the interval for resending the messages after failure recovered to prevent message storm.
- cleanCacheAtStop: whether to clean all caches when the rule stops to prevent a burst of resend for outdated messages when the rule restarts. If not set to true, the memory cache will be stored into the disk once the rule is stopping. Otherwise, the memory and disk rules are cleaned up.

Internal Configuration for Sqlite

The default storage will be sqlite. The scenario for cache storage have these characterstics:

1. Sequential writing
2. Adapt to limited CPU+Memory platform
3. Async, await mechanism (non-transaction)
4. Append-only, No Edit

We will use these sqlite configurations by default:

1. Set Page Size as same as OS's page size (getconf PAGESIZE)
   `PRAGMA page_size = 4096;`
2. Set as WAL mode
   `PRAGMA journal_mode=WAL;`
3. Set synchronous mode as full, so that it won't corrupt the database file when experiencing power down.
   `PRAGMA synchronous=FULL`
4. Set checkpoint to auto or disable it and mange it by self-define interval.
   Enable: `PRAGMA wal_autocheckpoint; or sqlite3_wal_autocheckpoint(sqlite3 *db, int N);`
   Disable: `PRAGMA wal_autocheckpoint=N;`

Implementation consideration

- If the disk storage is sqlite, all the caches will be save to `data/cache.db` file. Each sink will have an unique sqlite table to save the cache.
- Add cached count to the sink metrics
- Integrate into the checkpoint mechanism
- Limitation: implement sync mode firstly

1289

- :
- : Jiyong Huang
- : https://github.com/lf-edge/ekuiper/issues/1289

eKuiper sink /DBQOS

sinkeKuipersinksinksinkack

- sinkackack
- rotate
- AckAck Ack mem + disk

Sink `etc/kuiper.yaml` sink

- enableCachesink cache `etc/kuiper.yaml`
- memoryCacheThreshold
- maxDiskCache: FIFO
- bufferPageSize/IOeKuiper
- resendInterval:
- cleanCacheAtStoptrue

Sqlite

sqlite

1.
2. CPU+Memory
3. transaction)
4.

sqlite

- getconf PAGESIZE
  PRAGMA page_size = 4096

- WAL
  PRAGMA journal_mode=WAL

- full
  PRAGMA synchronous=FULL

-
  PRAGMA wal_autocheckpoint; sqlite3_wal_autocheckpoint(sqlite3 *db, int N)
  PRAGMA wal_autocheckpoint=N

- sqlite`data/cache.db`sinksqlite
- sink metric
- checkpoint
- sync sink async