KIP-346 - Improve LogCleaner behavior on error

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: KAFKA-7215

***Released**: 2.1*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Historically, there have been numerous issues where the log compaction has failed for some reason, most commonly bugs in code (

KAFKA-3330 - Getting issue details... STATUS	**KAFKA-6264** - Getting issue details... STATUS	**KAFKA-6834** - Getting issue details... STATUS
KAFKA-6854 - Getting issue details... STATUS	**KAFKA-6762** - Getting issue details... STATUS	

to name some).

Currently, during log compaction, if the compaction of one log fails unexpectedly the whole `CleanerThread` responsible for compacting and deleting old logs exits. It is then not automatically restarted at any point. This results in a Kafka broker that runs seemingly fine but does not delete old log segments at all. This makes the broker a ticking time bomb - it is only a matter of time until the broker runs out of disk space and then all sorts of fatal scenarios ensue.

The situation has been improving - we have a metric showing the time since the last run of the `CleanerThread` (`kafka.log:type=LogCleanerManager, name=time-since-last-run-ms`) and Kafka 1.1 (KIP-226) provided functionality allowing us to restart the log cleaner thread without restarting the broker.

Then again, these improvements still require manual intervention or at the very least complex infrastructure code that automates the process.
It would be very useful if Kafka had a way to quarantine unexpected failures in certain logs such that they don't affect the cleaning of other logs. While this would not fix the issue, it would significantly slow down the process and provide users with adequate time for detection and repair.

Public Interfaces

New metrics:

- `uncleanable-partitions-count` (Int) - Count of partitions that are uncleanable per logDir
- `uncleanable-bytes` (Long) - The current number of uncleanable bytes per logDir. This is the sum of uncleanable bytes for every uncleanable partition in a certain log directory

Proposed Changes

Catch any unexpected (non-IO) exceptions in `CleanerThread#cleanOrSleep()`.

Properly log the exception and mark the partition that caused the exception as "uncleanable" in a collection in `LogCleaner`'s `LogCleanerManager`.

When evaluating which logs to compact, skip the marked as uncleanable ones.

Compatibility, Deprecation, and Migration Plan

The "time-since-last-run" metric will slightly change its behavior, since the LogCleaner will now continue to run once it encounters an error. Previous implementations that track the "time-since-last-run" metric for potential disk failures might be affected, but at least disk damage is maximally mitigated by marking the log directory as offline. If all log directories are offline, "time-since-last-run" will not be updated.

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

- *Restart `CleanerThread` - it will most likely inevitably hit the same problem before it is able to compact more*
- *Mark disk volumes as "uncleanable" on first encountered error. While this would work, in practice it would not help as most deployments use a single volume. Also, if the error is caused by a bug in the partition itself (as shown by most JIRA issues in the Motivation paragraph), this will unnecessarily stop compaction of all other partitions.*
- *Mark log directories as offline after a certain threshold of uncleanable bytes or number of uncleanable partitions. - uncleanable partitions threshold proved insufficient since previous problems that have been encountered affected a small number of partitions (__consumer_offsets topic). threshold of uncleanable bytes is hard to get right, as it should be different for each user and the default value should best be -1 (disabled)*