KIP-490: New metric to count offsets expired without being consumed by a consumer group

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA:

Motivation

Messages stored in Kafka will expire and disappear silently based on retention time and size configuration. Consumers have no way to know they have missed messages.

I propose to expose a JMX metric after a message has been removed due to topic time/size retention settings, for a set of consumer groups specified on the topic configuration.

This could be implemented because the kafka brokers know the information needed for the task:

- offset of the message that has been removed.
- last offset consumed from a consumer group.

The compacted topics are out of scope, because when users chose compacted topics they are interested on the last value of they key, not on the intermediary states.

Public Interfaces

A new property at topic level should be created:

Name	Description	Type	Default	Valid Values	Server default property	Importance
non.consumed. offsets.groups	comma separated list of consumer groups that will expose a metric with the number of messages that expired before being consumed	List	""		""	medium

A new JMX metric should be created to be exposed by the broker:

Metric / Attribute Name	Description	MBEAN NAME
non-consumed-total	Number of messages expired without being consumed by a consumer group, per topic and partition	

Proposed Changes

Currently the LogManager schedule the "kafka-delete-logs" thread, that will call the deleteLogs() method. Is possible to add into that method the metric to expose the number of offsets non consumed by a list of consumer groups.

The pseudo code starts on the line 19:

LogManager.scala deleteLogs()

```
  private def deleteLogs(): Unit = {
    var nextDelayMs = 0L
    try {
      def nextDeleteDelayMs: Long = {
        if (!logsToBeDeleted.isEmpty) {
          val (_, scheduleTimeMs) = logsToBeDeleted.peek()
          scheduleTimeMs + currentDefaultConfig.fileDeleteDelayMs - time.milliseconds()
        } else
          currentDefaultConfig.fileDeleteDelayMs
      }

      while ({nextDelayMs = nextDeleteDelayMs; nextDelayMs <= 0}) {
        val (removedLog, _) = logsToBeDeleted.take()
        if (removedLog != null) {
          try {
            removedLog.delete()
            info(s"Deleted log for partition ${removedLog.topicPartition} in ${removedLog.dir.
getAbsolutePath}.")



                        //
                        //  KIP-490: log when consumer groups lose a message because offset has been deleted
                        //

                        val consumerGroupsSubscribed : Seq[String] = getConsumerGroups(removedLog.
topicPartition.topic() );
                        val groupsToNotify : Seq[String] = consumerGroupsSubscribed intersect groupsTobeNotify
// value get from topic config property 'retention.notify.groups'
                        groupsToNotify.forEach( {
                          val lastCosumedOffsetGroup : Integer = getLastOffsetConsumed( _, removedLog.
topicPartition);
              if(lastCosumedOffsetGroup < removedLog.nextOffsetMetadata) {
                // increment and expose JMS metric non-consumed-total.
                          }
                        })



          } catch {
            case e: KafkaStorageException =>
              error(s"Exception while deleting $removedLog in dir ${removedLog.dir.getParent}.", e)
          }
        }
      }
    } catch {
      case e: Throwable =>
        error(s"Exception in kafka-delete-logs thread.", e)
    } finally {
      try {
        scheduler.schedule("kafka-delete-logs",
          deleteLogs _,
          delay = nextDelayMs,
          unit = TimeUnit.MILLISECONDS)
      } catch {
        case e: Throwable =>
          if (scheduler.isStarted) {
            // No errors should occur unless scheduler has been shutdown
            error(s"Failed to schedule next delete in kafka-delete-logs thread", e)
          }
      }
    }
  }
```

Compatibility, Deprecation, and Migration Plan

There is no impact on existing features.

Rejected Alternatives

Logging on broker side

To write logs on the broker to side to alert that messages have not been consumed has been rejected in favour of standard monitoring. Main reason is the big amount of data that could be generated.