package com.twitter.search.earlybird.partition;

import java.io.IOException;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import com.google.common.util.concurrent.SimpleTimeLimiter;

import com.google.common.util.concurrent.TimeLimiter;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.metrics.SearchTimer;

import com.twitter.search.common.metrics.SearchTimerStats;

/\*\*

\* Abstracts details of making time limited calls to hadoop.

\*

\* During IM-3556 we discovered that hadoop API calls can take a long time (seconds, minutes)

\* if the Hadoop clsuter is in a bad state. Our code was generally not prepared for that and

\* this caused various issues. This class is a fix on top of the Hadoop API's exists call and

\* it introduces a timeout.

\*

\* The main motivation for having this as an external class is for testability.

\*/

public class TimeLimitedHadoopExistsCall {

private final TimeLimiter hadoopCallsTimeLimiter;

private final FileSystem fileSystem;

private final int timeLimitInSeconds;

private static final SearchTimerStats EXISTS\_CALLS\_TIMER =

SearchTimerStats.export("hadoop\_exists\_calls");

private static final SearchCounter EXISTS\_CALLS\_EXCEPTION =

SearchCounter.export("hadoop\_exists\_calls\_exception");

public TimeLimitedHadoopExistsCall(FileSystem fileSystem) {

// This times varies. Sometimes it's very quick, sometimes it takes some amount of seconds.

// Do a rate on hadoop\_exists\_calls\_latency\_ms to see for yourself.

this(fileSystem, 30);

}

public TimeLimitedHadoopExistsCall(FileSystem fileSystem, int timeLimitInSeconds) {

// We do hadoop calls once every "FLUSH\_CHECK\_PERIOD" minutes. If a call takes

// a long time (say 10 minutes), we'll use a new thread for the next call, to give it

// a chance to complete.

//

// Let's say every call takes 2 hours. After 5 calls, the 6th call won't be able

// to take a thread out of the thread pool and it will time out. That's fair, we don't

// want to keep sending requests to Hadoop if the situation is so dire.

ExecutorService executorService = Executors.newFixedThreadPool(5);

this.hadoopCallsTimeLimiter = SimpleTimeLimiter.create(executorService);

this.fileSystem = fileSystem;

this.timeLimitInSeconds = timeLimitInSeconds;

}

protected boolean hadoopExistsCall(Path path) throws IOException {

SearchTimer timer = EXISTS\_CALLS\_TIMER.startNewTimer();

boolean res = fileSystem.exists(path);

EXISTS\_CALLS\_TIMER.stopTimerAndIncrement(timer);

return res;

}

/\*\*

\* Checks if a path exists on Hadoop.

\*

\* @return true if the path exists.

\* @throws Exception see exceptions thrown by callWithTimeout

\*/

boolean exists(Path path) throws Exception {

try {

boolean result = hadoopCallsTimeLimiter.callWithTimeout(new Callable<Boolean>() {

@Override

public Boolean call() throws Exception {

return hadoopExistsCall(path);

}

}, timeLimitInSeconds, TimeUnit.SECONDS);

return result;

} catch (Exception ex) {

EXISTS\_CALLS\_EXCEPTION.increment();

// No need to print and rethrow, it will be printed when caught upstream.

throw ex;

}

}

}