package com.twitter.search.ingester.pipeline.twitter;

import java.util.List;

import java.util.Optional;

import java.util.Set;

import java.util.concurrent.CompletableFuture;

import scala.Option;

import scala.Tuple2;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Lists;

import org.apache.commons.lang.StringUtils;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.validation.ConsumedTypes;

import org.apache.commons.pipeline.validation.ProducesConsumed;

import com.twitter.search.common.decider.DeciderUtil;

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

import com.twitter.search.common.relevance.entities.TwitterMessageUser;

import com.twitter.search.ingester.model.IngesterTwitterMessage;

import com.twitter.search.ingester.pipeline.strato\_fetchers.AudioSpaceCoreFetcher;

import com.twitter.search.ingester.pipeline.strato\_fetchers.AudioSpaceParticipantsFetcher;

import com.twitter.strato.catalog.Fetch;

import com.twitter.ubs.thriftjava.AudioSpace;

import com.twitter.ubs.thriftjava.ParticipantUser;

import com.twitter.ubs.thriftjava.Participants;

import com.twitter.util.Function;

import com.twitter.util.Future;

import com.twitter.util.Futures;

import com.twitter.util.Try;

@ConsumedTypes(IngesterTwitterMessage.class)

@ProducesConsumed

public class RetrieveSpaceAdminsAndTitleStage extends TwitterBaseStage

<IngesterTwitterMessage, CompletableFuture<IngesterTwitterMessage>> {

@VisibleForTesting

protected static final String RETRIEVE\_SPACE\_ADMINS\_AND\_TITLE\_DECIDER\_KEY =

"ingester\_all\_retrieve\_space\_admins\_and\_title";

private AudioSpaceCoreFetcher coreFetcher;

private AudioSpaceParticipantsFetcher participantsFetcher;

private SearchRateCounter tweetsWithSpaceAdmins;

private SearchRateCounter tweetsWithSpaceTitle;

private SearchRateCounter coreFetchSuccess;

private SearchRateCounter coreFetchFailure;

private SearchRateCounter participantsFetchSuccess;

private SearchRateCounter participantsFetchFailure;

private SearchRateCounter emptyCore;

private SearchRateCounter emptyParticipants;

private SearchRateCounter emptySpaceTitle;

private SearchRateCounter emptySpaceAdmins;

private SearchRateCounter parallelFetchAttempts;

private SearchRateCounter parallelFetchFailure;

@Override

protected void doInnerPreprocess() {

innerSetup();

}

@Override

protected void innerSetup() {

coreFetcher = wireModule.getAudioSpaceCoreFetcher();

participantsFetcher = wireModule.getAudioSpaceParticipantsFetcher();

tweetsWithSpaceAdmins = getStageStat("tweets\_with\_audio\_space\_admins");

tweetsWithSpaceTitle = getStageStat("tweets\_with\_audio\_space\_title");

coreFetchSuccess = getStageStat("core\_fetch\_success");

coreFetchFailure = getStageStat("core\_fetch\_failure");

participantsFetchSuccess = getStageStat("participants\_fetch\_success");

participantsFetchFailure = getStageStat("participants\_fetch\_failure");

emptyCore = getStageStat("empty\_core");

emptyParticipants = getStageStat("empty\_participants");

emptySpaceTitle = getStageStat("empty\_space\_title");

emptySpaceAdmins = getStageStat("empty\_space\_admins");

parallelFetchAttempts = getStageStat("parallel\_fetch\_attempts");

parallelFetchFailure = getStageStat("parallel\_fetch\_failure");

}

private SearchRateCounter getStageStat(String statSuffix) {

return SearchRateCounter.export(getStageNamePrefix() + "\_" + statSuffix);

}

private Future<Tuple2<Try<Fetch.Result<AudioSpace>>, Try<Fetch.Result<Participants>>>>

tryRetrieveSpaceAdminAndTitle(IngesterTwitterMessage twitterMessage) {

Set<String> spaceIds = twitterMessage.getSpaceIds();

if (spaceIds.isEmpty()) {

return null;

}

if (!(DeciderUtil.isAvailableForRandomRecipient(decider,

RETRIEVE\_SPACE\_ADMINS\_AND\_TITLE\_DECIDER\_KEY))) {

return null;

}

String spaceId = spaceIds.iterator().next();

// Query both columns in parallel.

parallelFetchAttempts.increment();

Future<Fetch.Result<AudioSpace>> core = coreFetcher.fetch(spaceId);

Future<Fetch.Result<Participants>> participants = participantsFetcher.fetch(spaceId);

return Futures.join(core.liftToTry(), participants.liftToTry());

}

@Override

protected CompletableFuture<IngesterTwitterMessage> innerRunStageV2(IngesterTwitterMessage

twitterMessage) {

Future<Tuple2<Try<Fetch.Result<AudioSpace>>, Try<Fetch.Result<Participants>>>>

tryRetrieveSpaceAdminAndTitle = tryRetrieveSpaceAdminAndTitle(twitterMessage);

CompletableFuture<IngesterTwitterMessage> cf = new CompletableFuture<>();

if (tryRetrieveSpaceAdminAndTitle == null) {

cf.complete(twitterMessage);

} else {

tryRetrieveSpaceAdminAndTitle.onSuccess(Function.cons(tries -> {

handleFutureOnSuccess(tries, twitterMessage);

cf.complete(twitterMessage);

})).onFailure(Function.cons(throwable -> {

handleFutureOnFailure();

cf.complete(twitterMessage);

}));

}

return cf;

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof IngesterTwitterMessage)) {

throw new StageException(this, "Object is not a IngesterTwitterMessage object: " + obj);

}

IngesterTwitterMessage twitterMessage = (IngesterTwitterMessage) obj;

Future<Tuple2<Try<Fetch.Result<AudioSpace>>, Try<Fetch.Result<Participants>>>>

tryRetrieveSpaceAdminAndTitle = tryRetrieveSpaceAdminAndTitle(twitterMessage);

if (tryRetrieveSpaceAdminAndTitle == null) {

emitAndCount(twitterMessage);

return;

}

tryRetrieveSpaceAdminAndTitle.onSuccess(Function.cons(tries -> {

handleFutureOnSuccess(tries, twitterMessage);

emitAndCount(twitterMessage);

})).onFailure(Function.cons(throwable -> {

handleFutureOnFailure();

emitAndCount(twitterMessage);

}));

}

private void handleFutureOnSuccess(Tuple2<Try<Fetch.Result<AudioSpace>>,

Try<Fetch.Result<Participants>>> tries, IngesterTwitterMessage twitterMessage) {

handleCoreFetchTry(tries.\_1(), twitterMessage);

handleParticipantsFetchTry(tries.\_2(), twitterMessage);

}

private void handleFutureOnFailure() {

parallelFetchFailure.increment();

}

private void handleCoreFetchTry(

Try<Fetch.Result<AudioSpace>> fetchTry,

IngesterTwitterMessage twitterMessage) {

if (fetchTry.isReturn()) {

coreFetchSuccess.increment();

addSpaceTitleToMessage(twitterMessage, fetchTry.get().v());

} else {

coreFetchFailure.increment();

}

}

private void handleParticipantsFetchTry(

Try<Fetch.Result<Participants>> fetchTry,

IngesterTwitterMessage twitterMessage) {

if (fetchTry.isReturn()) {

participantsFetchSuccess.increment();

addSpaceAdminsToMessage(twitterMessage, fetchTry.get().v());

} else {

participantsFetchFailure.increment();

}

}

private void addSpaceTitleToMessage(

IngesterTwitterMessage twitterMessage,

Option<AudioSpace> audioSpace) {

if (audioSpace.isDefined()) {

String audioSpaceTitle = audioSpace.get().getTitle();

if (StringUtils.isNotEmpty(audioSpaceTitle)) {

twitterMessage.setSpaceTitle(audioSpaceTitle);

tweetsWithSpaceTitle.increment();

} else {

emptySpaceTitle.increment();

}

} else {

emptyCore.increment();

}

}

private void addSpaceAdminsToMessage(

IngesterTwitterMessage twitterMessage,

Option<Participants> participants) {

if (participants.isDefined()) {

List<ParticipantUser> admins = getAdminsFromParticipants(participants.get());

if (!admins.isEmpty()) {

for (ParticipantUser admin : admins) {

addSpaceAdminToMessage(twitterMessage, admin);

}

tweetsWithSpaceAdmins.increment();

} else {

emptySpaceAdmins.increment();

}

} else {

emptyParticipants.increment();

}

}

private List<ParticipantUser> getAdminsFromParticipants(Participants participants) {

if (!participants.isSetAdmins()) {

return Lists.newArrayList();

}

return participants.getAdmins();

}

private void addSpaceAdminToMessage(IngesterTwitterMessage twitterMessage,

ParticipantUser admin) {

TwitterMessageUser.Builder userBuilder = new TwitterMessageUser.Builder();

if (admin.isSetTwitter\_screen\_name()

&& StringUtils.isNotEmpty(admin.getTwitter\_screen\_name())) {

userBuilder.withScreenName(Optional.of(admin.getTwitter\_screen\_name()));

}

if (admin.isSetDisplay\_name() && StringUtils.isNotEmpty(admin.getDisplay\_name())) {

userBuilder.withDisplayName(Optional.of(admin.getDisplay\_name()));

}

twitterMessage.addSpaceAdmin(userBuilder.build());

}

}