package com.twitter.search.core.earlybird.index.column;

import java.io.IOException;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.ConcurrentHashMap;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import com.twitter.search.common.schema.base.EarlybirdFieldType;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.DataSerializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.common.util.io.flushable.Flushable;

import com.twitter.search.core.earlybird.index.DocIDToTweetIDMapper;

public abstract class DocValuesManager implements Flushable {

protected final Schema schema;

protected final int segmentSize;

protected final ConcurrentHashMap<String, ColumnStrideFieldIndex> columnStrideFields;

public DocValuesManager(Schema schema, int segmentSize) {

this(schema, segmentSize, new ConcurrentHashMap<>());

}

protected DocValuesManager(Schema schema,

int segmentSize,

ConcurrentHashMap<String, ColumnStrideFieldIndex> columnStrideFields) {

this.schema = Preconditions.checkNotNull(schema);

this.segmentSize = segmentSize;

this.columnStrideFields = columnStrideFields;

}

protected abstract ColumnStrideFieldIndex newByteCSF(String field);

protected abstract ColumnStrideFieldIndex newIntCSF(String field);

protected abstract ColumnStrideFieldIndex newLongCSF(String field);

protected abstract ColumnStrideFieldIndex newMultiIntCSF(String field, int numIntsPerField);

/\*\*

\* Optimize this doc values manager, and return a doc values manager a more compact and fast

\* encoding for doc values (but that we can't add new doc IDs to).

\*/

public abstract DocValuesManager optimize(

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException;

public Set<String> getDocValueNames() {

return columnStrideFields.keySet();

}

/\*\*

\* Creates a new {@link ColumnStrideFieldIndex} for the given field and returns it.

\*/

public ColumnStrideFieldIndex addColumnStrideField(String field, EarlybirdFieldType fieldType) {

// For CSF view fields, we will perform the same check on the base field when we try to create

// a ColumnStrideFieldIndex for them in newIntViewCSF().

if (!fieldType.isCsfViewField()) {

Preconditions.checkState(

fieldType.isCsfLoadIntoRam(), "Field %s is not loaded in RAM", field);

}

if (columnStrideFields.containsKey(field)) {

return columnStrideFields.get(field);

}

final ColumnStrideFieldIndex index;

switch (fieldType.getCsfType()) {

case BYTE:

index = newByteCSF(field);

break;

case INT:

if (fieldType.getCsfFixedLengthNumValuesPerDoc() > 1) {

index = newMultiIntCSF(field, fieldType.getCsfFixedLengthNumValuesPerDoc());

} else if (fieldType.isCsfViewField()) {

index = newIntViewCSF(field);

} else {

index = newIntCSF(field);

}

break;

case LONG:

index = newLongCSF(field);

break;

default:

throw new RuntimeException("Invalid CsfType.");

}

columnStrideFields.put(field, index);

return index;

}

protected ColumnStrideFieldIndex newIntViewCSF(String field) {

Schema.FieldInfo info = Preconditions.checkNotNull(schema.getFieldInfo(field));

Schema.FieldInfo baseFieldInfo = Preconditions.checkNotNull(

schema.getFieldInfo(info.getFieldType().getCsfViewBaseFieldId()));

Preconditions.checkState(

baseFieldInfo.getFieldType().isCsfLoadIntoRam(),

"Field %s has a base field (%s) that is not loaded in RAM",

field, baseFieldInfo.getName());

// We might not have a CSF for the base field yet.

ColumnStrideFieldIndex baseFieldIndex =

addColumnStrideField(baseFieldInfo.getName(), baseFieldInfo.getFieldType());

Preconditions.checkNotNull(baseFieldIndex);

Preconditions.checkState(baseFieldIndex instanceof AbstractColumnStrideMultiIntIndex);

return new ColumnStrideIntViewIndex(info, (AbstractColumnStrideMultiIntIndex) baseFieldIndex);

}

/\*\*

\* Returns the ColumnStrideFieldIndex instance for the given field.

\*/

public ColumnStrideFieldIndex getColumnStrideFieldIndex(String field) {

ColumnStrideFieldIndex docValues = columnStrideFields.get(field);

if (docValues == null) {

Schema.FieldInfo info = schema.getFieldInfo(field);

if (info != null && info.getFieldType().isCsfDefaultValueSet()) {

return new ConstantColumnStrideFieldIndex(field, info.getFieldType().getCsfDefaultValue());

}

}

return docValues;

}

private static final String CSF\_INDEX\_CLASS\_NAME\_PROP\_NAME = "csfIndexClassName";

private static final String CSF\_PROP\_NAME = "column\_stride\_fields";

protected static final String MAX\_SEGMENT\_SIZE\_PROP\_NAME = "maxSegmentSize";

private static Map<String, Set<Schema.FieldInfo>> getIntViewFields(Schema schema) {

Map<String, Set<Schema.FieldInfo>> intViewFields = Maps.newHashMap();

for (Schema.FieldInfo fieldInfo : schema.getFieldInfos()) {

if (fieldInfo.getFieldType().isCsfViewField()) {

Schema.FieldInfo baseFieldInfo = Preconditions.checkNotNull(

schema.getFieldInfo(fieldInfo.getFieldType().getCsfViewBaseFieldId()));

String baseFieldName = baseFieldInfo.getName();

Set<Schema.FieldInfo> intViewFieldsForBaseField =

intViewFields.computeIfAbsent(baseFieldName, k -> Sets.newHashSet());

intViewFieldsForBaseField.add(fieldInfo);

}

}

return intViewFields;

}

public abstract static class FlushHandler extends Handler<DocValuesManager> {

private final Schema schema;

public FlushHandler(Schema schema) {

this.schema = schema;

}

public FlushHandler(DocValuesManager docValuesManager) {

super(docValuesManager);

this.schema = docValuesManager.schema;

}

@Override

public void doFlush(FlushInfo flushInfo, DataSerializer out) throws IOException {

long startTime = getClock().nowMillis();

DocValuesManager docValuesManager = getObjectToFlush();

flushInfo.addIntProperty(MAX\_SEGMENT\_SIZE\_PROP\_NAME, docValuesManager.segmentSize);

long sizeBeforeFlush = out.length();

FlushInfo csfProps = flushInfo.newSubProperties(CSF\_PROP\_NAME);

for (ColumnStrideFieldIndex csf : docValuesManager.columnStrideFields.values()) {

if (!(csf instanceof ColumnStrideIntViewIndex)) {

Preconditions.checkState(

csf instanceof Flushable,

"Cannot flush column stride field {} of type {}",

csf.getName(), csf.getClass().getCanonicalName());

FlushInfo info = csfProps.newSubProperties(csf.getName());

info.addStringProperty(CSF\_INDEX\_CLASS\_NAME\_PROP\_NAME, csf.getClass().getCanonicalName());

((Flushable) csf).getFlushHandler().flush(info, out);

}

}

csfProps.setSizeInBytes(out.length() - sizeBeforeFlush);

getFlushTimerStats().timerIncrement(getClock().nowMillis() - startTime);

}

@Override

public DocValuesManager doLoad(FlushInfo flushInfo, DataDeserializer in)

throws IOException {

long startTime = getClock().nowMillis();

Map<String, Set<Schema.FieldInfo>> intViewFields = getIntViewFields(schema);

FlushInfo csfProps = flushInfo.getSubProperties(CSF\_PROP\_NAME);

ConcurrentHashMap<String, ColumnStrideFieldIndex> columnStrideFields =

new ConcurrentHashMap<>();

Iterator<String> csfPropIter = csfProps.getKeyIterator();

while (csfPropIter.hasNext()) {

String fieldName = csfPropIter.next();

try {

FlushInfo info = csfProps.getSubProperties(fieldName);

String className = info.getStringProperty(CSF\_INDEX\_CLASS\_NAME\_PROP\_NAME);

Class<? extends ColumnStrideFieldIndex> fieldIndexType =

(Class<? extends ColumnStrideFieldIndex>) Class.forName(className);

Preconditions.checkNotNull(

fieldIndexType,

"Invalid field configuration: field " + fieldName + " not found in config.");

for (Class<?> c : fieldIndexType.getDeclaredClasses()) {

if (Handler.class.isAssignableFrom(c)) {

@SuppressWarnings("rawtypes")

Handler handler = (Handler) c.newInstance();

ColumnStrideFieldIndex index = (ColumnStrideFieldIndex) handler.load(

csfProps.getSubProperties(fieldName), in);

columnStrideFields.put(fieldName, index);

// If this is a base field, create ColumnStrideIntViewIndex instances for all the

// view fields based on it.

if (index instanceof AbstractColumnStrideMultiIntIndex) {

AbstractColumnStrideMultiIntIndex multiIntIndex =

(AbstractColumnStrideMultiIntIndex) index;

// We should have AbstractColumnStrideMultiIntIndex instances only for base fields

// and all our base fields have views defined on top of them.

for (Schema.FieldInfo intViewFieldInfo : intViewFields.get(fieldName)) {

columnStrideFields.put(

intViewFieldInfo.getName(),

new ColumnStrideIntViewIndex(intViewFieldInfo, multiIntIndex));

}

}

break;

}

}

} catch (ClassNotFoundException | IllegalAccessException | InstantiationException e) {

throw new IOException(

"Invalid field configuration for column stride field: " + fieldName, e);

}

}

getLoadTimerStats().timerIncrement(getClock().nowMillis() - startTime);

return createDocValuesManager(

schema,

flushInfo.getIntProperty(MAX\_SEGMENT\_SIZE\_PROP\_NAME),

columnStrideFields);

}

protected abstract DocValuesManager createDocValuesManager(

Schema docValuesSchema,

int maxSegmentSize,

ConcurrentHashMap<String, ColumnStrideFieldIndex> columnStrideFields);

}

}