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

import java.io.IOException;

import java.util.Arrays;

import com.google.common.annotations.VisibleForTesting;

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

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;

// Modeled after TwitterCharBlockPool, with a lot of simplification.

public class IntBlockPool implements Flushable {

private static final SearchLongGauge INT\_BLOCK\_POOL\_MAX\_LENGTH =

SearchLongGauge.export("twitter\_int\_block\_pool\_max\_size");

private static final String STAT\_PREFIX = "twitter\_int\_block\_pool\_size\_";

private static final int BLOCK\_SHIFT = 14;

public static final int BLOCK\_SIZE = 1 << BLOCK\_SHIFT;

private static final int BLOCK\_MASK = BLOCK\_SIZE - 1;

// We can address up to 2^31 elements with an int. We use 1 << 14 bits for the block offset,

// so we can use the remaining 17 bits for the blocks index. Therefore the maximum number of

// addressable blocks is 1 << 17 or maxInt >> 14.

private static final int MAX\_NUM\_BLOCKS = Integer.MAX\_VALUE >> BLOCK\_SHIFT;

// Initial value written into the blocks.

private final int initialValue;

// Extra object with final array is necessary to guarantee visibility

// to other threads without synchronization / volatiles. See comment

// in TwitterCharBlockPool.

public static final class Pool {

public final int[][] blocks;

Pool(int[][] blocks) {

this.blocks = blocks;

// Adjust max size if exceeded maximum value.

synchronized (INT\_BLOCK\_POOL\_MAX\_LENGTH) {

if (this.blocks != null) {

final long currentSize = (long) (this.blocks.length \* BLOCK\_SIZE);

if (currentSize > INT\_BLOCK\_POOL\_MAX\_LENGTH.get()) {

INT\_BLOCK\_POOL\_MAX\_LENGTH.set(currentSize);

}

}

}

}

}

public Pool pool;

private int currBlockIndex; // Index into blocks array.

private int[] currBlock = null;

private int currBlockOffset; // Index into current block.

private final String poolName;

private final SearchLongGauge sizeGauge;

public IntBlockPool(String poolName) {

this(0, poolName);

}

public IntBlockPool(int initialValue, String poolName) {

// Start with room for 16 initial blocks (does not allocate these blocks).

this.pool = new Pool(new int[16][]);

this.initialValue = initialValue;

// Start at the end of a previous, non-existent blocks.

this.currBlockIndex = -1;

this.currBlock = null;

this.currBlockOffset = BLOCK\_SIZE;

this.poolName = poolName;

this.sizeGauge = createGauge(poolName, pool);

}

// Constructor for FlushHandler.

protected IntBlockPool(

int currBlockIndex,

int currBlockOffset,

int[][]blocks,

String poolName) {

this.initialValue = 0;

this.pool = new Pool(blocks);

this.currBlockIndex = currBlockIndex;

this.currBlockOffset = currBlockOffset;

if (currBlockIndex >= 0) {

this.currBlock = this.pool.blocks[currBlockIndex];

}

this.poolName = poolName;

this.sizeGauge = createGauge(poolName, pool);

}

private static SearchLongGauge createGauge(String suffix, Pool pool) {

SearchLongGauge gauge = SearchLongGauge.export(STAT\_PREFIX + suffix);

if (pool.blocks != null) {

gauge.set(pool.blocks.length \* BLOCK\_SIZE);

}

return gauge;

}

/\*\*

\* Adds an int to the current block and returns it's overall index.

\*/

public int add(int value) {

if (currBlockOffset == BLOCK\_SIZE) {

newBlock();

}

currBlock[currBlockOffset++] = value;

return (currBlockIndex << BLOCK\_SHIFT) + currBlockOffset - 1;

}

// Returns number of ints in this blocks

public int length() {

return currBlockOffset + currBlockIndex \* BLOCK\_SIZE;

}

// Gets an int from the specified index.

public final int get(int index) {

return getBlock(index)[getOffsetInBlock(index)];

}

public static int getBlockStart(int index) {

return (index >>> BLOCK\_SHIFT) \* BLOCK\_SIZE;

}

public static int getOffsetInBlock(int index) {

return index & BLOCK\_MASK;

}

public final int[] getBlock(int index) {

final int blockIndex = index >>> BLOCK\_SHIFT;

return pool.blocks[blockIndex];

}

// Sets an int value at the specified index.

public void set(int index, int value) {

final int blockIndex = index >>> BLOCK\_SHIFT;

final int offset = index & BLOCK\_MASK;

pool.blocks[blockIndex][offset] = value;

}

/\*\*

\* Evaluates whether two instances of IntBlockPool are equal by value. It is

\* slow because it has to check every element in the pool.

\*/

@VisibleForTesting

public boolean verySlowEqualsForTests(IntBlockPool that) {

if (length() != that.length()) {

return false;

}

for (int i = 0; i < length(); i++) {

if (get(i) != that.get(i)) {

return false;

}

}

return true;

}

private void newBlock() {

final int newBlockIndex = 1 + currBlockIndex;

if (newBlockIndex >= MAX\_NUM\_BLOCKS) {

throw new RuntimeException(

"Too many blocks, would overflow int index for blocks " + poolName);

}

if (newBlockIndex == pool.blocks.length) {

// Blocks array is too small to add a new block. Resize.

int[][] newBlocks = new int[pool.blocks.length \* 2][];

System.arraycopy(pool.blocks, 0, newBlocks, 0, pool.blocks.length);

pool = new Pool(newBlocks);

sizeGauge.set(pool.blocks.length \* BLOCK\_SIZE);

}

currBlock = pool.blocks[newBlockIndex] = allocateBlock();

currBlockOffset = 0;

currBlockIndex = newBlockIndex;

}

private int[] allocateBlock() {

int[] block = new int[BLOCK\_SIZE];

Arrays.fill(block, initialValue);

return block;

}

@SuppressWarnings("unchecked")

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

public static final class FlushHandler extends Flushable.Handler<IntBlockPool> {

private static final String CURRENT\_BLOCK\_INDEX\_PROP\_NAME = "currentBlockIndex";

private static final String CURRENT\_BLOCK\_OFFSET\_PROP\_NAME = "currentBlockOffset";

private static final String POOL\_NAME = "poolName";

public FlushHandler() {

super();

}

public FlushHandler(IntBlockPool objToFlush) {

super(objToFlush);

}

@Override

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

IntBlockPool pool = getObjectToFlush();

flushInfo.addIntProperty(CURRENT\_BLOCK\_INDEX\_PROP\_NAME, pool.currBlockIndex);

flushInfo.addIntProperty(CURRENT\_BLOCK\_OFFSET\_PROP\_NAME, pool.currBlockOffset);

flushInfo.addStringProperty(POOL\_NAME, pool.poolName);

out.writeIntArray2D(pool.pool.blocks, pool.currBlockIndex + 1);

}

@Override

protected IntBlockPool doLoad(FlushInfo flushInfo, DataDeserializer in) throws IOException {

String poolName = flushInfo.getStringProperty(POOL\_NAME);

return new IntBlockPool(

flushInfo.getIntProperty(CURRENT\_BLOCK\_INDEX\_PROP\_NAME),

flushInfo.getIntProperty(CURRENT\_BLOCK\_OFFSET\_PROP\_NAME),

in.readIntArray2D(),

poolName);

}

}

}