#include "internal/error.h"

#include "internal/thrift.h"

#include <map>

#include <twml/ThriftWriter.h>

#include <twml/TensorRecordWriter.h>

#include <twml/io/IOError.h>

using namespace twml::io;

namespace twml {

static int32\_t getRawThriftType(twml\_type dtype) {

// convert twml enum to tensor.thrift enum

switch (dtype) {

case TWML\_TYPE\_FLOAT:

return DATA\_TYPE\_FLOAT;

case TWML\_TYPE\_DOUBLE:

return DATA\_TYPE\_DOUBLE;

case TWML\_TYPE\_INT64:

return DATA\_TYPE\_INT64;

case TWML\_TYPE\_INT32:

return DATA\_TYPE\_INT32;

case TWML\_TYPE\_UINT8:

return DATA\_TYPE\_UINT8;

case TWML\_TYPE\_STRING:

return DATA\_TYPE\_STRING;

case TWML\_TYPE\_BOOL:

return DATA\_TYPE\_BOOL;

default:

throw IOError(IOError::UNSUPPORTED\_OUTPUT\_TYPE);

}

}

void TensorRecordWriter::writeTensor(const RawTensor &tensor) {

if (tensor.getType() == TWML\_TYPE\_INT32) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_INT32);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_I32, tensor.getNumElements());

const int32\_t \*data = tensor.getData<int32\_t>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeInt32(data[i]);

} else if (tensor.getType() == TWML\_TYPE\_INT64) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_INT64);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_I64, tensor.getNumElements());

const int64\_t \*data = tensor.getData<int64\_t>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeInt64(data[i]);

} else if (tensor.getType() == TWML\_TYPE\_FLOAT) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_FLOAT);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_DOUBLE, tensor.getNumElements());

const float \*data = tensor.getData<float>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeDouble(static\_cast<double>(data[i]));

} else if (tensor.getType() == TWML\_TYPE\_DOUBLE) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_DOUBLE);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_DOUBLE, tensor.getNumElements());

const double \*data = tensor.getData<double>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeDouble(data[i]);

} else if (tensor.getType() == TWML\_TYPE\_STRING) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_STRING);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_STRING, tensor.getNumElements());

const std::string \*data = tensor.getData<std::string>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeString(data[i]);

} else if (tensor.getType() == TWML\_TYPE\_BOOL) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_BOOL);

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 1);

m\_thrift\_writer.writeListHeader(TTYPE\_BOOL, tensor.getNumElements());

const bool \*data = tensor.getData<bool>();

for (uint64\_t i = 0; i < tensor.getNumElements(); i++)

m\_thrift\_writer.writeBool(data[i]);

} else {

throw IOError(IOError::UNSUPPORTED\_OUTPUT\_TYPE);

}

// write tensor shape field

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 2);

m\_thrift\_writer.writeListHeader(TTYPE\_I64, tensor.getNumDims());

for (uint64\_t i = 0; i < tensor.getNumDims(); i++)

m\_thrift\_writer.writeInt64(tensor.getDim(i));

m\_thrift\_writer.writeStructStop();

m\_thrift\_writer.writeStructStop();

}

void TensorRecordWriter::writeRawTensor(const RawTensor &tensor) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRUCT, GT\_RAW);

// dataType field

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_I32, 1);

m\_thrift\_writer.writeInt32(getRawThriftType(tensor.getType()));

// content field

uint64\_t type\_size = getSizeOf(tensor.getType());

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_STRING, 2);

const uint8\_t \*data = reinterpret\_cast<const uint8\_t \*>(tensor.getData<void>());

m\_thrift\_writer.writeBinary(data, tensor.getNumElements() \* type\_size);

// shape field

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_LIST, 3);

m\_thrift\_writer.writeListHeader(TTYPE\_I64, tensor.getNumDims());

for (uint64\_t i = 0; i < tensor.getNumDims(); i++)

m\_thrift\_writer.writeInt64(tensor.getDim(i));

m\_thrift\_writer.writeStructStop();

m\_thrift\_writer.writeStructStop();

}

TWMLAPI uint32\_t TensorRecordWriter::getRecordsWritten() {

return m\_records\_written;

}

// Caller (usually DataRecordWriter) must precede with struct header field

// like thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_GENERAL\_TENSOR)

TWMLAPI uint64\_t TensorRecordWriter::write(twml::TensorRecord &record) {

uint64\_t bytes\_written\_before = m\_thrift\_writer.getBytesWritten();

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_STRUCT, record.getRawTensors().size());

for (auto id\_tensor\_pairs : record.getRawTensors()) {

m\_thrift\_writer.writeInt64(id\_tensor\_pairs.first);

// all tensors written as RawTensor Thrift except for StringTensors

// this avoids the overhead of converting little endian to big endian

if (id\_tensor\_pairs.second.getType() == TWML\_TYPE\_STRING)

writeTensor(id\_tensor\_pairs.second);

else

writeRawTensor(id\_tensor\_pairs.second);

}

m\_records\_written++;

return m\_thrift\_writer.getBytesWritten() - bytes\_written\_before;

}

} // namespace twml