#pragma once

#include <twml.h>

#include <atomic>

#include <string>

#include <vector>

// Add these to make gcc ignore the warnings from tensorflow.

#pragma GCC diagnostic push

#pragma GCC diagnostic ignored "-Wsign-compare"

#include "tensorflow/core/framework/resource\_mgr.h"

#include "tensorflow/core/framework/resource\_op\_kernel.h"

#pragma GCC diagnostic pop

#include <memory>

#include <functional>

template<typename T>

void unrefHandle(T \*handle) {

handle->Unref();

}

template <typename T>

using unique\_handle = std::unique\_ptr<T, std::function<void(T \*)> >;

// as std::type\_index is not abi compatible, we bypass the hash\_code checks.

// https://github.com/tensorflow/tensorflow/commit/15275d3a14c77e2244ae1155f93243256f08e3ed

#ifdef \_\_APPLE\_\_

template <typename T>

Status CreateTwmlResource(OpKernelContext\* ctx, const ResourceHandle& p, T\* value) {

return ctx->resource\_manager()->Create(p.container(), p.name(), value);

}

template <typename T>

Status LookupTwmlResource(OpKernelContext\* ctx, const ResourceHandle& p,

T\*\* value) {

return ctx->resource\_manager()->Lookup(p.container(), p.name(), value);

}

#endif // \_\_APPLE\_\_

template<typename T>

unique\_handle<T> getHandle(tensorflow::OpKernelContext\* context, int input\_idx) {

using namespace tensorflow;

T \*ptr = nullptr;

#ifdef \_\_APPLE\_\_

auto s = LookupTwmlResource(context, HandleFromInput(context, input\_idx), &ptr);

#else

auto s = LookupResource(context, HandleFromInput(context, input\_idx), &ptr);

#endif // \_\_APPLE\_\_

if (!s.ok()) {

throw std::runtime\_error("Failed to get resource handle");

}

return unique\_handle<T>(ptr, unrefHandle<T>);

}

template<typename InputType>

const uint8\_t \*getInputBytes(const Tensor &input, int id) {

return reinterpret\_cast<const uint8\_t \*>(input.flat<InputType>().data());

}

template<>

inline const uint8\_t \*getInputBytes<string>(const Tensor &input, int id) {

return reinterpret\_cast<const uint8\_t \*>(input.flat<string>()(id).c\_str());

}

template<typename InputType>

const int getBatchSize(const Tensor &input) {

return 1;

}

template<>

inline const int getBatchSize<string>(const Tensor &input) {

return static\_cast<int>(input.NumElements());

}

class DataRecordResource : public ResourceBase {

public:

Tensor input;

int64 num\_labels;

int64 num\_weights;

twml::DataRecord common;

std::vector<twml::DataRecord> records;

twml::Map<int64\_t, int64\_t> \*keep\_map;

string DebugString() const override { return "DataRecords resource"; }

};

// A thin layer around batch of HashedDataRecords

class HashedDataRecordResource : public ResourceBase {

public:

Tensor input;

int64 total\_size;

int64 num\_labels;

int64 num\_weights;

twml::HashedDataRecord common;

std::vector<twml::HashedDataRecord> records;

string DebugString() const override { return "HashedDataRecord Resource"; }

};

#define TF\_CHECK\_STATUS(fn) do { \

Status s = fn; \

if (!s.ok()) return s; \

} while (0)

template<typename ResourceType>

Status makeResourceHandle(OpKernelContext\* context, int out\_idx, ResourceType \*\*resource\_) {

static std::atomic<int64> id;

Tensor\* handle\_tensor;

TF\_CHECK\_STATUS(context->allocate\_output(out\_idx, TensorShape({}), &handle\_tensor));

ResourceType \*resource = new ResourceType();

const auto resource\_name = typeid(ResourceType).name() + std::to\_string(id++);

ResourceHandle handle = MakePerStepResourceHandle<ResourceType>(context, resource\_name);

#ifdef \_\_APPLE\_\_

TF\_CHECK\_STATUS(CreateTwmlResource(context, handle, resource));

#else

TF\_CHECK\_STATUS(CreateResource(context, handle, resource));

#endif // \_\_APPLE\_\_

handle\_tensor->scalar<ResourceHandle>()() = handle;

\*resource\_ = resource;

return Status::OK();

}