tensorflow / **tensorflow** Public

Code | Issues 2.1k | Pull requests 284 | Actions | Projects 1 | ...

f3b9bf4c3c

tensorflow / tensorflow / core / kernels / **session_ops.cc**

quintinwang5 add DEVICE_DEFAULT for session/transpose ops ✗ History

9 contributors

152 lines (126 sloc) | 5.78 KB

```
/* Copyright 2015 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
==============================================================================*/

// See docs in ../ops/data_flow_ops.cc.

#include <limits.h>

#include <vector>

#include "tensorflow/core/common_runtime/device.h"
#include "tensorflow/core/framework/device_base.h"
#include "tensorflow/core/framework/op_kernel.h"
#include "tensorflow/core/framework/register_types.h"
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/framework/tensor_shape.h"
#include "tensorflow/core/framework/types.h"
#include "tensorflow/core/lib/core/errors.h"
```

```cpp
#include "tensorflow/core/lib/gtl/map_util.h"
#include "tensorflow/core/platform/errors.h"
#include "tensorflow/core/platform/logging.h"
#include "tensorflow/core/platform/macros.h"
#include "tensorflow/core/platform/mutex.h"
#include "tensorflow/core/platform/thread_annotations.h"
#include "tensorflow/core/platform/types.h"

namespace tensorflow {

class GetSessionHandleOp : public OpKernel {
 public:
  explicit GetSessionHandleOp(OpKernelConstruction* context)
      : OpKernel(context) {}

  void Compute(OpKernelContext* ctx) override {
    const Tensor& val = ctx->input(0);
    auto session_state = ctx->session_state();
    OP_REQUIRES(ctx, session_state != nullptr,
                errors::FailedPrecondition(
                    "GetSessionHandle called on null session state"));
    int64_t id = session_state->GetNewId();
    TensorStore::TensorAndKey tk{val, id, requested_device()};
    OP_REQUIRES_OK(ctx, ctx->tensor_store()->AddTensor(name(), tk));

    Tensor* handle = nullptr;
    OP_REQUIRES_OK(ctx, ctx->allocate_output(0, TensorShape({}), &handle));
    if (ctx->expected_output_dtype(0) == DT_RESOURCE) {
      ResourceHandle resource_handle = MakeResourceHandle<Tensor>(
          ctx, SessionState::kTensorHandleResourceTypeName,
          tk.GetHandle(name()));
      resource_handle.set_maybe_type_name(
          SessionState::kTensorHandleResourceTypeName);
      handle->scalar<ResourceHandle>()() = resource_handle;
    } else {
      // Legacy behavior in V1.
      handle->flat<tstring>().setConstant(tk.GetHandle(name()));
    }
  }

  TF_DISALLOW_COPY_AND_ASSIGN(GetSessionHandleOp);
};

REGISTER_KERNEL_BUILDER(Name("GetSessionHandle").Device(DEVICE_CPU),
                        GetSessionHandleOp);
REGISTER_KERNEL_BUILDER(Name("GetSessionHandleV2").Device(DEVICE_CPU),
                        GetSessionHandleOp);

#define REGISTER_DEFAULT_KERNEL(type)                       \
  REGISTER_KERNEL_BUILDER(Name("GetSessionHandle")        \
                              .Device(DEVICE_DEFAULT)     \
                              .HostMemory("handle")       \
                              .TypeConstraint<type>("T"), \
                          GetSessionHandleOp)             \
  REGISTER_KERNEL_BUILDER(Name("GetSessionHandleV2")      \
                              .Device(DEVICE_DEFAULT)     \
                              .HostMemory("handle")       \
                              .TypeConstraint<type>("T"), \
                          GetSessionHandleOp)

TF_CALL_NUMBER_TYPES(REGISTER_DEFAULT_KERNEL);
REGISTER_DEFAULT_KERNEL(bool);
#undef REGISTER_DEFAULT_KERNEL

class GetSessionTensorOp : public OpKernel {
 public:
  explicit GetSessionTensorOp(OpKernelConstruction* context)
      : OpKernel(context) {}

  void Compute(OpKernelContext* ctx) override {
    const Tensor& handle = ctx->input(0);
    const string& name = handle.scalar<tstring>()();
    Tensor val;
    auto session_state = ctx->session_state();
    OP_REQUIRES(ctx, session_state != nullptr,
                errors::FailedPrecondition(
                    "GetSessionTensor called on null session state"));
    OP_REQUIRES_OK(ctx, session_state->GetTensor(name, &val));
    ctx->set_output(0, val);
  }

  TF_DISALLOW_COPY_AND_ASSIGN(GetSessionTensorOp);
};

REGISTER_KERNEL_BUILDER(Name("GetSessionTensor").Device(DEVICE_CPU),
                        GetSessionTensorOp);

#define REGISTER_DEFAULT_KERNEL(type)                         \
  REGISTER_KERNEL_BUILDER(Name("GetSessionTensor")            \
                              .Device(DEVICE_DEFAULT)         \
                              .HostMemory("handle")           \
                              .TypeConstraint<type>("dtype"), \
                          GetSessionTensorOp)

TF_CALL_NUMBER_TYPES(REGISTER_DEFAULT_KERNEL);
REGISTER_DEFAULT_KERNEL(bool);
#undef REGISTER_DEFAULT_KERNEL

```

```
class DeleteSessionTensorOp : public OpKernel {
 public:
  explicit DeleteSessionTensorOp(OpKernelConstruction* context)
      : OpKernel(context) {}

  void Compute(OpKernelContext* ctx) override {
    const Tensor& handle = ctx->input(0);
    const string& name = handle.scalar<tstring>()();
    auto session_state = ctx->session_state();
    OP_REQUIRES(ctx, session_state != nullptr,
                errors::FailedPrecondition(
                    "DeleteSessionTensor called on null session state"));
    OP_REQUIRES_OK(ctx, session_state->DeleteTensor(name));
  }

  TF_DISALLOW_COPY_AND_ASSIGN(DeleteSessionTensorOp);
};

REGISTER_KERNEL_BUILDER(Name("DeleteSessionTensor").Device(DEVICE_CPU),
                        DeleteSessionTensorOp);
REGISTER_KERNEL_BUILDER(
    Name("DeleteSessionTensor").Device(DEVICE_DEFAULT).HostMemory("handle"),
    DeleteSessionTensorOp);

}  // namespace tensorflow
```