

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

...

```

1  /* Copyright 2015 The TensorFlow Authors. All Rights Reserved.
2
3  Licensed under the Apache License, Version 2.0 (the "License");
4  you may not use this file except in compliance with the License.
5  You may obtain a copy of the License at
6
7      http://www.apache.org/licenses/LICENSE-2.0
8
9  Unless required by applicable law or agreed to in writing, software
10 distributed under the License is distributed on an "AS IS" BASIS,
11 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 See the License for the specific language governing permissions and
13 limitations under the License.
14 =====*/
15
16 // See docs in ../ops/data_flow_ops.cc.
17
18 #include <limits.h>
19
20 #include <vector>
21
22 #include "tensorflow/core/common_runtime/device.h"
23 #include "tensorflow/core/framework/device_base.h"
24 #include "tensorflow/core/framework/op_kernel.h"
25 #include "tensorflow/core/framework/register_types.h"
26 #include "tensorflow/core/framework/tensor.h"
27 #include "tensorflow/core/framework/tensor_shape.h"
28 #include "tensorflow/core/framework/types.h"
29 #include "tensorflow/core/lib/core/errors.h"

```

```

30 #include "tensorflow/core/lib/gtl/map_util.h"
31 #include "tensorflow/core/platform/errors.h"
32 #include "tensorflow/core/platform/logging.h"
33 #include "tensorflow/core/platform/macros.h"
34 #include "tensorflow/core/platform/mutex.h"
35 #include "tensorflow/core/platform/thread_annotations.h"
36 #include "tensorflow/core/platform/types.h"
37
38 namespace tensorflow {
39
40 class GetSessionHandleOp : public OpKernel {
41 public:
42     explicit GetSessionHandleOp(OpKernelConstruction* context)
43         : OpKernel(context) {}
44
45     void Compute(OpKernelContext* ctx) override {
46         const Tensor& val = ctx->input(0);
47         auto session_state = ctx->session_state();
48         OP_REQUIRES(ctx, session_state != nullptr,
49                     errors::FailedPrecondition(
50                         "GetSessionHandle called on null session state"));
51         int64_t id = session_state->GetNewId();
52         TensorStore::TensorAndKey tk{val, id, requested_device()};
53         OP_REQUIRES_OK(ctx, ctx->tensor_store()->AddTensor(name(), tk));
54
55         Tensor* handle = nullptr;
56         OP_REQUIRES_OK(ctx, ctx->allocate_output(0, TensorShape({}), &handle));
57         if (ctx->expected_output_dtype(0) == DT_RESOURCE) {
58             ResourceHandle resource_handle = MakeResourceHandle<Tensor>(
59                 ctx, SessionState::kTensorHandleResourceTypeName,
60                 tk.GetHandle(name()));
61             resource_handle.set_maybe_type_name(
62                 SessionState::kTensorHandleResourceTypeName);
63             handle->scalar<ResourceHandle>()() = resource_handle;
64         } else {
65             // Legacy behavior in V1.
66             handle->flat<tstring>().setConstant(tk.GetHandle(name()));
67         }
68     }
69
70     TF_DISALLOW_COPY_AND_ASSIGN(GetSessionHandleOp);
71 };
72
73 REGISTER_KERNEL_BUILDER(Name("GetSessionHandle").Device(DEVICE_CPU),
74                         GetSessionHandleOp);
75 REGISTER_KERNEL_BUILDER(Name("GetSessionHandleV2").Device(DEVICE_CPU),
76                         GetSessionHandleOp);
77
78 #define REGISTER_DEFAULT_KERNEL(type) \

```

```

79 REGISTER_KERNEL_BUILDER(Name("GetSessionHandle") \
80                          .Device(DEVICE_DEFAULT) \
81                          .HostMemory("handle") \
82                          .TypeConstraint<type>("T"), \
83                          GetSessionHandleOp) \
84 REGISTER_KERNEL_BUILDER(Name("GetSessionHandleV2") \
85                          .Device(DEVICE_DEFAULT) \
86                          .HostMemory("handle") \
87                          .TypeConstraint<type>("T"), \
88                          GetSessionHandleOp)
89
90 TF_CALL_NUMBER_TYPES(REGISTER_DEFAULT_KERNEL);
91 REGISTER_DEFAULT_KERNEL(bool);
92 #undef REGISTER_DEFAULT_KERNEL
93
94 class GetSessionTensorOp : public OpKernel {
95 public:
96     explicit GetSessionTensorOp(OpKernelConstruction* context)
97         : OpKernel(context) {}
98
99     void Compute(OpKernelContext* ctx) override {
100         const Tensor& handle = ctx->input(0);
101         const string& name = handle.scalar<tstring>()();
102         Tensor val;
103         auto session_state = ctx->session_state();
104         OP_REQUIRES(ctx, session_state != nullptr,
105                     errors::FailedPrecondition(
106                         "GetSessionTensor called on null session state"));
107         OP_REQUIRES_OK(ctx, session_state->GetTensor(name, &val));
108         ctx->set_output(0, val);
109     }
110
111     TF_DISALLOW_COPY_AND_ASSIGN(GetSessionTensorOp);
112 };
113
114 REGISTER_KERNEL_BUILDER(Name("GetSessionTensor").Device(DEVICE_CPU),
115                         GetSessionTensorOp);
116
117 #define REGISTER_DEFAULT_KERNEL(type) \
118     REGISTER_KERNEL_BUILDER(Name("GetSessionTensor") \
119                             .Device(DEVICE_DEFAULT) \
120                             .HostMemory("handle") \
121                             .TypeConstraint<type>("dtype"), \
122                             GetSessionTensorOp)
123
124 TF_CALL_NUMBER_TYPES(REGISTER_DEFAULT_KERNEL);
125 REGISTER_DEFAULT_KERNEL(bool);
126 #undef REGISTER_DEFAULT_KERNEL
127

```

```

128 class DeleteSessionTensorOp : public OpKernel {
129     public:
130         explicit DeleteSessionTensorOp(OpKernelConstruction* context)
131             : OpKernel(context) {}
132
133         void Compute(OpKernelContext* ctx) override {
134             const Tensor& handle = ctx->input(0);
135             const string& name = handle.scalar<tstring>()();
136             auto session_state = ctx->session_state();
137             OP_REQUIRES(ctx, session_state != nullptr,
138                 errors::FailedPrecondition(
139                     "DeleteSessionTensor called on null session state"));
140             OP_REQUIRES_OK(ctx, session_state->DeleteTensor(name));
141         }
142
143         TF_DISALLOW_COPY_AND_ASSIGN(DeleteSessionTensorOp);
144     };
145
146     REGISTER_KERNEL_BUILDER(Name("DeleteSessionTensor").Device(DEVICE_CPU),
147                             DeleteSessionTensorOp);
148     REGISTER_KERNEL_BUILDER(
149         Name("DeleteSessionTensor").Device(DEVICE_DEFAULT).HostMemory("handle"),
150         DeleteSessionTensorOp);
151
152 } // namespace tensorflow

```