/\* Copyright 2016 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.

==============================================================================\*/

syntax = "proto3";

package tensorflow;

import "tensorflow/core/framework/device\_attributes.proto";

import "tensorflow/core/framework/graph.proto";

import "tensorflow/core/framework/tensor.proto";

import "tensorflow/core/protobuf/config.proto";

import "tensorflow/core/protobuf/error\_codes.proto";

import "tensorflow/core/protobuf/named\_tensor.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "DistributedRuntimeProtos";

option java\_multiple\_files = true;

option java\_package = "org.tensorflow.distruntime";

option go\_package = "github.com/tensorflow/tensorflow/tensorflow/go/core/protobuf/for\_core\_protos\_go\_proto";

////////////////////////////////////////////////////////////////////////////////

//

// CreateSession method request/response protos.

//

////////////////////////////////////////////////////////////////////////////////

message CreateSessionRequest {

// The initial graph definition.

GraphDef graph\_def = 1;

// Configuration options.

ConfigProto config = 2;

// The target string used from the client's perspective.

string target = 3;

}

message CreateSessionResponse {

// The session handle to be used in subsequent calls for the created session.

//

// The client must arrange to call CloseSession with this returned

// session handle to close the session.

string session\_handle = 1;

// The initial version number for the graph, to be used in the next call

// to ExtendSession.

int64 graph\_version = 2;

}

////////////////////////////////////////////////////////////////////////////////

//

// ExtendSession method request/response protos.

//

// The "graph\_def" specifies a set of nodes to be added to the session's graph.

//

// A typical "graph\_def" will contain:

//

// \* Zero or more new nodes with names that do not exist in the server-side

// graph. These will be added to the graph.

//

// PRECONDITION: The server-side current version is req.current\_version.

// None of the names in req.graph\_def appeared in previous successful calls to

// CreateSession or ExtendSession with the same session\_handle.

// POSTCONDITION: The server-side current version is resp.new\_version.

//

////////////////////////////////////////////////////////////////////////////////

message ExtendSessionRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// REQUIRED: The nodes to be added to the session's graph. If any node has

// the same name as an existing node, the operation will fail with

// ILLEGAL\_ARGUMENT.

GraphDef graph\_def = 2;

// REQUIRED: The version number of the graph to be extended. This will be

// tested against the current server-side version number, and the operation

// will fail with FAILED\_PRECONDITION if they do not match.

int64 current\_graph\_version = 3;

}

message ExtendSessionResponse {

// TODO(mrry): Return something about the operation?

// The new version number for the extended graph, to be used in the next call

// to ExtendSession.

int64 new\_graph\_version = 4;

}

////////////////////////////////////////////////////////////////////////////////

//

// RunStep method request/response protos.

//

// The caller should provide the feeds needed by the graph and specify

// what nodes should be fetched.

//

////////////////////////////////////////////////////////////////////////////////

message RunStepRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// Tensors to be fed in the step. Each feed is a named tensor.

repeated NamedTensorProto feed = 2;

// Fetches. A list of tensor names. The caller expects a tensor to

// be returned for each fetch[i] (see RunStepResponse.tensor). The

// order of specified fetches does not change the execution order.

repeated string fetch = 3;

// Target Nodes. A list of node names. The named nodes will be run

// to but their outputs will not be fetched.

repeated string target = 4;

// Options for the run call.

RunOptions options = 5;

// Partial run handle (optional). If specified, this will be a partial run

// execution, run up to the specified fetches.

string partial\_run\_handle = 6;

// If true then some errors, e.g., execution errors that have long

// error messages, may return an OK RunStepResponse with the actual

// error saved in the status\_code/status\_error\_message fields of the

// response body. This is a workaround since the RPC subsystem may

// truncate long metadata messages.

bool store\_errors\_in\_response\_body = 7;

// Unique identifier for this request. Every RunStepRequest must

// have a unique request\_id, and retried RunStepRequest must have

// the same request\_id. If request\_id is zero, retry detection is disabled.

int64 request\_id = 8;

}

message RunStepResponse {

// NOTE: The order of the returned tensors may or may not match

// the fetch order specified in RunStepRequest.

repeated NamedTensorProto tensor = 1;

// Returned metadata if requested in the options.

RunMetadata metadata = 2;

// If store\_errors\_in\_response\_body is true in the request, then

// optionally the server may return an OK status for the RPC and

// fill the true status into the fields below, to allow for messages

// that are too long to fit in metadata.

error.Code status\_code = 3;

string status\_error\_message = 4;

}

////////////////////////////////////////////////////////////////////////////////

//

// PartialRunSetup method request/response protos.

//

// The caller should provide the future partial run feeds, fetches, and targets.

// Then the caller can use RunStepRequest with is\_partial set to make partial

// run calls.

//

////////////////////////////////////////////////////////////////////////////////

message PartialRunSetupRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// Tensors to be fed in future steps.

repeated string feed = 2;

// Fetches. A list of tensor names. The caller expects a tensor to be returned

// for each fetch[i] (see RunStepResponse.tensor), for corresponding partial

// RunStepRequests. The order of specified fetches does not change the

// execution order.

repeated string fetch = 3;

// Target Nodes. A list of node names. The named nodes will be run in future

// steps, but their outputs will not be fetched.

repeated string target = 4;

// Unique identifier for this request. Every PartialRunSetupRequest must

// have a unique request\_id, and retried PartialRunSetupRequest must have

// the same request\_id. If request\_id is zero, retry detection is disabled.

int64 request\_id = 5;

}

message PartialRunSetupResponse {

// The unique handle corresponding to the ongoing partial run call setup by

// the invocation to PartialRunSetup. This handle may be passed to

// RunStepRequest to send and receive tensors for this partial run.

string partial\_run\_handle = 1;

}

////////////////////////////////////////////////////////////////////////////////

//

// CloseSession method request/response protos.

//

////////////////////////////////////////////////////////////////////////////////

message CloseSessionRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

}

message CloseSessionResponse {}

// Reset() allows misbehaving or slow sessions to be aborted and closed, and

// causes their resources eventually to be released. Reset() does not wait

// for the computations in old sessions to cease; it merely starts the

// process of tearing them down. However, if a new session is started after

// a Reset(), the new session is isolated from changes that old sessions

// (started prior to the Reset()) may continue to make to resources, provided

// all those resources are in containers listed in "containers".

//

// Old sessions may continue to have side-effects on resources not in

// containers listed in "containers", and thus may affect future

// sessions' results in ways that are hard to predict. Thus, if well-defined

// behavior is desired, is it recommended that all containers be listed in

// "containers". Similarly, if a device\_filter is specified, results may be

// hard to predict.

message ResetRequest {

// A list of container names, which may be empty.

//

// If 'container' is not empty, releases resources in the given

// containers in all devices.

//

// If 'container' is empty, releases resources in the default

// container in all devices.

repeated string container = 1;

// When any filters are present, only devices that match the filters

// will be reset. Each filter can be partially specified,

// e.g. "/job:ps" "/job:worker/replica:3", etc.

repeated string device\_filters = 2;

}

message ResetResponse {}

////////////////////////////////////////////////////////////////////////////////

//

// ListDevices method request/response protos.

//

// Returns information about the TensorFlow devices that are available

// to this master.

//

////////////////////////////////////////////////////////////////////////////////

message ListDevicesRequest {

// Optional: session\_handle must be returned by a CreateSession call to the

// same master service.

//

// When session\_handle is empty, the ClusterSpec provided when the master was

// started is used to compute the available devices. If the session\_handle is

// provided but not recognized, an error is returned. Finally, if a valid

// session\_handle is provided, the cluster configuration for that session is

// used when computing the response.

string session\_handle = 1;

}

message ListDevicesResponse {

repeated DeviceAttributes local\_device = 1;

repeated DeviceAttributes remote\_device = 2;

}

////////////////////////////////////////////////////////////////////////////////

//

// MakeCallable method request/response protos.

//

////////////////////////////////////////////////////////////////////////////////

message MakeCallableRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// Options that define the behavior of the created callable.

CallableOptions options = 2;

// Unique identifier for this request. Every MakeCallableRequest must

// have a unique request\_id, and retried MakeCallableRequest must have

// the same request\_id. If request\_id is zero, retry detection is disabled.

int64 request\_id = 3;

}

message MakeCallableResponse {

// A handle to the created callable.

int64 handle = 1;

}

////////////////////////////////////////////////////////////////////////////////

//

// RunCallable method request/response protos.

//

////////////////////////////////////////////////////////////////////////////////

message RunCallableRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// REQUIRED: handle must be returned by a MakeCallable call to the same

// master service.

int64 handle = 2;

// Values of the tensors passed as arguments to the callable, in the order

// defined in the CallableOptions.feed field passed to MakeCallable.

repeated TensorProto feed = 3;

// Unique identifier for this request. Every RunCallableRequest must

// have a unique request\_id, and retried RunCallableRequest must have

// the same request\_id. If request\_id is zero, retry detection is disabled.

int64 request\_id = 4;

}

message RunCallableResponse {

// Values of the tensors returned by the callable, in the order defined in the

// CallableOptions.fetch field passed to MakeCallable.

repeated TensorProto fetch = 1;

// Returned metadata if requested in the options.

RunMetadata metadata = 2;

}

////////////////////////////////////////////////////////////////////////////////

//

// ReleaseCallable method request/response protos.

//

////////////////////////////////////////////////////////////////////////////////

message ReleaseCallableRequest {

// REQUIRED: session\_handle must be returned by a CreateSession call

// to the same master service.

string session\_handle = 1;

// REQUIRED: handle must be returned by a MakeCallable call to the same

// master service.

int64 handle = 2;

}

message ReleaseCallableResponse {}