RFC: Embedding Variable

Drafted by: Chen Ding (candy.dc@alibaba-inc.com)

Designed by: Alibaba PAI Large Scale Learning Team

Background

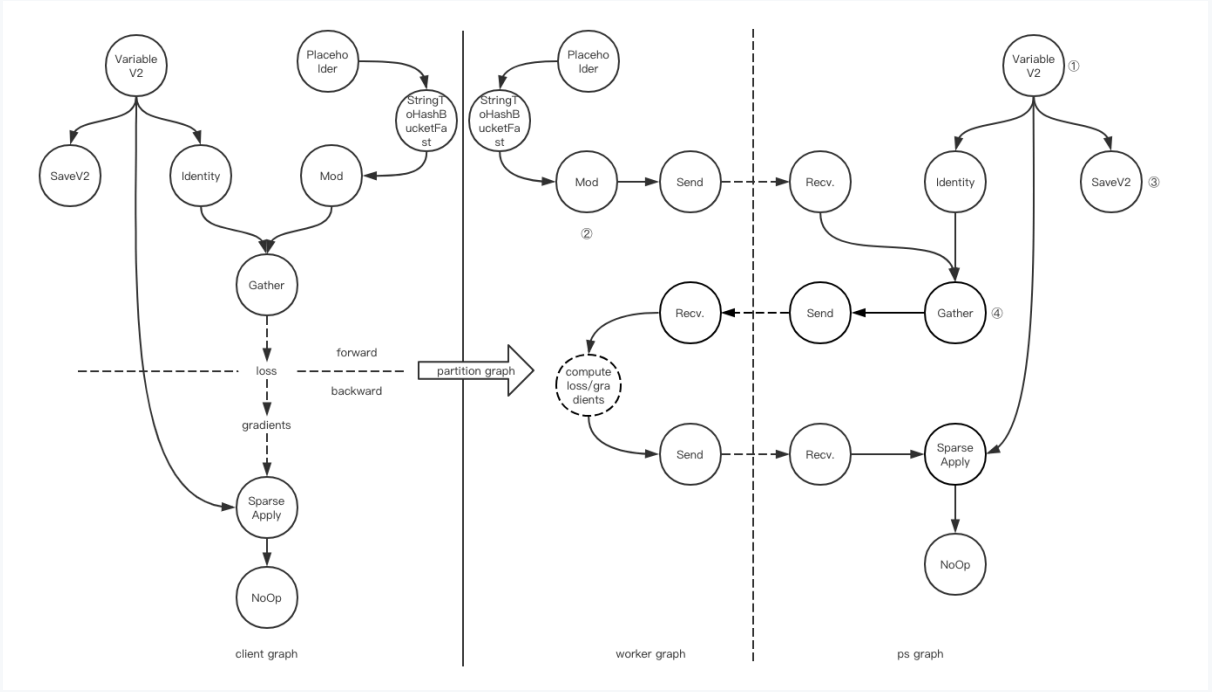
In advertising, search and recommendation scenarios, CTR (Click-Through Rate) is one of the most popular models. Increasing the number of features is an effective approach to improve the performance of the model. Model developers use billions of features to train their model, mostly of features are sparse and need to be converted to a dense vector by embedding. Therefore, it brings some problems in large scale sparse feature training, including memory usage, original feature encoding, model storage, etc.

Problem & Motivation

"In TensorFlow, we model all data as tensors (n-dimensional arrays) …... At the lowest level, all TensorFlow tensors are dense…... " which quote from TensorFlow white paper[1] states that Tensor is the basic data structure in TensorFlow. Every component (Operation, Rendezvous, BundleReader, BundleWriter etc.) in TensorFlow framework depends on Tensor. It works well with the simple interface, clear hierarchy, and elegant design.

In large scale sparse data training, embedding_lookup is the first layer in the train graph, the data storage and transmission is rigid. Usually, users invoke the API and build graph below:

```
...
col0 = tf.feature_column.categorical_column_with_hash_bucket(
          column_name="col0",
          hash_bucket_size=10000)
feature_columns = [tf.feature_column.embedding_column(
                      categorical_column=col0,
                      dimension=16)]
input_data = tf.contrib.layers.input_from_feature_columns(
                columns_to_tensors=features,
                feature_columns=feature_columns)
...
```



In my example, Variable is a single variable, not partitioned variable, some OP (UniqueOp, UnsortedSegmentSumOp) in ps are ignored. VariableV2 holds real data that refer to a dense Tensor whose shape is fixed after users finish building graphs, eg: [10000, 16]. All original features map to the Tensor with a set of Ops, which cause a lot of problems:

1. Memory waste, TensorBuffer may have some empty holes① because valid data in Tensor is sparse.
2. Feature conflict, hash function and mod operator may result in index collision② and make negative effect in model performance.
3. Static tensor shape, it doesn't work in online-learning with the increasing features①. If a user wants to train a model with larger dataset, the embedding table shape can not be changed, otherwise the ckpt does not match with Graph.
4. Inefficient IO, large Tensor makes save/restore process slow③, Tensor should rebuild because sparse data cannot transmiss via Rendervous④.

Objective

TensorFlow framework supports sparse data/KV data storage.

Design Proposal

Overview

We create a KV to store sparse data, named "Embedding Variable (EV)", which is compatible with graph execution.

1. EV, like Variable, is stateful operation. It stores sparse parameters. It is only used in Embedding Layers.
2. EV can connect necessary Operations to build a whole graph, such as: Gather, SparseApply, Save, Send Op.
3. EV's first dimension is dynamic. EV can enlarge or shrink in reasonable range according to the user's configuration.
4. EV send/receive via rendezvous without extra memory copy.
5. EV supports incremental checkpoint to speed up model save/restore and save disk space.

API

Two level API:

- Low-level (like tf.get_variable)

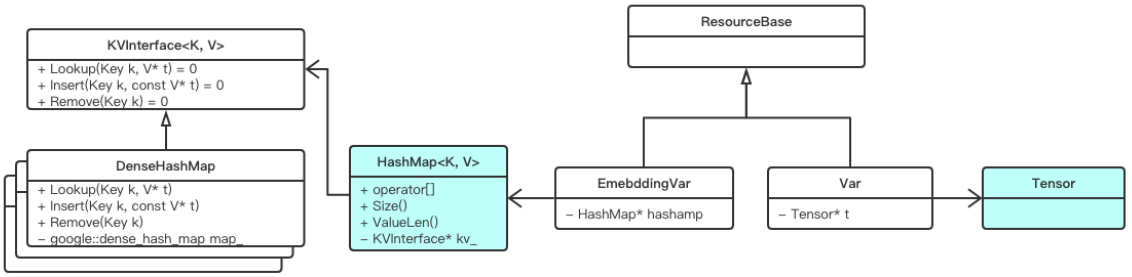
```
# tf.Variable
def get_embedding_variable(name,
                           embedding_dim,
                           key_dtype=dtypes.int64,
                           value_dtype=None,
                           initializer=None,
                           regularizer=None,
                           trainable=True,
                           collections=None,
                           caching_device=None,
                           partitioner=None,
                           validate_shape=True,
                           custom_getter=None,
                           constraint=None,
                           steps_to_live=None):
  # @embedding_dim: embedding dimension size
  # @steps_to_live: ev lifetime
  # create an embedding variable or a partitioned embedding variable
```

- High-level (like tf.feature_column.categorical_column_with_identity)

```
# feature_column
def tf.feature_column.categorical_column_with_embedding(column_name):
    # create an embedding column
```

Detailed Design

Framework



Graph/Operations

1. A new set of Operations
    - EVHandleOp
    - EVShapeOp
    - DestroyEVOp
    - InitializeEVOp
    - EVIsInitializedOp
    - EVGatherOp
    - EVSparseApplyAdagrad(Adam/Ftrl/...)Op
2. A new set of compute_gradiend subgraph
    - ops.RegisterGradient("EVGatherOp")
3. We regard EV as trainable variable and compute gradients using TensorFlow auto-differential

Resource

```
class EmbeddingVar : public ResourceBase {
   private:
      HashMap<K, V>* hash_map_;
};
```

```
template <class K, class V>
class HashMap {
 public:
   typename TTypes<V>::Flat flat(K key);
};
```

Primary OpDef

```
REGISTER_OP("KvVarHandleOp")
   .Attr("container: string = ''")
   .Attr("shared_name: string = ''")
   .Attr("dtype: type")
   .Attr("shape: shape")
   .Attr("Tkeys: {int64, int32}")
   .Output("resource: resource")
   .SetIsStateful()
```

```
REGISTER_OP("KvResourceGather")
   .Input("resource: resource")
   .Input("indices: Tkeys")
   .Input("default_value: dtype")
   .Attr("validate_indices: bool = true")
   .Output("output: dtype")
   .Attr("dtype: type")
   .Attr("Tkeys: {int64, int32}")
```

```
REGISTER_OP("KvResourceSparseApplyAdagrad")
   .Input("var: resource")
   .Input("accum: resource")
```

```
    .Input("lr: T")
    .Input("grad: T")
    .Input("indices: Tindices")
    .Attr("T: numbertype")
    .Attr("Tindices: {int32, int64}")
    .Attr("use_locking: bool = false")
    .Attr("update_slots: bool = true")
```

OpKernel Impl.

KvVarGather op new impl.
Optimizers opkernel minor change

```
template <typename TKey, typename T, typename ResourceVar>
class ResourceSparseApplyAdagradOp : public OpKernel {
    // ResourceVar can be Tensor or HashMap
    ResourceVar rv = ...
    // this op kernel change is minor
    // Tensor or HashMap have same Interface(flat(K key))
}

REGISTER_KERNEL_BUILDER(Name("ResourceSparseApplyAdagrad")      \
                        .Device(DEVICE_CPU)                     \
                        .TypeConstraint<T>("T")                 \
                        .TypeConstraint<Tindices>("Tindices"), \
                    ResourceSparseApplyAdagradOp<Tindices, T, Tensor>);
REGISTER_KERNEL_BUILDER(Name("KvResourceSparseApplyAdagrad")      \
                        .Device(DEVICE_CPU)                     \
                        .TypeConstraint<T>("T")                 \
                        .TypeConstraint<Tindices>("Tindices"), \
                    ResourceSparseApplyAdagradOp<Tindices, T, HashMap>);
```

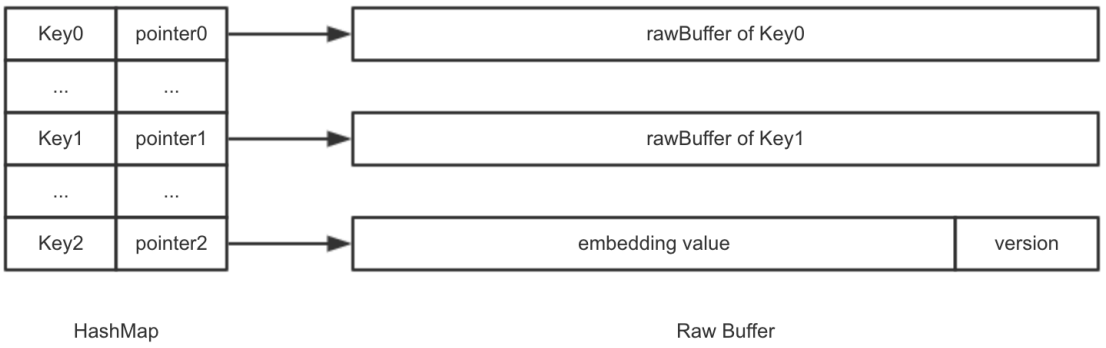
Optimizer/Processor

```
class _SparseResourceVariableProcessor(_OptimizableVariable)
```

Data Structure

Embedding Variable consists of an efficient hashmap and rawBuffers, hashmap's value indicates the pointer of rawBuffer, memory allocation happens during insertion of a new KV (feature).

Memory Management



- Use 3rd-party hashmap as internal HashMap, such: google::dense_hash_map
    - Do some optimizations outside the hashmap
    - or more effective hashmap?
- Use tensorflow::allocator as RawBuffer‘s allocator directly, not Tensor / TensorBuffer
- RawBuffer contains embedding_value and version
    - embedding value: storage parameter for training
    - version: storage global_step in which latest kv updates, indicate whether keep this kv or not, it works with steps_to_live parameter, remove kv if the kv is out of date
- Behavior
    - lookup
        - if key not in hashmap, insert a new key-value according users config, and return value
        - else, return existing value
        - invoked by EVGatherOp
    - update
        - update existing value
        - invoked by EVSparseApplyXXXOp
    - shrink
        - according parameter steps_to_live to remove expired key-value, do it in some time interval, such as save_checkpoint_secs

        - invoked by TF framework, such TF::Hook

EV Send/Recv. in Network

ZeroCopy in Rendezvous

- EVGatherOp's output is not a dense Tensor, is a set of {pointer, len}, pass to iovec directly without any memory copy, See Appendix[3]

EV Persistence in Checkpoint

SaveV2 support EV (EVHandleOp)

- SaveV2 Op support resource as input Tensor, flush to disk without rebuild a new Tensor

Incremental Checkpoint

- For Sparse parameter update, saving whole data to ckpt is redundant. It is more efficient to save updated kv data.
    - checkpoint: full checkpoint + delta checkpoint (like checkpoint + oplog)
    - Save: save full checkpoint per 1 hour, save delta checkpoint per 10min
    - Restore: full checkpoint + a series of delta checkpoints
- This point is a part of sparse data checkpoint design, proposed in other RFCs

Appendix

GitHub Issue

[1] https://github.com/tensorflow/tensorflow/issues/19324

[2] https://github.com/tensorflow/tensorflow/issues/24539

[3] https://github.com/tensorflow/tensorflow/issues/24575

Reference

[1] https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf