To enable random access to TFRecords, we implement four new functions and modify two existing functions as explained below. You can read our paper "Analyzing the Interplay Between Random Shuffling and Storage Devices for Efficient Machine Learning" for more details.

I. Implementation of LIRS with TFRecord file format

1. TFRecordLIRS.read(instance id)

This new module reads out a training instance with the specified instance ID when being called. It takes the specified instance ID as input and searches the offset table, which records the offset of each instance, to get the offset and instance length calculated by Equation (1). It then reads and returns the instance. This function is used inside the ***TFRecordLIRS.next()***, which gets a random instance when being called. Programmers do not need to directly use this function to randomly shuffle TFRecords, but they can still use it to read out any instance they want to access.

instance length = next instance offset - current instance offset (1)

2. TFRecordLIRS.next()

This modified module returns training instances in random order when being called. It should be used after the training instances are randomly shuffled (i.e., after using ***TFRecordLIRS.shuffle_order(page_aware)***). Instead of returning the instance on the top of the TensorFlow input pipeline, it returns instances following the order generated by ***TFRecordLIRS.shuffle_order(page_aware)***.

3. OffsetTableBuilder.calc_offset(instance)

This new module uses Equation (2) to calculate the file offset of each instance and store it inside an array (i.e., the offset table). The maximum size of this array (i.e., the size of the offset table) is shown in Equation (3). Calculating the offset of each training instance is necessary when the training instances are stored in sparse format. If a user decides to apply padding-based page alignment to eliminate redundant page reads, the offset of each instance is determined by the padding algorithm (e.g., next-fit, best-fit, etc) rather than Equation (2).

current instance offset = previous instance offset + previous instance length (2)

Offset table size = # of instances * 8 Bytes (3)

4. OffsetTableBuilder.write(offset_table_filename)

This new module writes the offset table built by OffsetTableBuilder.cal_offset() to an offset table file. The offset table file will be used during training to locate the position of each training instance in the file.

5. TFRecordLIRS.shuffle_order(page_aware)

This new module randomly shuffles the order of the training instances in the entire dataset. Users can decide to apply page-aware random shuffling (page_aware = 1) or instance-based random shuffling (page_aware = 0). Page-aware random shuffling uses a page as the minimum random shuffling unit and groups the training instances within the same page into the same batch. This approach sacrifices some degree of randomness but can prevent redundant page transfers that happen when a page is evicted from caches before all the training instances within that page are trained.

6. TFRecordWrite.write(instance or padding)

This modified module writes an instance or padding to the TFRecord file.

II. Examples

The following examples show how to apply our implementation to train a DNN model using randomly shuffled training data stored in TFRecord file format. The first example shows how to construct a TFRecord file that supports random access.

```
input: instances [], whether_to_use_padding
output: offset_table_file, TFRecord_file

offset_table = OffsetTableBuilder ( whether_to_use_padding )

for all instance in instances []:
        # …
        # Do whatever preprocessing you want, e.g., resizing the image, filtering low-quality data
        # …
        if  whether_to_use_padding is True:
                # write the padding to TFRecord_file
                TFRecordWriter.write ( padding )
        # calculate the offset of this instance
        OffsetTableBuilder.calc_offset ( instance )
        # write the instance to TFRecord_file
        TFRecordWriter.write ( instance )

OffsetTableBuiler.write ( offset_table_file )
```

The second example shows how to train the model with the constructed TFRecord file and offset table using the random shuffling function we provided. Note that whether to use padding is decided when constructing the TFRecord file. As the offset table contains the location of each training instance, we do not need to know whether the dataset is padded (i.e., page-aligned) during training.

```
input: offset_table_file, TFRecord_file, whether_page_aware
output: the trained model

# Initialize the TFRrcordLIRS object, this initialization also opens the TFRecord file and the offset table file
train_dataset = TFRecordLIRS ( TFRecord_file, offset_table_file )

for number of epochs:
    train_dataset.shuffle_order ( whether_page_aware )
    for number of steps:
        batch = []
        for batch size:
            # form a batch by calling next()
            batch.append ( train_dataset.next ( ) )
        # note that this train step is executed by GPU
        train ( model, batch )
```