# TensorFlow– A system for large-scale machine learning

Hong-Yunn Chen

Graduate Institute of Networking and Multimedia (GINM),

National Taiwan University, Taipei, Taiwan

# Introduction to TensorFlow

- Originated from Google DistBelief.
- Uses dataflow graphs to represent computation, shared states, and the operations that mutates that state.
- This paper focuses on neural network training as a challenging system problem

# Previous system: DistBelief

- DistBelief adopts a parameter server architecture.
- A job comprises two disjoint sets of processes: worker process and parameter server process.
  - Worker processes
    - Stateless
    - Performs the bulk of computation when training a model
  - Parameter server process
    - Stateful.
    - Maintain the current version of the model parameters
- DistBelief is similar to Caffe's:
  - User defines a neural network as a Directed acyclic graph (DAG)
  - Terminate with a loss function

# Flexibility

- DistBelief’s layers are C++ classes, which maybe barrier for researcher seeking for new layer architectures.
- Researchers often want to experiment with new optimization models, but doing that in DistBelief involves modifying the parameter server implementation.
- DistBelief workers follow a fixed execution pattern. This pattern works for training simple feed-forward neural networks, but fails for more advanced models, such as recurrent neural networks, which contain loops.

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                 # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                  # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

An image classifier written using TensorFlow's Python API. This program is a simple solution to the MNIST digit classification problem, with 784-pixel images and 10 output classes.

# Design principles

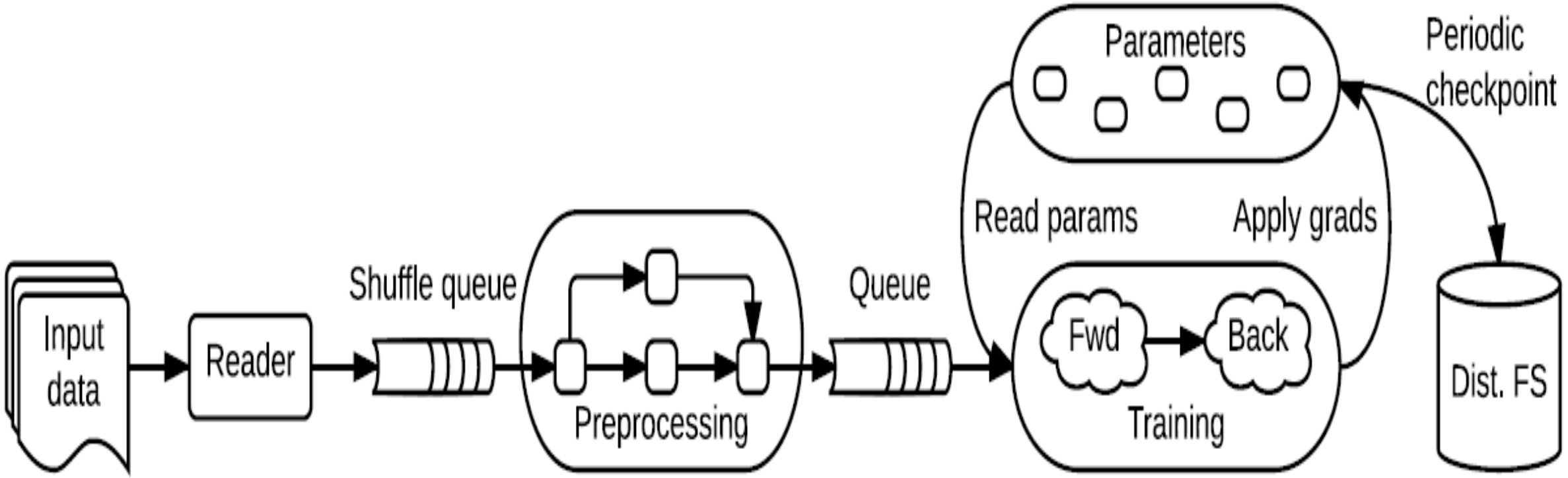
- Both TensorFlow and DistBelief use a dataflow for models. But DistBelief comprises relatively few complex “layers”, whereas the corresponding TensorFlow model represents individual mathematics operators as nodes.
- TensorFlow adopts deferred execution, which has two phases:
  - The first phase defines the program as a symbolic dataflow graph.
  - The second phase executes an optimized version of the program.
- There is no such thing as a parameter server. On a cluster, we deploy TensorFlow as a set of tasks

# Related work

- Single-machine frameworks: Many machine learning researchers carry out their work on a single—often GPU equipped—computer, and several single-machine frameworks support this scenario. Caffe is a high performance framework for training declaratively specified neural networks on multicore CPUs and GPUs.
- Batch dataflow systems: Starting with MapReduce, batch dataflow systems have been applied to a large number of machine learning algorithms, and more recent systems have focused on increasing expressivity and performance.
- Parameter servers: a parameter server architecture uses a set of servers to manage shared state that is updated by a set of parallel workers.

# TensorFlow execution model

- TensorFlow differs from batch dataflow systems in two respects:
  - The model supports multiple concurrent executions on overlapping subgraphs of the overall graph.
  - Individual vertices may have mutable state that can be shared between different executions of the graph.
- Mutable state is crucial when training very large models, because it becomes possible to make in-place updates to very large parameters, and propagate those updates to parallel training steps as quickly as possible.




A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state.

# TensorFlow dataflow graph elements

- In a TensorFlow graph, each vertex represents a unit of local computation, and each edge represents the output from, or input to, a vertex. We refer to the computation at vertices as operations, and the values that flow along edges as tensors.
- Tensors: In TensorFlow, we model all data as tensors (n-dimensional arrays) with the elements having one of a small number of primitive types, such as int32, float32, or string.
- Operations: An operation has a named “type” (such as Const, MatMul, or Assign) and may have zero or more compile-time attributes that determine its behavior.
- Stateful operations: variables An operation can contain mutable state that is read and/or written each time it executes. A Variable operation owns a mutable buffer that may be used to store the shared parameters of a model as it is trained.
- Stateful operations: queues TensorFlow includes several queue implementations, which support more advanced forms of coordination.

# Partial and concurrent execution

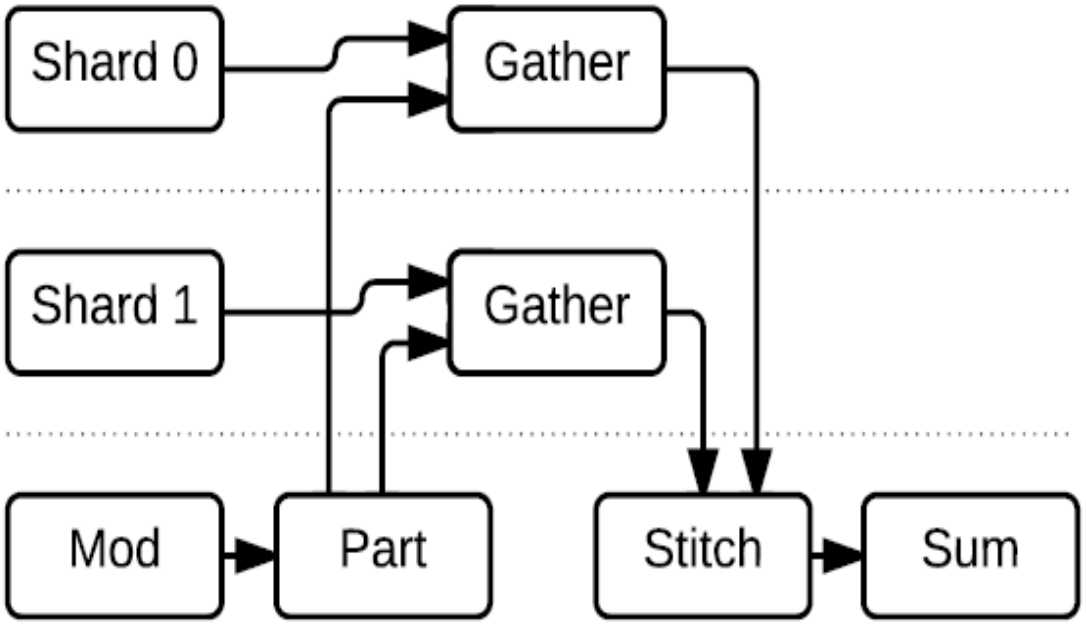
- The API for executing a graph allows the client to specify declaratively the subgraph that should be executed.
- By default, concurrent execution of a TensorFlow subgraph  run asynchronously with respect to one another. This asynchrony makes it straightforward to implement machine learning algorithms with weak consistency requirements.

# Distributed execution

- Each operation resides on a particular device, such as a CPU or GPU in a particular task. A device is responsible for executing a kernel for each operation assigned to it.
- A per-device subgraph for device d contains all of the operations that were assigned to d, with additional Send and Recv operations that replace edges across device boundaries.

# Dynamic control flow

- TensorFlow uses Switch and Merge to control the conditional flow
- Schematic dataflow for an embedding layer with a two-way sharded embedding matrix.



# Differentiation and optimization

- Many learning algorithms train a set of parameters using some variant of SGD, which entails computing the gradients of a loss function with respect to those parameters, then updating the parameters based on those gradients.
- TensorFlow includes a user-level library that differentiates a symbolic expression for a loss function and produces a new symbolic expression representing the gradients.

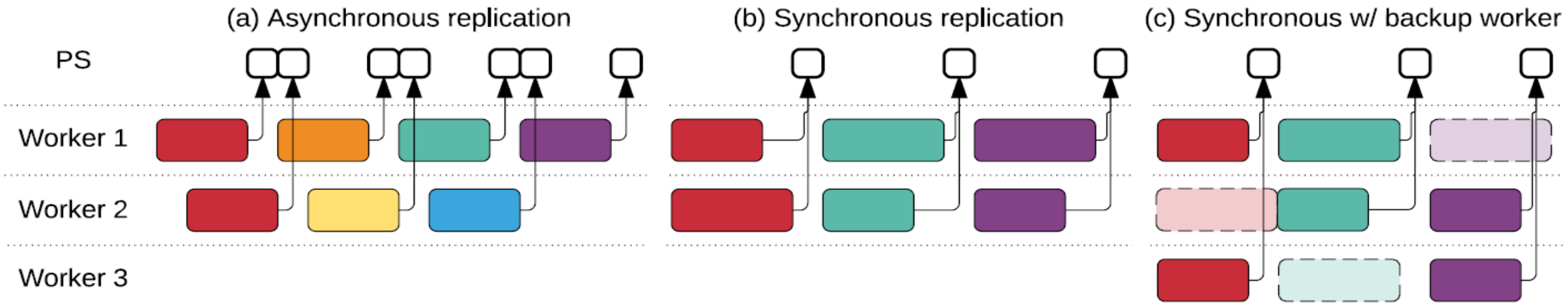
# Training very large models

- To train a model on high-dimensional data, such as words in a corpus of text, it is common to use a distributed representation, which embeds a training example as a pattern of activity across several neurons, and which can be learned by backpropagation.

# Fault tolerance

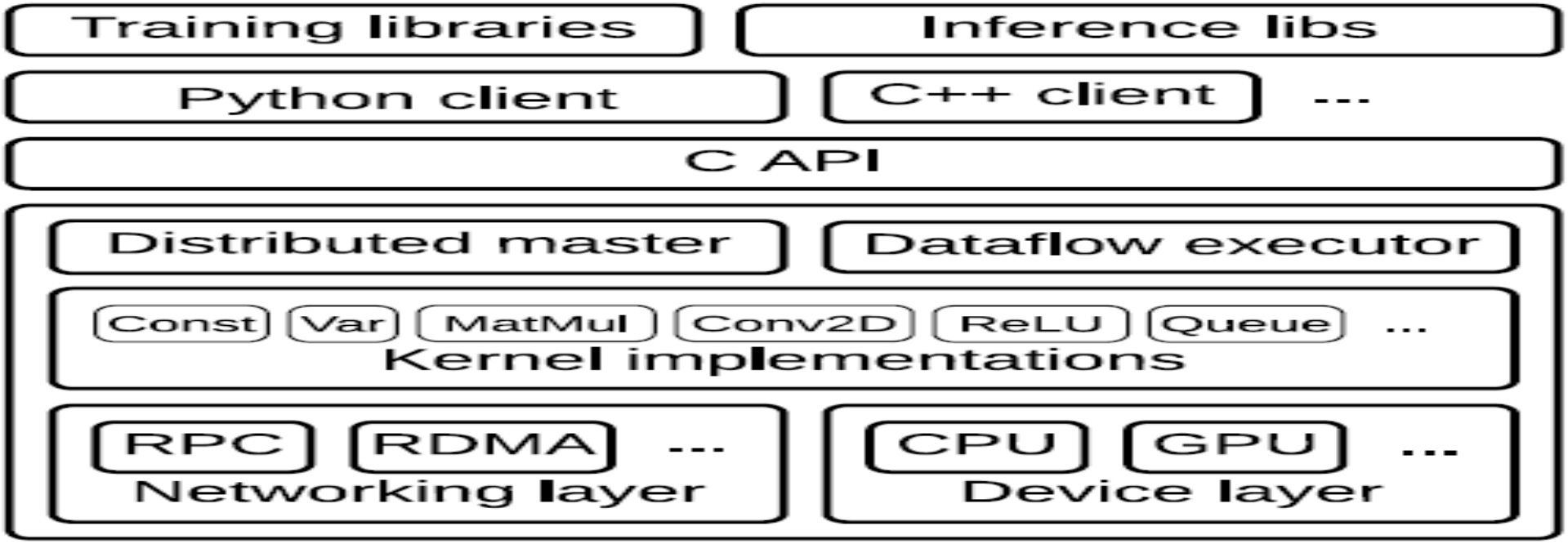
- TensorFlow implements sparse embedding layers in the TensorFlow graph as a composition of primitive operations.
- It is unlikely that tasks will fail so often that individual operations need fault tolerance. So Spark's RDD helps less.
- Many learning algorithms do not require strong consistency.
- TensorFlow implements user-level checkpointing for fault tolerance– Save writes one or more tensors to a checkpoint file, and Restore reads one or more tensors from a check point file.
- The checkpointing library does not attempt to produce consistent checkpoints, which is compatible with the relaxed guarantees of asynchronous SGD.

# Synchronous replica coordination




- Three synchronization schemes for parallel SGD. Each color represents a different starting parameter value; a white square is a parameter update. In (c), a dashed rectangle represents a backup worker whose result is discarded.

# Implementation




- A C API separates user-level code from the core runtime.
- The distributed master translates user requests into execution across a set of tasks
- The dataflow executor in each task handles a request from the master and schedules the execution of the kernels that comprise a local subgraph.

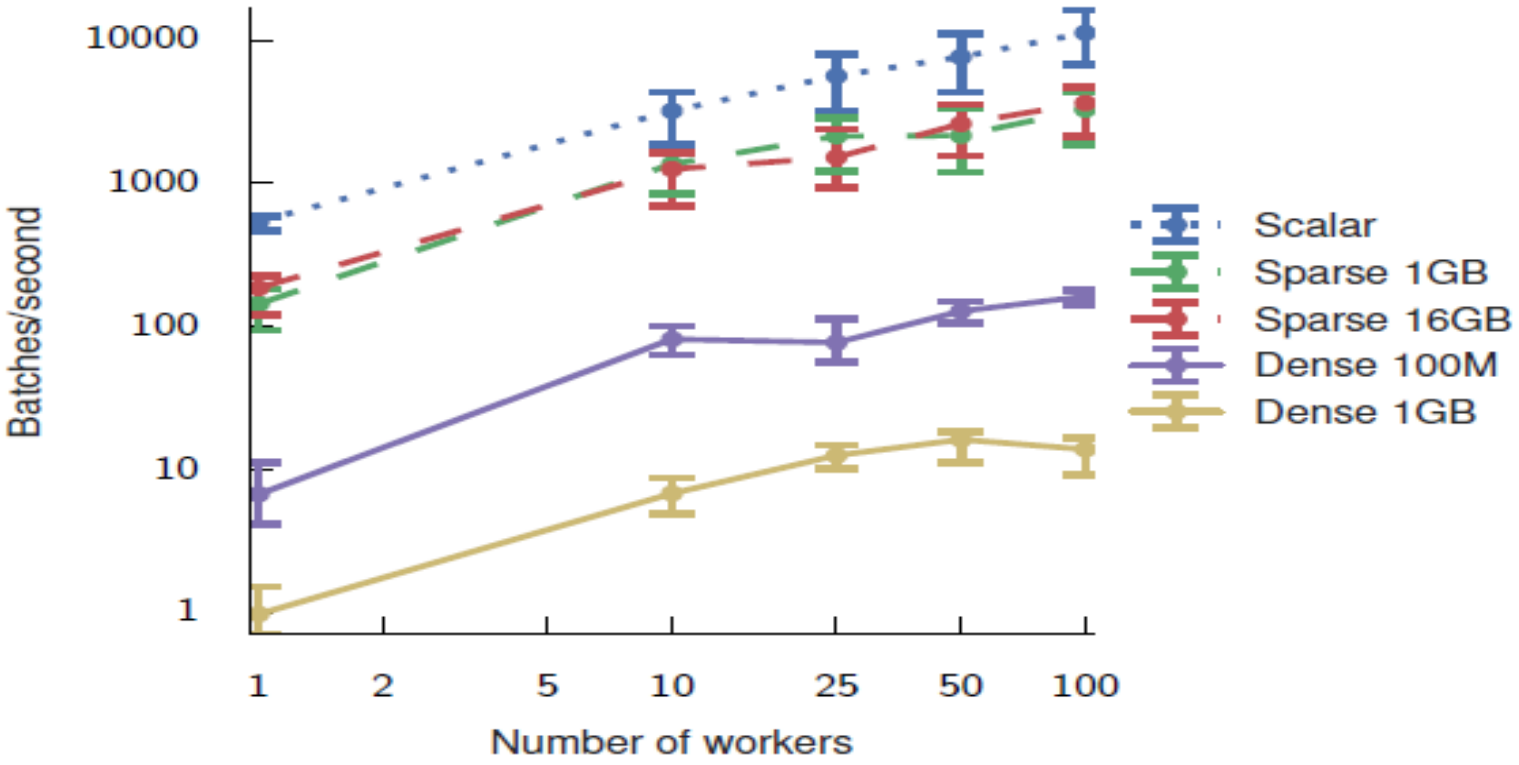
# Evaluation 1: Single-machine benchmarks

- Table 1: Step times for training four convolutional models with different libraries, using one GPU. All results are for training with 32-bit floats. The fastest time for each model is shown in bold.

| Library | Training step time (ms) | | | |
|---|---|---|---|---|
| | AlexNet | Overfeat | OxfordNet | GoogleNet |
| Caffe [38] | 324 | 823 | 1068 | 1935 |
| Neon [58] | 87 | **211** | **320** | **270** |
| Torch [17] | **81** | 268 | 529 | 470 |
| TensorFlow | **81** | 279 | 540 | 445 |

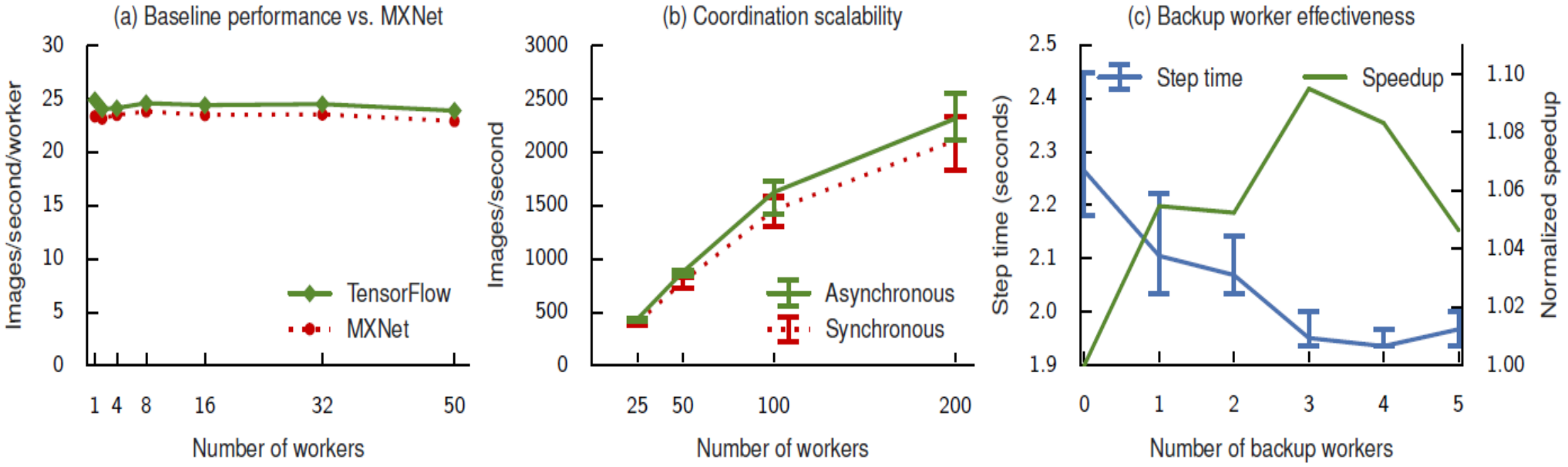
# Evaluation 2: Synchronous replica microbenchmark

- Baseline throughput for synchronous replication with a null model. Sparse accesses enable TensorFlow to handle larger models, such as embedding matrices.



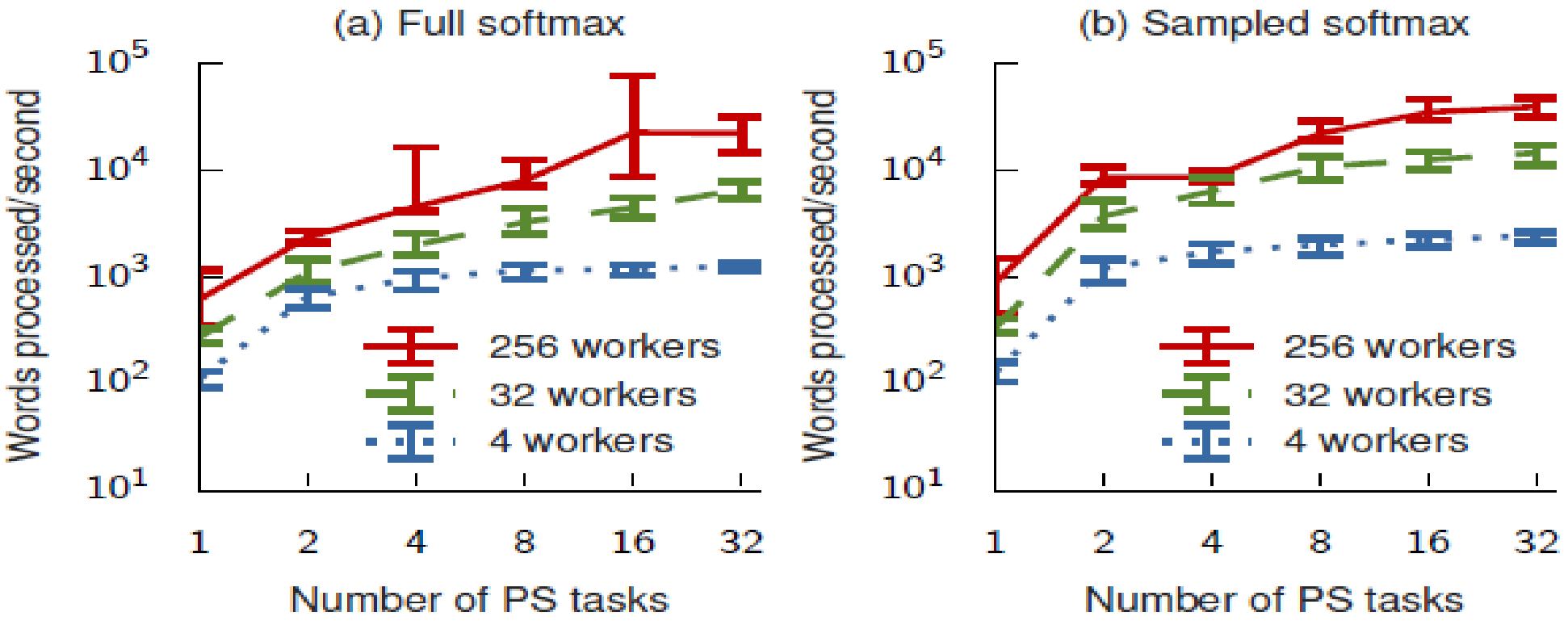
# Evaluation 3: Image classification

- Results of the performance evaluation for Inception-v3 training. (a) TensorFlow achieves slightly better throughput than MXNet for asynchronous training. (b) Asynchronous and synchronous training throughput increases with up to 200 workers. (c) Adding backup workers to a 50-worker training job can reduce the overall step time, and improve performance even when normalized for resource consumption.



# Evaluation 4: Language modeling

- Increasing the number of PS tasks leads to increased throughput for language model training, by parallelizing the softmax computation. Sampled softmax increases throughput by performing less computation.



# Conclusions

- While the current implementations of mutable state and fault tolerance suffice for applications with weak consistency requirements, we expect that some TensorFlow applications will require stronger consistency, and we are investigating how to build such policies at user-level.
- Finally, some users have begun to chafe at the limitations of a static dataflow graph, especially for algorithms like deep reinforcement learning.
- Therefore, we face the intriguing problem of providing a system that transparently and efficiently uses distributed resources, even when the structure of the computation unfolds dynamically.