CERN

Distributed Machine Learning with Apache Spark and Keras

Joeri Hermans (Technical Student)
Maastricht University

Problem Statement

- CMS would like to use a deep learning model in the high level trigger.
- **Specific questions from CMS:**
  - Equivalent amount of CPU cores compared to a single GPU?
  - Can we reduce training time using distributed machine learning?
- CMS use-case requires a lot of data (~1 TB of training data)
- **Keras** (modeling framework in Python, interfacing TensorFlow or Theano).

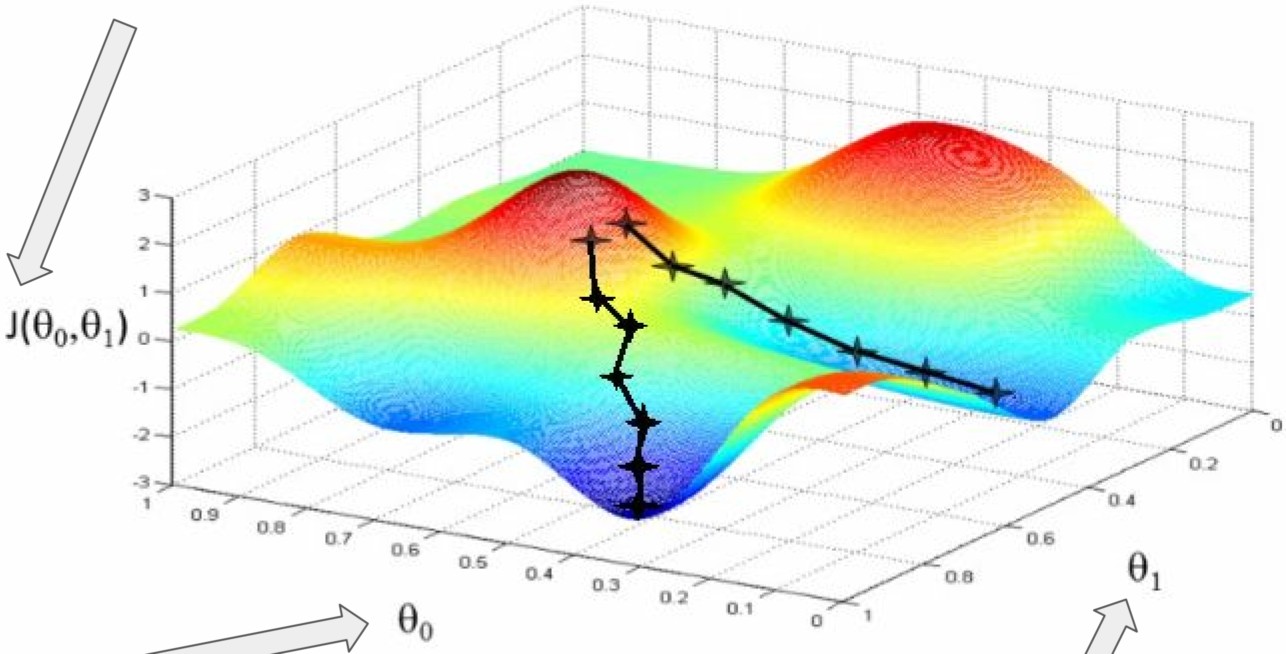
Keras

```
model = Sequential()
model.add(Dense(600, input_shape=(num_features,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(600))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Gradient Descent

Goal: optimize parameters in such a way that it minimizes the error.
Problem: Gradient can only be evaluated locally.
Solution: Iteratively, add gradient (slope) until convergence.

Additional dimension to indicate error of expected value, and predicted value.



A single dimension corresponds with one parameter (weight) of a Neural Network.

Related Work (1)

- Elephas (https://github.com/maxpumperla/elephas):
  - **Fits requirements** (Keras, Apache Spark).
  - Patched to work with Keras 1.0+, but testing indicates that it does not work on a cluster.
  - Distributed **gradient update is naive** (but could not test).
  - Does not support Spark 2.0, nor it supports Spark DataFrames.
  - **Not easily extendable** to implement new algorithms with different communication protocols.

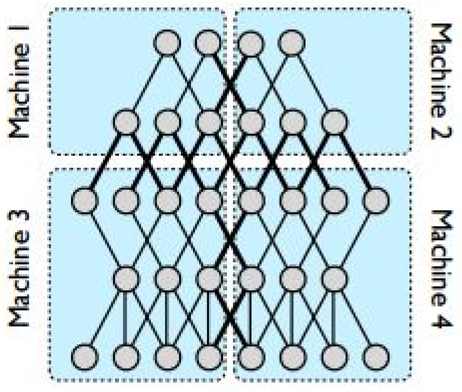
Related Work (2)

- SparkNet (https://github.com/amplab/SparkNet)
  - Built using **Caffe**
  - TensorFlow supported
  - A lot of **freedom**, **but** distributed optimizer needs to be implemented with every model.

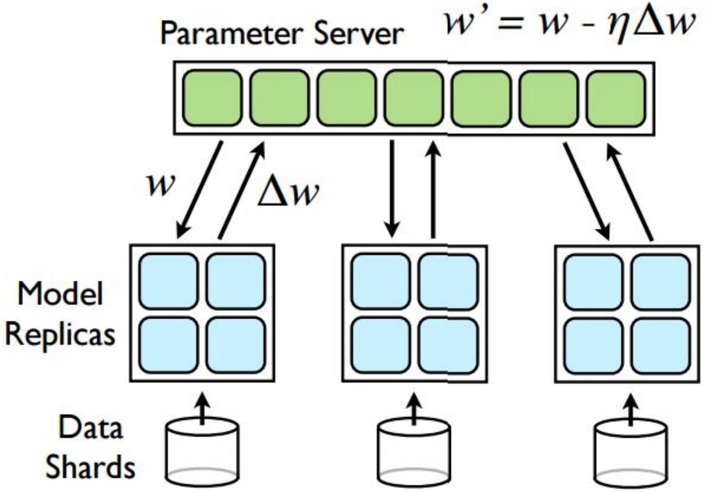
- Deeplearning4j (http://deeplearning4j.org/spark)
  - Java API
  - Again, **naive gradient update** (averaging)

Main approaches

Model parallelism



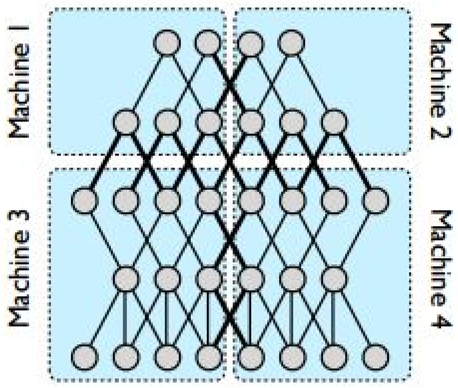
Data parallelism



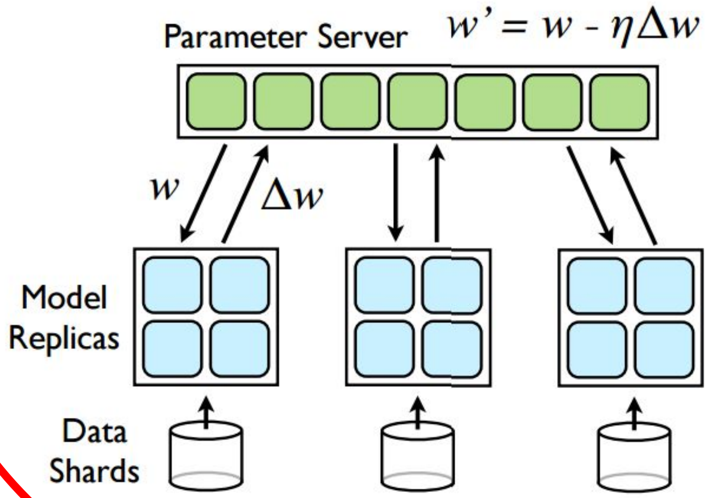
Source: Large scale distributed Deep Networks [1]

Main approaches

Model parallelism

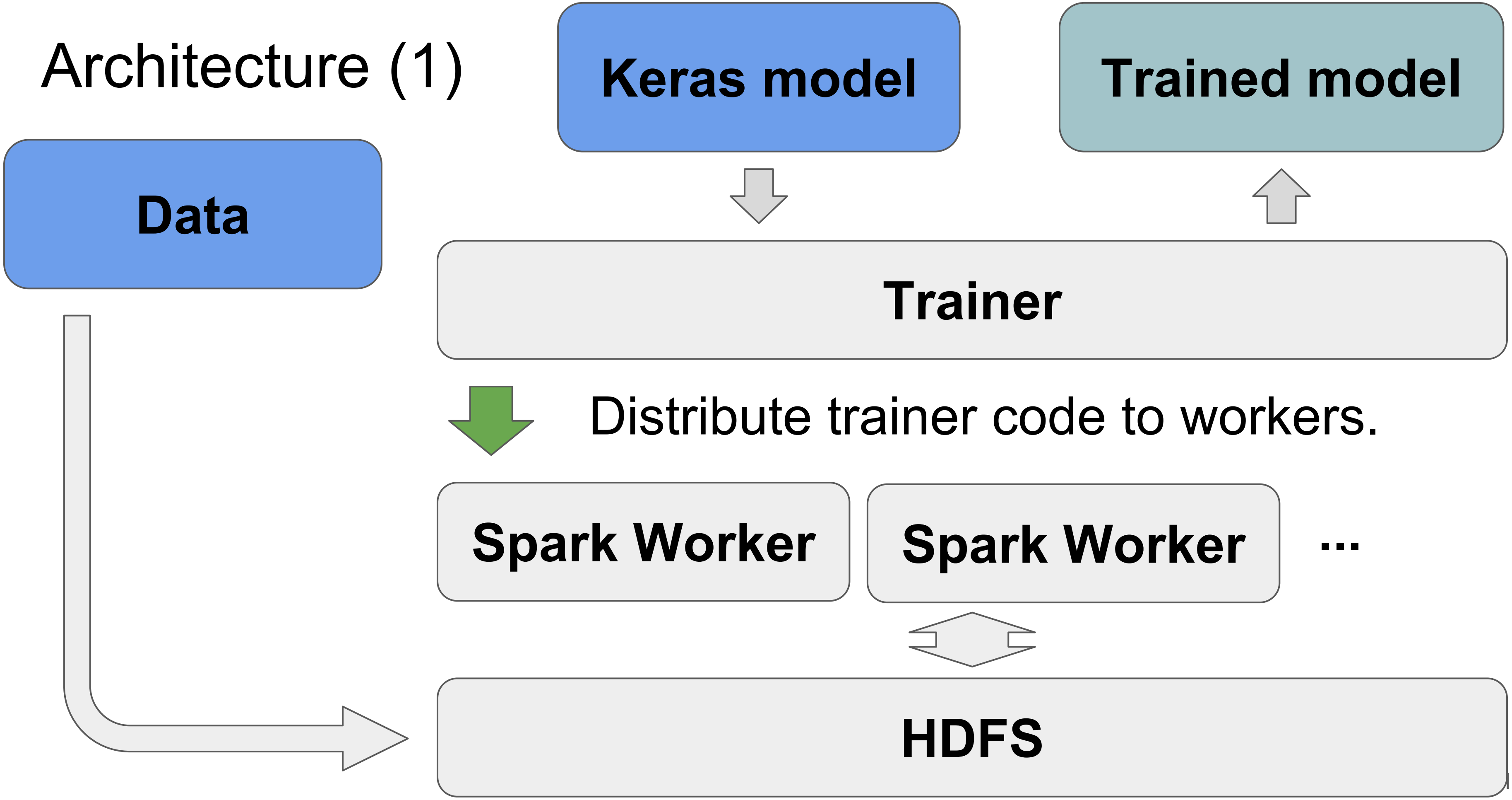


Data parallelism

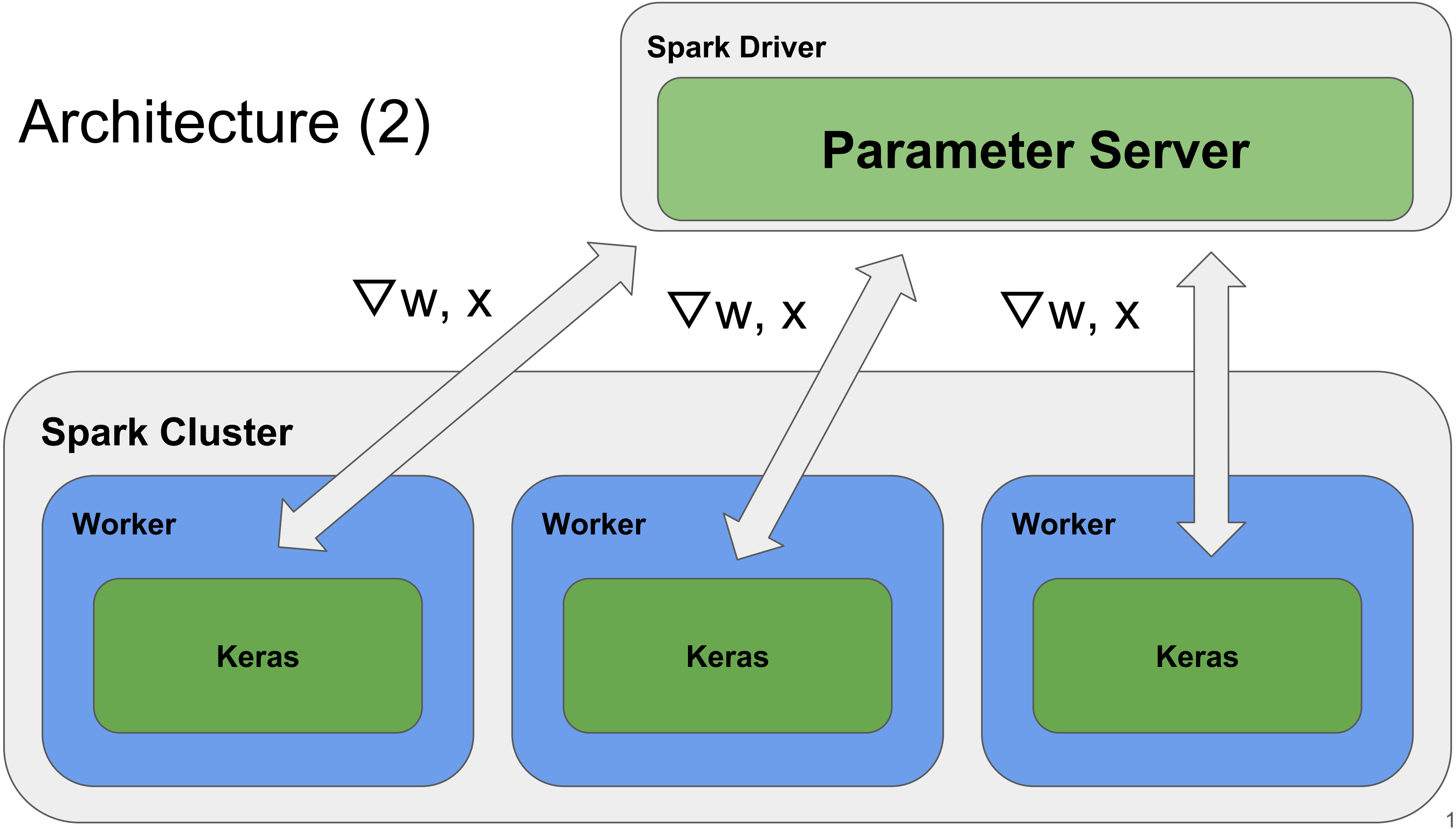


Source: Large scale distributed Deep Networks [1]

Architecture (1)



Architecture (2)



Architecture (3)

```
def train(self, data):
    self.start_service()  # Start parameter server
    worker = self.allocate_worker()
    numPartitions = data.rdd.getNumPartitions()
    if numPartitions > self.num_workers:
        data = data.coalesce(self.num_workers)
    else:
        data = data.repartition(self.num_workers)
    data.rdd.mapPartitionsWithIndex(worker.train).collect() # Train models
    self.stop_service()
```

Worker code as a Spark mapping function.

Source: github.com/JoeriHermans/dist-keras/blob/master/distkeras/distributed.py

Advantages of using Apache Spark

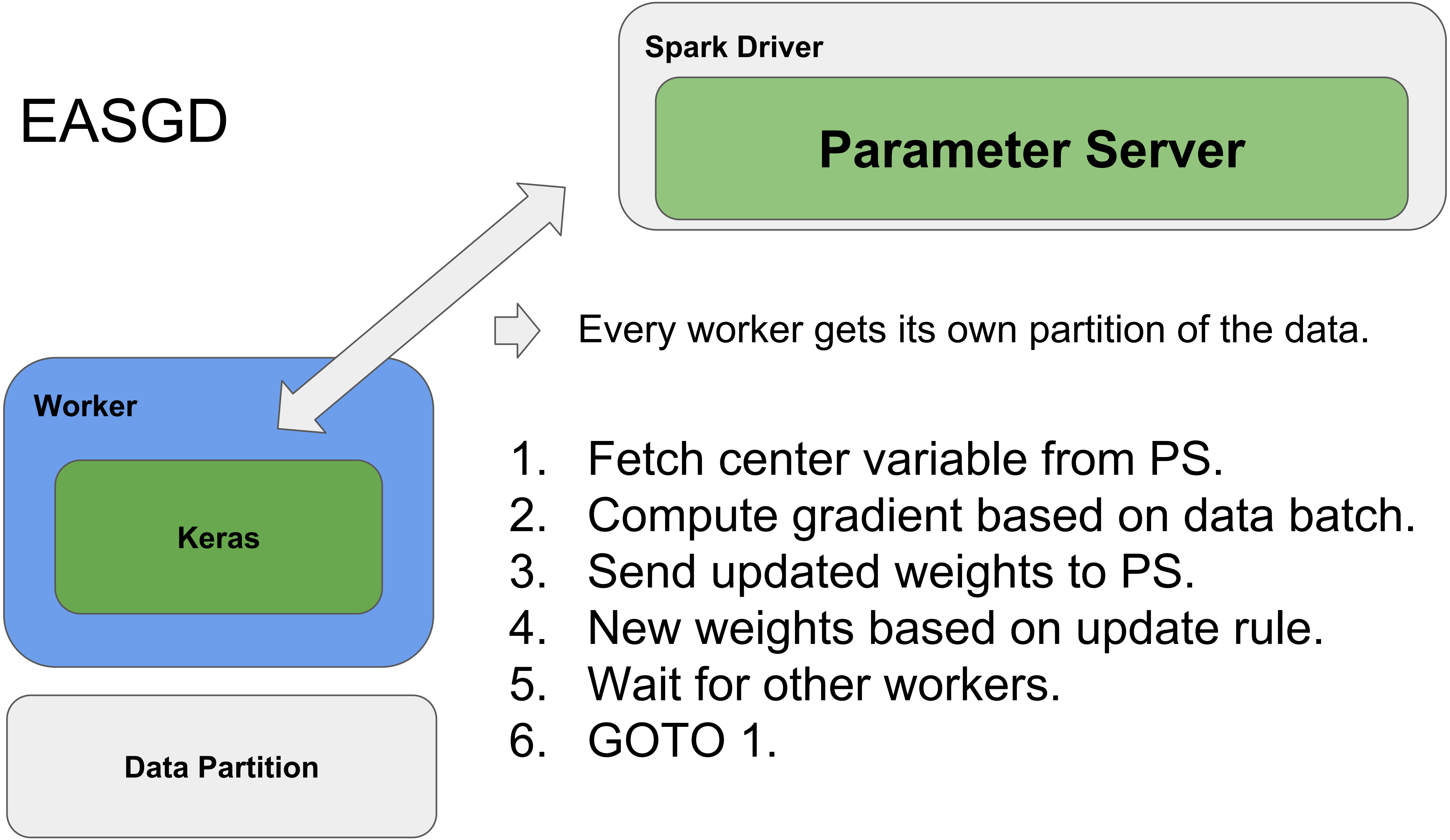
- **Scale out** on cluster level (more cores, faster training).
- Spark will handle “**bigger than memory**” issue and **parallelization**.
- Easy to provide compatibility with Spark API’s / Python:
    - SparkSQL (DataFrames)
    - Distributed preprocessing of dataset.
    - Evaluation metrics.
    - …

Experimental setup

- Evaluating EASGD [1] (Elastic Averaging SGD)
- 1 iteration over the training data (nb_epoch = 1)
- 24 trainers used when running EASGD
- **Model:** 380402 trainable parameters
- **Data:** only 250000 instances
    - **Training:** 90% (variable)
    - **Test:** 10%

- Evaluation criteria:
    - Time
    - Convergence
    - Scaling

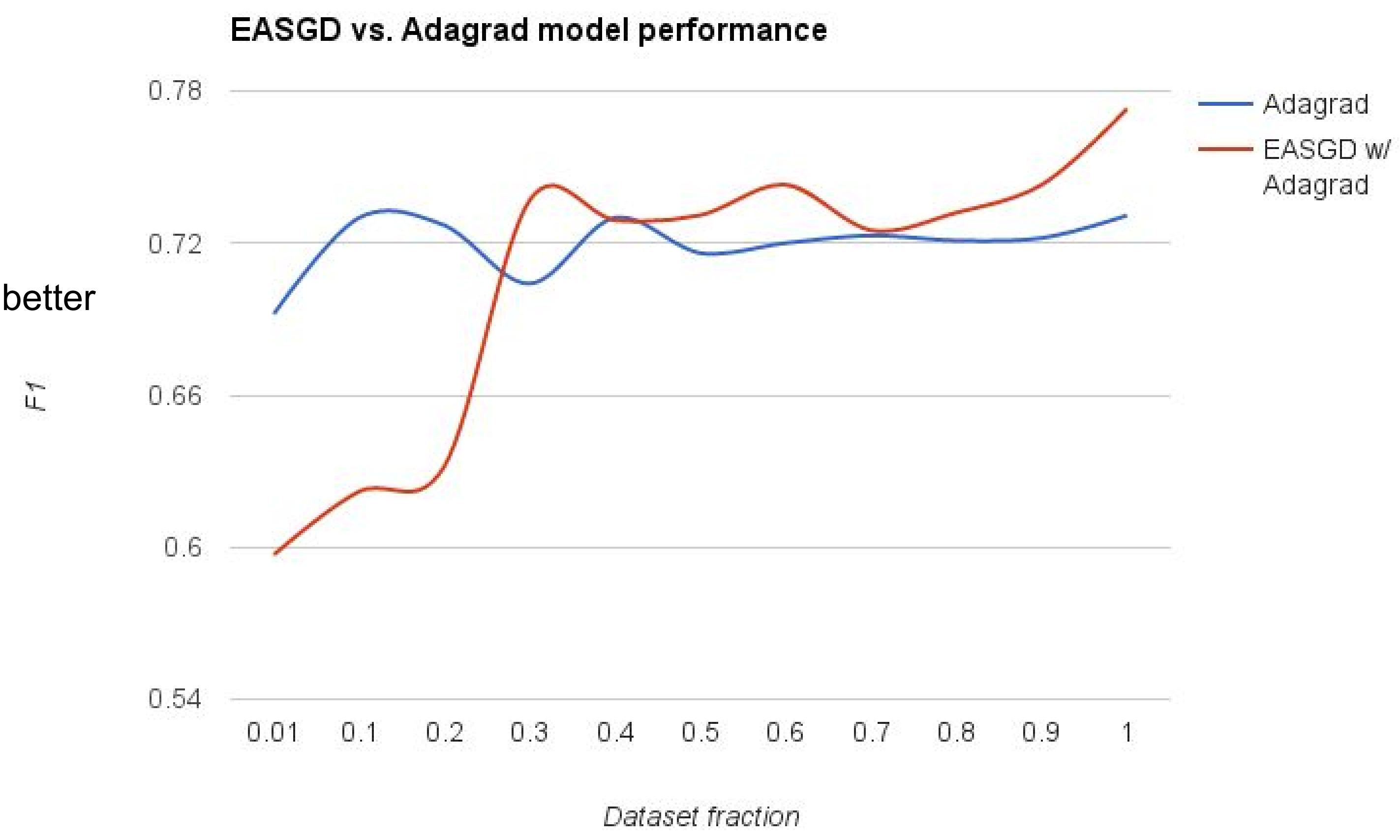
EASGD



Every worker gets its own partition of the data.

1. Fetch center variable from PS.
2. Compute gradient based on data batch.
3. Send updated weights to PS.
4. New weights based on update rule.
5. Wait for other workers.
6. GOTO 1.

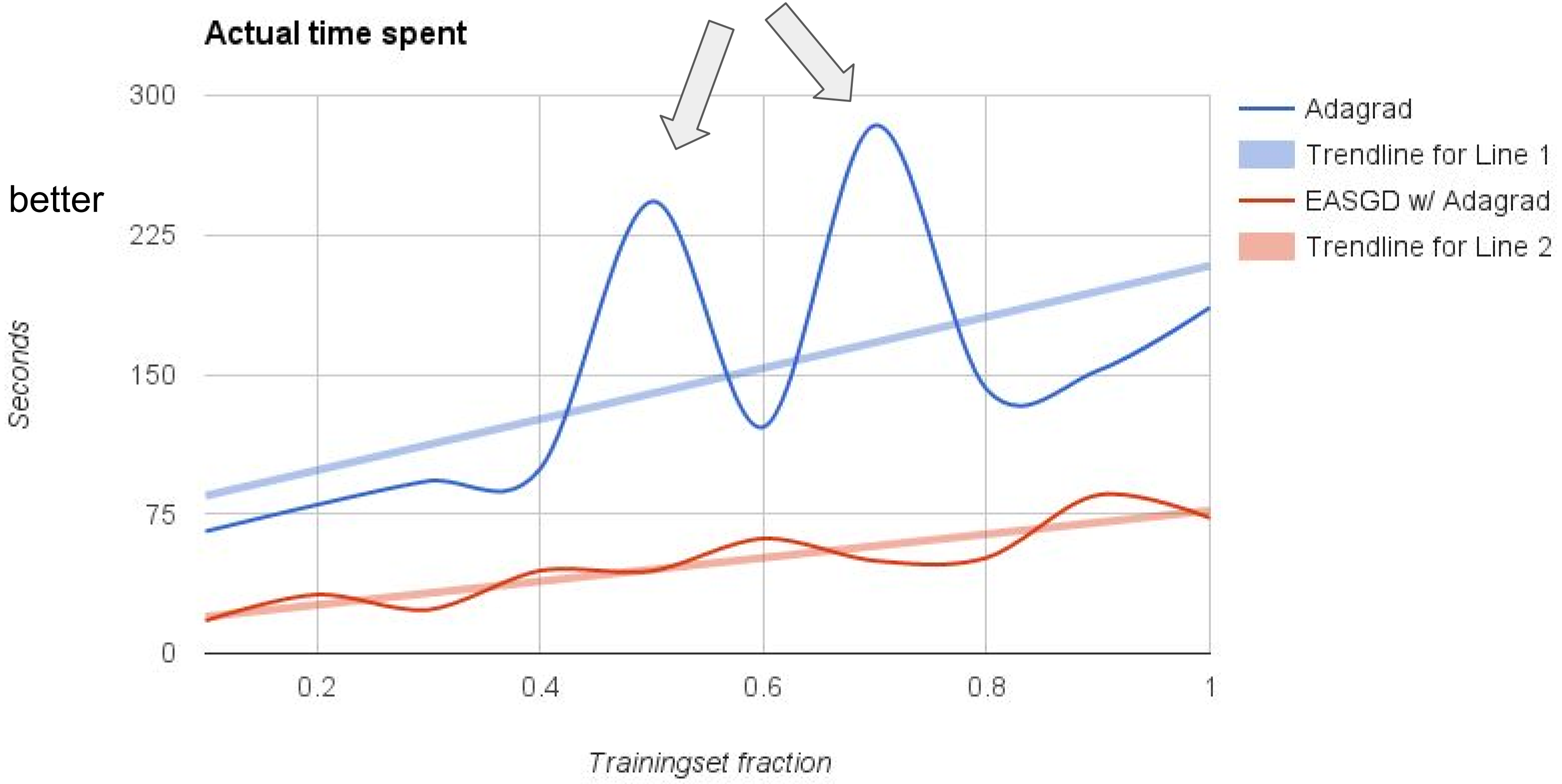
Higher is better



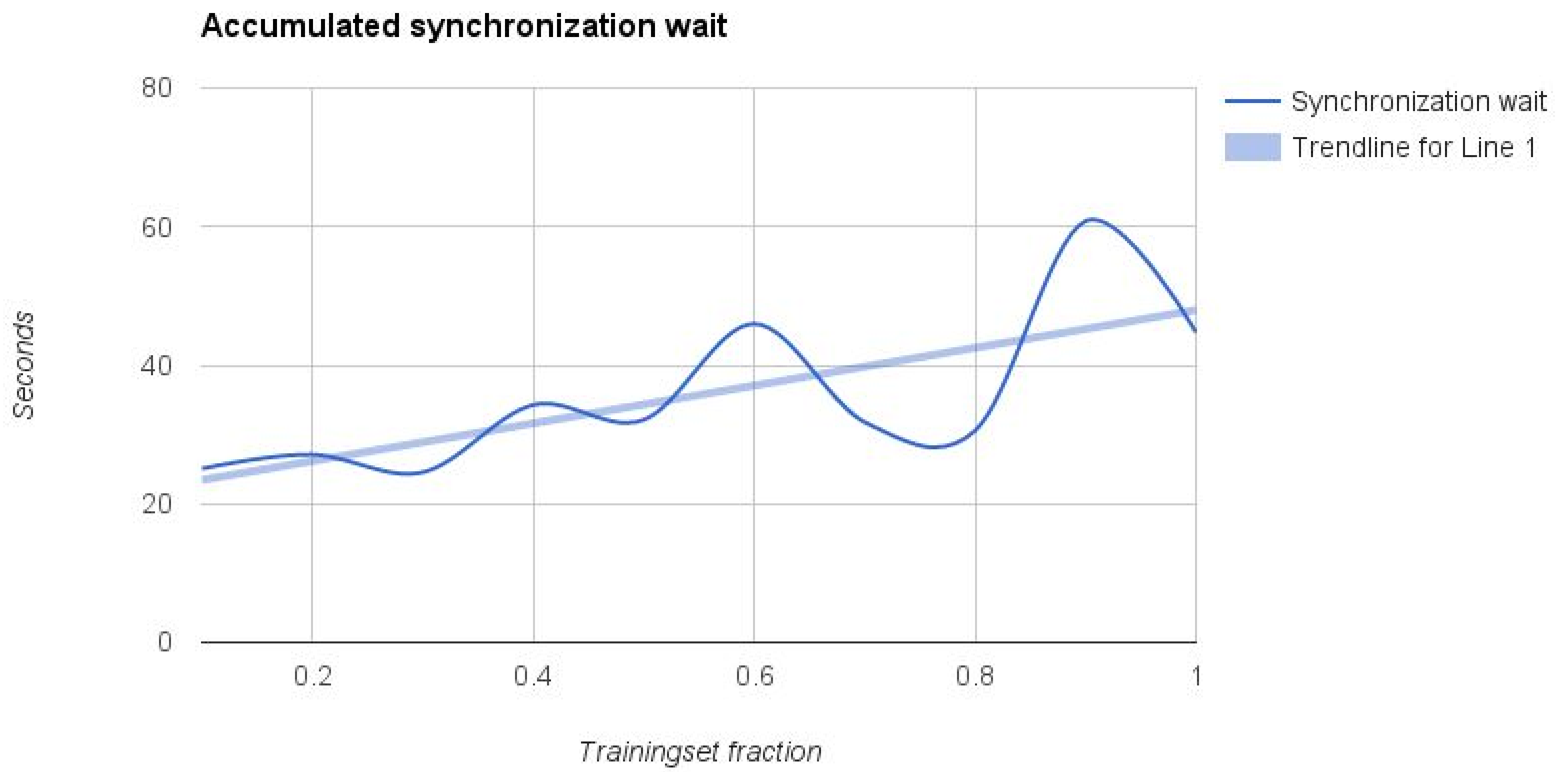
EASGD needs more data to converge, but achieves better model performance.

Outliers (loaded clusternode)

Lower is better

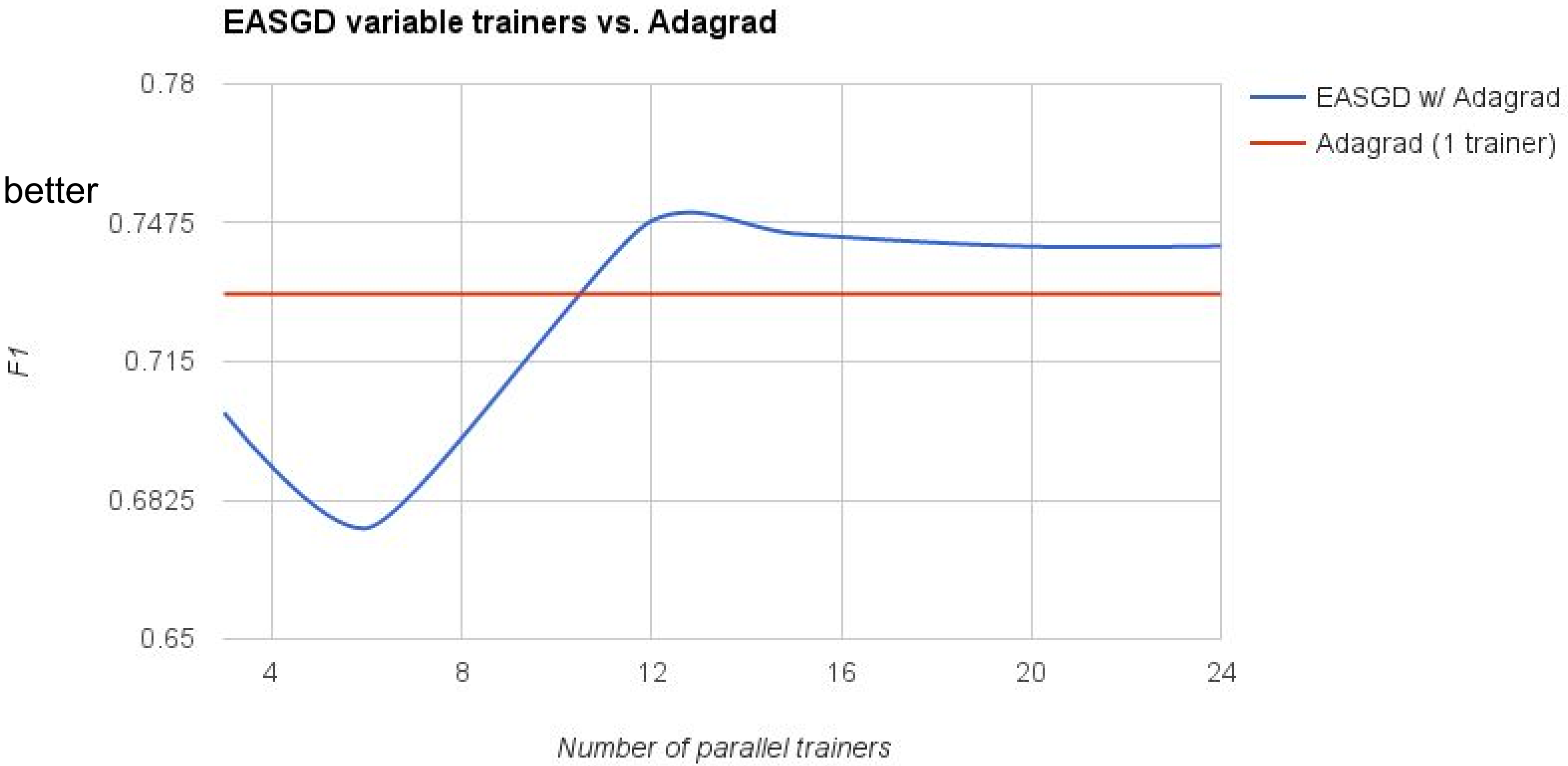


EASGD needs less time to converge.

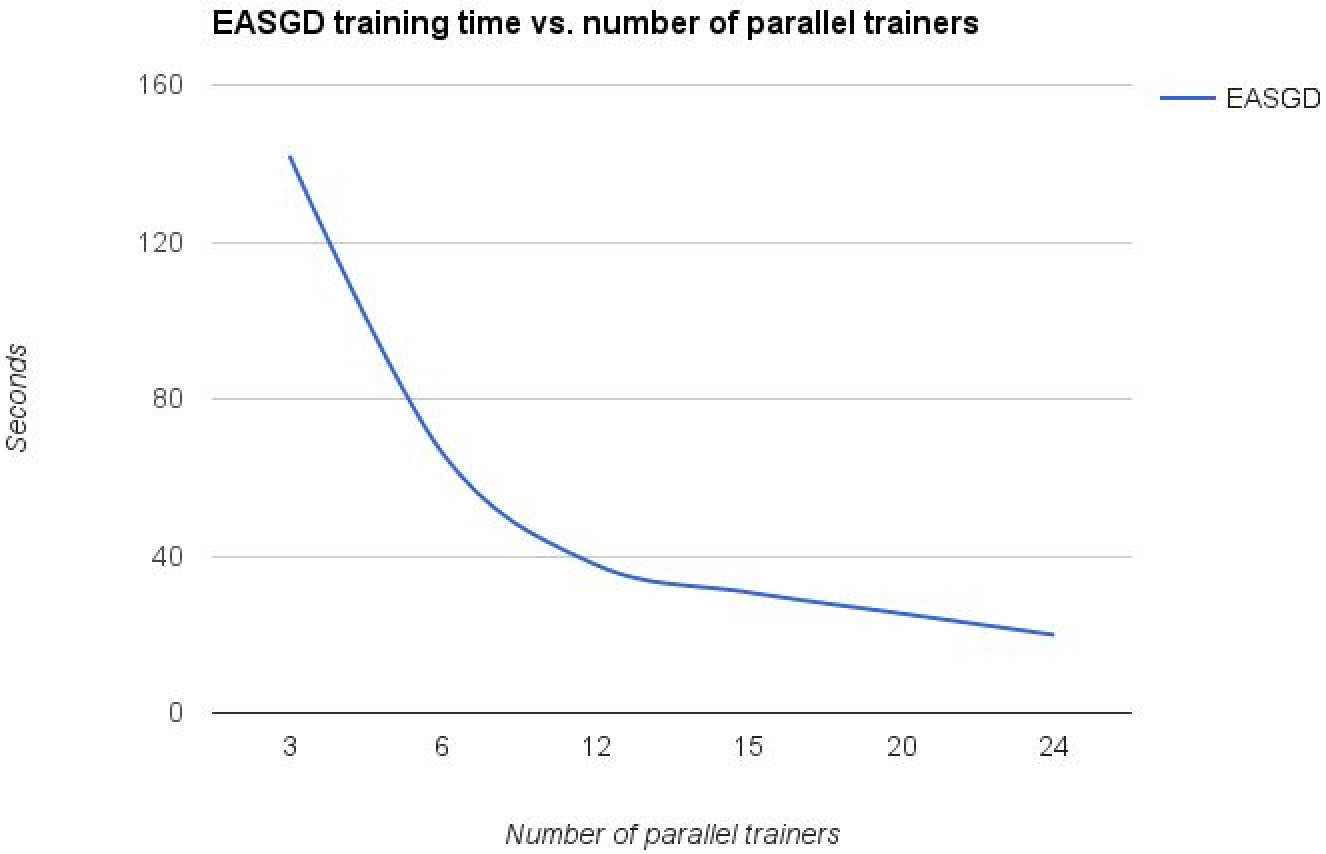


A lot of time is spent on the synchronization of the workers!

Higher is better



Can be improved by correcting inefficiencies regarding batch retrieval and batch size parameterization.



Summary

- Abstraction to easily experiment new distributed algorithms.
- Source code of framework is available as Open Source software.
- EASGD implemented and evaluated.
- Claims of paper are confirmed:
  - Faster training.
  - Eventual moder will have (slightly) better performance.

Future Work

- More experiments to understand scaling and convergence (10F-CV).
- Asynchronous algorithms
- Experiments with parameterization of EASGD.
- **Other features requested by community?**

References

- *Large Scale Distributed Deep Networks*. Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang and Andrew Y. Ng. NIPS 2012. [1]

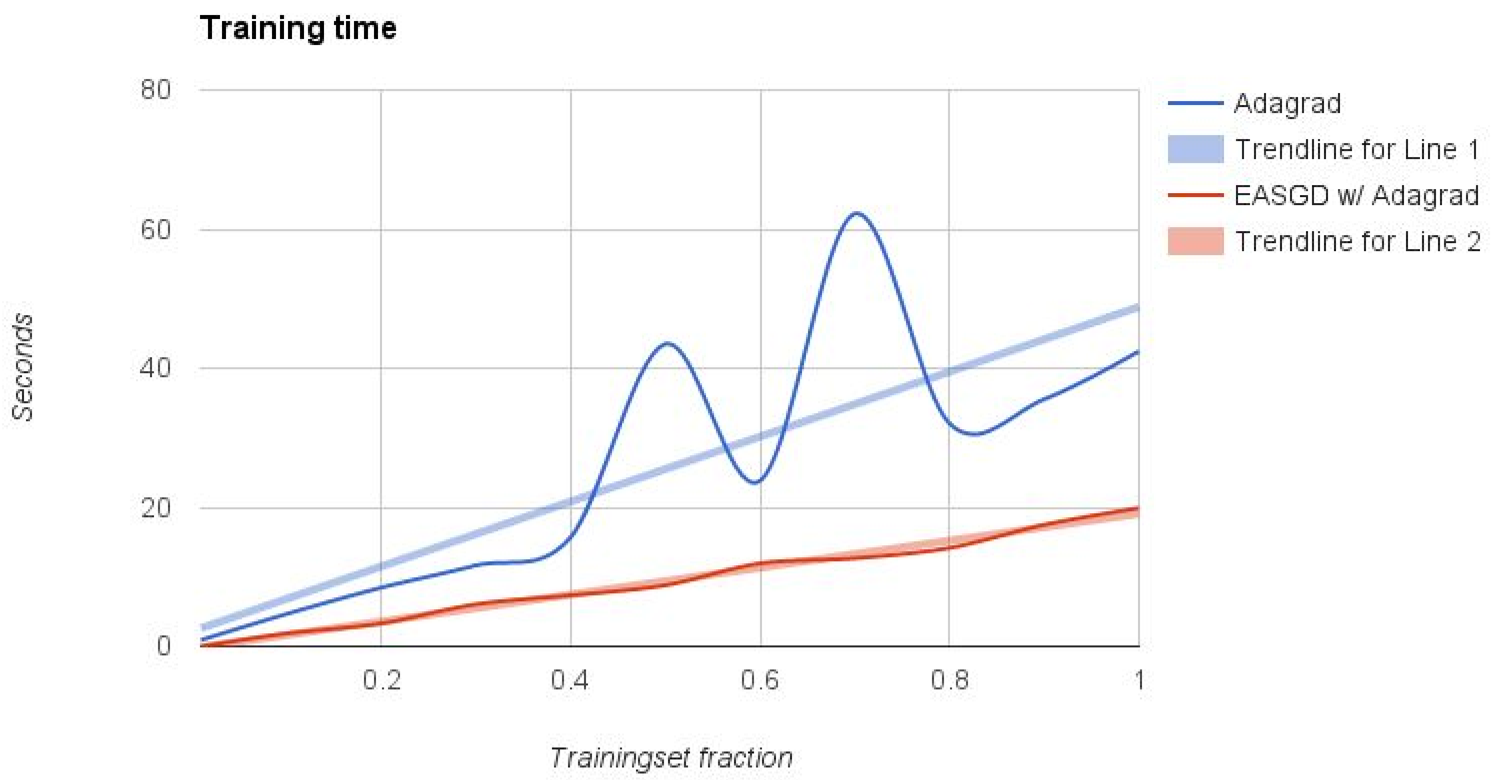
- Zhang, S., Choromanska, A. E., & LeCun, Y. (2015). Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems* (pp. 685-693). [2]

- https://github.com/JoeriHermans/dist-keras/ [3]

- SparkNet: Training Deep Networks in Spark. http://arxiv.org/pdf/1511.06051v4.pdf [4]

Questions?

Appendices


Training time
80
60
40
20
0
Seconds
0.2
0.4
0.6
0.8
1
Trainingset fraction
Adagrad
Trendline for Line 1
EASGD w/ Adagrad
Trendline for Line 2