Distributed ML Introduction

Шугаепов Ильнур
VK.com
ilnur.shug@gmail.com

февраль 2020 г.

Problem formulation [4, 3]

- Dataset $X^n = \{z_i\}_{i=1}^n$, где $z_i = (x_i, y_i) \sim P(z)$

Problem formulation [4, 3]

- Dataset $X^n = \{z_i\}_{i=1}^n$, где $z_i = (x_i, y_i) \sim P(z)$
- Loss function $l(\hat{y}, y)$

Problem formulation [4, 3]

- Dataset $X^n = \{z_i\}_{i=1}^n$, где $z_i = (x_i, y_i) \sim P(z)$
- Loss function $l(\hat{y}, y)$
- Function

$$f_w \in \mathcal{F} \quad : \quad \frac{1}{n}\sum_{i=1}^{n} l(f_w(x_i), y_i) \quad \rightarrow \quad \min_w$$

Loss

- Expected risk

$$E(f) = \int l(f(x), y) dP(z)$$

Loss

- Expected risk

$$E(f) = \int l(f(x), y) dP(z)$$

- Emperical risk

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l\left(f\left(x_i\right), y_i\right)$$

Gradient descent

- GD

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{1}$$

Gradient descent

- GD

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{1}$$

- SGD

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t) \tag{2}$$

It is hoped that (2) behaves like its expectation (1)

Online learning

- SGD can process examples on the fly in a deployed system
- SGD directly optimizes the expected risk, since the examples are randomly drawn from $P(z)$

- $f^* = \arg\min_f E(f)$ — best possible prediction function

- $f^* = \arg\min_f E(f)$ — best possible prediction function
- $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} E(f)$ — best function in the family $\mathcal{F}$

- $f^* = \arg\min_f E(f)$ — best possible prediction function
- $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} E(f)$ — best function in the family $\mathcal{F}$
- $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ — empirical optimum

- $f^* = \arg\min_f E(f)$ — best possible prediction function
- $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} E(f)$ — best function in the family $\mathcal{F}$
- $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ — empirical optimum
- $\tilde{f}_n$ minimizes the objective function with a predefined accuracy $E_n(\tilde{f}_n) < E_n(f_n) + \rho$

The tradeoffs of large scale learning

Excess error $\mathcal{E} = \mathbb{E}\left[E(\tilde{f}_n) - E\left(f^*\right)\right]$ can be decomposed in three terms

$$\mathcal{E} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right] + \mathbb{E}\left[E\left(f_n\right) - E\left(f^*_{\mathcal{F}}\right)\right] + \mathbb{E}\left[E(\tilde{f}_n) - E\left(f_n\right)\right]$$

- $\mathcal{E}_{app} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right]$ — approximation error

The tradeoffs of large scale learning

Excess error $\mathcal{E} = \mathbb{E}\left[E(\tilde{f}_n) - E\left(f^*\right)\right]$ can be decomposed in three terms

$$\mathcal{E} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right] + \mathbb{E}\left[E\left(f_n\right) - E\left(f^*_{\mathcal{F}}\right)\right] + \mathbb{E}\left[E(\tilde{f}_n) - E\left(f_n\right)\right]$$

- $\mathcal{E}_{app} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right]$ — approximation error
- $\mathcal{E}_{est} = \mathbb{E}\left[E\left(f_n\right) - E\left(f^*_{\mathcal{F}}\right)\right]$ — estimation error

The tradeoffs of large scale learning

Excess error $\mathcal{E} = \mathbb{E}\left[E(\tilde{f}_n) - E\left(f^*\right)\right]$ can be decomposed in three terms

$$\mathcal{E} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right] + \mathbb{E}\left[E\left(f_n\right) - E\left(f^*_{\mathcal{F}}\right)\right] + \mathbb{E}\left[E(\tilde{f}_n) - E\left(f_n\right)\right]$$

- $\mathcal{E}_{app} = \mathbb{E}\left[E\left(f^*_{\mathcal{F}}\right) - E\left(f^*\right)\right]$ — approximation error
- $\mathcal{E}_{est} = \mathbb{E}\left[E\left(f_n\right) - E\left(f^*_{\mathcal{F}}\right)\right]$ — estimation error
- $\mathcal{E}_{opt} = \mathbb{E}\left[E(\tilde{f}_n) - E\left(f_n\right)\right]$ — optimization error

Optimization
Two cases

$$\min_{\mathcal{F},\rho,n} \mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \quad \text{subject to} \begin{cases} n \leq n_{\max} \\ T(\mathcal{F},\rho,n) \leq T_{\max} \end{cases}$$

Optimization
Two cases

$$\min_{\mathcal{F},\rho,n} \mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \quad \text{subject to} \begin{cases} n \leq n_{\max} \\ T(\mathcal{F}, \rho, n) \leq T_{\max} \end{cases}$$

- *Small-scale* constrained by the maximal number of examples. Choose ρ arbitrary small and set $n = n_{\max}$.
- *Large-scale* constrained by the maximal computing time. Process more examples during allowed time.

Typical variations when $\mathcal{F}$, n, and ρ increase

	$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$ (approximation error)			

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)			

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$		

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)			

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$		

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)			

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$		

Typical variations when $\mathcal{F}$, n, and ρ increase

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	

Typical variations when $\mathcal{F}$, n, and ρ increase

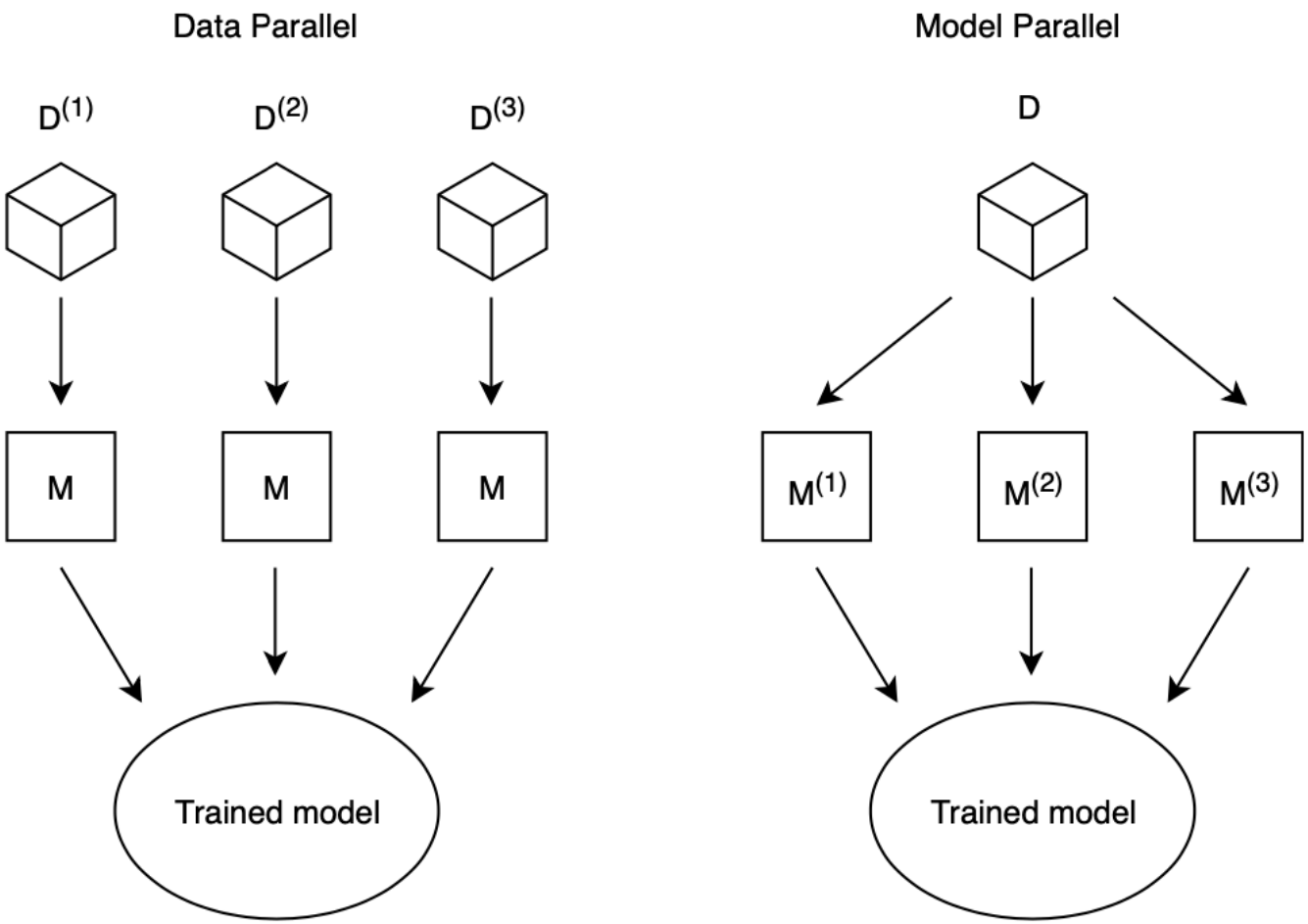
		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$

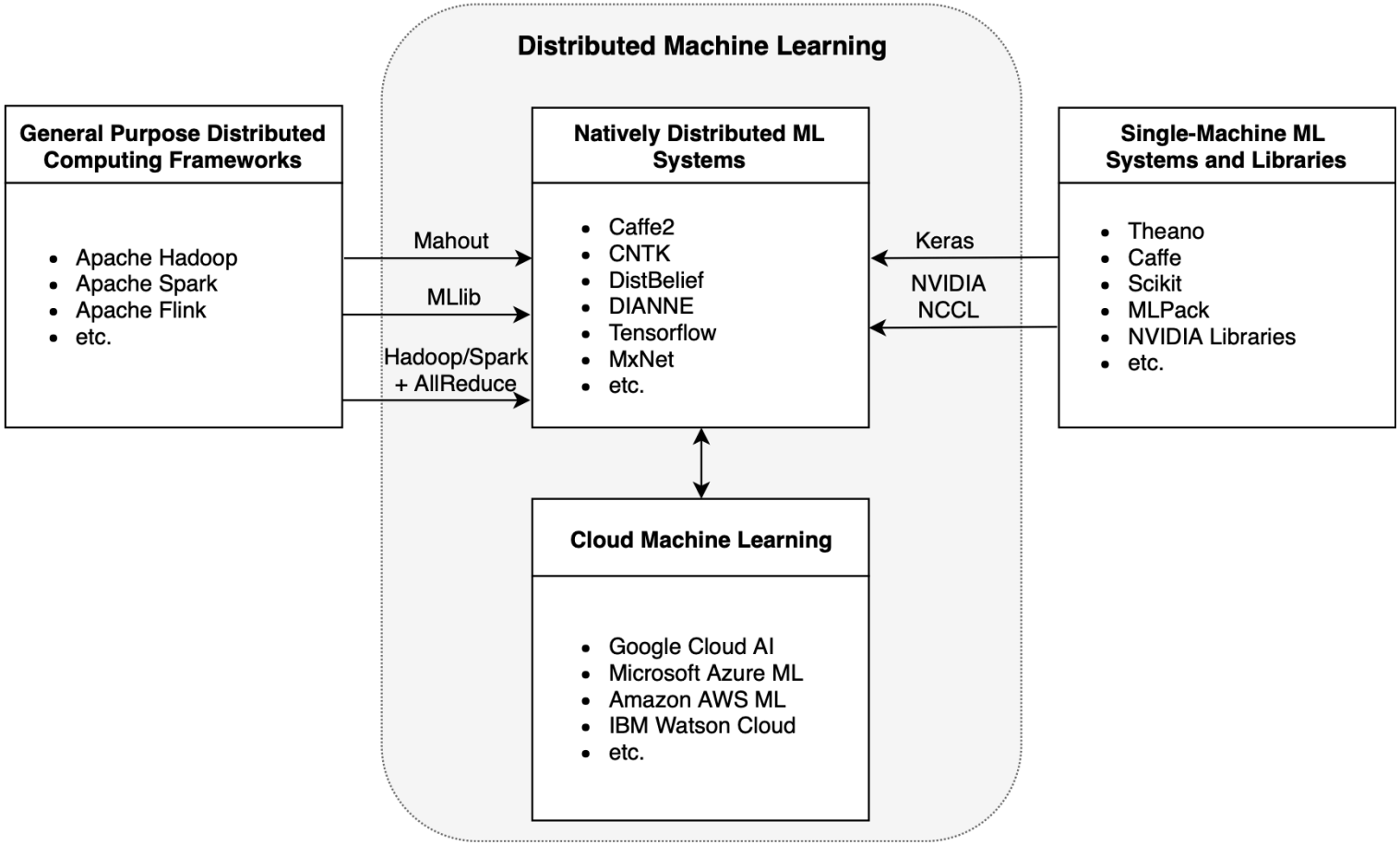
Resume

The data sizes have grown faster than the speed of processors. In this context, the capabilities of statistical machine learning methods is limited by the computing time rather than the sample size.

DISTRIBUTED ML

Data & Model Parallelism [14]




Distributed Machine Learning
General Purpose Distributed Computing Frameworks
Apache Hadoop
Apache Spark
Apache Flink
etc.
Mahout
MLlib
Hadoop/Spark + AllReduce
Natively Distributed ML Systems
Caffe2
CNTK
DistBelief
DIANNE
Tensorflow
MxNet
etc.
Keras
NVIDIA NCCL
Single-Machine ML Systems and Libraries
Theano
Caffe
Scikit
MLPack
NVIDIA Libraries
etc.
Cloud Machine Learning
Google Cloud AI
Microsoft Azure ML
Amazon AWS ML
IBM Watson Cloud
etc.

GENERAL PURPOSE DISTRIBUTED COMPUTING

MapReduce for ML [6]

Statistical Query Model

- Exact implementation of ML algorithms, not parallel approximations to algorithms

Adopted Algorithms

- Logistic Regression (LR)
- Neural Networks (NN)
- k-Means
- ...

Adopted Algorithms

- Logistic Regression (LR)
- Neural Networks (NN)
- k-Means
- ...

MapReduce gives us Data Parallelism

Limitations

- Exact implementation of ML algorithms could be slow
  - it requires many passes through the dataset for convergence
  - it requires many synchronization sweeps (i.e. MapReduce iterations)

Limitations

- Exact implementation of ML algorithms could be slow
  - it requires many passes through the dataset for convergence
  - it requires many synchronization sweeps (i.e. MapReduce iterations)
- Broadcasting parameters of LR, NN could be slow

Limitations

- Exact implementation of ML algorithms could be slow
  - it requires many passes through the dataset for convergence
  - it requires many synchronization sweeps (i.e. MapReduce iterations)
- Broadcasting parameters of LR, NN could be slow
- Using HDFS at each iteration is slow

Parallel SGD [15]

Algorithm 3.1 SGD$(X^n = \{z_i\}_{i=1}^n, T, \gamma, w_0)$

for $t = 1, \ldots, T$ **do**
  Draw $t \in \{1, \ldots, n\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \gamma \nabla_w Q\left(z_t, w_{t-1}\right)$
return w_t

Parallel SGD [15]

Algorithm 3.3 SGD$(X^n = \{z_i\}_{i=1}^n, T, \gamma, w_0)$

for $t = 1, \ldots, T$ **do**
  Draw $t \in \{1, \ldots, n\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \gamma \nabla_w Q(z_t, w_{t-1})$
return w_t

Algorithm 3.4 ParallelSGD$(X^n = \{z_i\}_{i=1}^n, T, \gamma, w_0, K)$

for $i = 1, \ldots, K$ **parallel do**
  $v_i \leftarrow$ SGD(X^n, T, γ, w_0)
Aggregate from all computers $v \leftarrow \frac{1}{K} \sum_{i=1}^K v_i$
return v

Parallel SGD [15]

Algorithm 3.5 SGD$(X^n = \{z_i\}_{i=1}^n, T, \gamma, w_0)$

for $t = 1, \ldots, T$ **do**
  Draw $t \in \{1, \ldots, n\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \gamma \nabla_w Q(z_t, w_{t-1})$
return w_t

Algorithm 3.6 ParallelSGD$(X^n = \{z_i\}_{i=1}^n, T, \gamma, w_0, K)$

for $i = 1, \ldots, K$ **parallel do**
  $v_i \leftarrow$ SGD(X^n, T, γ, w_0)
Aggregate from all computers $v \leftarrow \frac{1}{K} \sum_{i=1}^K v_i$
return v

If T is much less than n, then it is only necessary for a machine to have access to the data it actually touches.

SimuParallelSGD [15]

Algorithm 3.7 SimuParallelSGD$(X^n = \{z_i\}_{i=1}^n, \gamma, K)$

$T \leftarrow \lfloor n/K \rfloor$
Randomly partition the X^n, giving T examples to each machine
for $i = 1, \ldots, K$ **parallel do**
  Randomly shuffle the data on machine i
  Initialize $w_{i,0} = 0$
  for $t = 1, \ldots, T$ **do**
    Get the t-th example on the i-th machine (this machine), $z^{i,t}$
    $w_{i,t} \leftarrow w_{i,t-1} - \gamma \nabla_w Q\left(z^{i,t}, w_{i,t-1}\right)$
Aggregate from all computers $v \leftarrow \frac{1}{K} \sum_{i=1}^{K} w_{i,t}$
return v

SimuParallelSGD
Experiments

- **Dataset**: 2.5M Training examples, 600K Test examples. $x_i \in \mathbb{R}^{2^{18}}$
- **Approach**:
  1. Trained up to 100 models, each on an independent, random permutation of the full training data.
  2. During training, the model is stored on disk after $k = 10^4 \cdot 2^i$ updates.
  3. Then models obtained for each i are averaged and evaluated.
  4. This approach evaluates performance for the algorithm after each machine has seen k samples.
- **Metrics**: normalized RMSE (1.0 is the RMSE obtained by training in one sequential pass)

SimuParallelSGD
Experiment Results

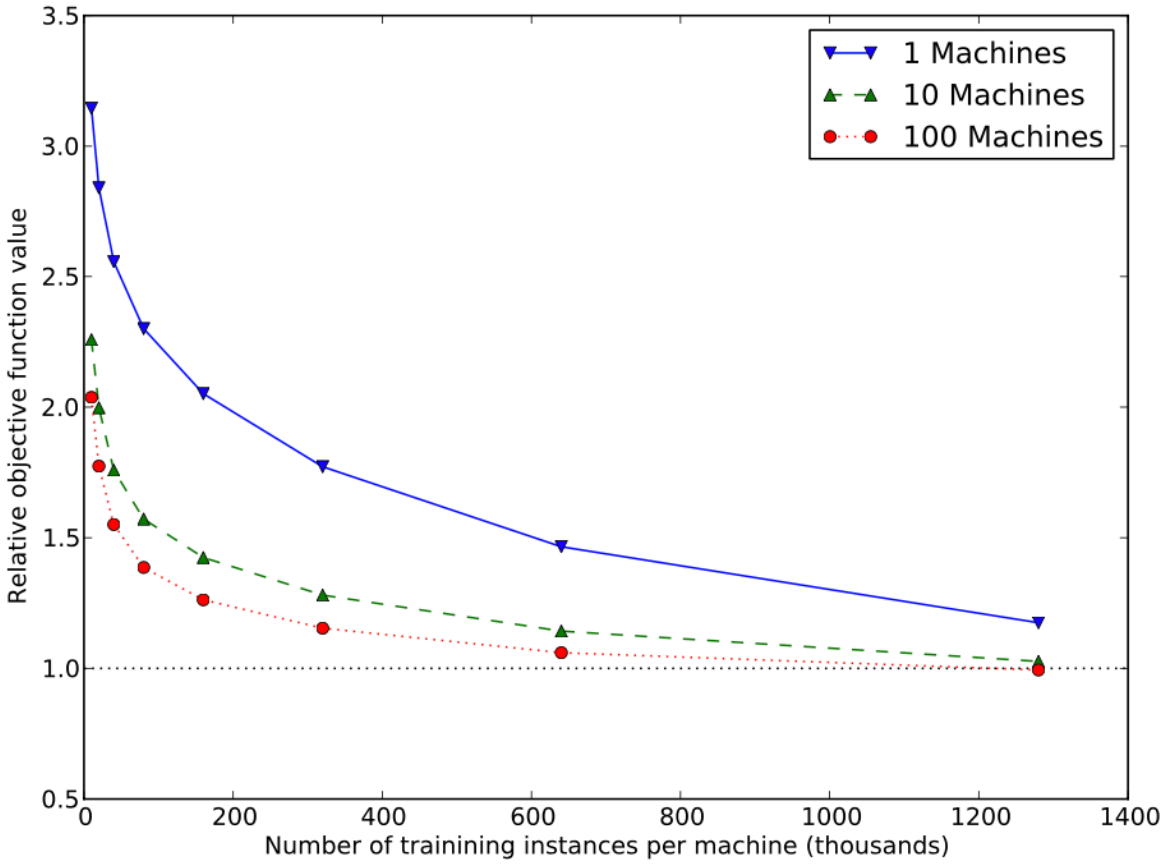


Рис.: Relative training error

SimuParallelSGD
Conclusion

- *Data-parallel* SGD
- Highly suitable for parallel, large-scale ML (communication at the very end)

SparkNet [11]
Motivation & Benefits

Motivation:

- Main goal is to address limitations of training Neural Networks using MapReduce

Benefits:

- Model training is integrated in existing data-processing pipelines

Implementation

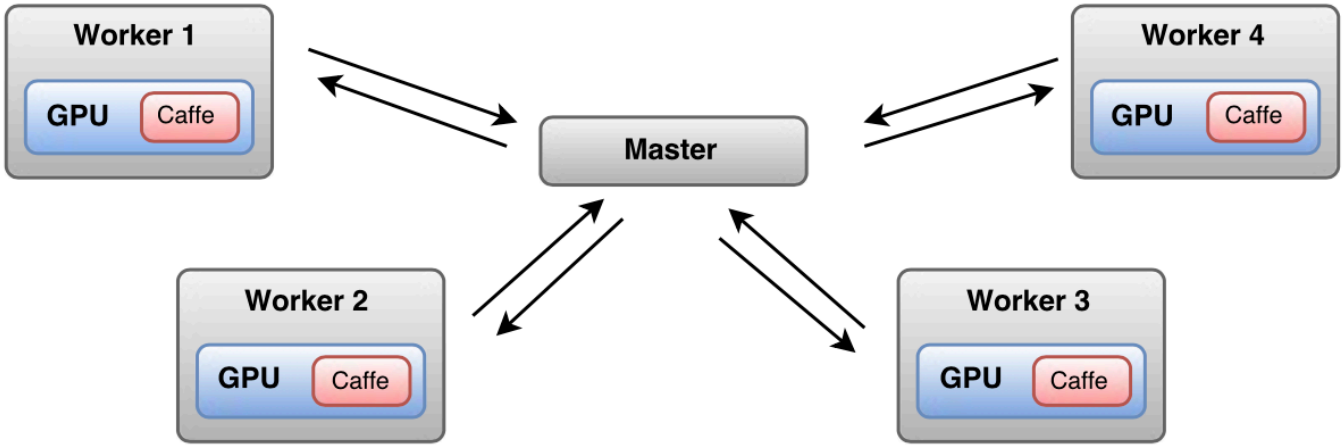


Рис.: SparkNet architecture

Parallelizing SGD
Serial SGD

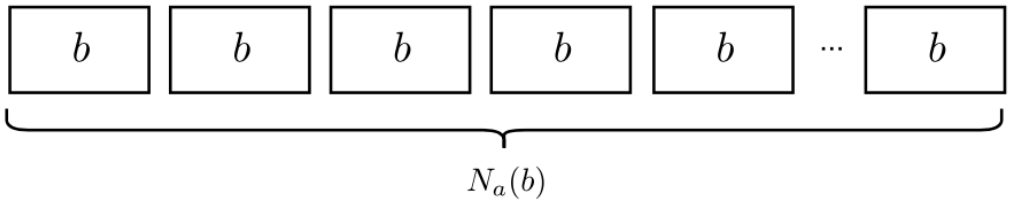


Рис.: Serial run of SGD. Each block corresponds to a single SGD update with batch size b. The quantity $N_a(b)$ is the number of iterations required to achieve an accuracy of a.

Parallelizing SGD

Naive parallelization

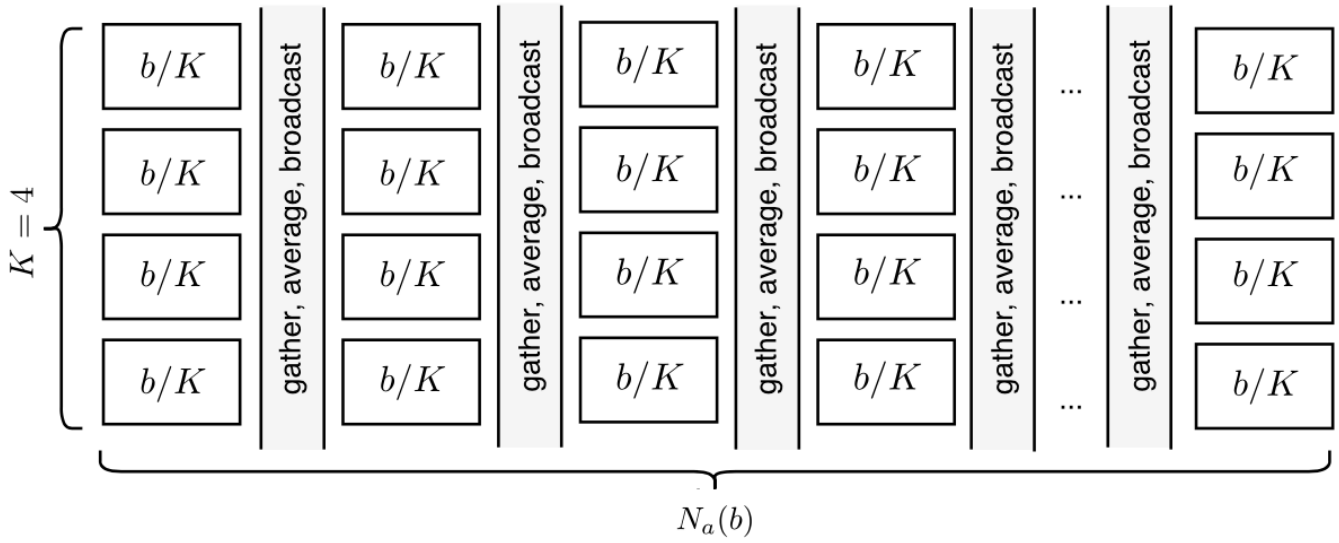


Рис.: At each iteration, each batch of size b is divided among the K machines. this is equivalent to the serial run of SGD in Figure 3 and so the number of iterations is the same — $N_a(b)$.

Parallelizing SGD

SparkNet's parallelization

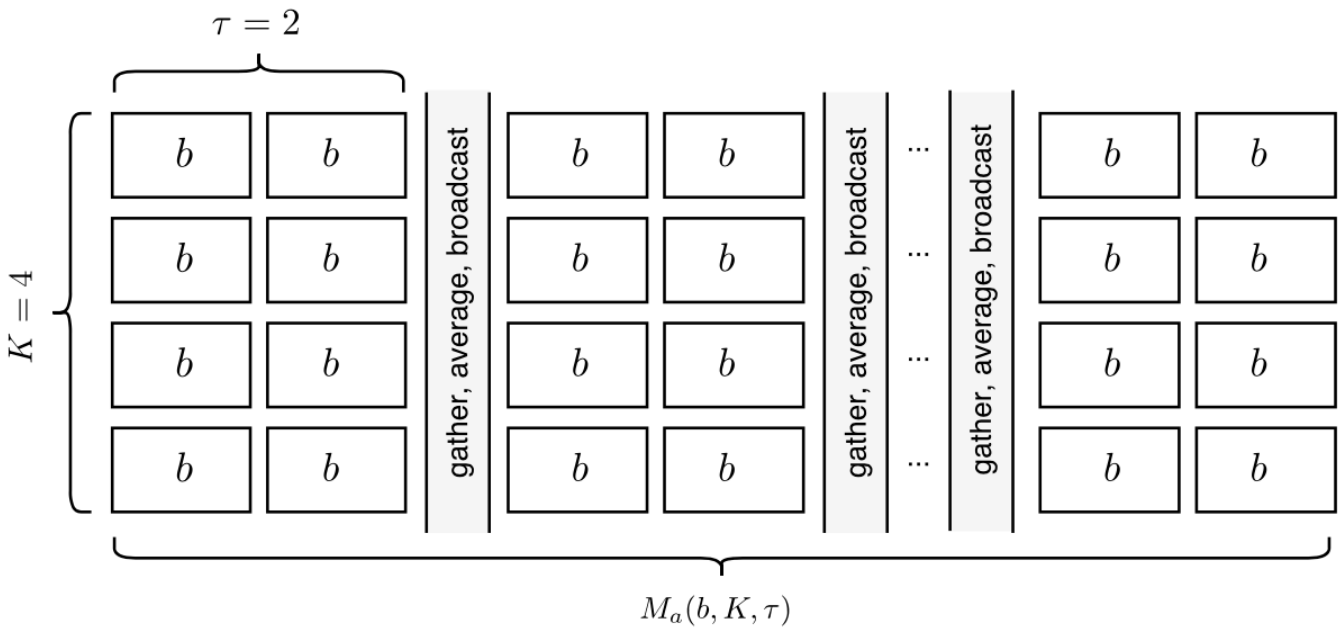


Рис.: The quantity $M_a(b, K, \tau)$ is the number of rounds (of τ iterations) required to obtain an accuracy of a. The total number of parallel iterations of SGD under SparkNet's parallelization scheme required to obtain an accuracy of a is then $\tau M_a(b, K, \tau)$.

Evaluation
ImageNet speedup

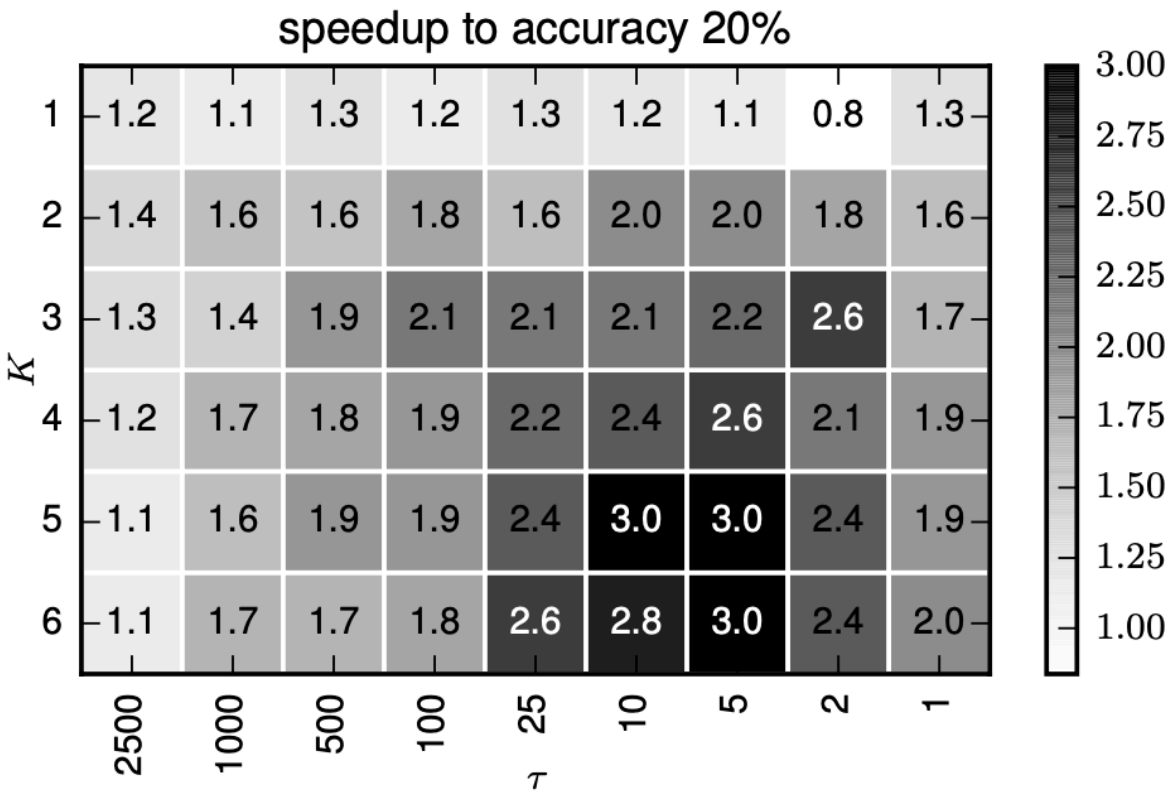


Рис.: Speedup $\tau M_a(b, K, \tau)/N_a(b)$ given by SparkNet's parallelization scheme relative to training on a single machine to obtain an accuracy of $a = 20\%$

MLlib [10]

MLlib[1]

- Classification and Regression (...)
- Clustering (k-Means, LDA, GMM)
- Collaborative Filtering (ALS)
- Dimensionality Reduction (SVD, PCA)

[1] https://spark.apache.org/docs/latest/ml-guide.html

NATIVELY DISTRIBUTED ML SYSTEMS

Parameter Server [9]

Classic Parameter Server

- All parameters are stored in the PS
- Workers send gradients to PS and get updated parameters once in a while

Key observation

- 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.

DistBelief [7]

DistBelief provides both Data and Model parallelism.

Model parallelism

- User defines the computation that takes place at each node in each layer of the model, and the messages that should be passed during the upward and downward phases of computation

Model parallelism

- User defines the computation that takes place at each node in each layer of the model, and the messages that should be passed during the upward and downward phases of computation
- User may partition the model across several machines

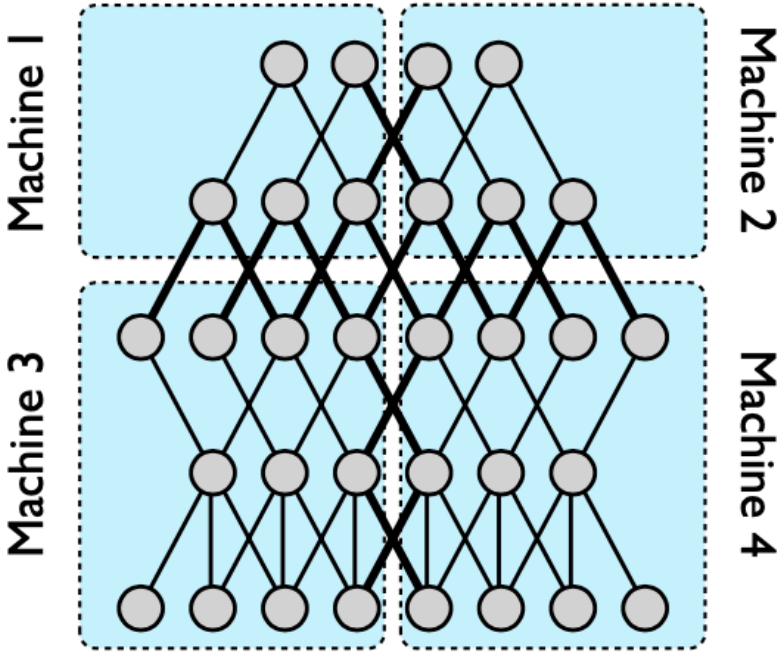


Рис.: A 5 layer DNN with local connectivity, partitioned across 4 machines. Nodes with edges that cross partition boundaries will need to have their state transmitted between machines.

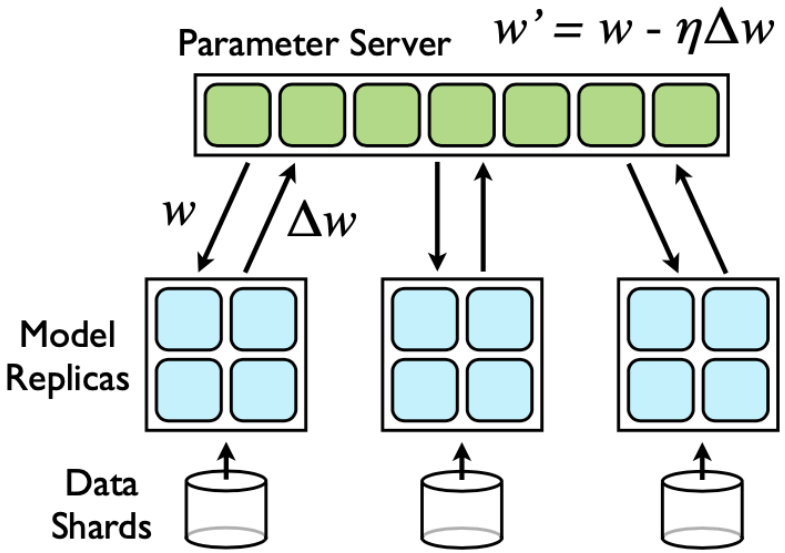
Model parallelism

Models with a large number of parameters or high computational demands typically benefit from access to more CPUs and memory, up to the point where communication costs dominate.

Distributed optimization algorithm

Downpour SGD

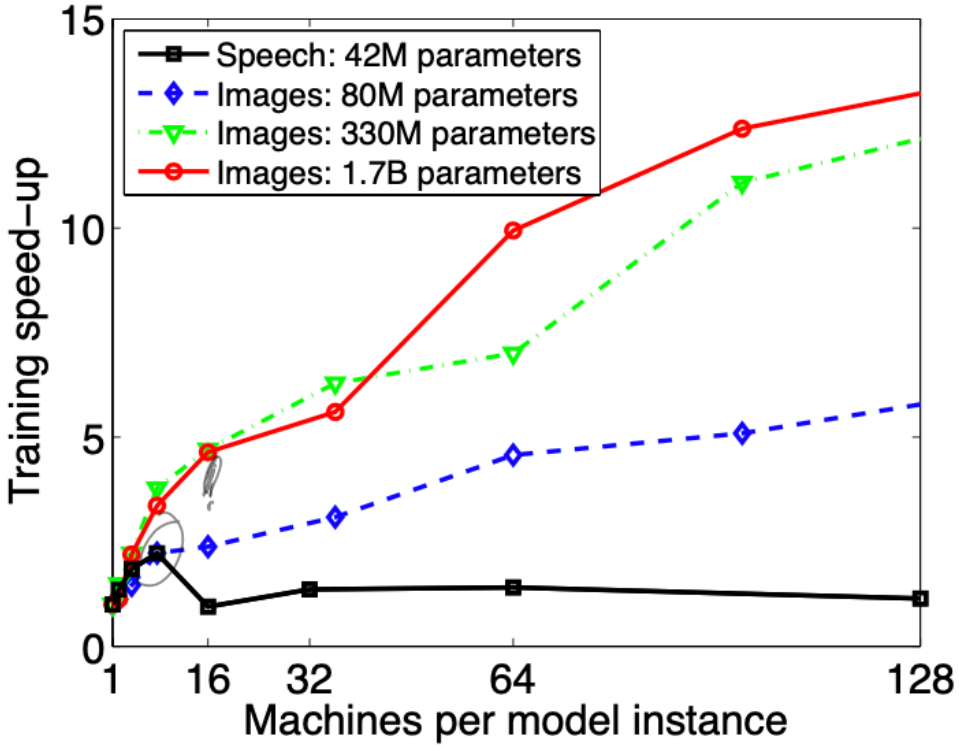
- We divide the training data into a number of subsets and run a copy of the model on each of these subsets



- Before processing each mini-batch, a model replica asks the parameter server for an updated copy of parameters
- We can limit replica to push updates only each n_{push} steps and ask for updates only each n_{fetch} steps

Experiments

Model parallelism benchmarks



smallest model

Рис.: Training speed-up for four different deep networks as a function of machines allocated to a single DistBelief model instance

Experiments

Optimization method comparisons

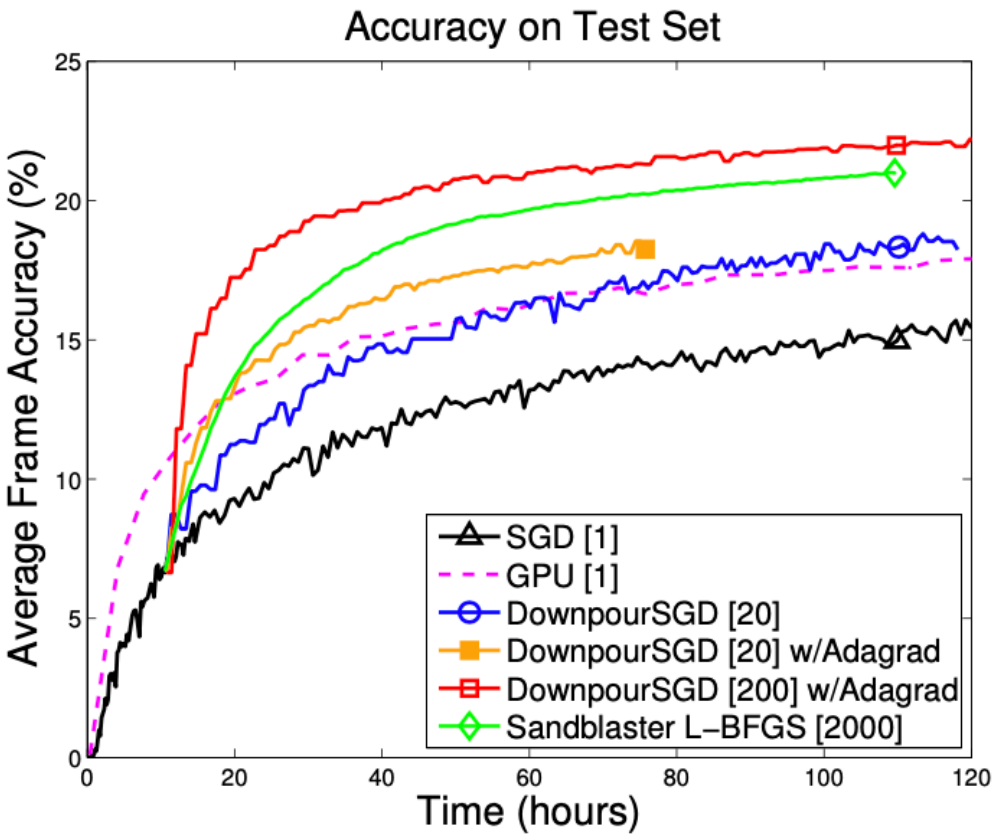


Рис.: ImageNet. Classification accuracy on the hold out test set as a function of training time. Downpour and Sandblaster experiments initialized using the same 10 hour warmstart of simple SGD.

Experiments

Optimization method comparisons

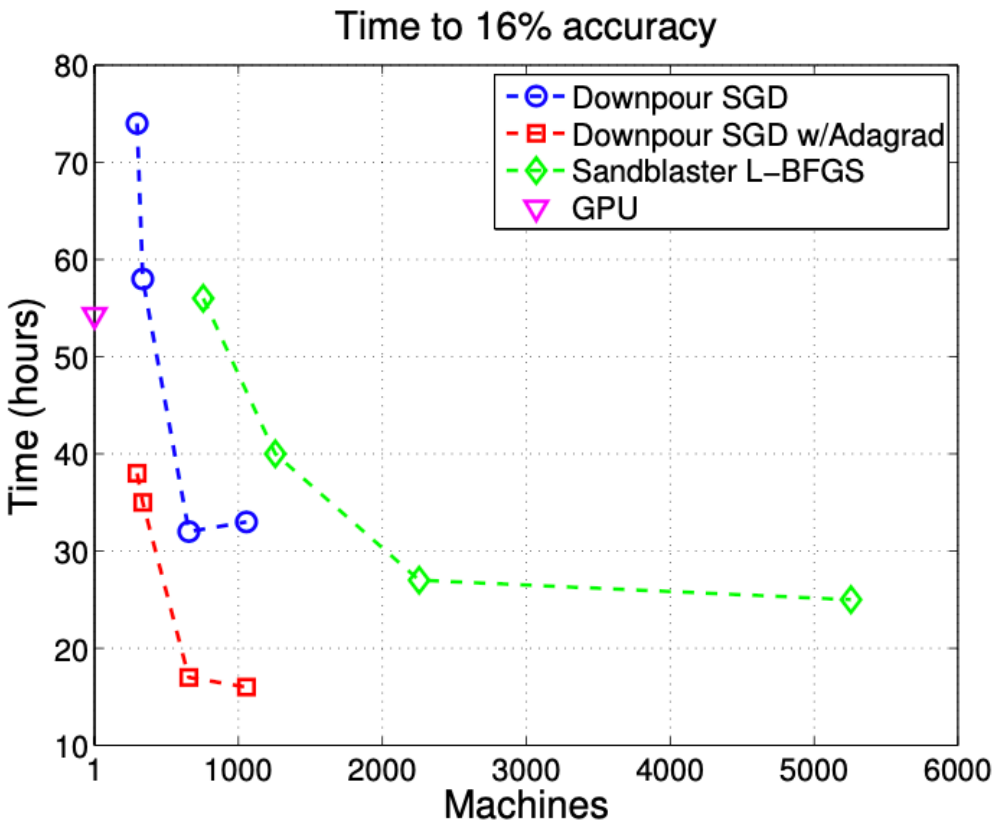


Рис.: ImageNet. Time to reach a fixed accuracy (16%)

Conclusions

- Downpour SGD, a highly asynchronous variant of SGD works surprisingly well for training nonconvex deep learning models
- Methods can use a cluster of machines to train even modestly sized deep networks significantly faster than a GPU, and without the GPU's limitation on the maximum size of the model

TensorFlow [1, 2] aka DistBelief 2.0

Basic concepts

- DataFlow graph (nodes, tensors, variables, operations)
- Kernel, Device
- Session
- Client, master, workers

Multi-Device Execution

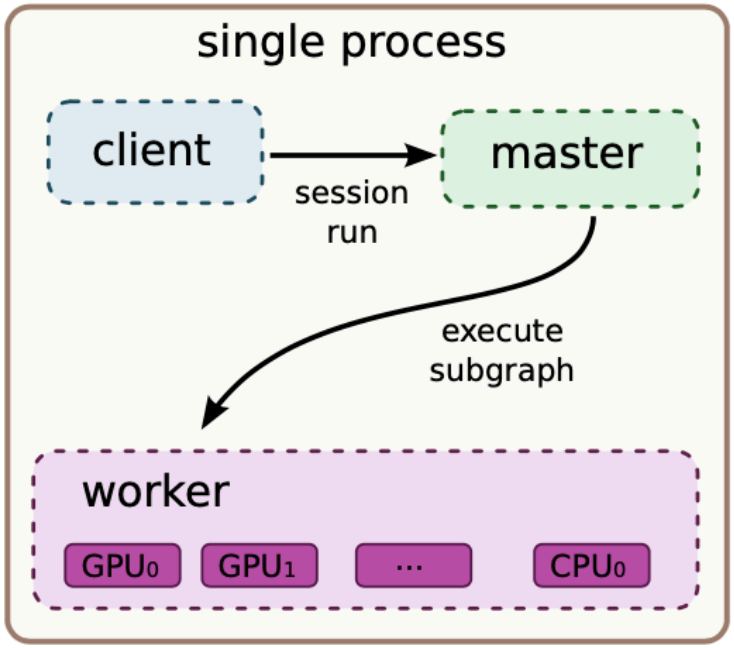


Рис.: Single machine structure

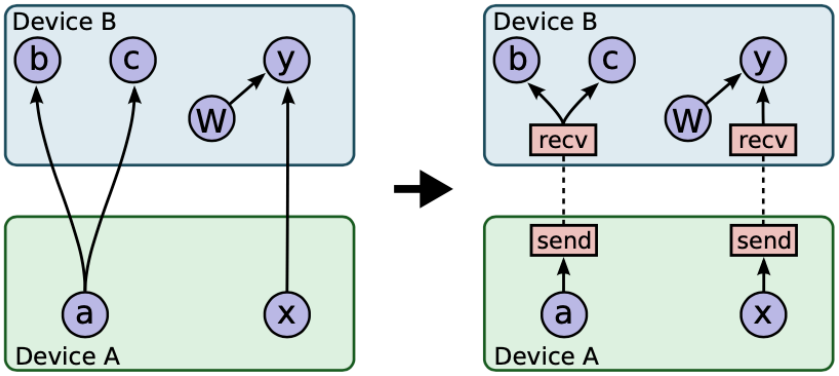


Рис.: Cross-device communication

Distributed Execution

- Distributed execution of a graph is very similar to multi- device execution
- TensorFlow supports both (A)synchronous Data Parallelism and Model Parallelism

Limitations of (TensorFlow) Parameter Server

- If one parameter server is used, it will likely become a networking or computational bottleneck.
- If multiple parameter servers are used, the communication pattern becomes “all-to-all” which may saturate network interconnects.

Ring Allreduce[12] & Horovod[13]

- Originally proposed and implemented at Baidu[2]
- Open-sourced at Uber[3][4][5]

[2] http://andrew.gibiansky.com/
[3] https://github.com/horovod/horovod/blob/master/docs/spark.rst
[4] https://docs.databricks.com/applications/deep-learning/distributed-training/index.html
[5] https://youtu.be/Ol3i_QKj8sY

REMARKS

- Literature with more math: Book [5, 8], Papers [15, 4, 3]
- What about non-SGD ML algorithms?

References I

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[4] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

References II

[5] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[6] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[8] E. Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

References III

[9] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[10] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[11] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan. Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*, 2015.

[12] B. Research. Baidu: Ring Allreduce. `http://andrew.gibiansky.com/`, 2017.

References IV

[13] A. Sergeev and M. Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[14] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer. A survey on distributed machine learning. *arXiv preprint arXiv:1912.09789*, 2019.

[15] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.