

LSML #6

Распараллеливание нейросетей

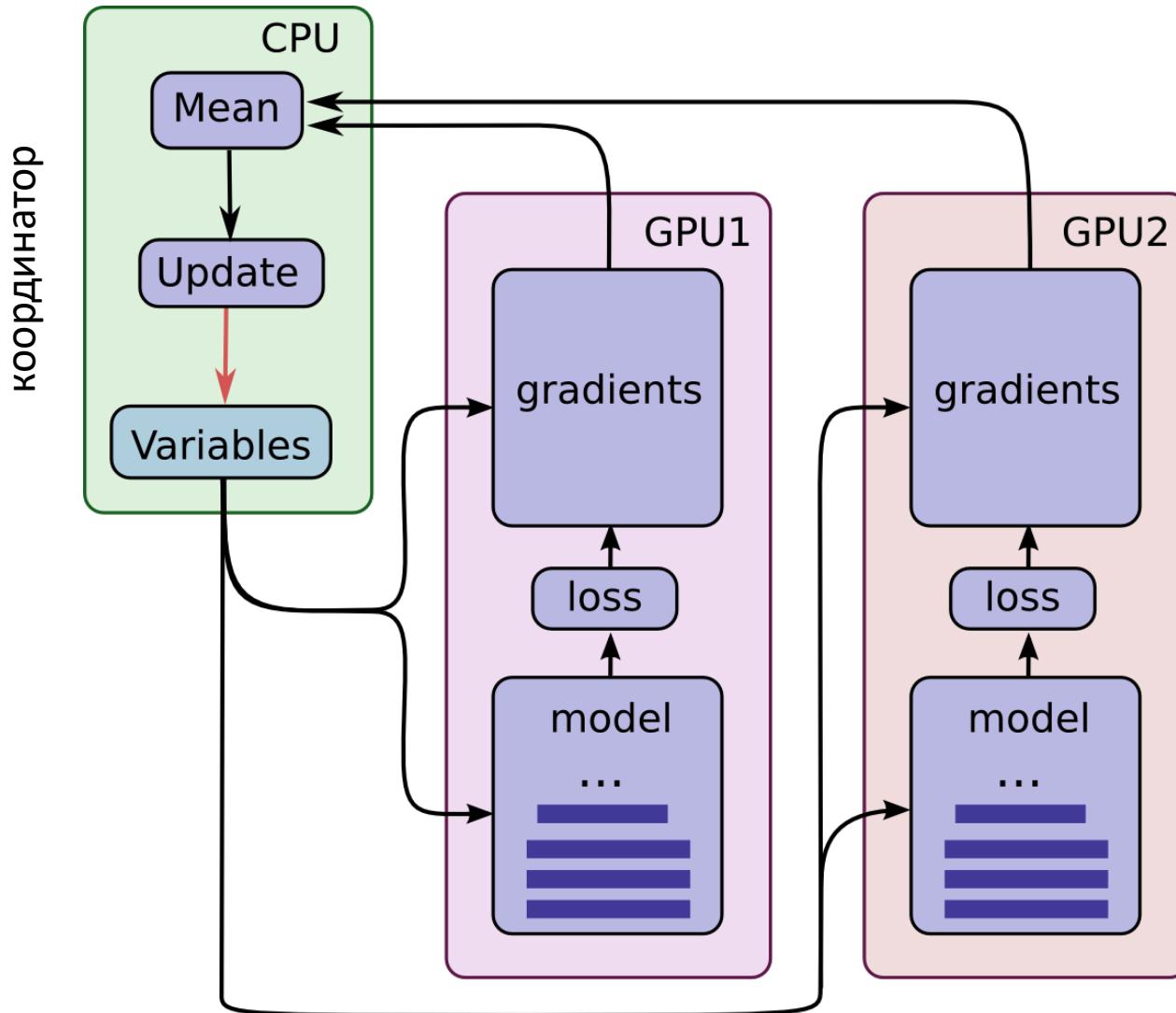
Как ускорять применение нейросети

- Если предсказываем для 1 примера:
 - Используют быстрый GPU (+ TensorRT) <https://venturebeat.com/2018/03/27/nvidia-speeds-up-deep-learning-inference-processing/>
 - Сжимают сети (teacher networks) <https://arxiv.org/pdf/1412.6550.pdf>
 - Квантуют веса сетей (быстрые INT8 операции) <https://www.tensorflow.org/performance/quantization> https://www.theregister.co.uk/2016/09/13/nvidia_p4_p40_gpu_ai/
 - Придумывают архитектуры для мобильных (MobileNet) <https://arxiv.org/pdf/1704.04861.pdf>
 - ...
- Если предсказываем для кучи примеров:
 - Embarrassingly parallel (предсказания делаются независимо)

Будем параллелизовать обучение

- Есть два подхода:
 - Data-parallel (по данным)
 - Model-parallel (по весам)

Data-parallel



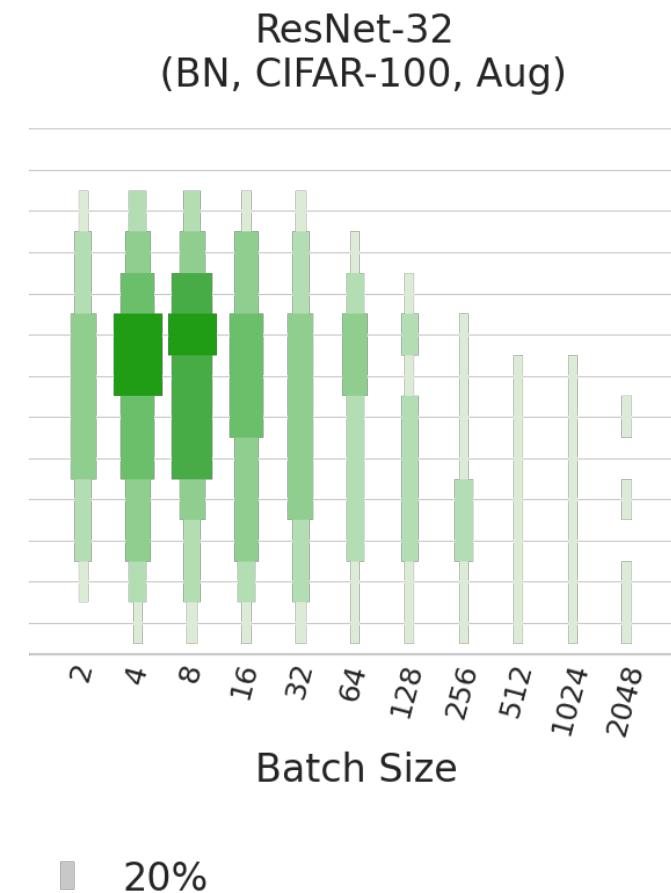
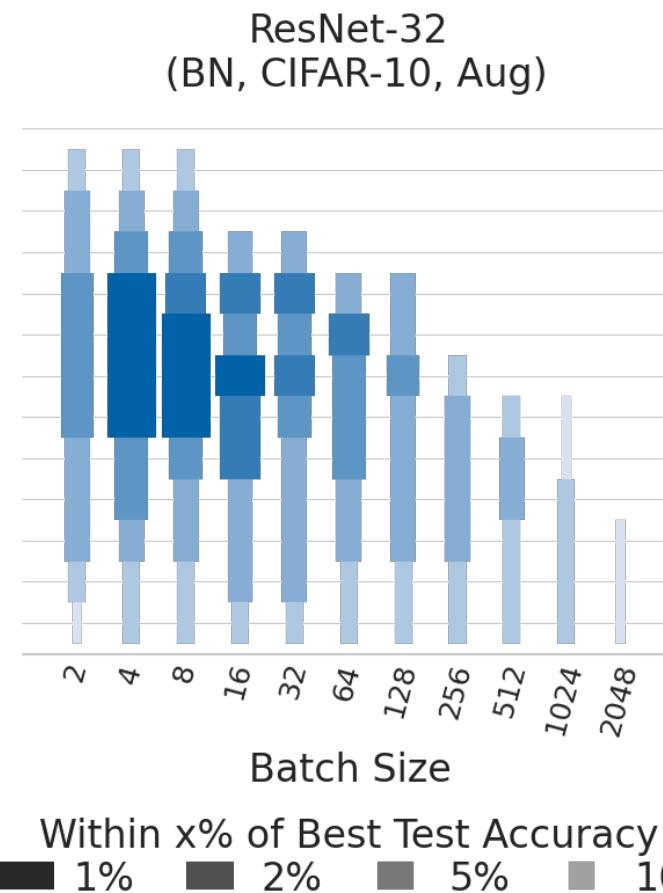
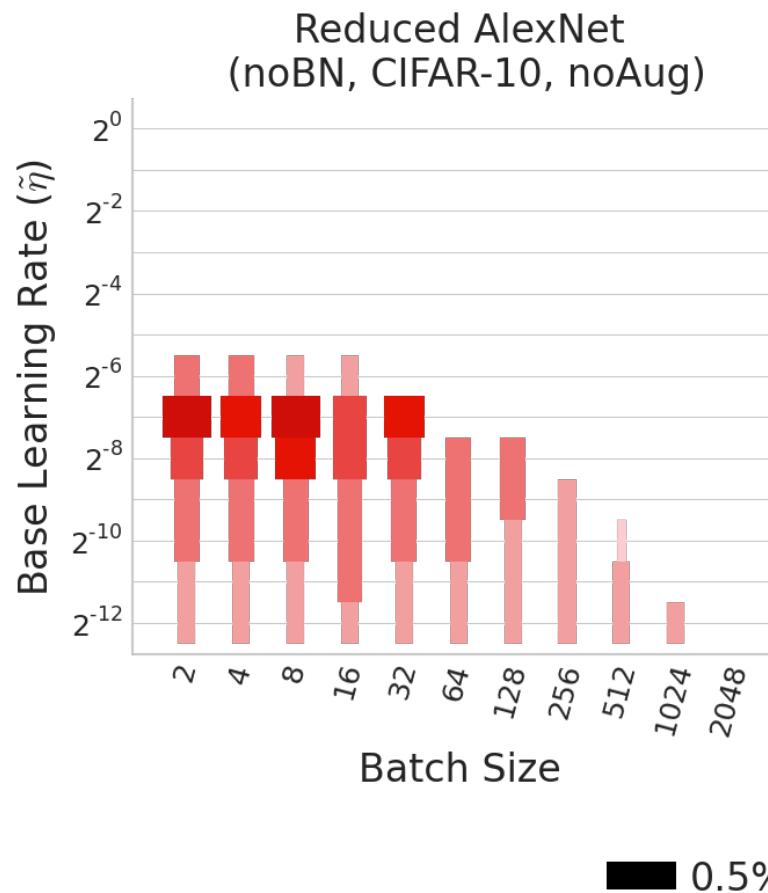
- Рассылаем копию модели на каждый GPU
- Применяем независимо к разным кусочкам батча
- Вклады в градиент отправляем **координатору** на CPU (или GPU), суммируем и заново рассылаем всем GPU

Data-parallel

- Все упирается в размер батча, сойдется ли с большим?
- Если модель большая, то долгие пересылки

Размер батча

- Кто-то говорит, что нужно брать маленький:



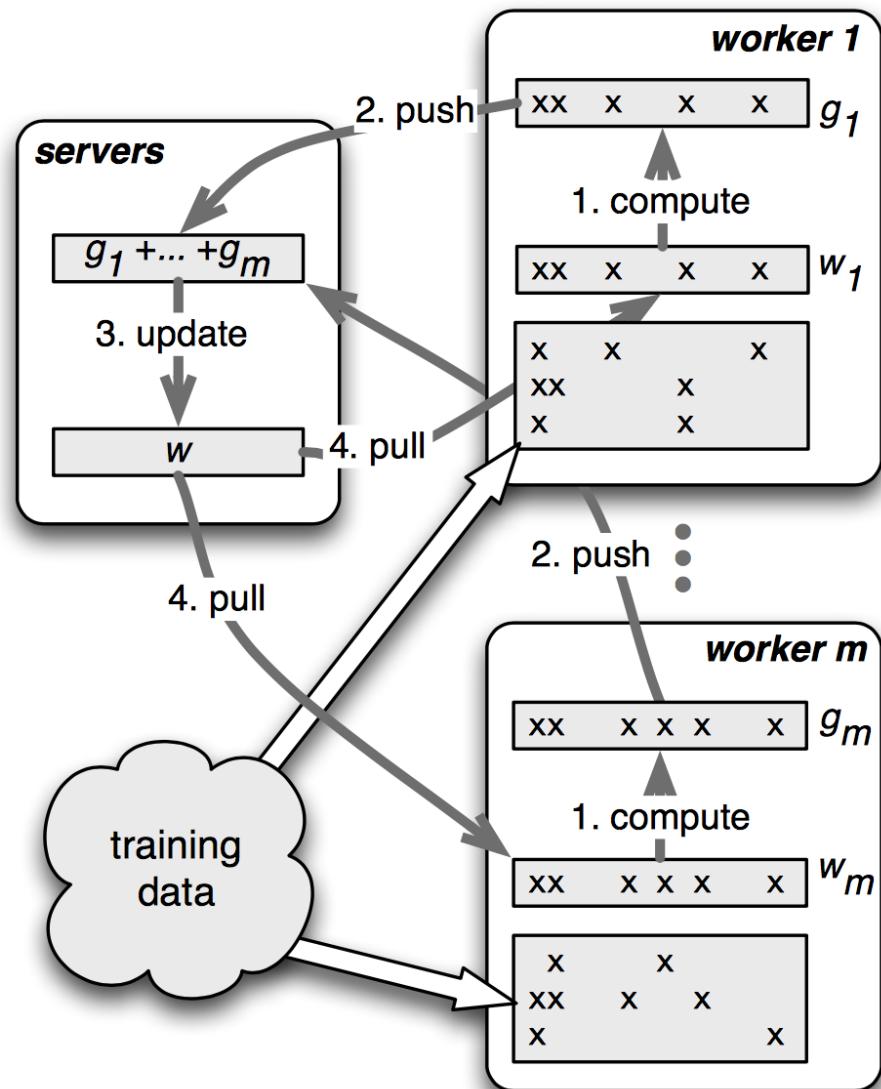
Размер батча

- Маленький плох для распараллеливания:
 - Не полностью утилизируется GPU
 - Маленькое количество воркеров

Data-parallel

- Все упирается в размер батча, сойдется ли с большим?
- **Если модель большая, то долгие пересылки**

Усложним координатор: Parameter Server (PS)



На примере линейной регрессии:

1. Воркеры читают свою часть данных и считают градиенты \mathbf{g}
2. Воркеры отсылают (**push to PS**) градиенты \mathbf{g} на сервер параметров
3. Сервер параметров обновляет свои веса \mathbf{w} на базе сообщений от воркеров
4. Воркеры просят у сервера параметров (**pull from PS**) обновленные веса \mathbf{w}

Parameter Server (PS)

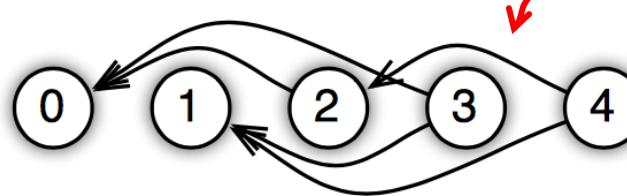
- PS может состоять из многих машин (шардирование параметров)
- Можно варьировать **консистентность** параметров, что позволяет делать вычисления асинхронно



(a) Sequential



(b) Eventual



(c) 1 Bounded delay

iter 10: gradient → push & pull → o

iter 11: gradient → push & pull → o

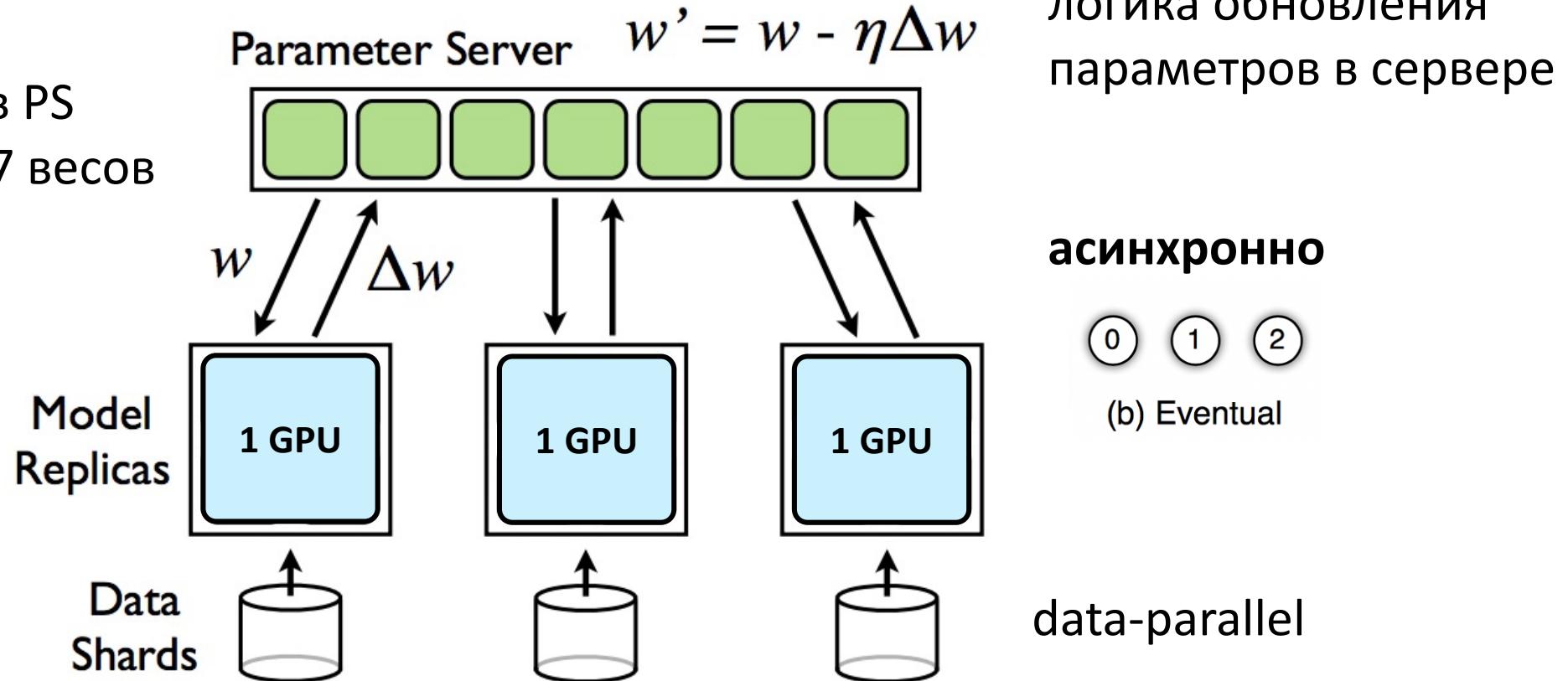
iter 12: gradient → pu

Пример:
отставать не
больше, чем
на 2 апдейта
весов

Уменьшим overhead от пересылок

- **Асинхронный SGD:** не будем ждать, пока прилетит вклад в градиент от всех машин, сделаем шаг с тем, что есть (возможно только локальный вклад)

каждая машина в PS
обрабатывает 1/7 весов



Асинхронный SGD от Google (DistBelief)

Algorithm 1.1: DOWNPOURSGDCLIENT($\alpha, n_{fetch}, n_{push}$)

procedure STARTASYNCHRONOUSLYFETCHINGPARAMETERS(*parameters*)

parameters \leftarrow GETPARAMETERSFROMPARAMSERVER()

procedure STARTASYNCHRONOUSLYPUSHINGGRADIENTS(*accruedgradients*)

SENDGRADIENTSTOPARAMSERVER(*accruedgradients*)

accruedgradients $\leftarrow 0$

Раз в сколько-то
итераций просим у
PS **свежее** состояние

main

global *parameters, accruedgradients*

step $\leftarrow 0$

accruedgradients $\leftarrow 0$

while true

if (*step mod n_{fetch}*) == 0

then STARTASYNCHRONOUSLYFETCHINGPARAMETERS(*parameters*)

data \leftarrow GETNEXTMINIBATCH()

gradient \leftarrow COMPUTEGRADIENT(*parameters, data*)

accruedgradients \leftarrow *accruedgradients* + *gradient*

parameters \leftarrow *parameters* - $\alpha * gradient$

if (*step mod n_{push}*) == 0

then STARTASYNCHRONOUSLYPUSHINGGRADIENTS(*accruedgradients*)

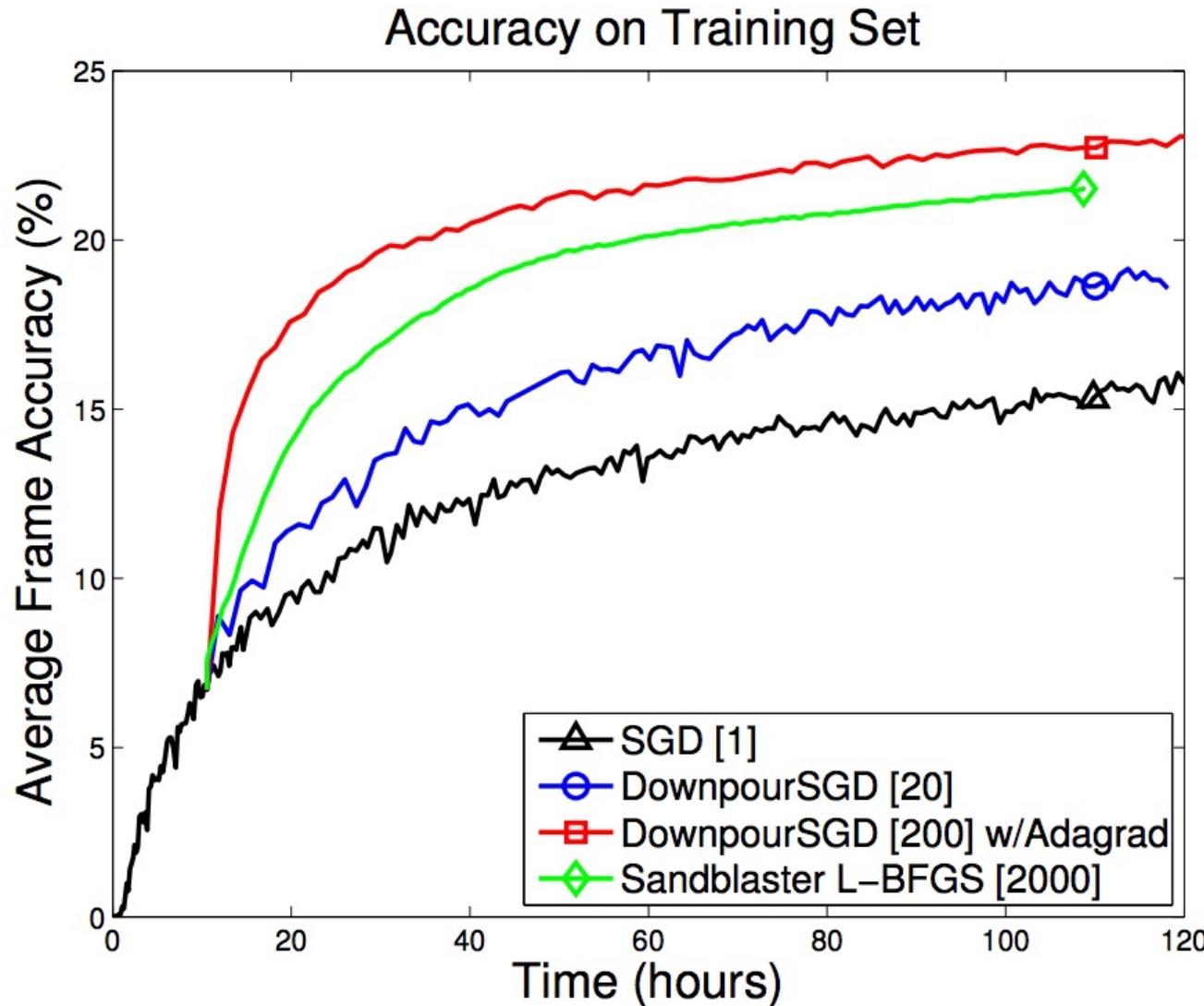
step \leftarrow *step* + 1

Раз в сколько-то
итераций посылаем
PS **накопленные**
градиенты

do

SGD на локальных данных

Асинхронный SGD от Google (DistBelief)



Запускают асинхронное обучение после 1 эпохи, видимо иначе ничего не сходится из-за запаздываний градиентов на ранних стадиях обучения

В [] скобках кол-во машин

Hogwild!

- «This work aims to show using novel theoretical analysis, algorithms, and implementation that SGD can be implemented **without any locking**.»
- «We show that when the associated **optimization problem is sparse**, meaning most gradient updates only modify small parts of the decision variable, then Hogwild! achieves a nearly optimal rate of convergence.»
- Где используется: w2v, матричные факторизации, итд

Delayed Block Proximal Gradient – пример с PS

Divide coordinates into B blocks. Set the order $b(1), \dots, b(T)$ and the maximal delay τ

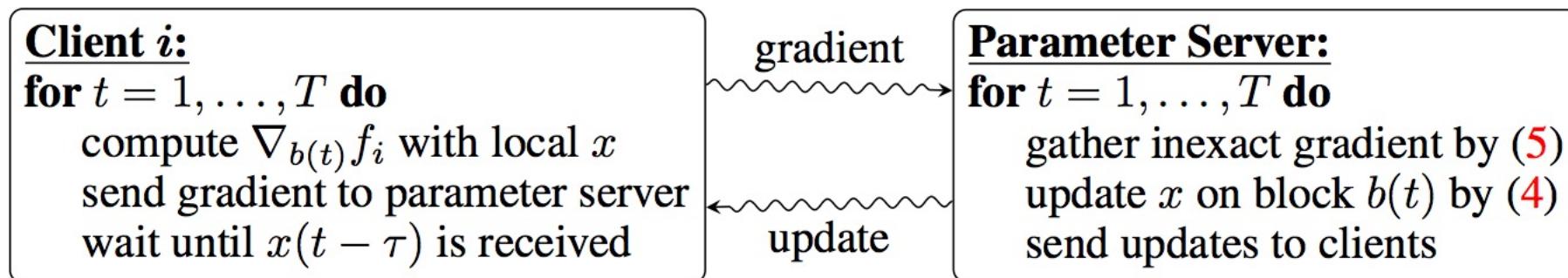
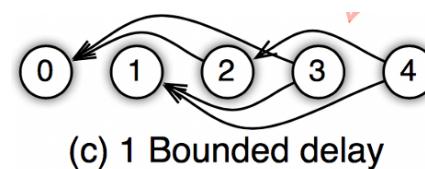


Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.

- Линейная регрессия
- На итерации t обновляем веса в блоке $b(t)$
- Хотим на каждом воркере отставать не больше, чем на τ блоков
- Асинхронно посылаем апдейты весов



Delayed Block Proximal Gradient – пример с PS

Divide coordinates into B blocks. Set the order $b(1), \dots, b(T)$ and the maximal delay τ

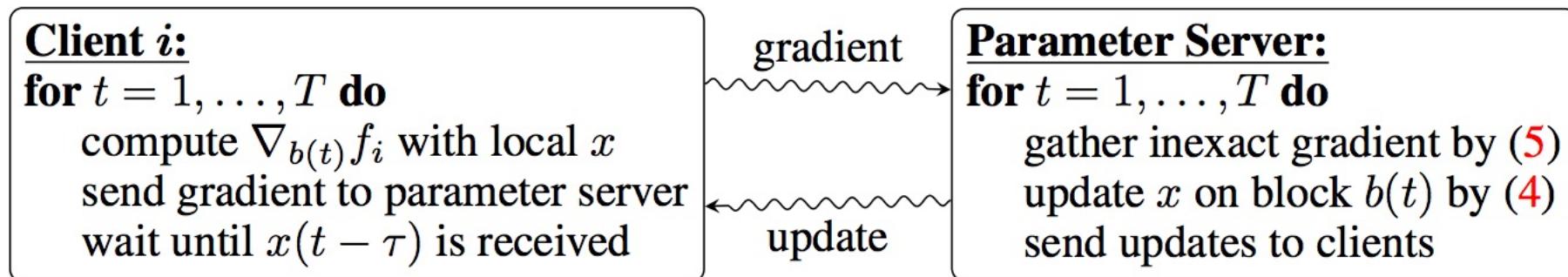
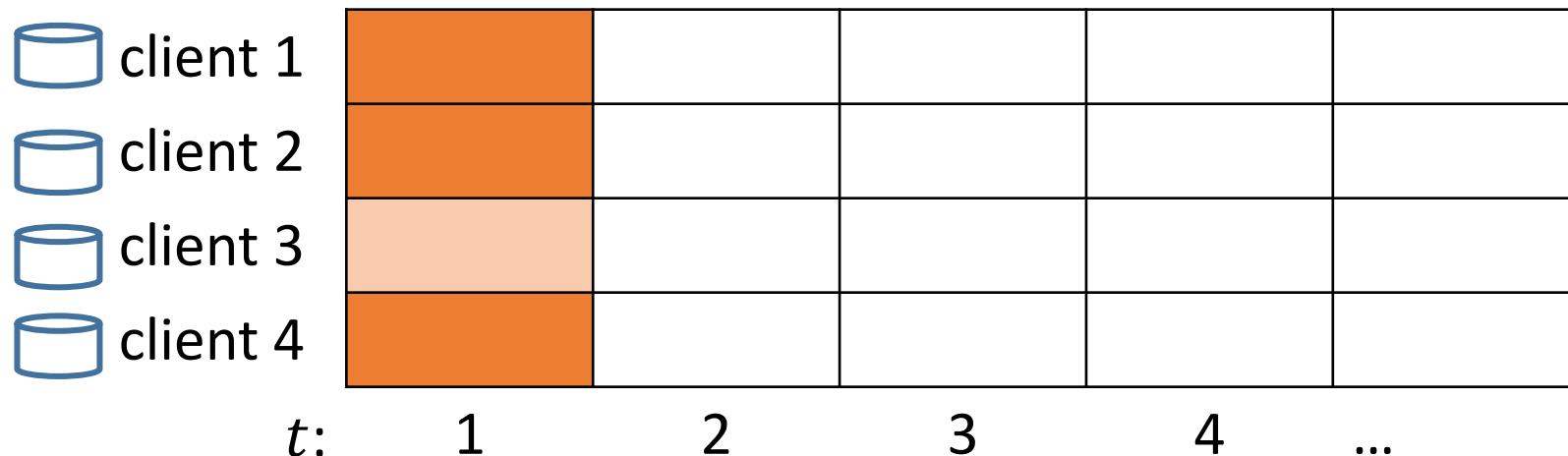


Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.



К какому-то
моменту первую
итерацию
завершили 1, 2 и 4
воркеры

Delayed Block Proximal Gradient – пример с PS

Divide coordinates into B blocks. Set the order $b(1), \dots, b(T)$ and the maximal delay τ

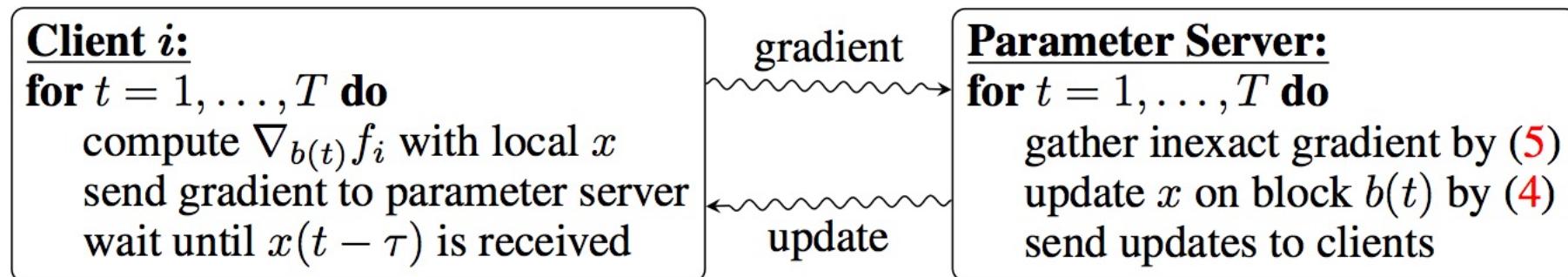
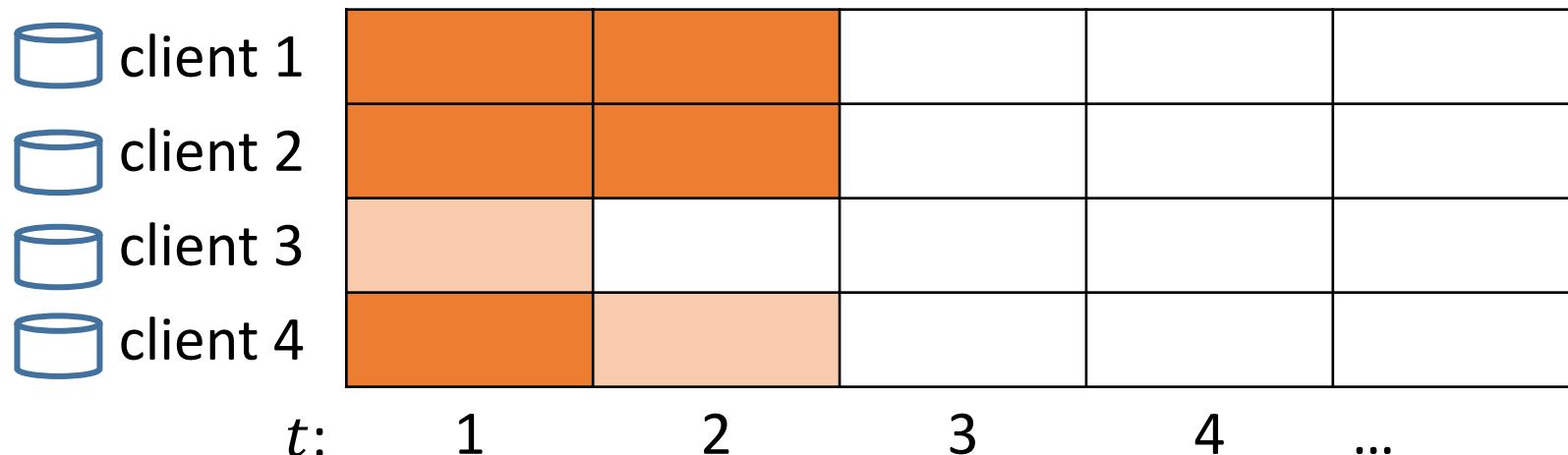


Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.



Теперь вторую итерацию закончили 1 и 2 воркеры

Delayed Block Proximal Gradient – пример с PS

Divide coordinates into B blocks. Set the order $b(1), \dots, b(T)$ and the maximal delay τ

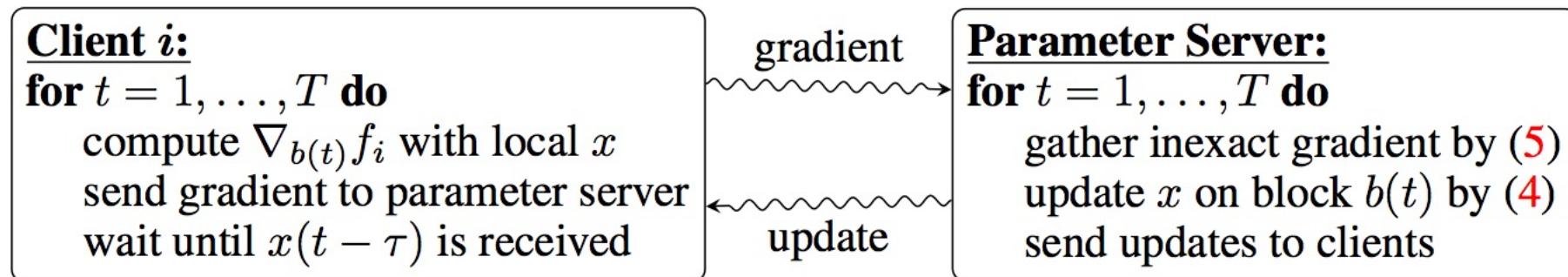
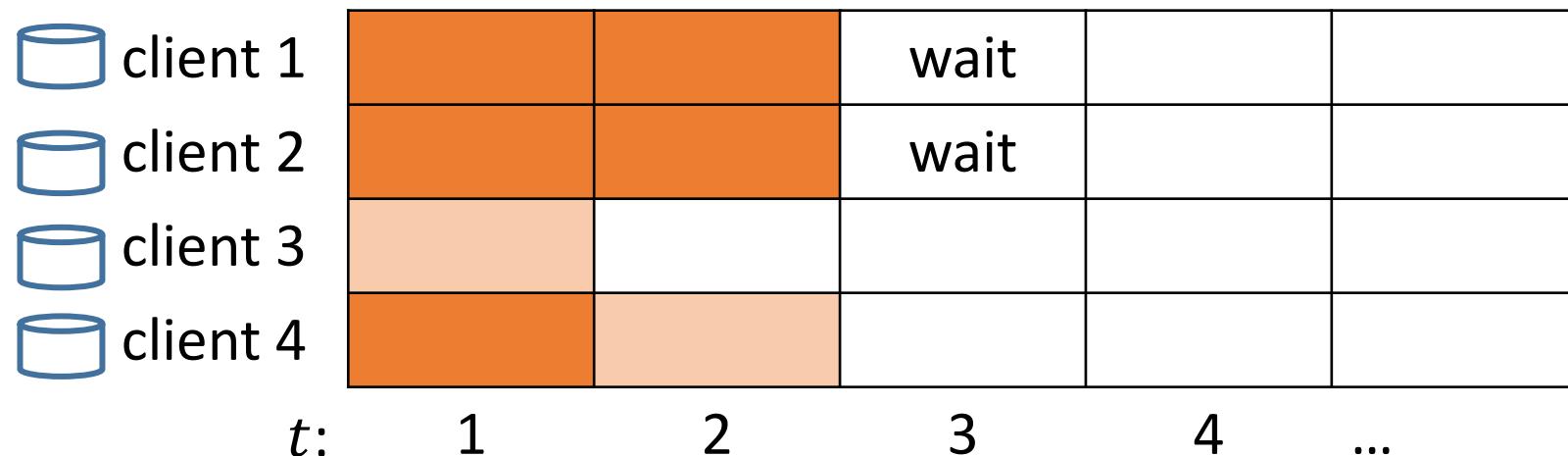


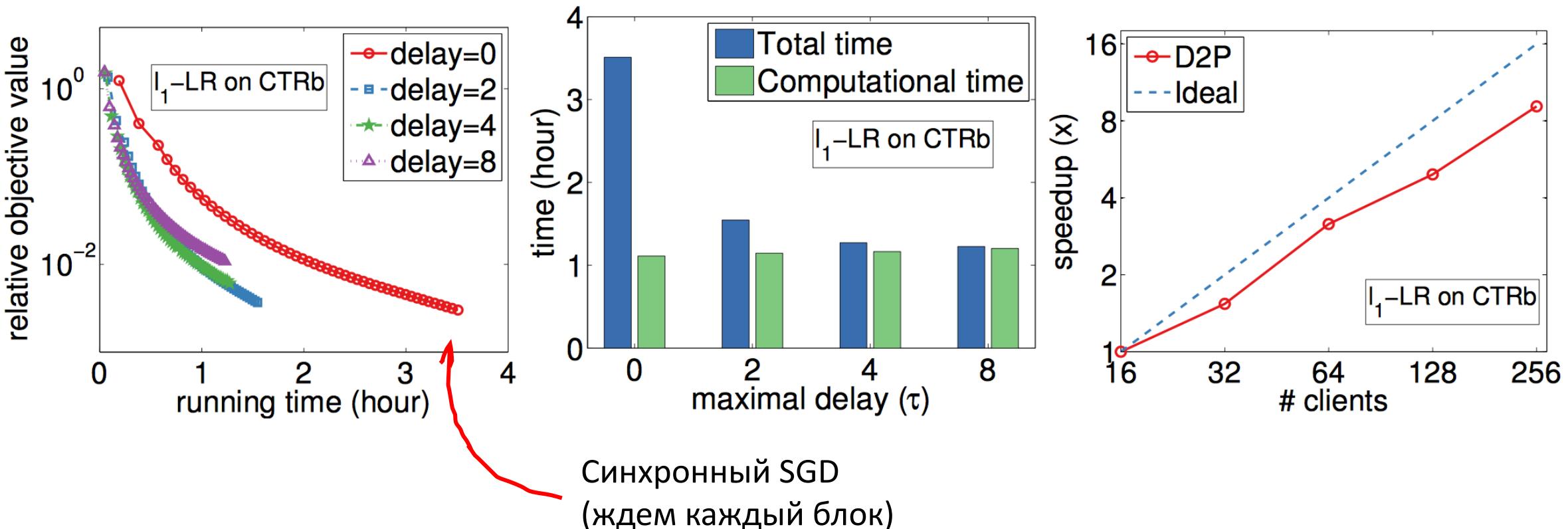
Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.



При $\tau = 2$ мы не
можем продолжить
вычисления на 1 и
2 воркере

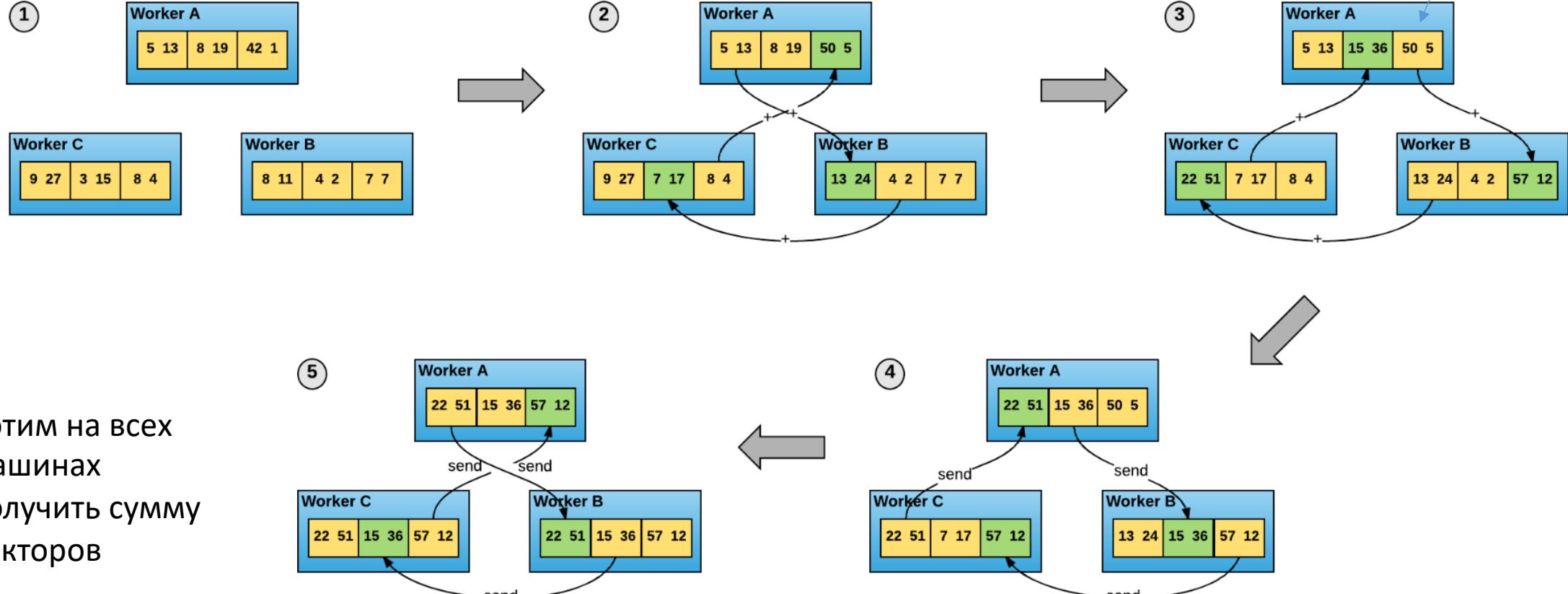
Delayed Block Proximal Gradient – пример с PS

- В τ раз меньше синхронизаций
- Замедление сходимости компенсируется ускорением одного шага



Синхронный SGD: Horovod

Как только посчитается на предыдущем шаге, сразу продолжаем

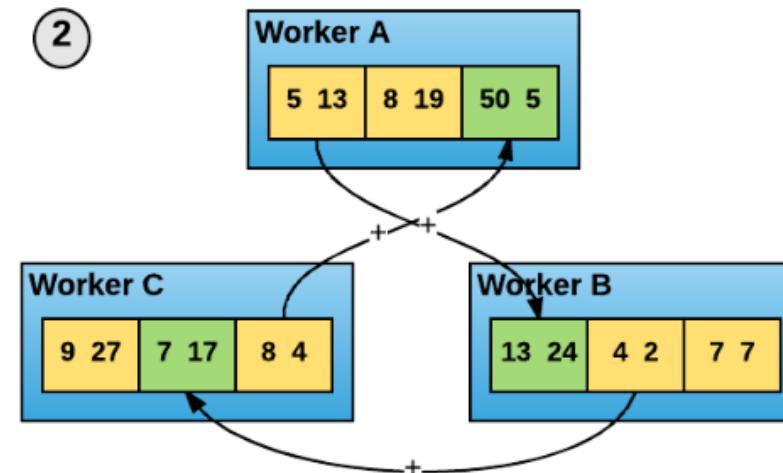


(a) Sequential

Чем-то похоже на AllReduce в VW, только без дерева

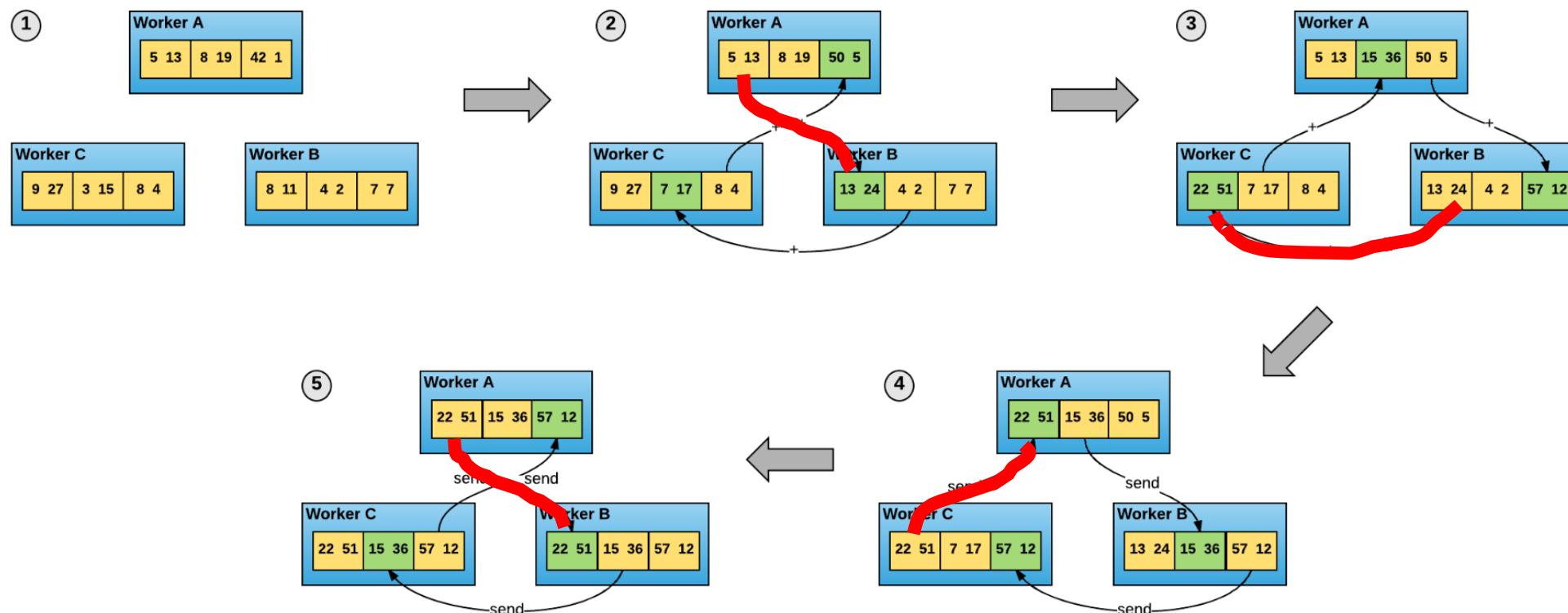
Синхронный SGD: Horovod

- Data-parallel (режем batch)
- Все пересылки peer-to-peer
- Вот тут нет блокировок (эффективно утилизирует сеть и GPU):



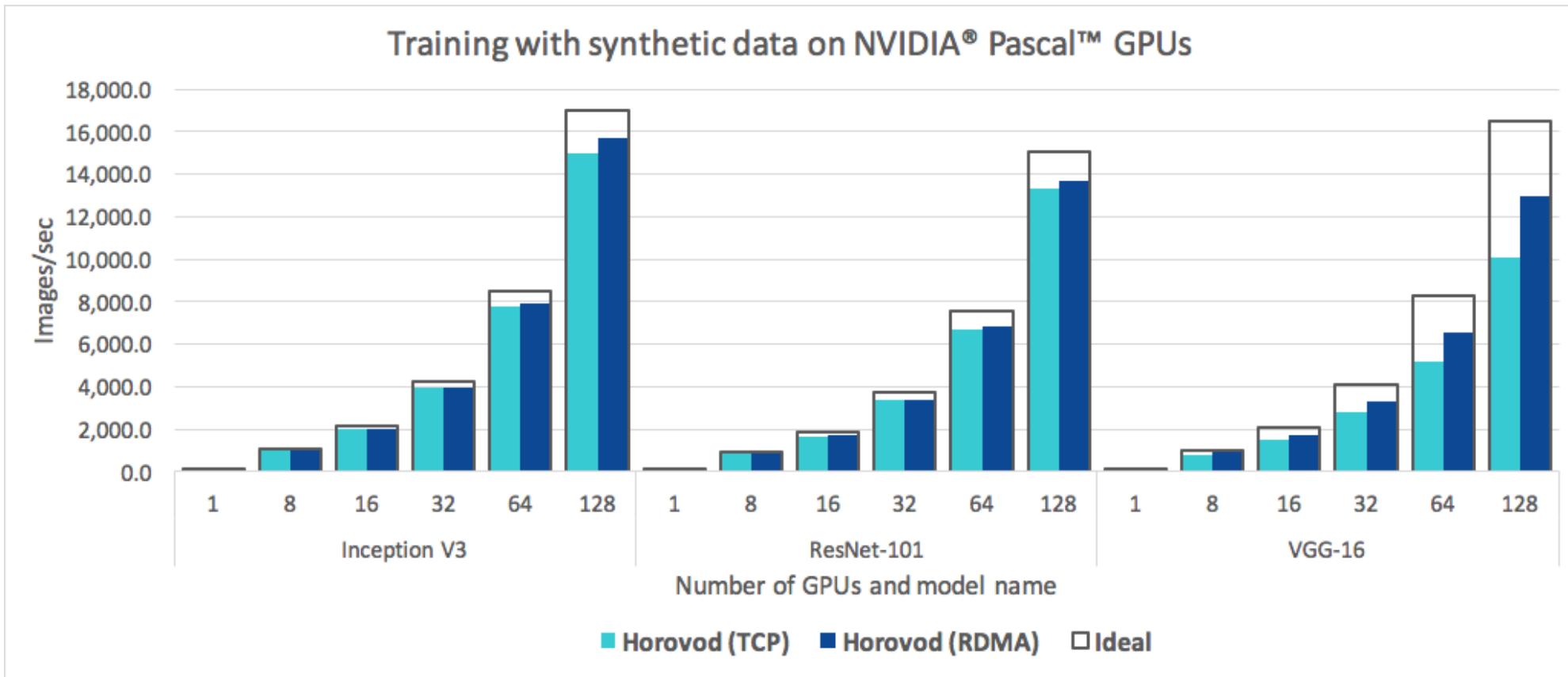
Синхронный SGD: Horovod

- По сути мы обмениваемся каждым блоком (независимо) по кругу (видимо поэтому так и назвали):



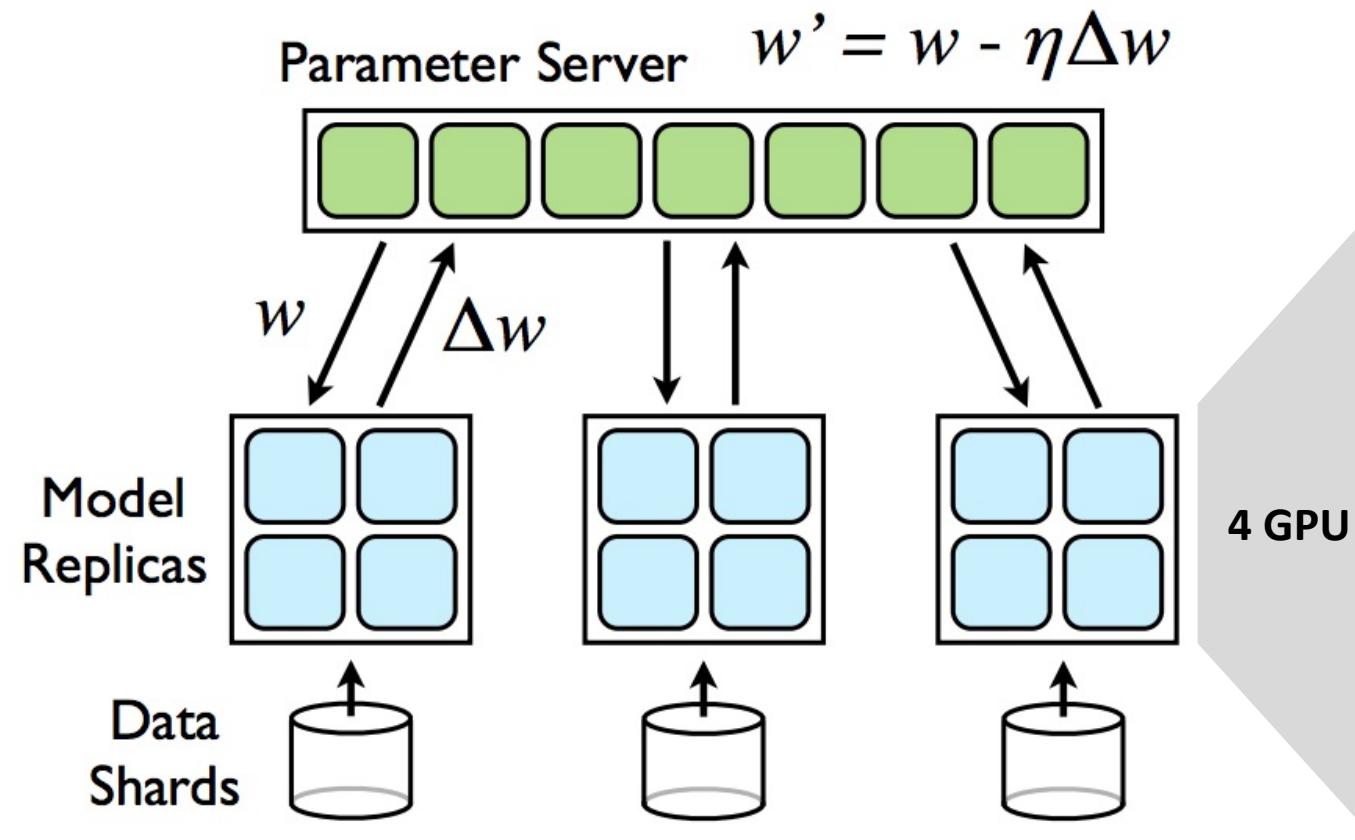
Синхронный SGD: Horovod

- Очень классно скейлится:

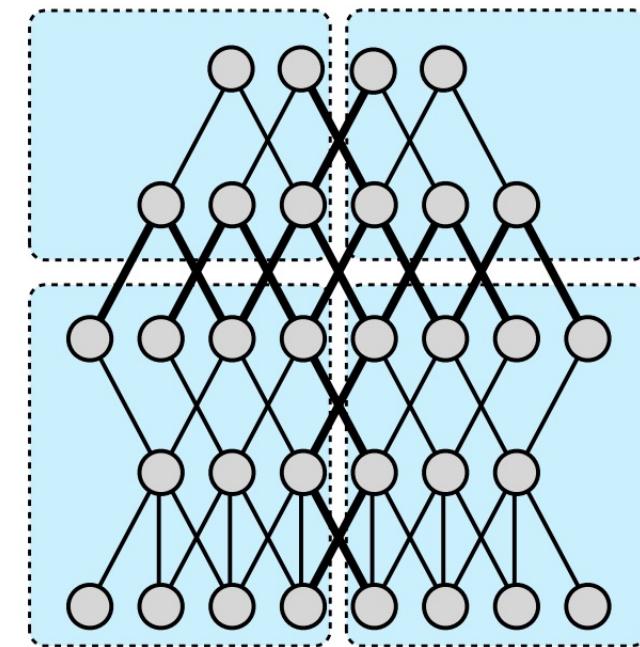


Model-parallel (мало кому нужен)

- Режем веса на партиции (можем их обновлять асинхронно)

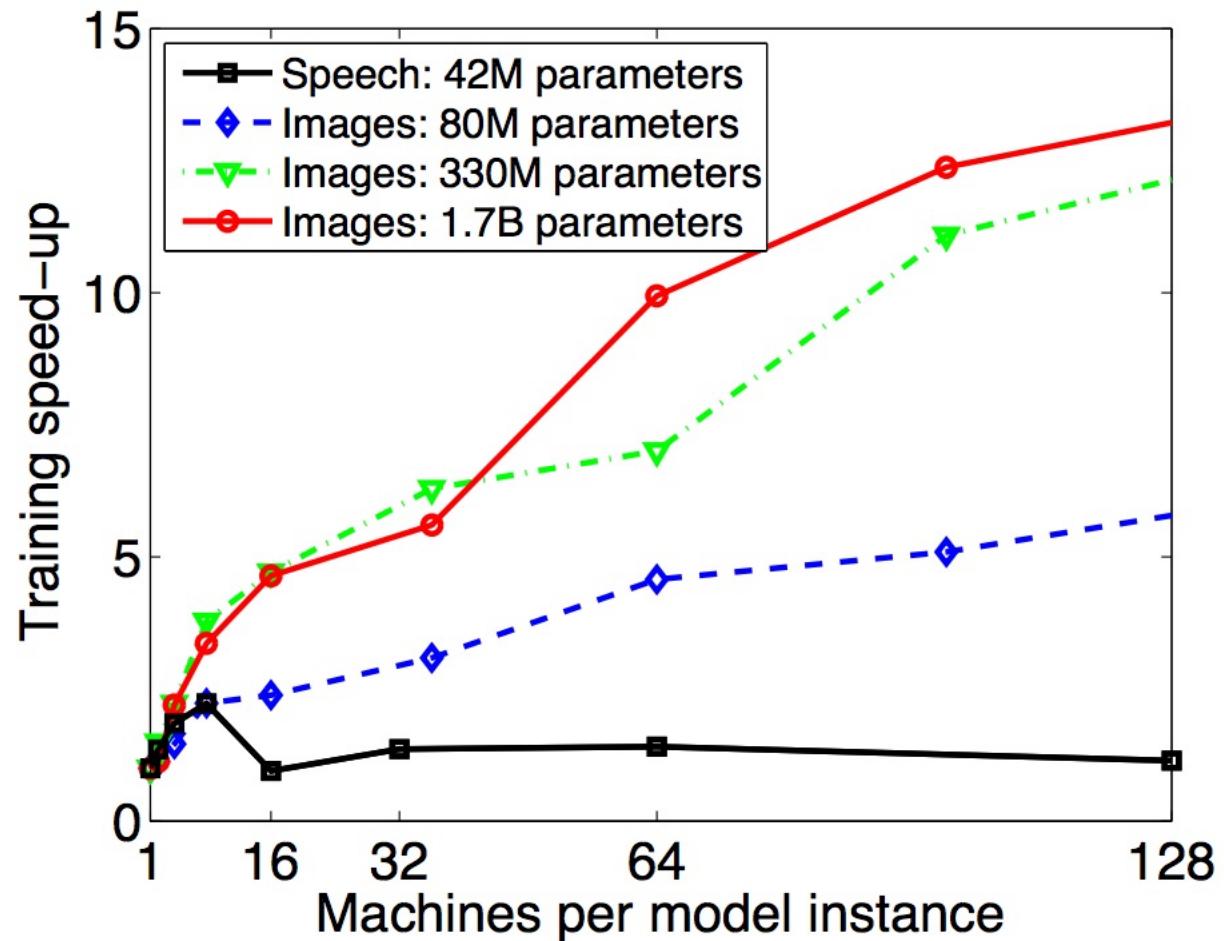


Для сверточных сетей ОК



Model-parallel в DistBelief

Для сверточных сетей ОК



Побили в свое время
state-of-the-art
на большом ImageNet

Model-parallel

- Сложнее реализовывать (нужны выходы нейронов от соседа)
- Не всегда нужно, как правило модель не очень большая

Ссылки

- Статья о Parameter Server

[https://www.cs.cmu.edu/~muli/file/parameter server osdi14.pdf](https://www.cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf)

- Еще про PS

[http://opt.kyb.tuebingen.mpg.de/papers/opt2013 submission 1.pdf](http://opt.kyb.tuebingen.mpg.de/papers/opt2013_submission_1.pdf)

- Hogwild!

<https://arxiv.org/pdf/1106.5730.pdf>