SCIENCE
PASSION
TECHNOLOGY

Architecture of ML Systems
06 Parameter Servers

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

Last update: May 03, 2019

Announcements/Org

- **#1 Programming/Analysis Projects**
  - **#1 Auto Differentiation**
  - **#5 LLVM Code Generator**
  - **#12 Information Extraction from Unstructured PDF/HTML**

- **#2 Study Abroad Fair 2019**

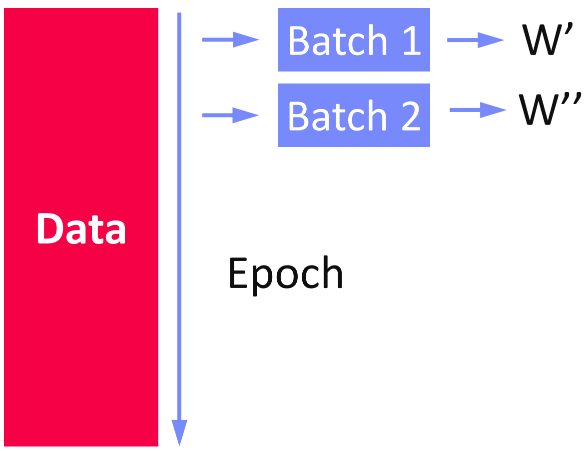
Agenda

- **Data-Parallel Parameter Servers**
- **Model-Parallel Parameter Servers**
- **Federated Machine Learning**

Data-Parallel Parameter Servers

Background: Mini-batch ML Algorithms

- **Mini-batch ML Algorithms**
  - Iterative ML algorithms, where each iteration only uses a **batch of rows** to make the next model update (in **epochs** over the data)
  - For large and **highly redundant training sets**
  - **Applies to almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)



- **Statistical vs Hardware Efficiency** (batch size)
  - **Statistical efficiency:** number of accessed data points to achieve certain accuracy
  - **Hardware efficiency:** number of independent computations to achieve high hardware utilization (parallelization at different levels)
  - **Beware higher variance / class skew for too small batches!**

➔ **Training Mini-batch ML Algortihms sequentially is hard to scale**

Background: Mini-batch DNN Training (LeNet)

```
# Initialize W1-W4, b1-b4
# Initialize SGD w/ Nesterov momentum optimizer
iters = ceil(N / batch_size)

for( e in 1:epochs ) {
   for( i in 1:iters ) {
      X_batch = X[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]
      y_batch = Y[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]

      ## layer 1: conv1 -> relu1 -> pool1
      ## layer 2: conv2 -> relu2 -> pool2
      ## layer 3:  affine3 -> relu3 -> dropout
      ## layer 4:  affine4 -> softmax
      outa4 = affine::forward(outd3, W4, b4)
      probs = softmax::forward(outa4)

      ## layer 4:  affine4 -> softmax
      douta4 = softmax::backward(dprobs, outa4)
      [doutd3, dW4, db4] = affine::backward(douta4, outr3, W4, b4)
      ## layer 3:  affine3 -> relu3 -> dropout
      ## layer 2: conv2 -> relu2 -> pool2
      ## layer 1: conv1 -> relu1 -> pool1

      # Optimize with SGD w/ Nesterov momentum W1-W4, b1-b4
      [W4, vW4] = sgd_nesterov::update(W4, dW4, lr, mu, vW4)
      [b4, vb4] = sgd_nesterov::update(b4, db4, lr, mu, vb4)
   }
}
```

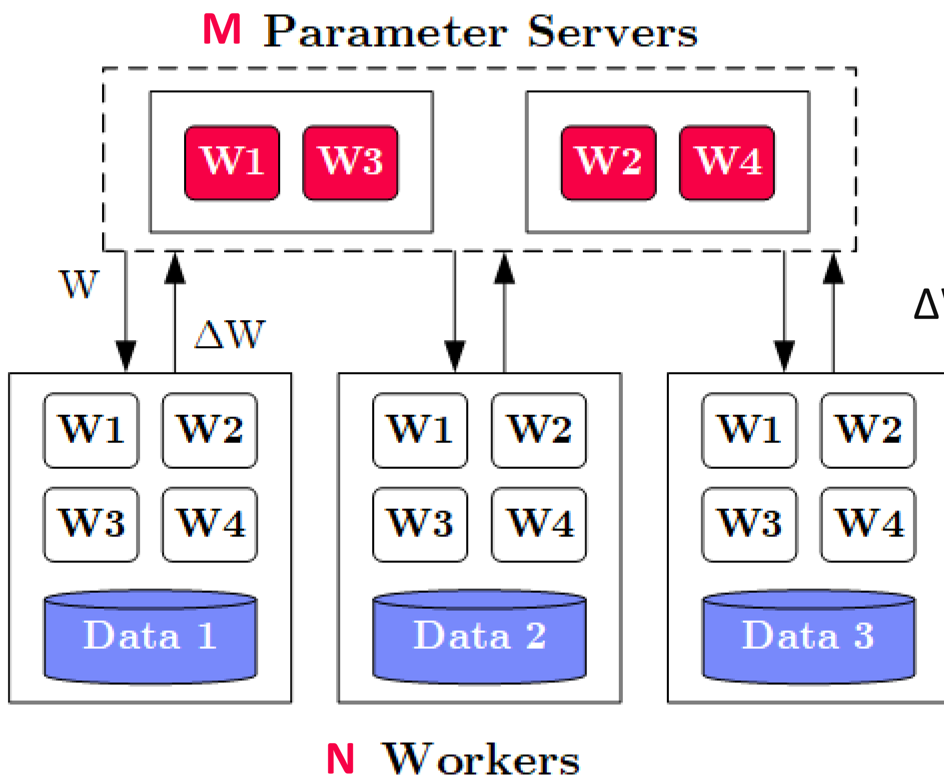
NN Forward Pass

NN Backward Pass
→ Gradients

Model Updates

Overview Parameter Servers

- **System Architecture**
  - M Parameter Servers
  - N Workers
  - Optional Coordinator



W .. Model
ΔW .. Gradient

- **Key Techniques**
  - Data partitioning D → workers Di (e.g., disjoint, reshuffling)
  - Updated strategies (e.g., synchronous, asynchronous)
  - Batch size strategies (small/large batches, hybrid methods)

History of Parameter Servers

- **1st Gen: Key/Value**
  - **Distributed key-value store** for parameter exchange and synchronization
  - Relatively high overhead
- **2nd Gen: Classic Parameter Servers**
  - **Parameters as dense/sparse matrices**
  - Different **update/consistency strategies**
  - Flexible configuration and fault tolerance
- **3rd Gen: Parameter Servers w/ improved data communication**
  - Prefetching and range-based pull/push
  - Lossy or lossless compression w/ compensations
- **Examples**
  - TensorFlow, MXNet, PyTorch, CNTK, Petuum

[Alexander J. Smola, Shravan M. Narayanamurthy: An Architecture for Parallel Topic Models. **PVLDB 2010**]

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

[Mu Li et al: Scaling Distributed Machine Learning with the Parameter Server. **OSDI 2014**]

[Jiawei Jiang, Bin Cui, Ce Zhang, Lele Yu: Heterogeneity-aware Distributed Parameter Servers. **SIGMOD 2017**]

[Jiawei Jiang et al: SketchML: Accelerating Distributed Machine Learning with Data Sketches. **SIGMOD 2018**]

Basic Worker Algorithm (batch)

```
for( i in 1:epochs ) {
   for( j in 1:iterations ) {
      params = pullModel(); # W1-W4, b1-b4 lr, mu
      batch = getNextMiniBatch(data, j);
      gradient = computeGradient(batch, params);
      pushGradients(gradient);
   }
}
```

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Extended Worker Algorithm (nfetch batches)

```
gradientAcc = matrix(0,...);
for( i in 1:epochs ) {
   for( j in 1:iterations ) {
      if( step mod nfetch = 0 )
         params = pullModel();
      batch = getNextMiniBatch(data, j);
      gradient = computeGradient(batch, params);
      gradientAcc += gradient;
      params = updateModel(params, gradients);
      if( step mod nfetch = 0 ) {
         pushGradients(gradientAcc); step = 0;
         gradientAcc = matrix(0, ...);
      }
      step++;
}  }
```

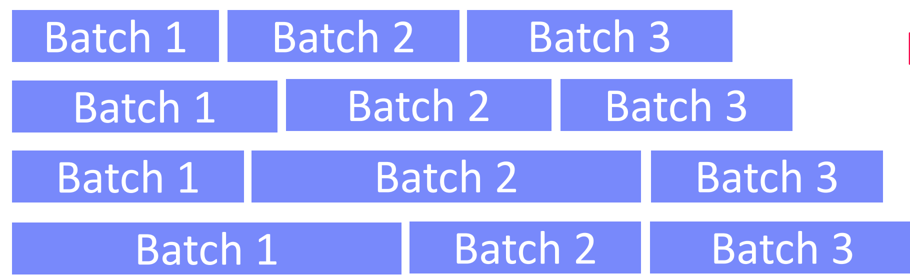
nfetch batches require
local gradient accrual and
local model update

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Update Strategies

- **Bulk Synchronous Parallel (BSP)**
  - Update model w/ accrued gradients
  - Barrier for N workers

- **Asynchronous Parallel (ASP)**
  - Update model for each gradient
  - No barrier



but, stale model updates

- **Synchronous w/ Backup Workers**
  - Update model w/ accrued gradients
  - Barrier for N of N+b workers

[Martín Abadi et al: TensorFlow: A System for Large-Scale Machine Learning. **OSDI 2016**]

Update Strategies, cont.

- **Stale-Synchronous Parallel (SSP)**
  - Similar to backup workers, **weak synchronization barrier**
  - Maximum staleness of s clocks between fastest and slowest worker → **if violated, block fastest**

[Qirong Ho et al: More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. **NIPS 2013**]

- **Hogwild!**
  - Even the model update completely **unsynchronized**
  - Shown to converge for **sparse model updates**

[Benjamin Recht, Christopher Ré, Stephen J. Wright, Feng Niu: Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. **NIPS 2011**]

- **Decentralized**
  - #1: Exchange partial gradients updates with local peers
  - #2: Peer-to-peer re-assignment of work
  - Other Examples: **Ako**, **FlexRR**

[Xiangru Lian et al: Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. **NIPS 2017**]

Data Partitioning Schemes

- **Goals Data Partitioning**
  - Even distribute data across workers, avoid skew regarding model updates

- **#1 Disjoint Contiguous**
  - Contiguous row partition of features/labels

```
Xp = X[id*blocksize+1:
  (id+1)*blocksize,];
```

- **#2 Disjoint Round Robin**
  - Rows of features distributed round robin

```
Xp = removeEmpty(X, 'rows',
  seq(1,nrow(X))%%N==id);
```

- **#3 Disjoint Random**
  - Random non-overlapping selection of rows

```
P = table(seq(1,nrow(X)),
sample(nrow(X),nrow(X),FALSE));

Xp = P[id*blocksize+1:
(id+1)*blocksize,] %*% X
```

- **#4 Overlap Reshuffle**
  - Each worker receives a reshuffled copy of the whole dataset

```
Xp = Pi %*% X
```

Example Distributed TensorFlow DP

```
# Create a cluster from the parameter server and worker hosts
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

# Create and start a server for the local task.
server = tf.train.Server(cluster, job_name=..., task_index=...)

# On worker: initialize loss
train_op = tf.train.AdagradOptimizer(0.01).minimize(
   loss, global_step=tf.contrib.framework.get_or_create_global_step())

# Create training session and run steps asynchronously
hooks=[tf.train.StopAtStepHook(last_step=1000000)]
with tf.train.MonitoredTrainingSession(master=server.target,
   is_chief=(task_index == 0), checkpoint_dir=..., hooks=hooks) as sess:
   while not mon_sess.should_stop():
      sess.run(train_op)

# Program needs to be started on ps and worker
```

But new experimental APIs and Keras Frontend

Example SystemML Parameter Server

```
# Initialize SGD w/ Adam optimizer
[mW1, vW1] = adam::init(W1); [mb1, vb1] = adam::init(b1); ...

# Create the model object
modelList = list(W1, W2, W3, W4, b1, b2, b3, b4, vW1, vW2, vW3, vW4,
  vb1, vb2, vb3, vb4, mW1, mW2, mW3, mW4, mb1, mb2, mb3, mb4);

# Create the hyper parameter list
params = list(lr=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8, t=0,
  C=C, Hin=Hin, Win=Win, Hf=Hf, Wf=Wf, stride=1, pad=2, lambda=5e-04,
  F1=F1, F2=F2, N3=N3)

# Use paramserv function
modelList2 = paramserv(model=modelList, features=X, labels=Y,
  upd=funGradients, aggregation=funUpdate, mode=REMOTE_SPARK, utype=ASP,
  freq=BATCH, epochs=200, batchsize=64, k=144, scheme=DISJOINT_RANDOM,
  hyperparams=params)
```

Selected Optimizers

- **Stochastic Gradient Descent (SGD)**
  - Vanilla SGD, basis for many other optimizers

```
X = X - lr*dX
```

- **SGD w/ Momentum**
  - Assumes parameter velocity w/ momentum

```
v = mu*v - lr*dX
X = X + v
```

- **SGD w/ Nesterov Momentum**
  - Assumes parameter velocity w/ momentum, but update from position **after** momentum

```
v0 = v
v = mu*v - lr*dX
X = X - mu*v0 + (1+mu)*v
```

- **AdaGrad**
  - Adaptive learning rate with regret guarantees

[John C. Duchi et al: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. **JMLR 2011**]

- **RMSprop**
  - Adaptive learning rate, extended AdaGrad

```
c = dr*c+(1-dr)*dX^2
X = X-(lr*dX/(sqrt(c)+eps))
```

- **Adam**
  - Individual adaptive learning rates for different parameters

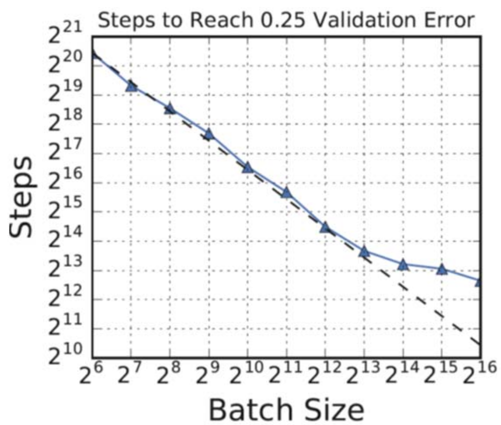
[Diederik P. Kingma, Jimmy Ba: Adam: A Method for Stochastic Optimization. **ICLR 2015**]

Batch Size Configuration

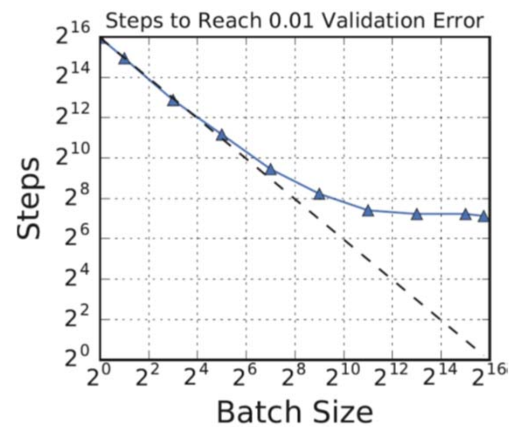
- **What is the right batch size for my data?**
  - Maximum useful batch size is dependent on data redundancy and model complexity

[Christopher J. Shallue et al.: Measuring the Effects of Data Parallelism on Neural Network Training. **CoRR 2018**]

ResNet-50 on ImageNet



VS



Simple CNN on MNIST

- **Additional Heuristics/Hybrid Methods**
  - #1 Increase the batch size instead of decaying the learning rate
  - #2 Combine batch and mini-batch algorithms (full batch + n online updates)

[Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le: Don't Decay the Learning Rate, Increase the Batch Size. **ICLR 2018**]

[Ashok Cutkosky, Róbert Busa-Fekete: Distributed Stochastic Optimization via Adaptive SGD. **NeurIPS 2018**]

Reducing Communication Overhead

- **Large Batch Sizes**
  - Larger batch sizes inherently reduce the relative communication overhead
- **Overlapping Computation/Communication**
  - For complex models with many weight/bias matrices, computation and push/pull communication can be overlapped according to data dependencies
  - This can be combined with prefetching of weights
- **Sparse and Compressed Communication**
  - For mini-batches of sparse data, sparse gradients can be communicated
  - Lossy (mantissa truncation, quantization), and lossless (delta, bitpacking) compression for weights and gradients

[Frank Seide et al: **1-bit stochastic gradient descent** and its application to data-parallel distributed training of speech DNNs. **INTERSPEECH 2014**]

  - Gradient sparsification (send gradients larger than a threshold)
  - Gradient dropping (drop fraction of positive/negative updates)

Model-Parallel Parameter Servers

Problem Setting

- **Limitations Data-Parallel Parameter Servers**
  - Need to fit entire model and activations of entire network into each worker node/device (or accept overhead for repeated eviction and restore)
  - Existence of very deep and wide networks (e.g., **ResNet-1001**)

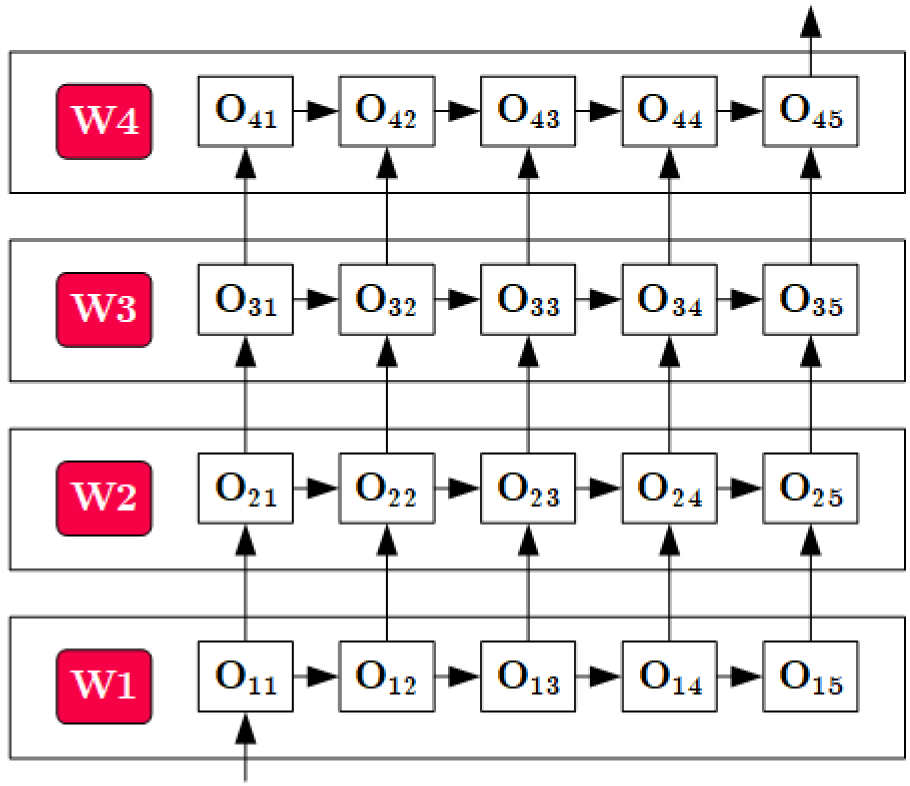
- **Model-Parallel Parameter Servers**
  - Workers responsible for **disjoint partitions of the network/model**
  - Exploit pipeline parallelism and independent subnetworks

- **Hybrid Parameter Servers**
  - *"To be successful, however, we believe that model parallelism must be combined with clever distributed optimization techniques that leverage data parallelism."*
  - *"[…] it is possible to use **tens of thousands of CPU cores** for training a single model"*

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Overview Model-Parallel Execution

- **System Architecture**
  - Nodes act as workers and parameter servers
  - Data Transfer for boundary-crossing data dependencies
- **Pipeline Parallelism**



Workers w/ disjoint network/model partitions

Example Distributed TensorFlow MP

```
# Place variables and ops on devices
with tf.device("/gpu:0"):
   a = tf.Variable(tf.random.uniform(...))
   a = tf.square(a)
with tf.device("/gpu:1"):
   b = tf.Variable(tf.random.uniform(...))
   b = tf.square(b)
with tf.device("/cpu:0"):
   loss = a+b

# Declare optimizer and parameters
opt = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = opt.minimize(loss)

# Force distributed graph evaluation
ret = sess.run([loss, train_op]))
```

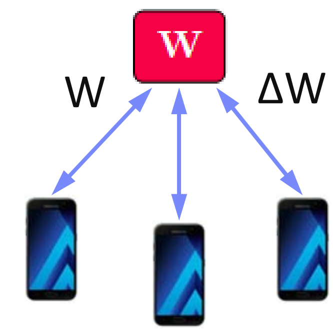
Explicit Placement of Operations

(shown via toy example)

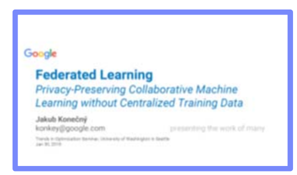
Federated Machine Learning

Problem Setting and Overview

- **Motivation Federated ML**
  - Learn model **w/o central data consolidation**
  - **Privacy + data/power caps** vs **personalization and sharing**
  - Applications Characteristics
    - #1 On-device data more relevant than server-side data
    - #2 On-device data is privacy-sensitive or large
    - #3 Labels can be inferred naturally from user interaction
  - **Example:** Language modeling for mobile keyboards and voice recognition



- **Challenges**
  - Massively distributed (data stored across many devices)
  - Limited and unreliable communication
  - Unbalanced data (skew in data size, non-IID)
  - Unreliable compute nodes / data availability



[Jakub Konečný: Federated Learning - Privacy-Preserving Collaborative Machine Learning without Centralized Training Data, **UW Seminar 2018**]

A Federated ML Training Algorithm

while(!converged) {

1. Select random subset (e.g. 1000) of the (online) clients

2. In parallel, send current parameters θ_t to those clients

At each client

2a. Receive parameters θ_t from server **[pull]**

2b. Run some number of minibatch SGD steps, producing θ'

2c. Return $\theta' - \theta_t$ (model averaging) **[push]**

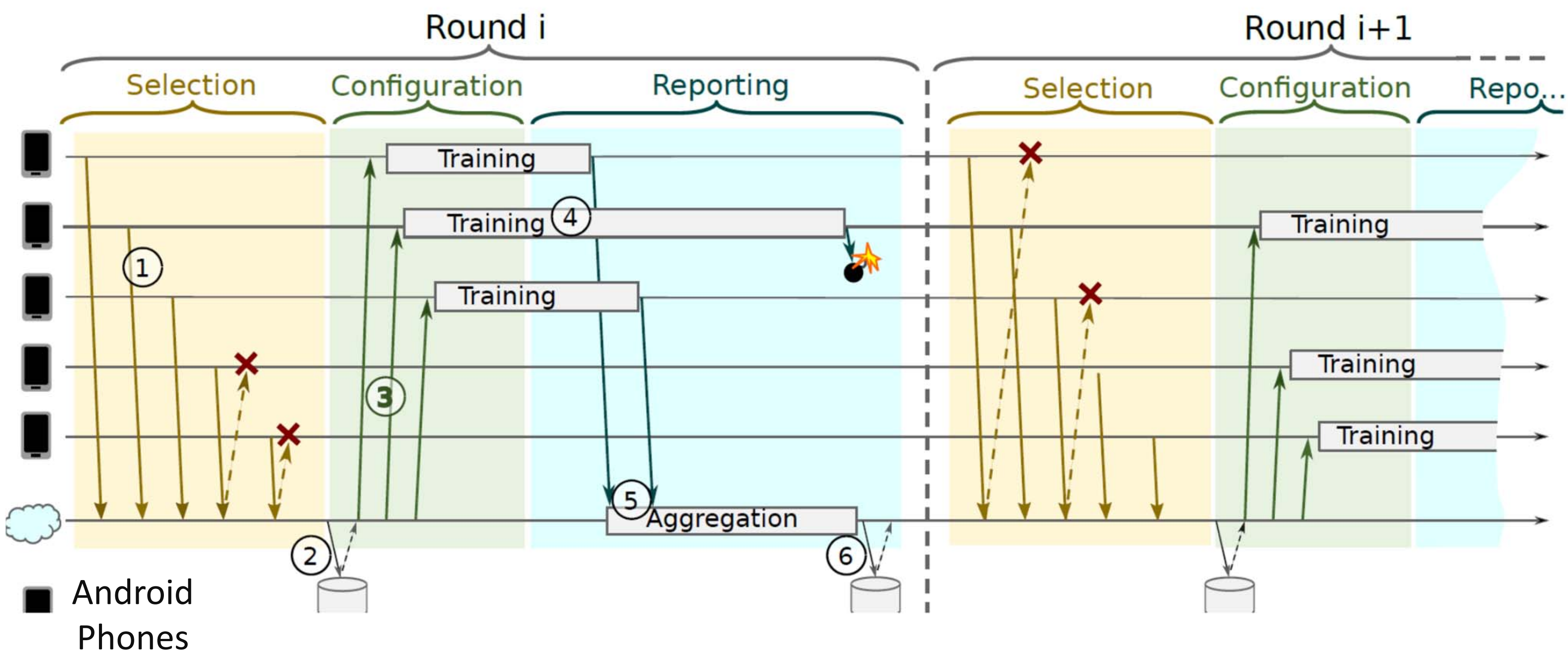
3. $\theta_{t+1} = \theta_t +$ data-weighted average of client updates

}

[Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y Arcas: Communication-Efficient Learning of Deep Networks from Decentralized Data. **AISTATS 2017**]

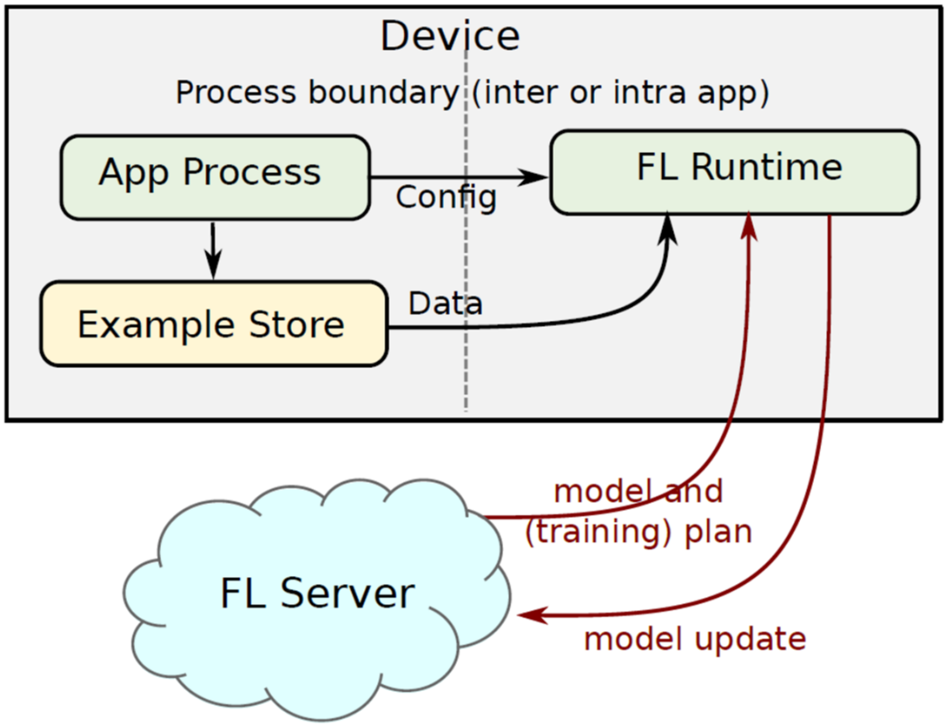
Federated Learning Protocol

- **Recommended Reading**
  - [Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, Jason Roselander: Towards Federated Learning at Scale: System Design. **SysML 2019**]



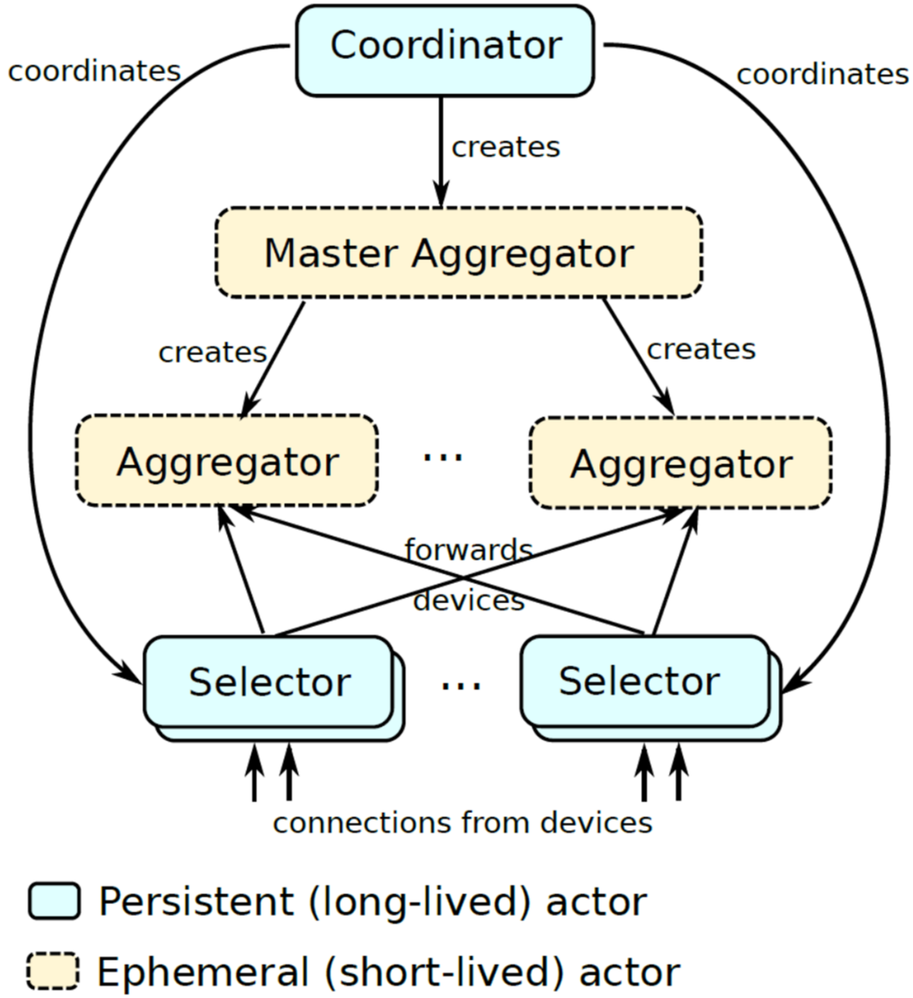
Federated Learning at the Device

- **Data Collection**
  - Maintain repository of locally collected data
  - Apps make data available via dedicated API
- **Configuration**
  - **Avoid negative impact** on data usage or battery life
  - Training and evaluation tasks
- **Multi-Tenancy**
  - Coordination between **multiple learning tasks** (apps and services)



Federated Learning at the Server

- **Actor Programming Model**
  - Comm. via message passing
  - Actors sequentially process stream of events/messages
  - ➔ **Scaling w/ # actors**
- **Coordinators**
  - Driver of overall learning algorithm
  - **Orchestration of aggregators** and selectors (conn handlers)
- **Robustness**
  - Pipelined selection and aggregation rounds
  - Fault Tolerance at aggregator/ master aggregator levels



Excursus: Data Ownership

- **Data Ownership Problem**
  - Vendor sells machine to middleman who uses it to test equipment of customer
    ➔ Who owns the data? Vendor, Middleman, or Customer? Why?
  - **Usually negotiated in bilateral contracts!**

- **A Thought on a Spectrum of Rights and Responsibilities**
  - **Federated ML creates new spectrum for data ownership** that might create new markets and business models
  - **#1** Data stays private with the customer
  - **#2** Gradients of individual machines shared with the vendor
  - **#3** Aggregated gradients shared with the vendor
  - **#4** Data completely shared with the vendor

Data Privacy

↑ ↓

Ability to train models

Summary and Conclusions

- **Data-Parallel Parameter Servers**
  - Data partitioning across workers, independent computation
  - Synchronous or asynchronous model updates

- **Model-parallel Parameter Servers**
  - Network/model partitioning across workers
  - Pipelined execution and independent network partitions

- **Federated Machine Learning**
  - Extended parameter server architecture w/ specialized techniques
  - High potential for use cases where data sharing not possible/practical

- **Next Lecture**
  - **07 Hybrid Execution and HW Accelerators** [May 10]