SCIENCE
PASSION
TECHNOLOGY

Architecture of ML Systems*
06 Execution and Parallelization

Matthias Boehm

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

Last update: Aug 25, 2022

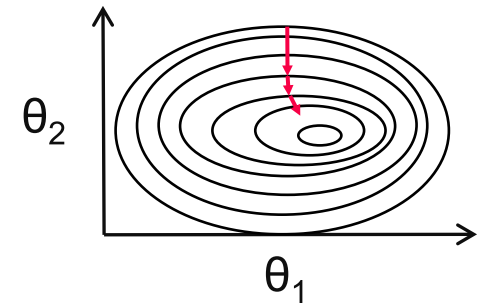
Agenda

- **Motivation and Terminology**
- **Data-Parallel Execution**
- **Task-Parallel Execution**
- **Parameter Servers**
- **Federated Machine Learning**

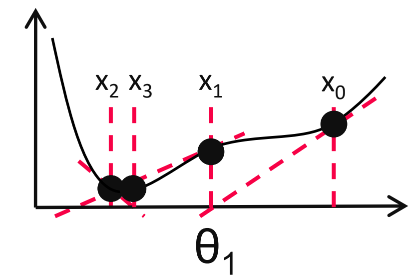
Motivation and Terminology

Terminology Optimization Methods

- **Problem: Given a continuous, differentiable function $f(D, \theta)$, find optimal parameters $\theta^* = \text{argmin}\,(f(D, \theta))$**

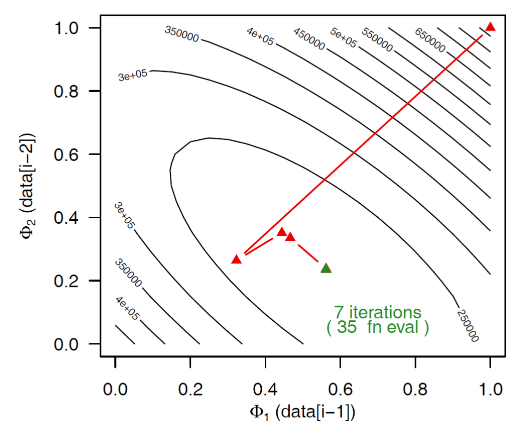


- **#1 Gradient Methods (1st order)**
  - Pick a starting point, compute gradient, descent in opposite direction of gradient $-\gamma \nabla f(D, \theta)$



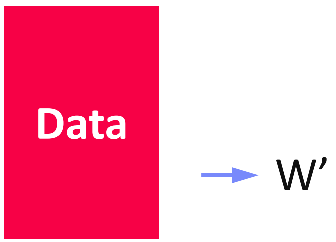
- **#2 Newton's Method (2nd order)**
  - Pick a starting point, compute gradient, descend to where derivative = 0 (via 2nd derivative)
  - Jacobian/Hessian matrices for multi-dimensional

- **#3 Quasi-Newton Methods**
  - Incremental approximation of Hessian
  - Algorithms: BFGS, L-BFGS, Conjugate Gradient (CG)
  - **Example:** L-BFGS-B, AR(2), MSE, N=100 EnBW energy-demand time series

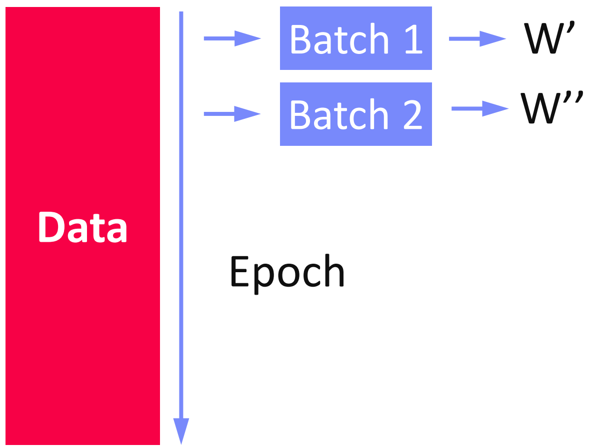


Terminology Batch/Mini-batch

- **Batch ML Algorithms**
  - Iterative ML algorithms, where each iteration uses the **entire dataset** to compute gradients ΔW
  - For (pseudo-)**second-order methods**, many features
  - **Dedicated optimizers** for traditional 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** or w/ **sampling**)
  - For large and **highly redundant training sets**
  - **Applies to almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)
  - **Stochastic Gradient Descent** (SGD)



Terminology Parallelism

- **Flynn's Classification**
  - SISD, SIMD
  - (MISD), MIMD

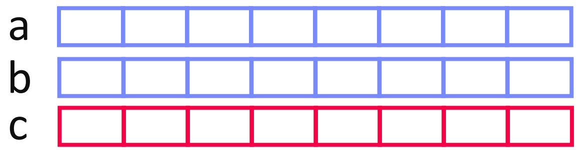
[Michael J. Flynn, Kevin W. Rudd: Parallel Architectures. **ACM Comput. Surv. 28(1) 1996**]

	Single Data	Multiple Data
Single Instruction	**SISD** (uni-core)	**SIMD** (vector)
Multiple Instruction	**MISD** (pipelining)	**MIMD** (multi-core)

- **Example: SIMD Processing**
  - Streaming SIMD Extensions (SSE)
  - Process the same operation on multiple elements at a time (**packed** vs scalar SSE instructions)
  - **Data parallelism** (aka: instruction-level parallelism)
  - Example: `VFMADD132PD`

2009 Nehalem: **128b** (2xFP64)
2012 Sandy Bridge: **256b** (4xFP64)
2017 Skylake: **512b** (8xFP64)

```
c = _mm512_fmadd_pd(a, b);
```



Terminology Parallelism, cont.

- **Distributed, Data-Parallel Computation**

```
Y = X.map(x -> foo(x))
```

  - Parallel computation of function foo() ➔ **single instruction**
  - Collection X of data items (key-value pairs) ➔ **multiple data**
  - Data parallelism similar to **SIMD** but more coarse-grained notion of “instruction” and “data” ➔ **SPMD** (single program, multiple data)

[Frederica Darema: The SPMD Model : Past, Present and Future. **PVM/MPI 2001**]

- **Additional Terminology**
  - **BSP:** Bulk Synchronous Parallel (global barriers)
  - **ASP:** Asynchronous Parallel (no barriers, often with accuracy impact)
  - **SSP:** Stale-synchronous parallel (staleness constraint on fastest-slowest)
  - Other: Fork&Join, Hogwild!, event-based, decentralized

- **Beware: data parallelism** used in very different contexts (e.g., Param Server)

Categories of Execution Strategies

Batch **SIMD/SPMD**	Batch/Mini-batch, Independent Tasks **MIMD**	Mini-batch
06_a Data-Parallel Execution	**06_b Task-Parallel Execution**	**06_c Parameter Servers (data**, model)

07_a Hybrid Execution and HW Accelerators

07_b Caching, Partitioning, Indexing, and Compression

Data-Parallel Execution

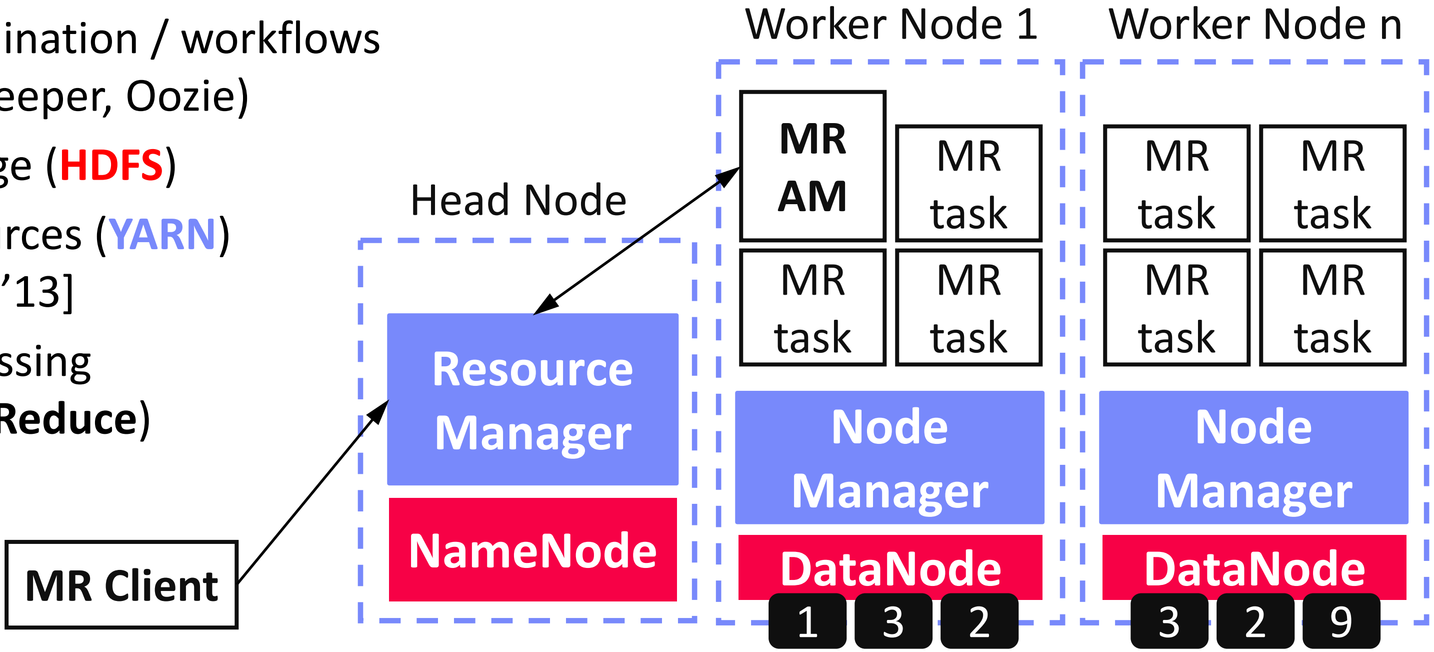
Batch ML Algorithms

Hadoop History and Architecture

- **Recap: Brief History**
  - Google's GFS [SOSP'03] + MapReduce → Apache Hadoop (2006)
  - Apache Hive (SQL), Pig (ETL), Mahout (ML), Giraph (Graph)

[Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. **OSDI 2004**]

- **Hadoop Architecture / Eco System**
  - Management (Ambari)
  - Coordination / workflows (Zookeeper, Oozie)
  - Storage (**HDFS**)
  - Resources (**YARN**) [SoCC'13]
  - Processing (**MapReduce**)



MapReduce – Programming Model

- **Overview Programming Model**
  - Inspired by functional programming languages
  - **Implicit parallelism** (abstracts distributed storage and processing)
  - Map function: key/value pair → set of intermediate key/value pairs
  - Reduce function: merge all intermediate values by key

- **Example** `SELECT Dep, count(*) FROM csv_files GROUP BY Dep`

Name	Dep
X	CS
Y	CS
A	EE
Z	CS

Collection of key/value pairs

```
map(Long pos, String line) {
  parts ← line.split(“,”)
  emit(parts[1], 1)
}
```

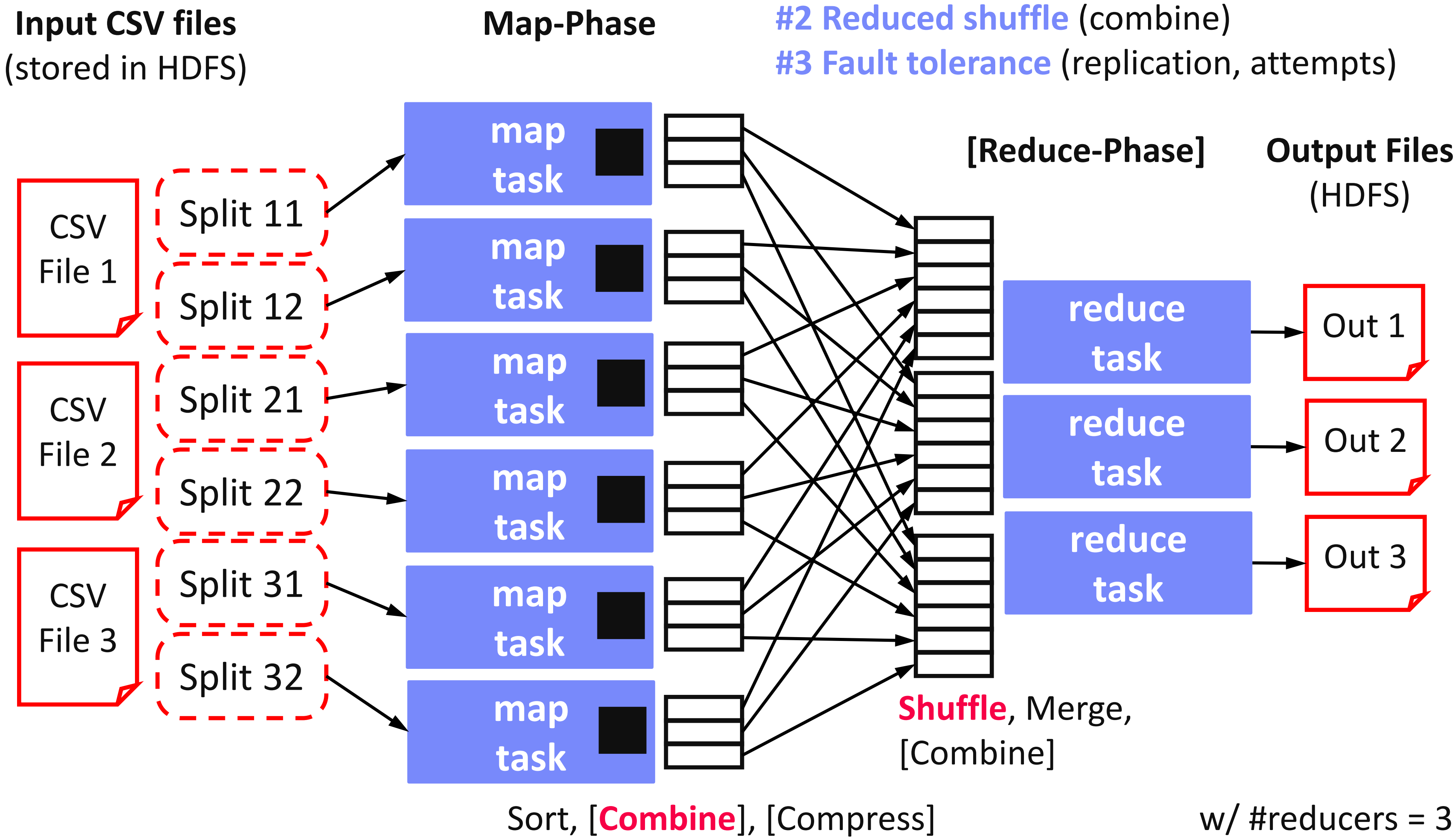
CS	1
CS	1
EE	1
CS	1

```
reduce(String dep,
       Iterator<Long> iter) {
  total ← iter.sum();
  emit(dep, total)
}
```

CS	3
EE	1

MapReduce – Execution Model

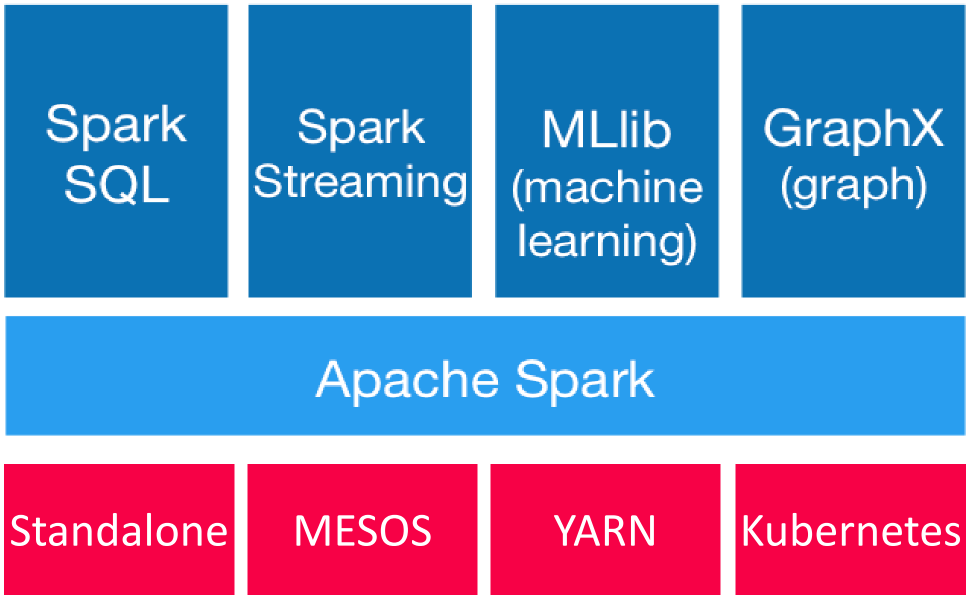
#1 Data Locality (delay sched., write affinity)
#2 Reduced shuffle (combine)
#3 Fault tolerance (replication, attempts)



Spark History and Architecture

- **High-Level Architecture**
  - **Different language bindings**: Scala, Java, Python, R
  - **Different libraries**: SQL, ML, Stream, Graph
  - Spark core (incl RDDs)
  - **Different cluster managers**: Standalone, Mesos, **Yarn**, **Kubernetes**
  - Different file systems/ formats, and data sources: **HDFS**, **S3**, SWIFT, **DBs**, **NoSQL**

[https://spark.apache.org/]



- **Focus on a unified platform for data-parallel computation** (**Apache Flink** w/ similar goals)

Spark Resilient Distributed Datasets (RDDs)

- **RDD Abstraction**

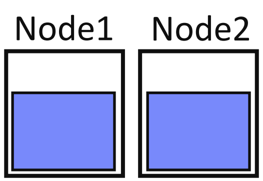
```
JavaPairRDD<MatrixIndexes,MatrixBlock>
```

  - **Immutable**, partitioned **collections of key-value pairs**
  - **Coarse-grained** deterministic operations (transformations/actions)
  - Fault tolerance via lineage-based re-computation

- **Operations**
  - Transformations: define new RDDs
  - Actions: return result to driver

Type	Examples
Transformation (**lazy**)	**map**, hadoopFile, textFile, flatMap, filter, sample, join, groupByKey, cogroup, reduceByKey, cross, sortByKey, mapValues
Action	**reduce**, save, collect, count, lookupKey

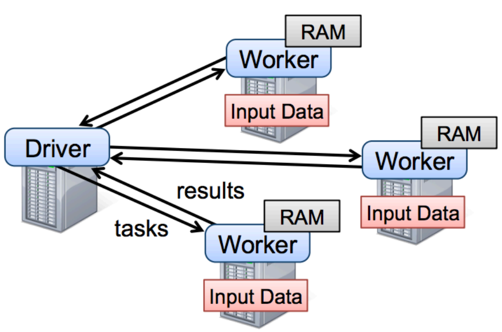
- **Distributed Caching**
  - Use fraction of worker **memory for caching**
  - Eviction at granularity of individual partitions
  - **Different storage levels** (e.g., mem/disk x serialization x compression)



Spark Resilient Distributed Datasets (RDDs), cont.

- **Lifecycle of an RDD**
  - Note: can't broadcast an RDD directly

X.filter(foo())
X.mapValues(foo())
X.reduceByKey(foo())
X.cache()/X.persist(…)
sc.parallelize(lst)
Local Data
(value, collection)
Distributed Collection
lst = X.collect()
v = X.reduce(foo())
sc.hadoopFile(f)
sc.textFile(f)
X.saveAsObjectFile(f)
X.saveAsTextFile(f)
File on DFS



Spark Partitions and Implicit/Explicit Partitioning

- **Spark Partitions**
  - Logical key-value collections are split into **physical partitions**
  - Partitions are granularity of **tasks, I/O, shuffling, evictions**

~128MB

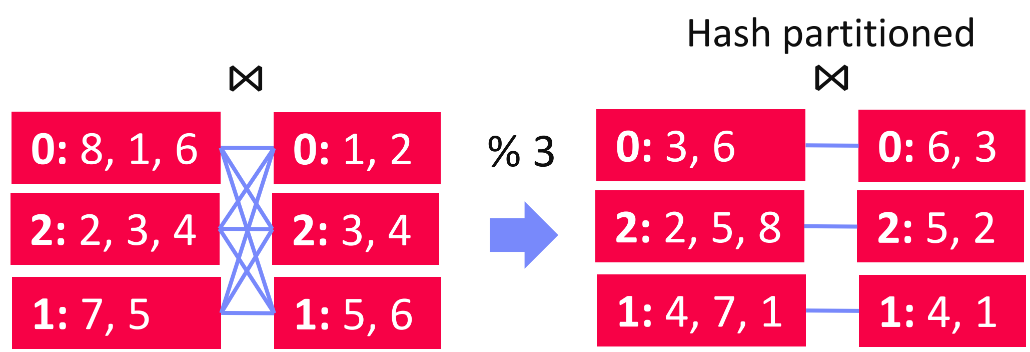
- **Partitioning via Partitioners**
  - Implicitly on every data shuffling
  - Explicitly via `R.repartition(n)`

Example Hash Partitioning:

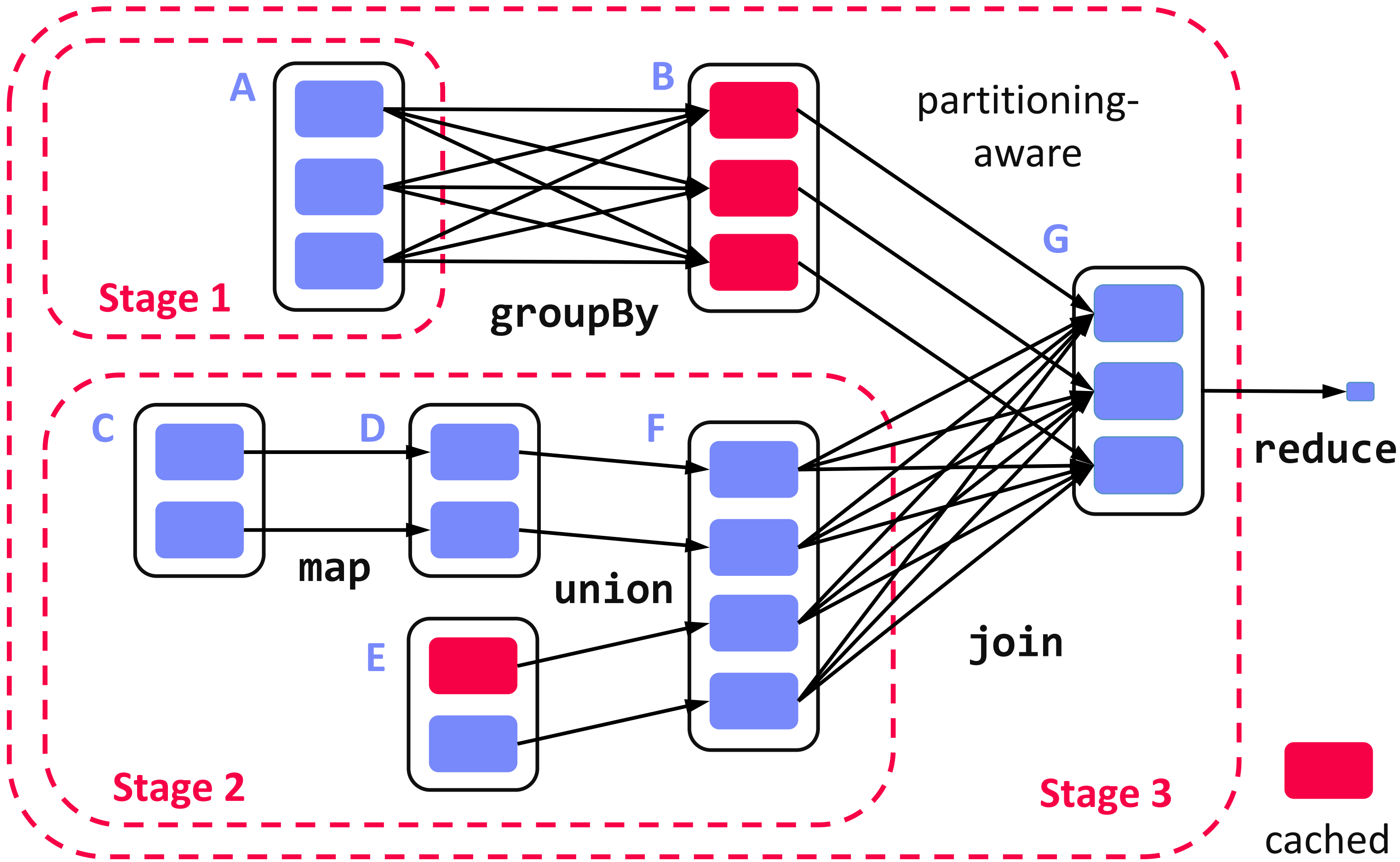
For all (k,v) of R:

pid = hash(k) % n

- **Partitioning-Preserving**
  - All operations that are guaranteed to keep keys unchanged (e.g. `mapValues()`, `mapPartitions()` w/ preservesPart flag)
- **Partitioning-Exploiting**
  - Join: R3 = R1.join(R2)
  - Lookups:
    `v = C.lookup(k)`



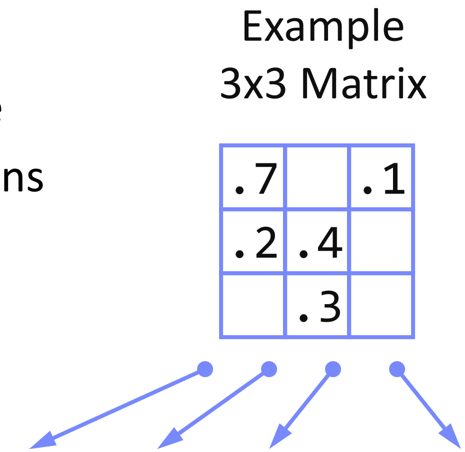
Spark Lazy Evaluation, Caching, and Lineage

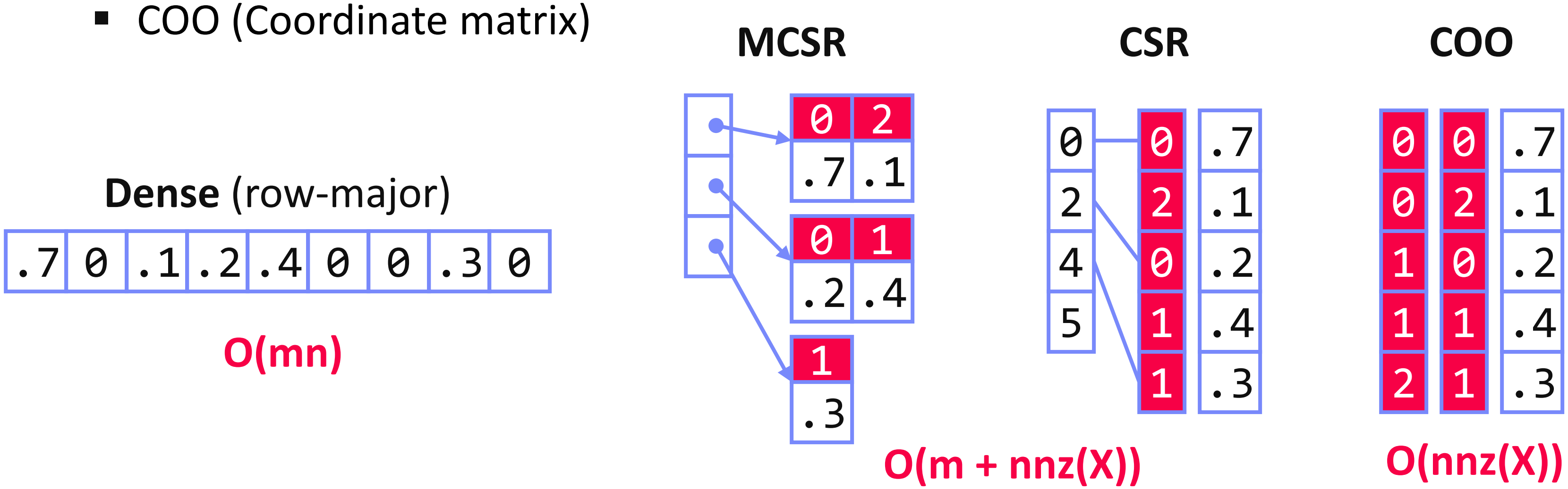


[Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, Ion Stoica: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. **NSDI 2012**]

Background: Matrix Formats

- **Matrix Block** (m x n)
  - A.k.a. tiles/chunks, most operations defined here
  - Local matrix: single block, different representations
- **Common Block Representations**
  - Dense (linearized arrays)
  - MCSR (modified CSR)
  - CSR (compressed sparse rows), CSC
  - COO (Coordinate matrix)





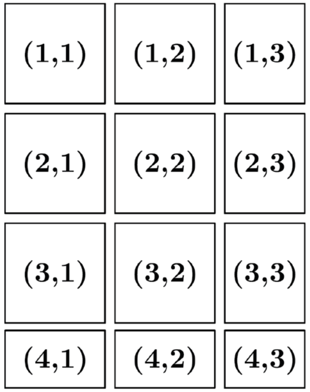
Distributed Matrix Representations

- **Collection of "Matrix Blocks" (and keys)**
  - Bag semantics (duplicates, unordered)
  - Logical (Fixed-Size) Blocking

    + join processing / independence

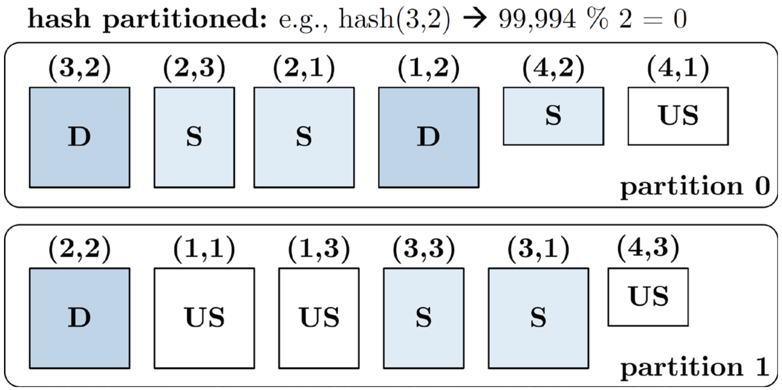
    - (sparsity skew)
  - E.g., SystemML on Spark: `JavaPairRDD<MatrixIndexes,MatrixBlock>`
  - Blocks encoded independently (dense/sparse)

Logical Blocking 3,400x2,700 Matrix (w/ B_c=1,000)



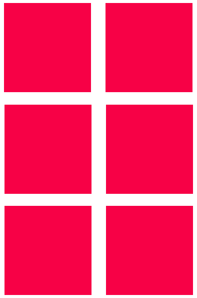
- **Partitioning**
  - Logical Partitioning (e.g., row-/column-wise)
  - Physical Partitioning (e.g., hash / grid)

Physical Blocking and Partitioning

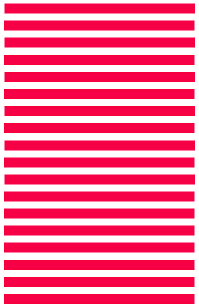


Distributed Matrix Representations, cont.

- **#1 Block-partitioned Matrices**
  - Fixed-size, square or rectangular blocks
  - Pros: Input/output alignment, block-local transpose, amortize block overheads, bounded mem, cache-conscious
  - Cons: Converting row-wise inputs (e.g., text) requires shuffle
  - **Examples: RIOT, PEGASUS, SystemML, SciDB, Cumulon, Distributed R, DMac, Spark Mllib, Gilbert, MatFast, and SimSQL**



- **#2 Row/Column-partitioned Matrices**
  - Collection of row indexes and rows (or columns respectively)
  - Pros: Seamless data conversion and access to entire rows
  - Cons: Storage overhead in Java, and cache unfriendly operations
  - Examples: **Spark MLlib**, **Mahout Samsara**, **Emma**, **SimSQL**

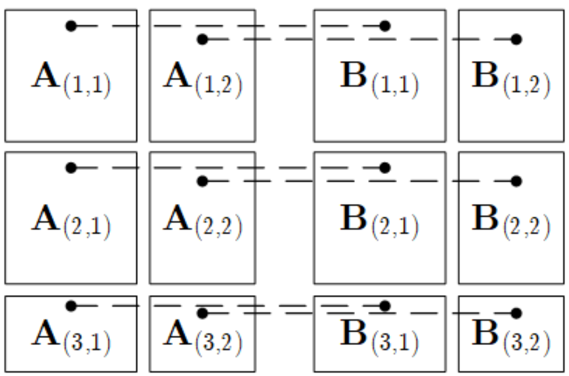


- **#3 Algorithm-specific Partitioning**
  - Operation and algorithm-centric data representations
  - Examples: matrix **inverse**, matrix **factorization**

Distributed Matrix Operations

Elementwise Multiplication
(Hadamard Product)

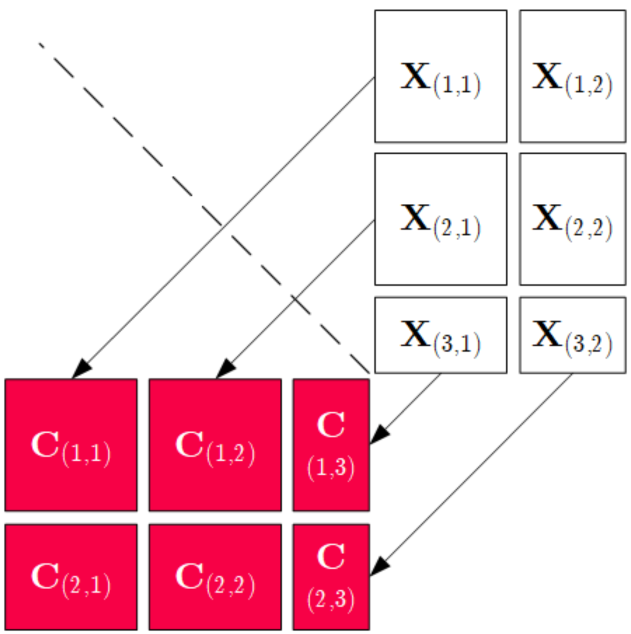
`C = A * B`



Note: also with
row/column vector rhs

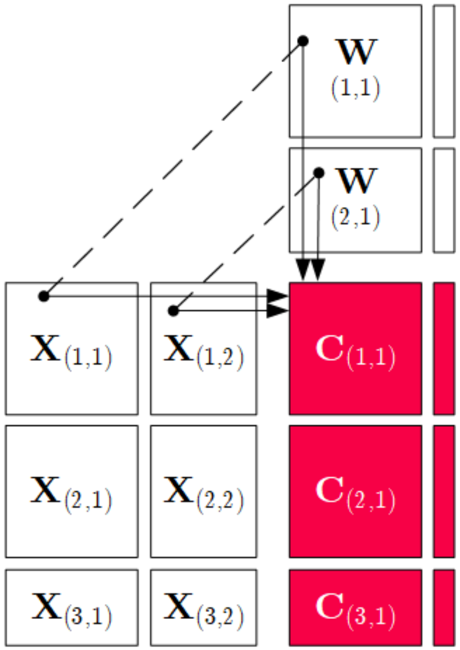
Transposition

`C = t(X)`



Matrix Multiplication

`C = X %*% W`

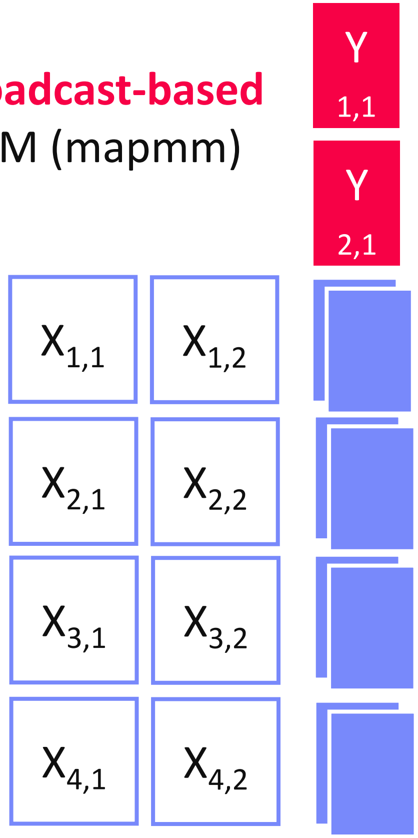


Note: 1:N join

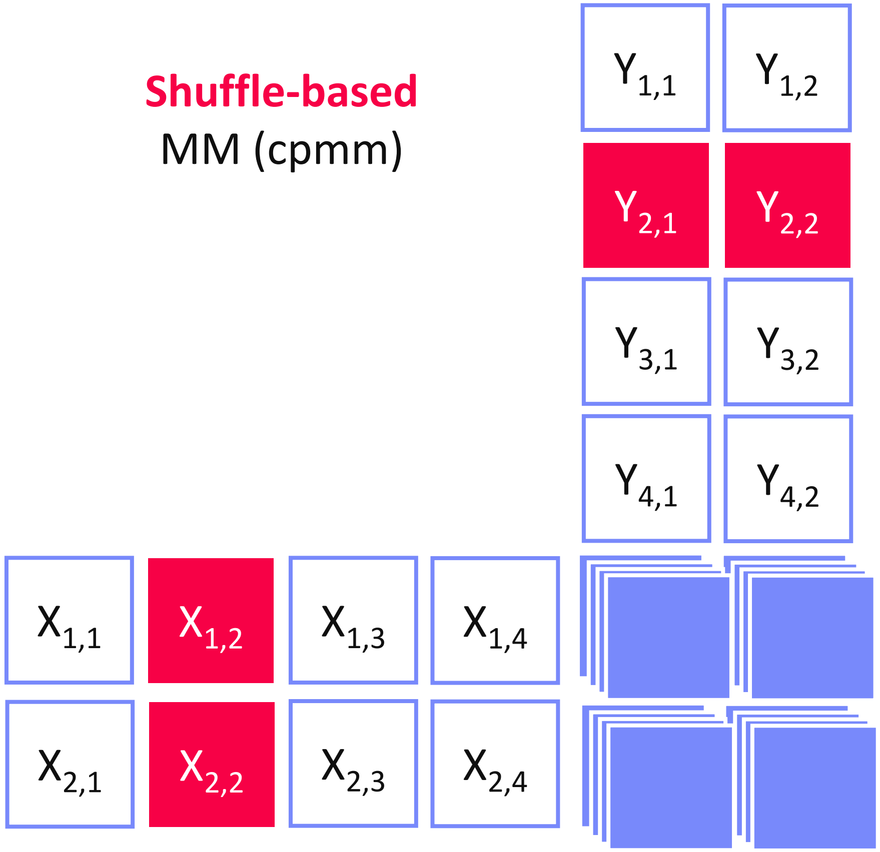
Physical MM Operator Selection, cont.

- **Examples Distributed MM Operators**

Broadcast-based MM (mapmm)



Shuffle-based MM (cpmm)



Partitioning-Preserving Operations

- **Shuffle is major bottleneck for ML on Spark**
- **Preserve Partitioning**
  - Op is partitioning-preserving if keys unchanged (guaranteed)
  - Implicit: Use restrictive APIs (`mapValues()` vs `mapToPair()`)
  - Explicit: Partition computation w/ declaration of partitioning-preserving
- **Exploit Partitioning**
  - Implicit: Operations based on `join`, `cogroup`, etc
  - Explicit: Custom operators (e.g., zipmm)
- **Example: Multiclass SVM**
  - Vectors fit neither into driver nor broadcast
  - ncol(X) ≤ B_c

repart, chkpt X MEM_DISK

```
parfor(iter_class in 1:num_classes) {
   Y_local = 2 * (Y == iter_class) - 1
   g_old = t(X) %*% Y_local
   ...
   while( continue ) {
      Xd = X %*% s
      ... inner while loop (compute step_sz)
      Xw = Xw + step_sz * Xd;
      out = 1 - Y_local * Xw;
      out = (out > 0) * out;
      g_new = t(X) %*% (out * Y_local) ...
```

chkpt y_local MEM_DISK

chkpt Xd, Xw MEM_DISK

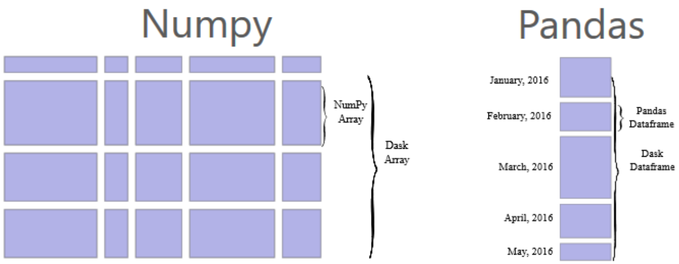
zipmm

Dask

DASK

[Matthew Rocklin: Dask: Parallel Computation with Blocked algorithms and Task Scheduling, **Python in Science 2015**]
[Dask Development Team: Dask: Library for dynamic task scheduling, 2016, https://dask.org]

- **Overview Dask**
  - Multi-threaded and distributed operations for arrays, bags, and dataframes
  - `dask.array:`
    list of numpy n-dim arrays
  - `dask.dataframe:`
    list of pandas data frames
  - `dask.bag:` unordered list of tuples (second order functions)
  - Local and distributed schedulers:
    threads, processes, YARN, Kubernetes, containers, HPC, and cloud, GPUs



- **Execution**
  - Lazy evaluation
  - Limitation: requires **static size inference**
  - Triggered via `compute()`

```
import dask.array as da

x = da.random.random(
    (10000,10000), chunks=(1000,1000))
y = x + x.T
y.persist() # cache in memory
z = y[::2, 5000:].mean(axis=1) # colMeans
ret = z.compute() # returns NumPy array
```

Task-Parallel Execution

Parallel Computation of Independent Tasks,
Emulation of Data-Parallel Operations/Programs

Overview Task-Parallelism

- **Historic Perspective**
  - Since 1980s: various parallel Fortran extensions, especially in HPC
  - DOALL parallel loops (independent iterations)
  - OpenMP (since 1997, Open Multi-Processing)

```
#pragma omp parallel for reduction(+: nnz)
for (int i = 0; i < N; i++) {
  int threadID = omp_get_thread_num();
  R[i] = foo(A[i]);
  nnz += (R[i]!=0) ? 1 : 0;
}
```

- **Motivation: Independent Tasks in ML Workloads**
  - Use cases: Ensemble learning, cross validation, hyper-parameter tuning, complex models with disjoint/overlapping/all data per task
  - **Challenge #1:** Adaptation to data and cluster characteristics
  - **Challenge #2:** Combination with data-parallelism

Parallel For Loops (ParFor)

Apache SystemML™

[M. Boehm et al.: Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. **PVLDB 2014**]

- **Hybrid Parallelization Strategies**
  - Combination of **data- and task-parallel** ops
  - Combination of **local and distributed** computation
- **Key Aspects**
  - Dependency Analysis
  - Task partitioning
  - Data partitioning, scan sharing, various rewrites
  - Execution strategies
  - Result agg strategies
  - ParFor optimizer

```
reg = 10^(seq(-1,-10))
B_all = matrix(0, nrow(reg), n)

parfor( i in 1:nrow(reg) ) {
   B = lm(X, y, reg[i,1]);
   B_all[i,] = t(B);
}
```

Local ParFor (multi-threaded), w/ local ops

Remote ParFor (distributed Spark job)

Local ParFor, w/ concurrent distributed ops

Additional ParFor Examples

- **Pairwise Pearson Correlation**
  - In practice: uni/bivariate stats
  - Pearson's R, Anova F, Chi-squared, Degree of freedom, P-value, Cramers V, Spearman, etc)

```
D = read("./input/D");
R = matrix(0, ncol(D), ncol(D));
parfor(i in 1:(ncol(D)-1)) {
   X = D[ ,i];
   sX = sd(X);
   parfor(j in (i+1):ncol(D)) {
      Y = D[ ,j];
      sY = sd(Y);
      R[i,j] = cov(X,Y)/(sX*sY);
} }
write(R, "./output/R");
```

- **Batch-wise CNN Scoring**
  - Emulate data-parallelism for complex functions

```
prob = matrix(0, Ni, Nc)
parfor( i in 1:ceil(Ni/B) ) {
   Xb = X[((i-1)*B+1):min(i*B,Ni),];
   prob[((i-1)*B+1):min(i*B,Ni),] =
      ... # CNN scoring
}
```

➔ **Conceptual Design:**
Coordinator/worker (task: group of parfor iterations)

ParFor Execution Strategies

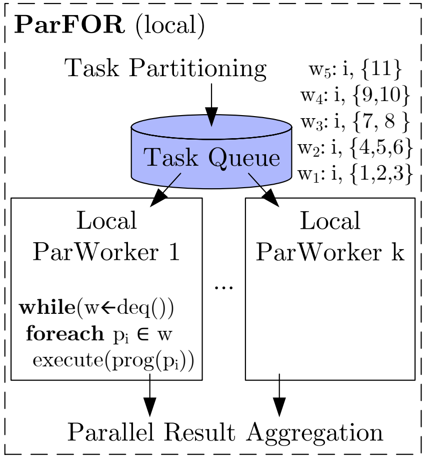
- **#1 Task Partitioning**
  - Fixed-size schemes: naive (1) , static (n/k), fixed (m)
  - Self-scheduling: e.g., guided self scheduling, factoring

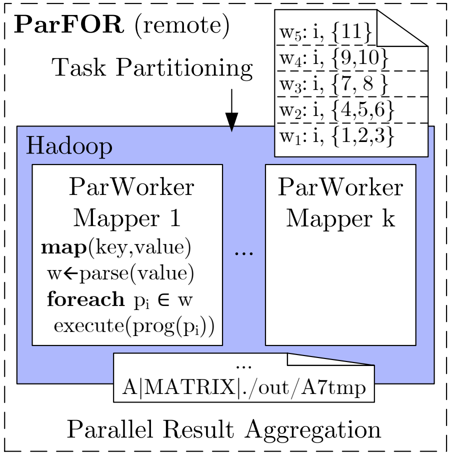
Factoring (n=101, k=4)

$$R_0 = N, \quad R_{i+1} = R_i - k \cdot l_i, \qquad l_i = \left\lceil \frac{R_i}{x_i \cdot k} \right\rceil = \left\lceil \left(\frac{1}{x_i}\right)^{i+1} \frac{N}{k} \right\rceil$$

(13,13,13,13, 7,7,7,7, 3,3,3,3, 2,2,2,2, 1)

- **#2 Data Partitioning**
  - Local or remote row/column partitioning (incl locality)
- **#3 Task Execution**
  - Local (multi-core) execution
  - Remote (MR/Spark) execution





- **#4 Result Aggregation**
  - With and without compare (non-empty output variable)
  - Local in-memory / remote MR/Spark result aggregation

Task-Parallelism in R

- **Multi-Threading**
  - doMC as multi-threaded foreach backend
  - Foreach w/ parallel (%dopar%) or sequential (%do%) execution

[https://cran.r-project.org/web/packages/doMC/vignettes/gettingstartedMC.pdf]

```
library(doMC)
registerDoMC(32)
R <- foreach(i=1:(ncol(D)-1),
             .combine=rbind) %dopar% {
   X = D[,i]; sX = sd(X);
   Ri = matrix(0, 1, ncol(D))
   for(j in (i+1):ncol(D)) {
      Y = D[,j]; sY = sd(Y)
      Ri[1,j] = cov(X,Y)/(sX*sY);
   }
   return(Ri);
}
```

- **Distribution**
  - doSNOW as distributed foreach backend
  - MPI/SOCK as comm methods

[https://cran.r-project.org/web/packages/doSNOW/doSNOW.pdf]

```
library(doSNOW)
clust = makeCluster(
   c(“192.168.0.1”, “192.168.0.2”,
   “192.168.0.3”), type=“SOCK”);
registerDoSNOW(clust);
... %dopar% ...
stopCluster(clust);
```

Task-Parallelism in Other Systems

- **MATLAB**
  - Parfor loops for multi-process & distributed loops
  - Use-defined par

```
matlabpool 32
c = pi; z = 0;
r = rand(1,10)
parfor i = 1 : 10
  z = z+1;  # reduction
  b(i) = r(i); # sliced
end
```

[Gaurav Sharma, Jos Martin: MATLAB®: A Language for Parallel Computing. Int. **Journal on Parallel Prog. 2009**]

- **Julia**
  - Dedicated macros: @threads @distributed

```
a = zeros(1000)
@threads for i in 1:1000
  a[i] = rand(r[threadid()])
end
```

[https://docs.julialang.org/en/v1/manual/parallel-computing/]

- **TensorFlow**
  - User-defined parallel iterations, responsible for correct results or acceptable approximate results

[https://www.tensorflow.org/api_docs/python/tf/while_loop]

```
tf.while_loop(cond, body, loop_vars, parallel_iterations=10,
   swap_memory=False, maximum_iterations=None, ...)
```

Task-Parallelism in Other Systems, cont.

- **sk-dist** [https://pypi.org/project/sk-dist/]
  - Distributed training of local scikit-learn models (via **PySpark**)
  - **Grid Search** / **Cross Validation** (hyper-parameter optimization)
  - **Multi-class Training** (one-against the rest)
  - **Tree Ensembles** (many decision trees)

- **Model Hopper Parallelism (MOP)**
  - Given a dataset D, p workers, and several NN configurations S
  - Partition D into worker-local partitions D_p
  - **Schedule tasks for sub-epochs** of $S' \subseteq S$ on p without moving the partitioned data
  - Checkpointing of models between tasks

[Supun Nakandala, Yuhao Zhang, Arun Kumar: Cerebro: Efficient and Reproducible Model Selection on Deep Learning Systems. **DEEM@SIGMOD 2019**]

[Supun Nakandala, Yuhao Zhang, Arun Kumar: Cerebro: A Data System for Optimized Deep Learning Model Selection. **PVLDB 2020**]

- **Reinforcement Learning Frameworks**
- **Future-based Task Graphs (Ray, Pathways,** UPLIFT**)**

Part of **Next Lecture**

Data-Parallel Parameter Servers

Background: Mini-batch DNN Training (LeNet)

[Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner: Gradient-Based Learning Applied to Document Recognition, **Proc of the IEEE 1998**]

```
# 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)
   }
}
```

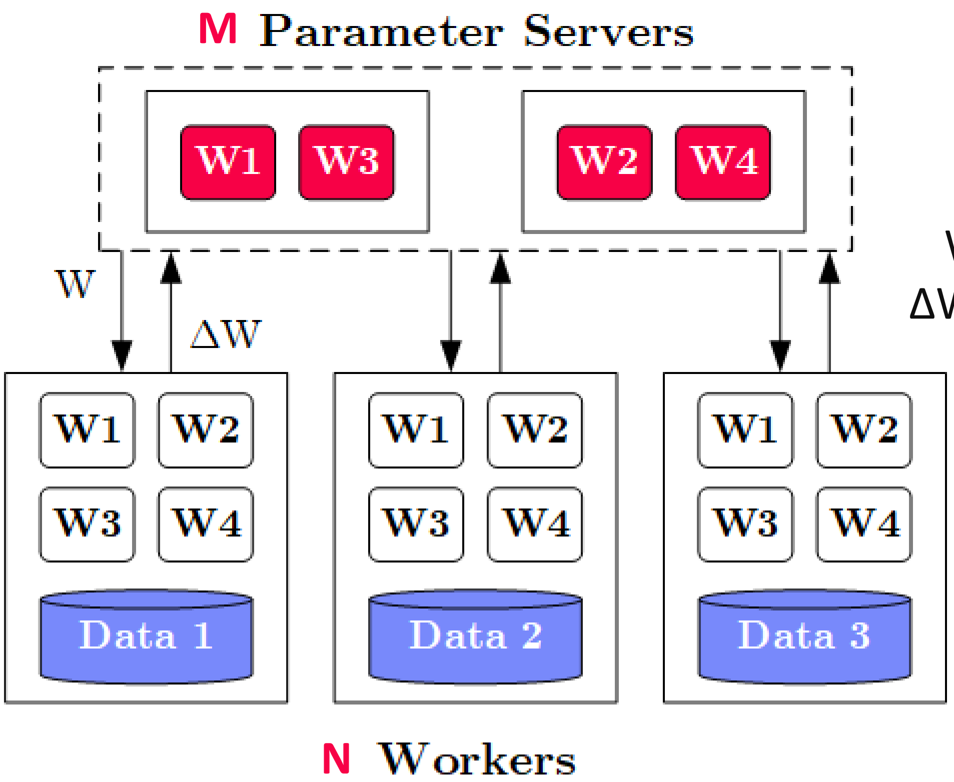
NN Forward Pass

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

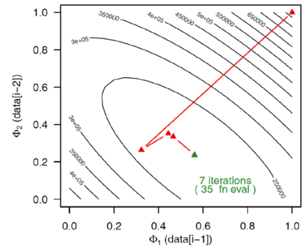
- **Asynchronous Parallel (ASP)**
  - Update model for each gradient
  - No barrier

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**]

Selected Optimizers (updateModel)



- **Stochastic Gradient Descent (SGD)**
  - Vanilla SGD, basis for many other optimizers
  - See **05 Data/Task-Parallel:** $-\gamma\nabla \boldsymbol{f}(\boldsymbol{D}, \boldsymbol{\theta})$

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

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

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

- **SGD w/ Nesterov Momentum**
  - Incorporates 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 w/ 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))
```

Selected Optimizers (updateModel), cont.

- **Adam**

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

  - Individual adaptive learning rates for different parameters

```
t = t + 1
m = beta1*m + (1-beta1)*dX    # update biased 1st moment est
v = beta2*v + (1-beta2)*dX^2 # update biased 2nd raw moment est
mhat = m / (1-beta1^t)        # bias-corrected 1st moment est
vhat = v / (1-beta2^t)        # bias-corrected 2nd raw moment est
X = X - (lr * mhat/(sqrt(vhat)+epsilon)) # param update
```

- **Shampoo**

[Vineet Gupta, Tomer Koren, Yoram Singer: Shampoo: Preconditioned Stochastic Tensor Optimization. **ICML 2018**]

  - Preconditioned gradient method (Newton's method, Quasi-Newton)
  - Retains gradients tensor structure by maintaining a preconditioner per dim
  - $O(m^2n^2) \rightarrow O(m^2 + n^2)$

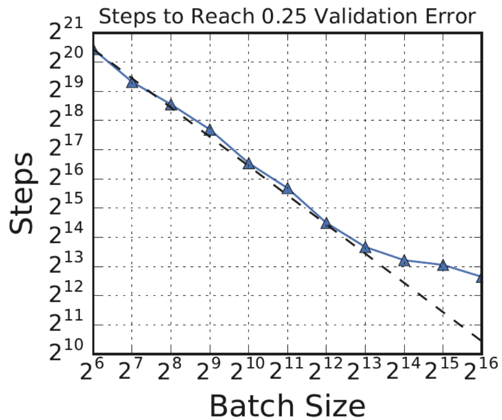
```
L = L + dX %*% t(dX)
R = R + t(dX) %*% dX
X = X - lr * pow(L,1/4)
      %*% dX %*% pow(R,1/4))
```

Batch Size Configuration

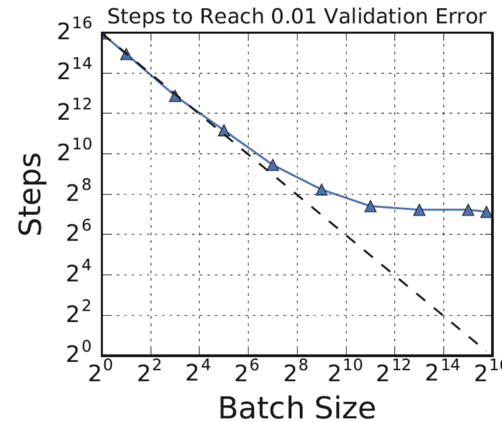
- **What is the right batch size for my data?**
  - Maximum useful batch size is dependent on data redundancy and model complexity

[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 reduce the relative communication overhead

[Priya Goyal et al: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. **CoRR 2017** (kn=8K, 256 GPUs)]

- **Overlapping Computation/Communication**
  - For deep NN w/ many weight/bias matrices, compute and comm. can be overlapped
  - Collective operations: all-Reduce / ring all-reduce / hierarchical all-reduce

`tf.distribute:`
`MirroredStrategy`
`MultiWorkerMirroredStrategy`

- **Sparse and Compressed Communication**
  - Mini-batches of sparse data → sparse dW
  - Lossy (mantissa truncation, quantization), and lossless (delta, bitpacking) for W and dW
  - Gradient sparsification/clipping (send gradients larger than a threshold)

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

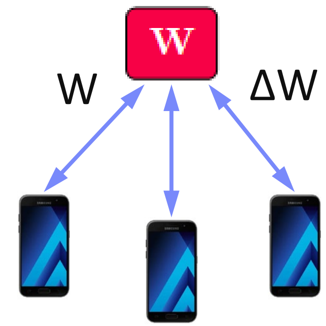
- **In-Network Aggregation** (SwitchML)
  - Aggregate worker updates in prog. switches
  - 32b fix-point, coordinated updates

[Amedeo Sapio et al: Scaling Distributed Machine Learning with In-Network Aggregation, **NSDI 2021**]

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


Google
Federated Learning
Privacy-Preserving Collaborative Machine Learning without Centralized Training Data
Jakub Konečný
konkey@google.com

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

Excursus: Spectrum of Data Sharing

- **Fine-grained Spectrum**
  - Spectrum of technologies with **performance/privacy/utility** tradeoffs
  - Different applications with different requirements
  - **Potential:** New markets for data-driven services in this spectrum

Increasing Privacy

Increasing Utility/Perf

Private Data (no sharing)

Privacy-enhancing Technologies (FHE, MPC, Differential Privacy)

Aggregates (Federated ML, Federated w/ secure comm.)

Anonymized Data (k-anonymity, pseudonyms)

Surrogate Data (char-preserving synthetic data, dataset distillation)

Public Data (full sharing, sharing w/ partners)

Key Property: no reconstruction of private raw data

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'

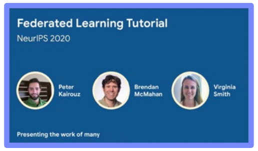
2c. **Return** $\theta' - \theta_t$ (model averaging) **[push]**

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**]

Algorithmic PS Extensions



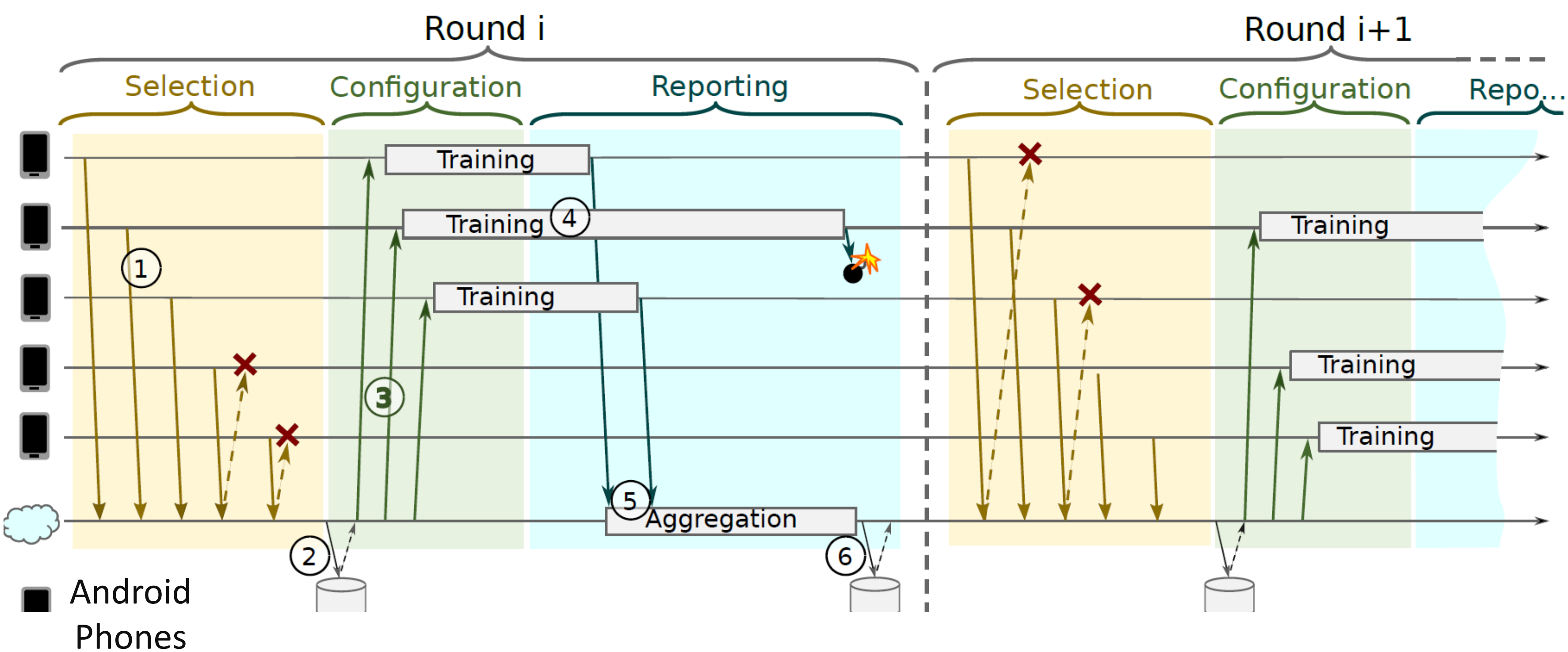
[Peter Kairouz, Brendan McMahan, Virginia Smith: Federated Learning Tutorial. **NeurIPS 2020**, https://slideslive.com/38935813/federated-learningtutorial]

- **#1 Client Sampling** (**FedAvg** w/ model averaging)
- **#2 Decentralized, Fault-tolerant Aggregation**
- **#3 Peer-to-peer Gradient and Model Exchange**
- **#4 Meta-learning for Private Models**
- **#5 Handling Statistical Heterogeneity** (non-IID data)
  - Reducing variance
  - Selecting relevant subsets of data
  - Tolerating partial client work
  - Partitioning clients into congruent groups
  - Adaptive Optimization (**FedOpt**, **FedAvgM**)

[Sashank J. Reddi et al: Adaptive Federated Optimization. **CoRR 2020**]

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.** **MLSys 2019**]

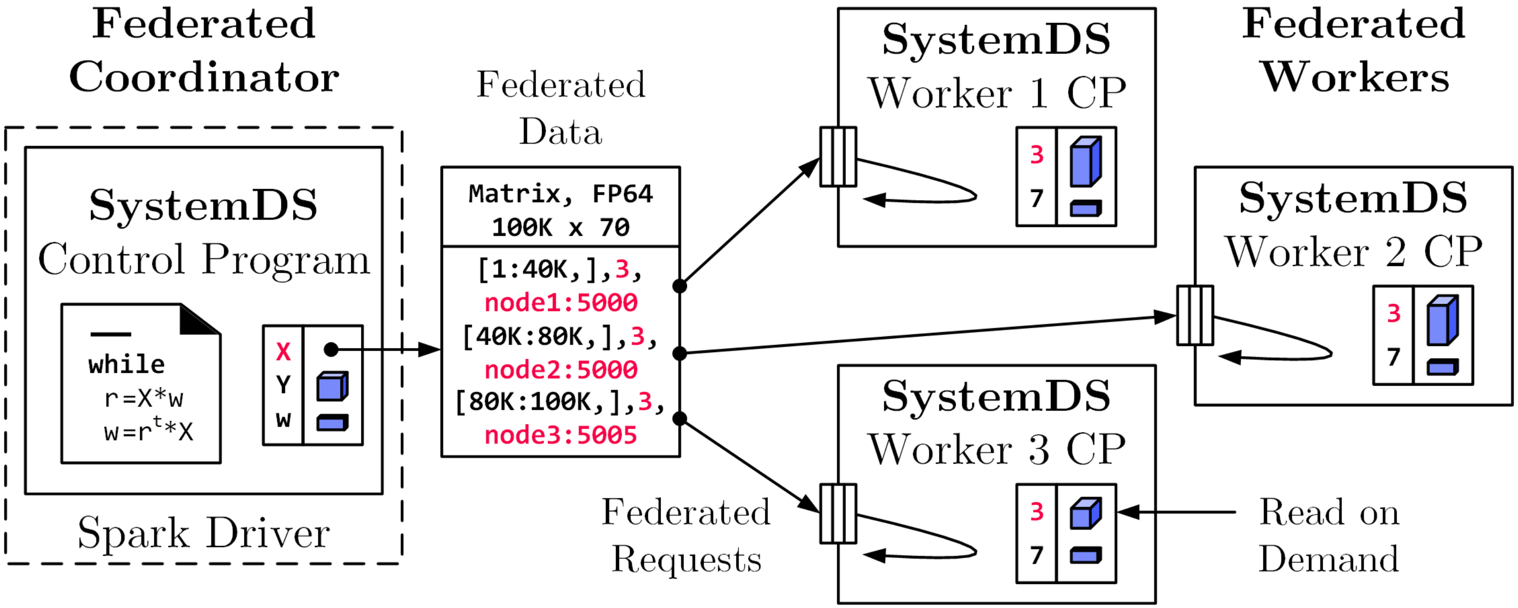


Federated Learning

[SIGMOD 2021b]

- **Federated Backend**
  - **Federated data** (matrices/frames) as meta data objects
  - **Federated linear algebra**, (and **federated parameter server**)

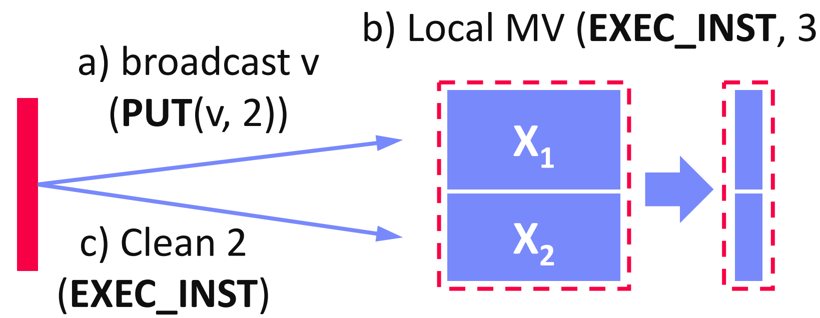
```
X = federated(addresses=list(node1, node2, node3),
  ranges=list(list(0,0),list(40K,70), ..., list(80K,0),list(100K,70)));
```



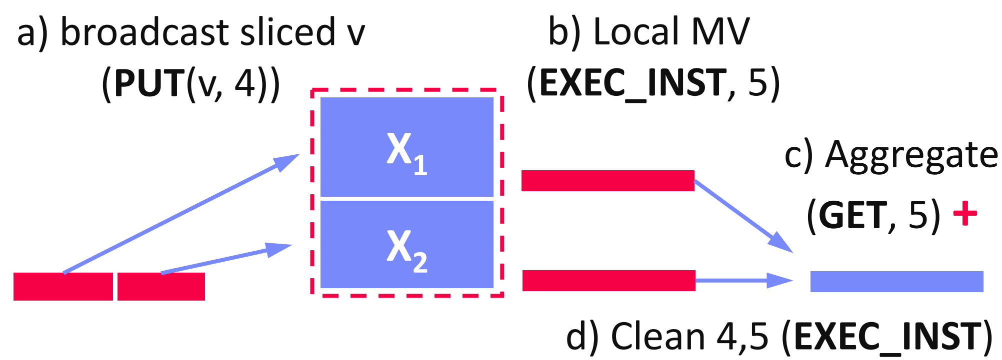
- **Federated Requests:** `READ, PUT, GET, EXEC_INST, EXEC_UDF, CLEAR`

Example Federated Operations

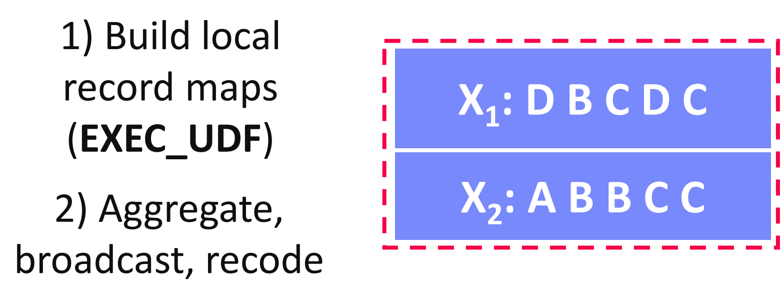
- **Matrix-Vector Multiplication**
  - `o = X %*% v`, local v
  - Row-partitioned, federated X
  - **Row-partitioned, federated o**



- **Vector-Matrix Multiplication**
  - `o = v %*% X`, local v
  - Row-partitioned, federated X, **local o**
  - **New broadcast handling**



- **Data Preparation**
  - `[X,M] = transformencode(F,spec)`
  - Recoding, feature hashing, binning, **one-hot encoding**

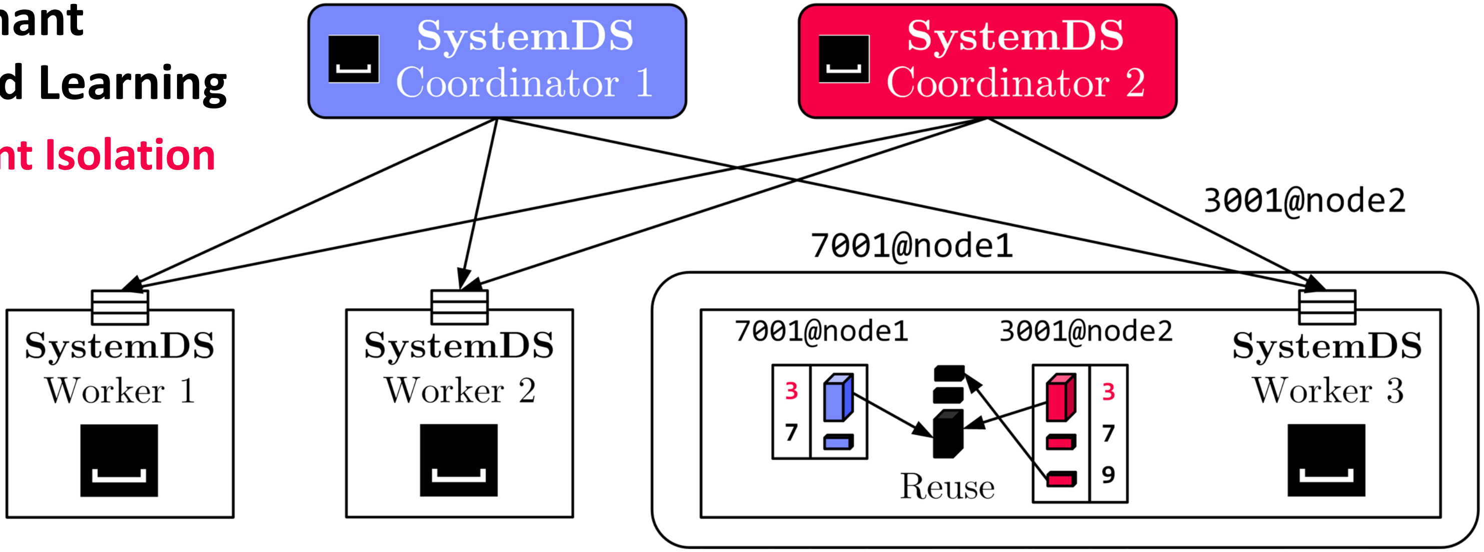


Federated Data Preparation, Learning, and Debugging

- **Federated Feature Transformations**
- **Federated Linear-algebra-based Data Cleaning, Data Preparation, and Model Debugging** (e.g., federated quantiles)
- **Multi-tenant Federated Learning**
  - Tenant Isolation

Lineage-based Reuse

Asynchronous Compression



TensorFlow Federated

[https://www.tensorflow.org/federated/]

- **Overview TFF**
  - **Federated PS algorithms** and **federated second order functions**
  - Primarily for simulating federated training, no OSS federated runtime
- **#1 Federated PS**

```
iterative_process = tff.learning.build_federated_averaging_process(
    model_fn, # function for created federated models
    client_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=0.02),
    server_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=1.0))
```

- **#2 Federated Analytics**
  - r = t(y) %*% X
  - User-level composition of federated algorithms
  - **PET primitives**

```
X = ... # tff.type_at_clients(tf.float32)
by = tff.federated_broadcast(y)
R  = tff.federated_sum(
       tff.federated_map(X, by, foo_mm), foo_s)
     # note: tff.federated_secure_sum
```

Summary and Q&A

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

- **#1 Different strategies (and systems) for different ML workloads**
  ➔ Specialization and abstraction
- **#2 Awareness of underlying execution frameworks**
- **#3 Awareness of effective compilation and runtime techniques**