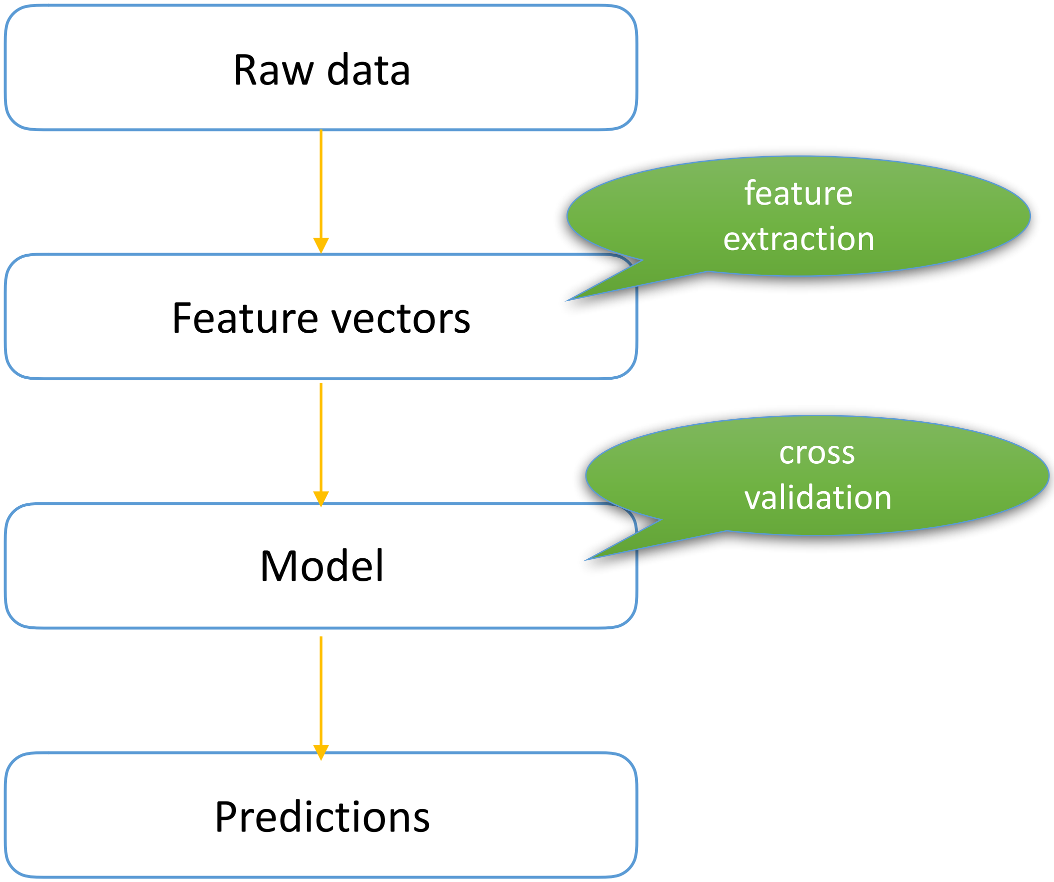
Beyond MLLib:
Scale up Advanced Machine Learning on Spark

Yizhi Liu

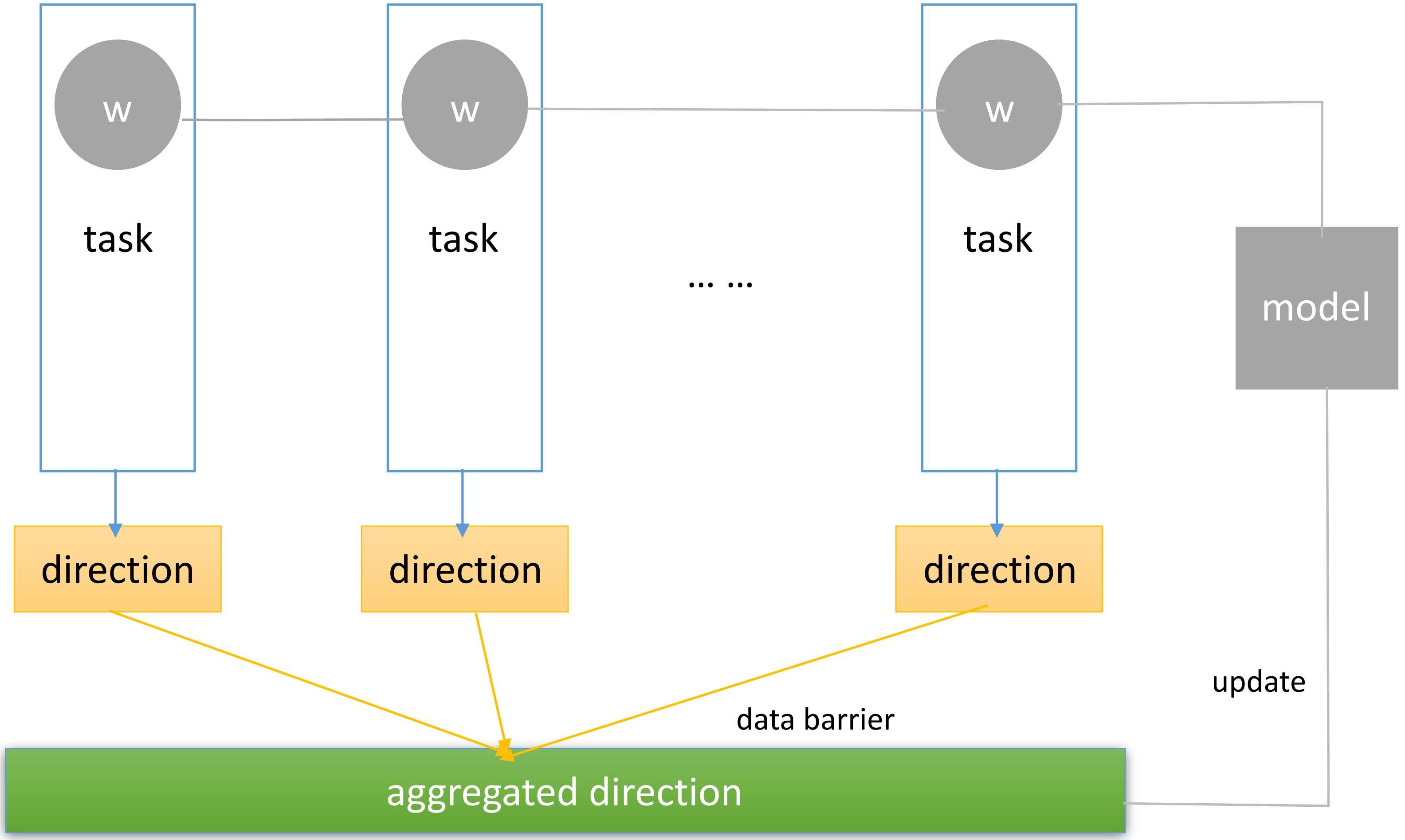
Qihoo 360 / MVAD
DMLC member
MXNet committer

Algorithms in Spark ML

- Classification and Regression
  - Linear models
  - Naive Bayes
  - Decision Trees
  - Ensembles of Trees
  - Isotonic Regression
- Collaborative Filtering
  - ALS
  - … …
- Clustering
  - Latent Dirichlet Allocation
  - … …
- Dimensionality Reduction
  - … …



Computational Model



Advanced ML Algorithms

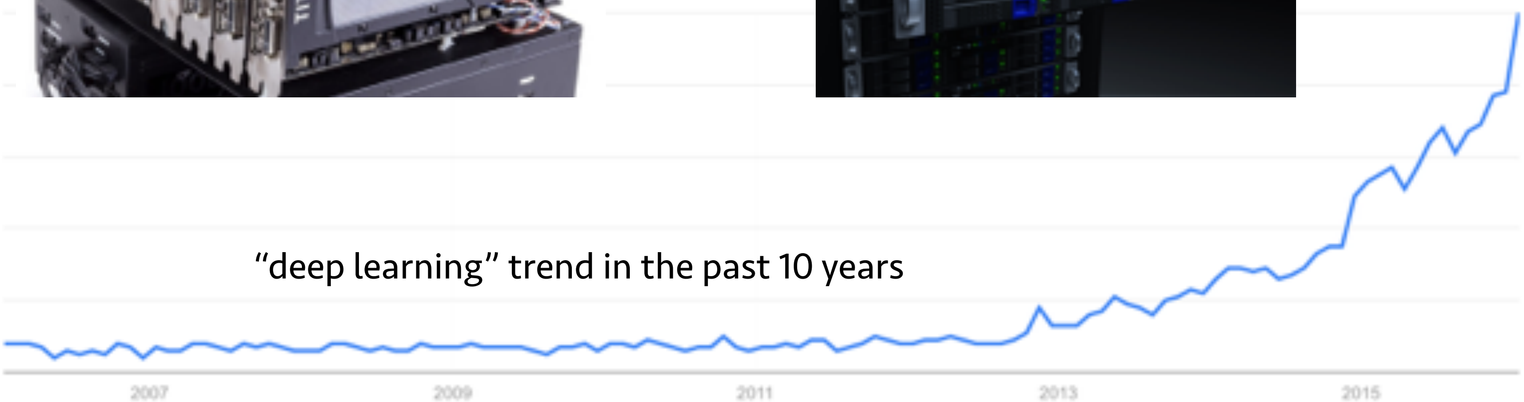
dmlc XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly ***efficient***, ***flexible*** and ***portable***

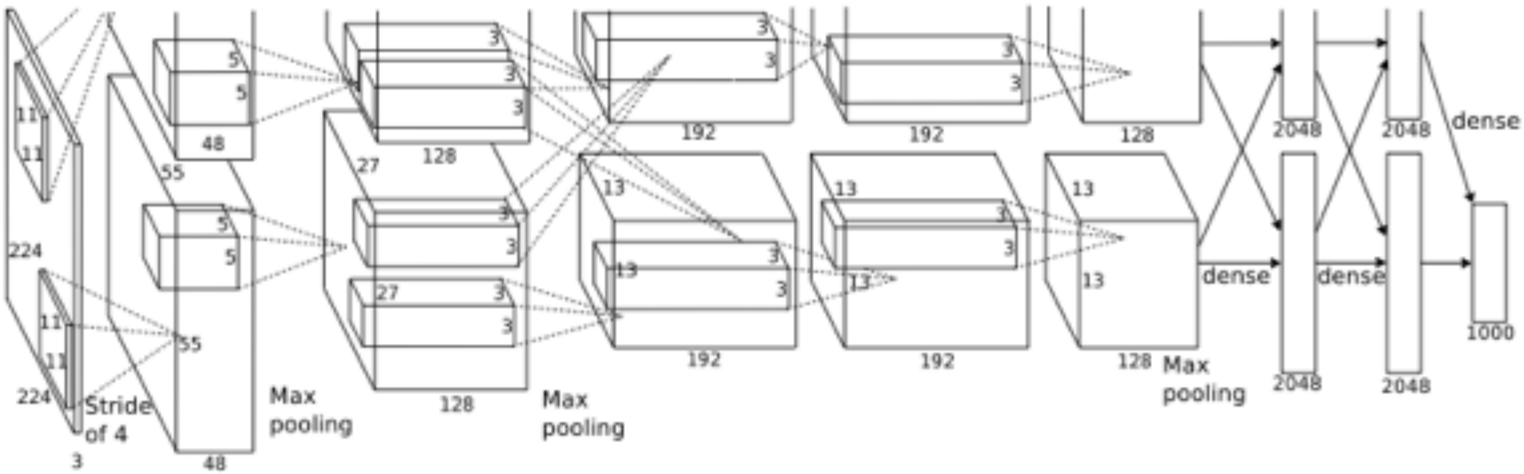
MXNet is a deep learning framework designed for both ***efficiency*** and ***flexibility***

dmlc mxnet

Deep Learning

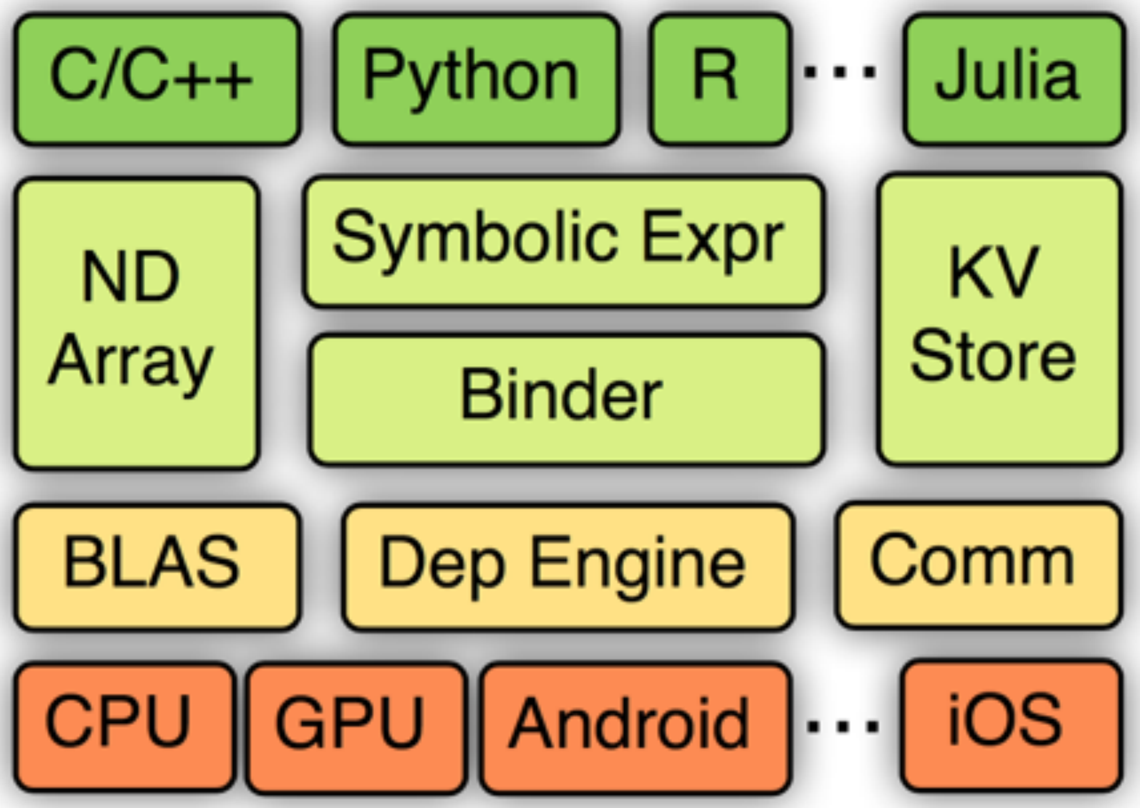


Deep Learning



- Hard to define the network
- Huge computational cost
  - Convolution layers
  - Fully connected layers
- Memory limit
- Way to distribute training process

MXNet Overview



Train Deep Network on MXNet

(declaratively) define layers

```
// model definition
val data = Symbol.Variable("data")
val fc1 = Symbol.FullyConnected(name = "fc1")(Map("data" -> data, "num_hidden" -> 128))
val act1 = Symbol.Activation(name = "relu1")(Map("data" -> fc1, "act_type" -> "relu"))
val fc2 = Symbol.FullyConnected(name = "fc2")(Map("data" -> act1, "num_hidden" -> 64))
val act2 = Symbol.Activation(name = "relu2")(Map("data" -> fc2, "act_type" -> "relu"))
val fc3 = Symbol.FullyConnected(name = "fc3")(Map("data" -> act2, "num_hidden" -> 10))
val mlp = Symbol.SoftmaxOutput(name = "sm")(Map("data" -> fc3))
```

set devices here, e.g., Context.gpu(0,1,2,3)

```
// setup model and fit the training set
val model = FeedForward.newBuilder(mlp)
      .setContext(Context.cpu())
      .setNumEpoch(10)
      .setOptimizer(new SGD(learningRate = 0.1f, momentum = 0.9f, wd = 0.0001f))
      .setTrainData(trainDataIter)
      .setEvalData(valDataIter)
      .build()
```

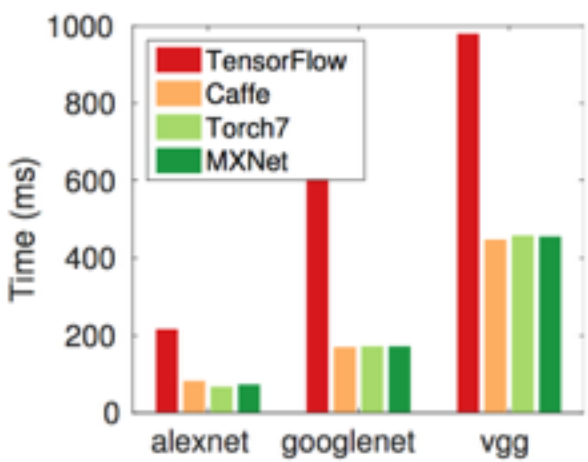
user defined optimizer

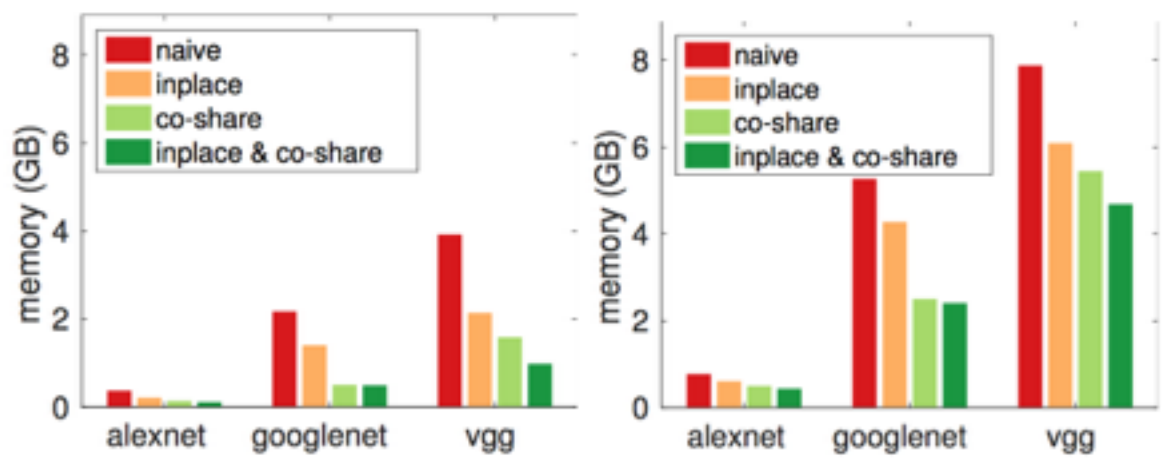
```
val probArrays = model.predict(valDataIter)
// in this case, we do not have multiple outputs
require(probArrays.length == 1)
val prob = probArrays(0)
// get predicted labels
val py = NDArray.argmaxChannel(prob)
// deal with predicted labels py
```

MXNet Benchmarks

Compare MXNet to others on a single forward-backward performance

Internal memory usage of MXNet under various allocation strategies





GPU Support

NDArray is imperative

specify device here

```
val weight = NDArray.empty(Shape(3, 2), Context.gpu(0))
weight -= eta * (grad + lambda * weight);

val weightOnCpu = weight.copyTo(Context.cpu())
```

Copy to another device

One code for both CPU and GPU,
(mshadow) translates at compile time

```
template<typename xpu>
void UpdateSGD(Tensor<xpu, 2> weight, const Tensor<xpu, 2> &grad,
               float eta, float lambda) {
  weight -= eta * (grad + lambda * weight);
}
```

GPU Support: mshadow

https://github.com/dmlc/mshadow

- Efficient: all the expression you write will be lazily evaluated and compiled into optimized code.
  - No temporal memory allocation will happen for expression you write.
  - mshadow will generate specific kernel for every expression you write in compile time.
- Device invariant: you can write one code and it will run on both CPU and GPU.
- Simple: mshadow allows you to write machine learning code using expressions.
- Whitebox: put a float* into the Tensor struct and take the benefit of the package, no memory allocation is happened unless explicitly called.
- Lightweight library: light amount of code to support frequently used functions in machine learning.
- Extendable: user can write simple functions that plugs into mshadow and run on GPU/CPU, no experience in CUDA is required.
- MultiGPU and Distributed ML: mshadow-ps interface allows user to write efficient MultiGPU and distributed programs in an unified way.

Distributed Training

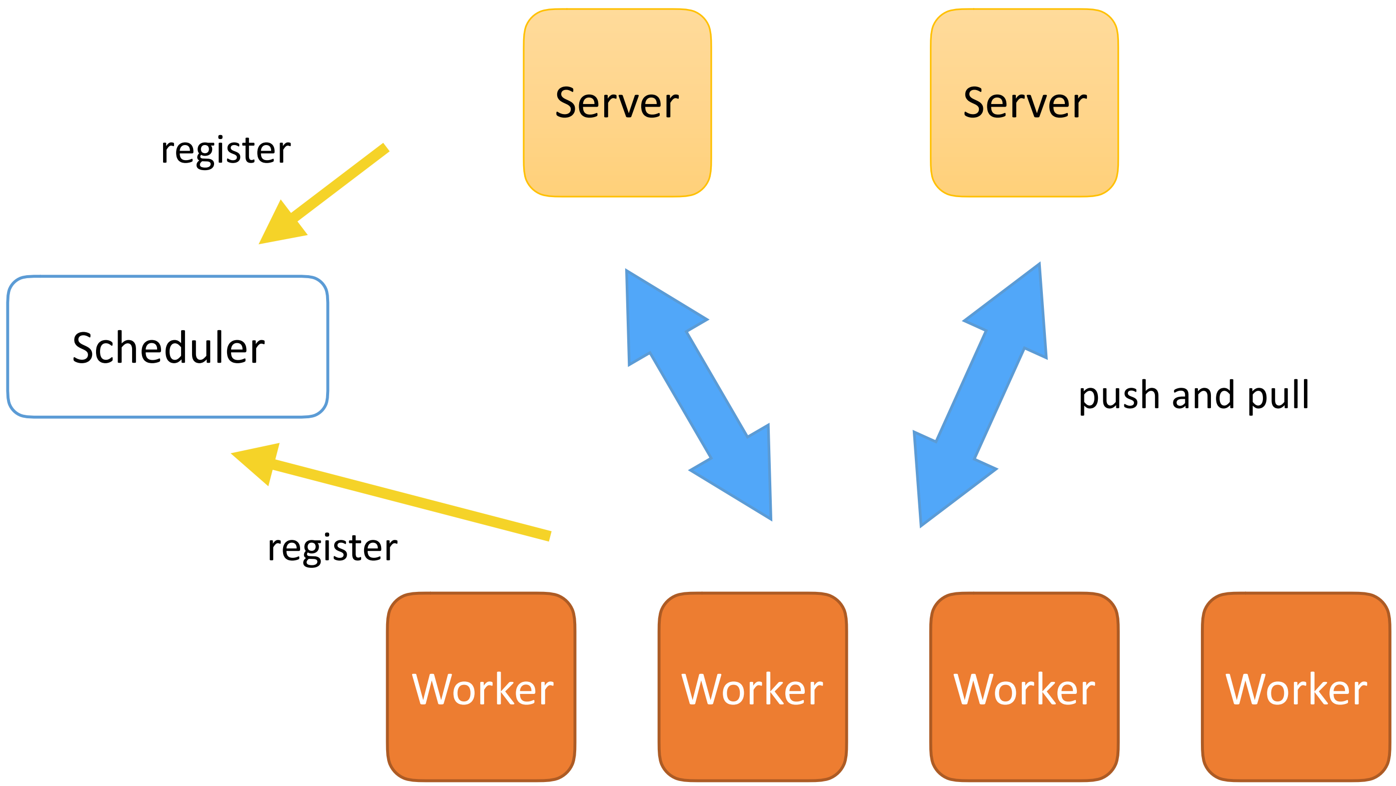
```
val envs = Map("DMLC_ROLE" -> role,
               "DMLC_PS_ROOT_URI" -> schedulerHost,
               "DMLC_PS_ROOT_PORT" -> schedulerPort,
               "DMLC_NUM_WORKER" -> numWorker,
               "DMLC_NUM_SERVER" -> numServer)
KVStoreServer.init(envs)

if (role == "server" || role == "scheduler") {
    // scheduler & server
    KVStoreServer.start()
} else {
    // worker
    val kv = KVStore.create("dist_sync")
    model.fit(trainData = train, kvStore = kv)
}
```

Start scheduler and servers on different nodes

BSP strategy. Or 'dist_async' for fully asynchronous update.

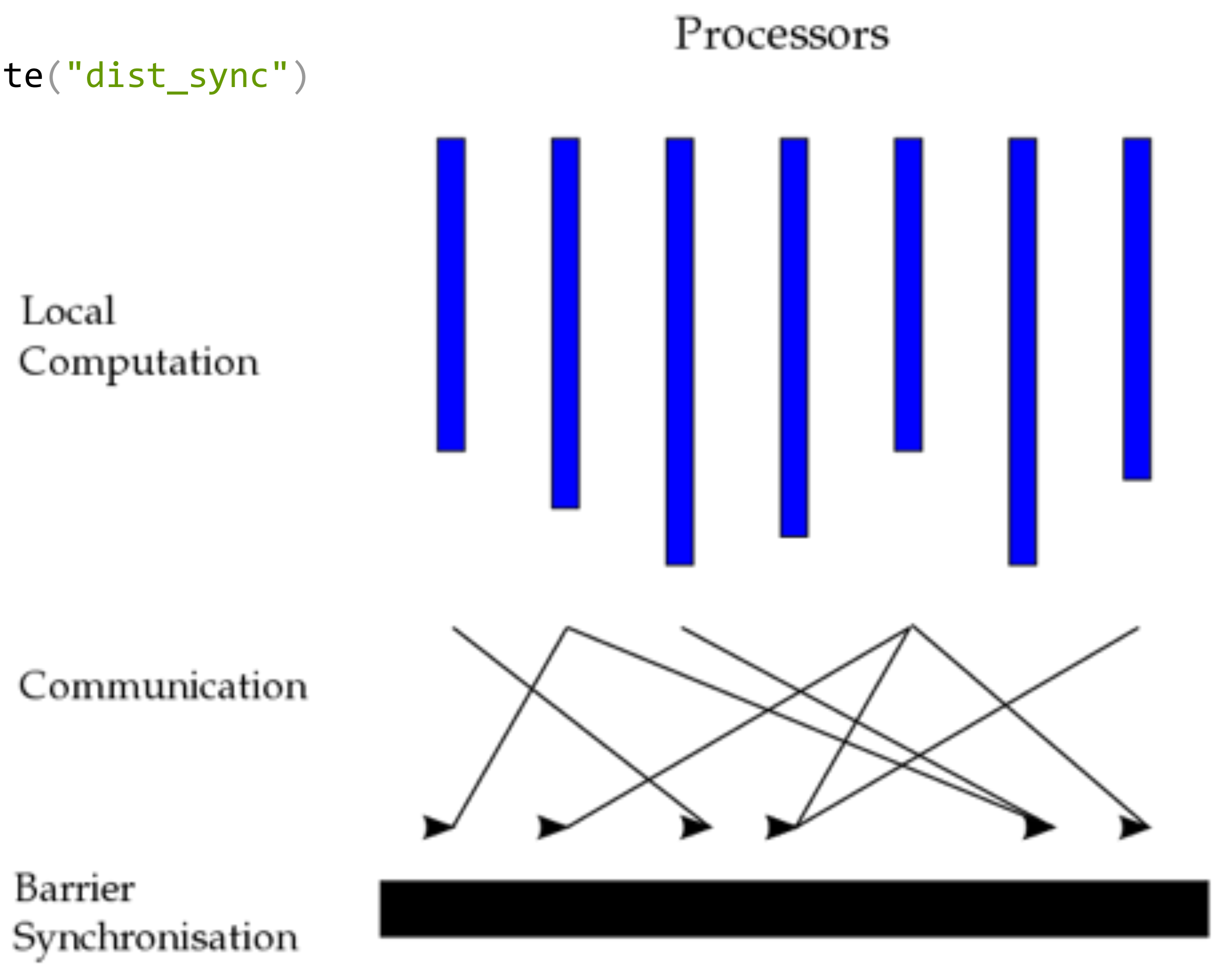
Run model fitting on worker node

Parameter Server

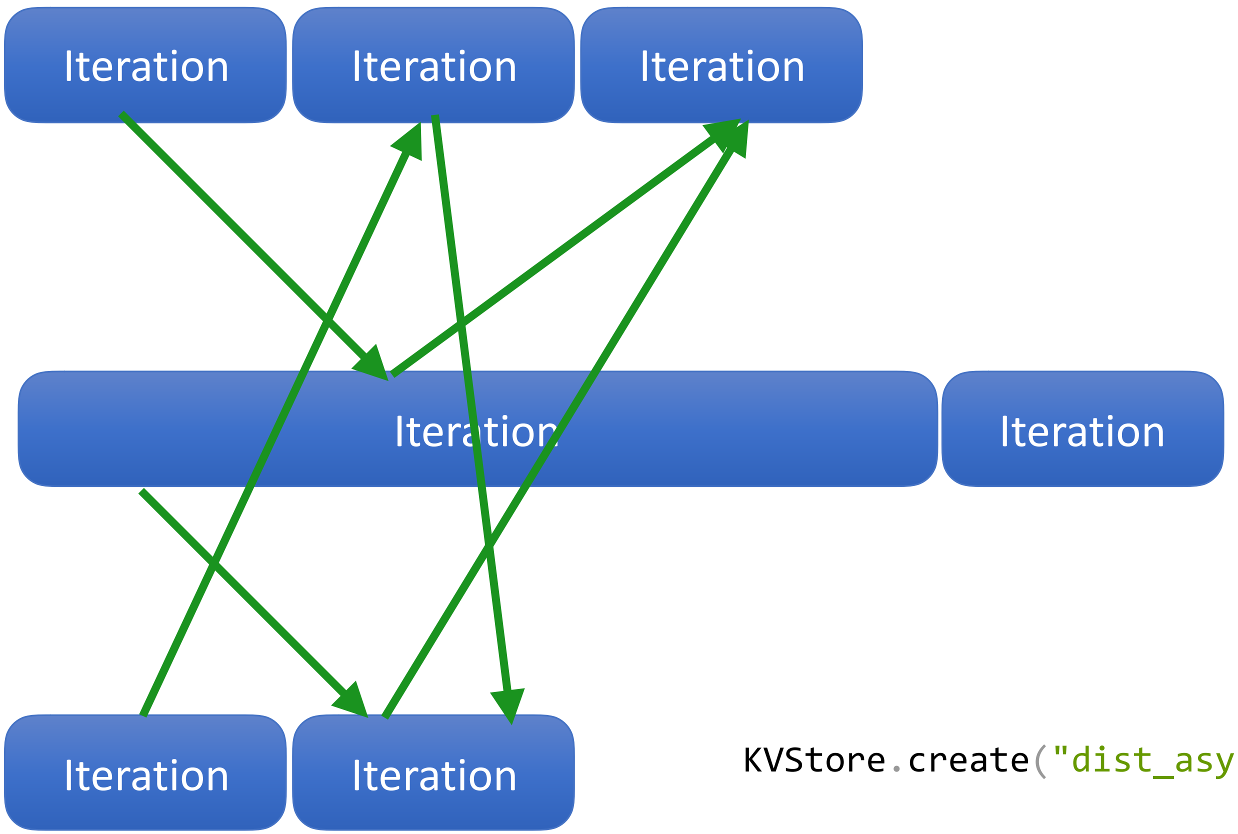


Bulk Synchronous Parallel

```
KVStore.create("dist_sync")
```



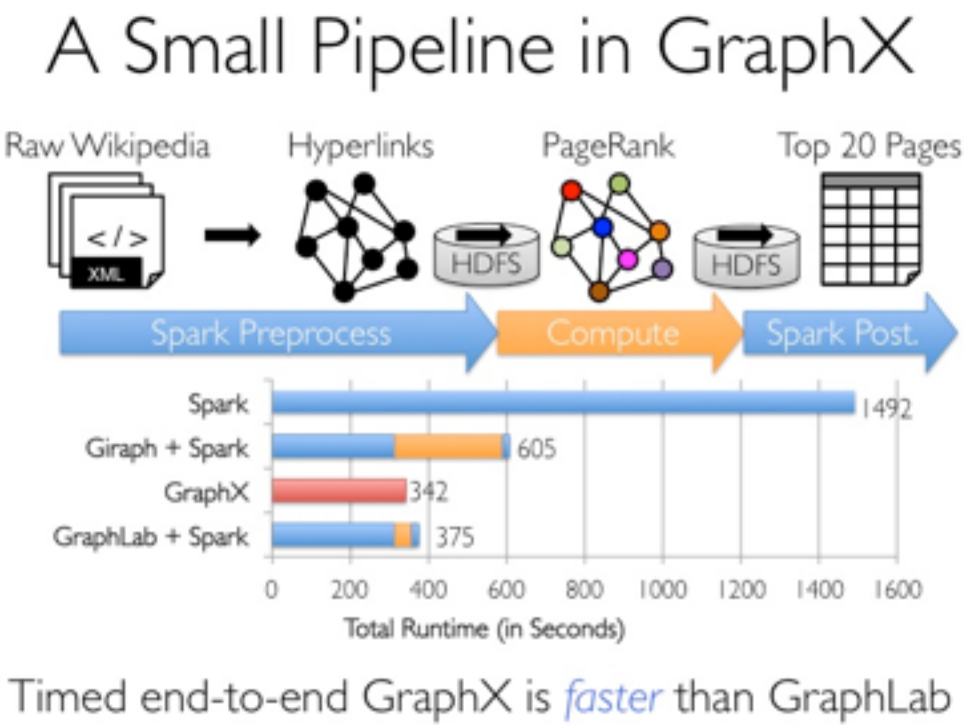
Asynchronous Execution



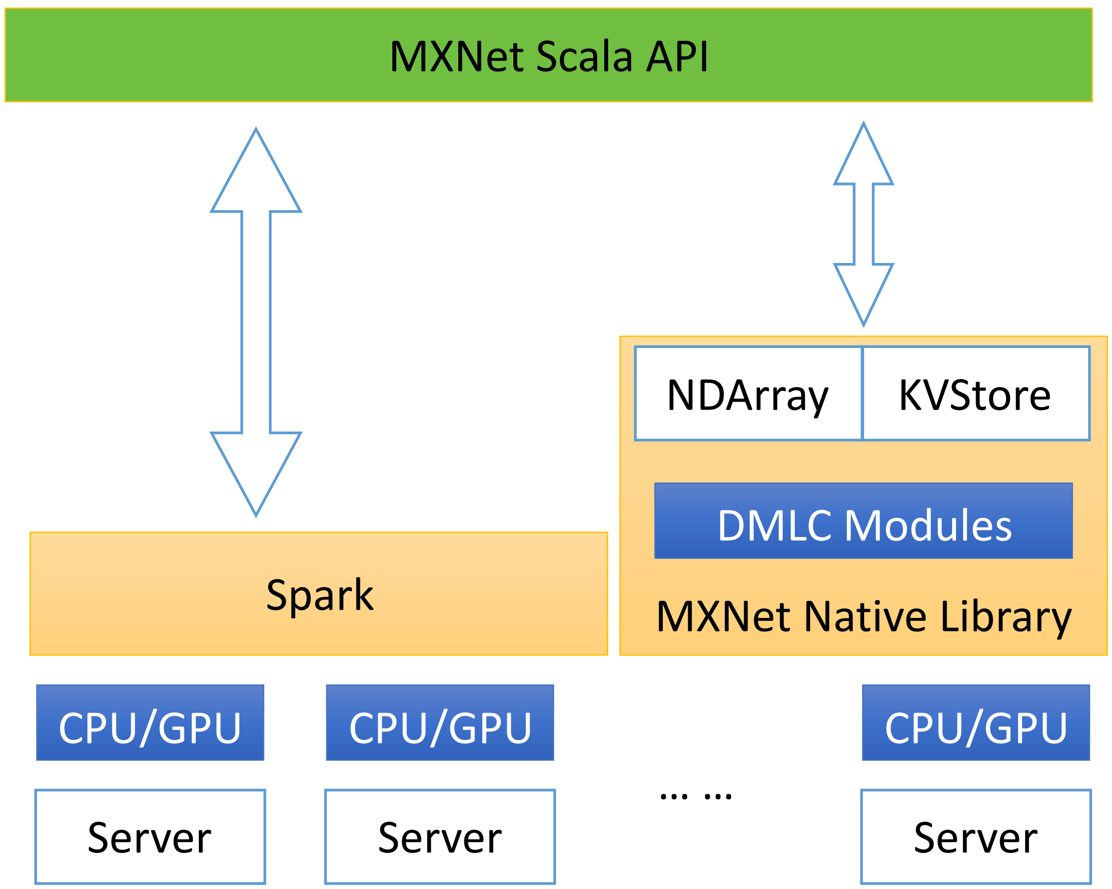

```
KVStore.create("dist_async")
```

Why Spark

- Spark has become the de facto standard for large-scale data processing.
- Combine ETL with machine learning pipeline.



MXNet on Spark



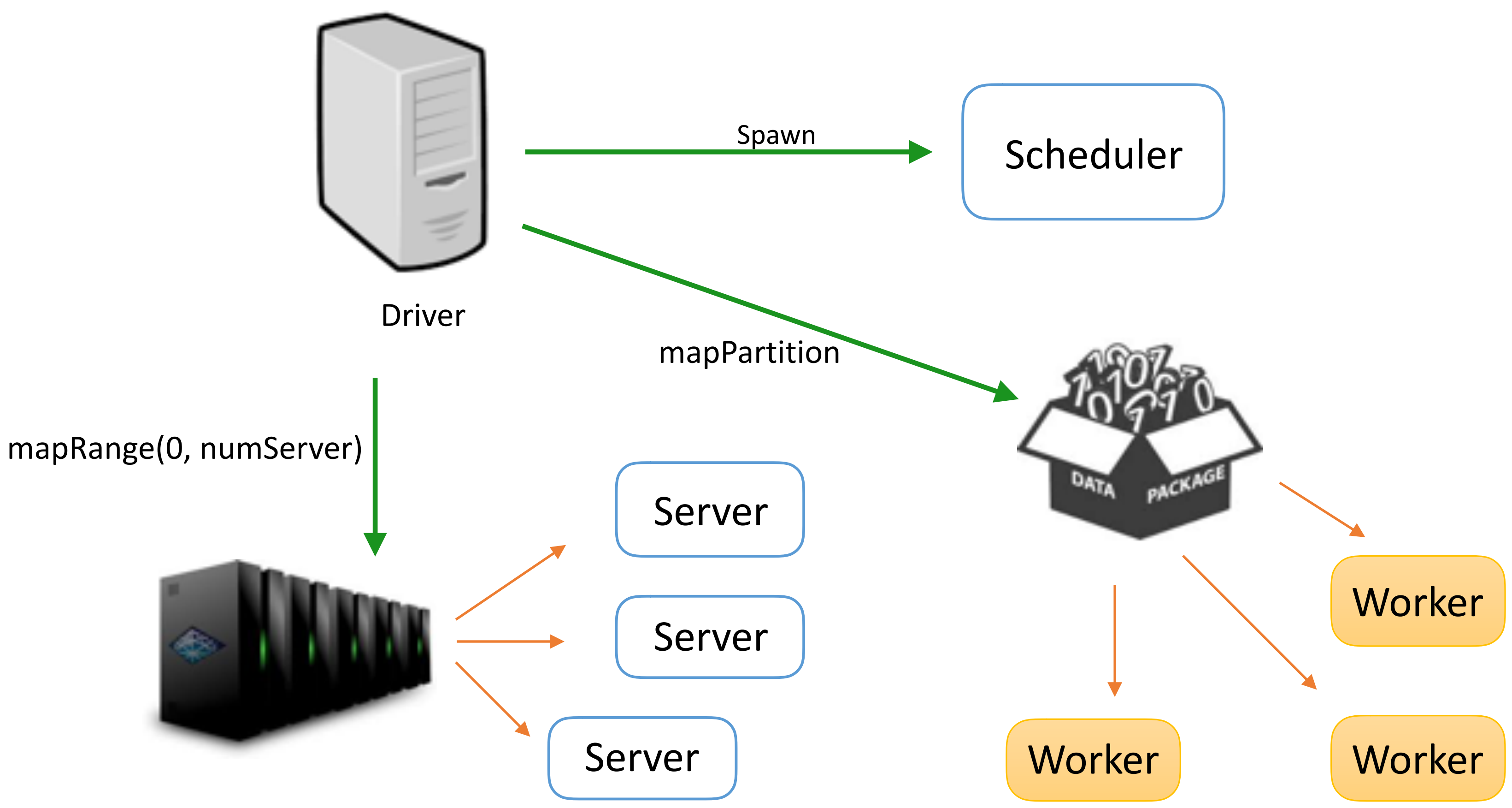
MXNet on Spark API

```
val mxnet = new MXNet()
    .setBatchSize(128)
    .setLabelName("softmax_label")
    .setContext(Context.gpu(0))
    .setDimension(Shape(784))
    .setNetwork(mlp)
    .setNumServer(2)
    .setNumWorker(4)
    .setExecutorClasspath(classpaths)
    .setOptimizer(
        new SGD(learningRate = 0.01f, momentum = 0.9f, wd = 0.00001f)

val model = mxnet.train(trainData)
model.save(modelPath)

val predictions = model.predict(testData)
```

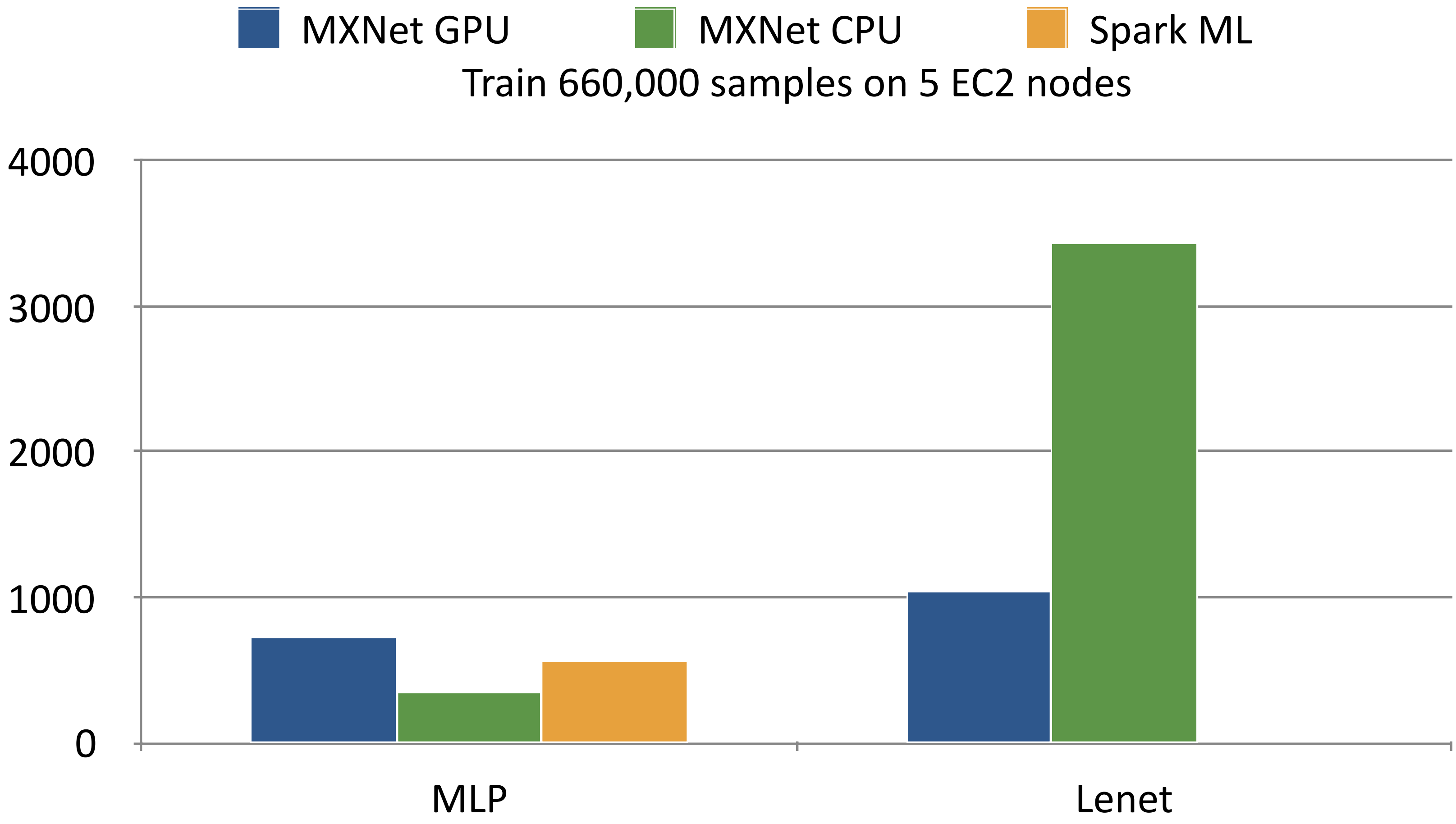
mlp will be serialized to workers

setup server & worker number

Parameter Server Components on Spark

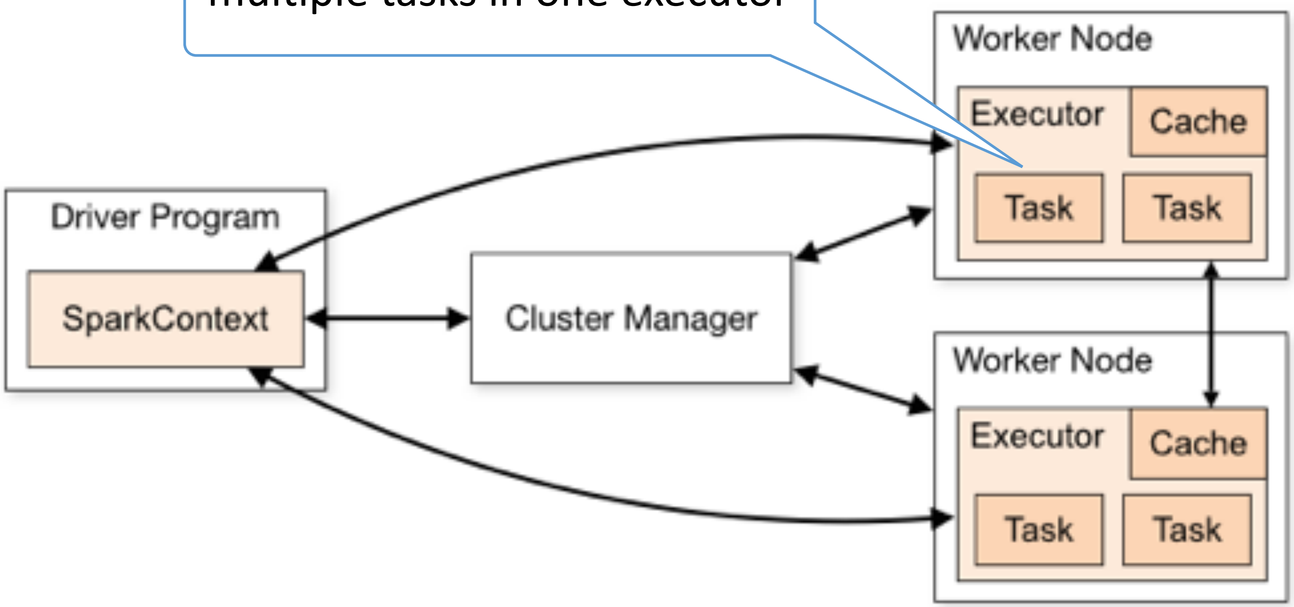


Performance



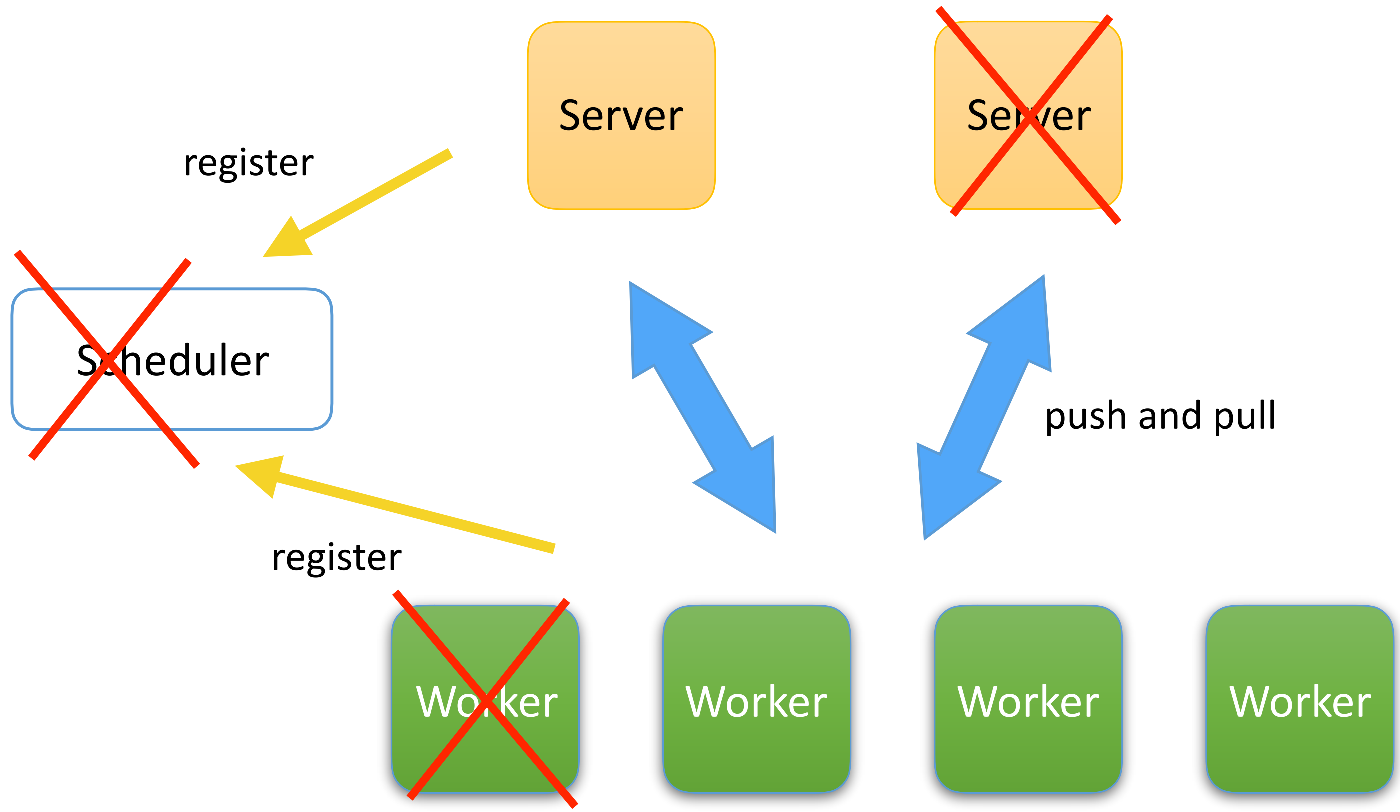
Executors and Processes

multiple tasks in one executor



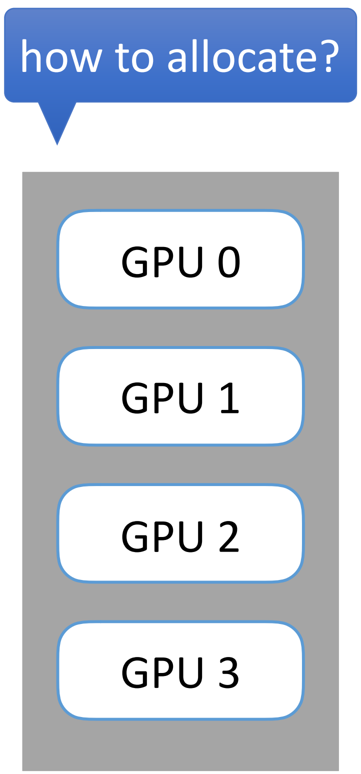
while parameter server and mx engine is singleton

Failover


register
Server
Server
Scheduler
push and pull
register
Worker
Worker
Worker
Worker

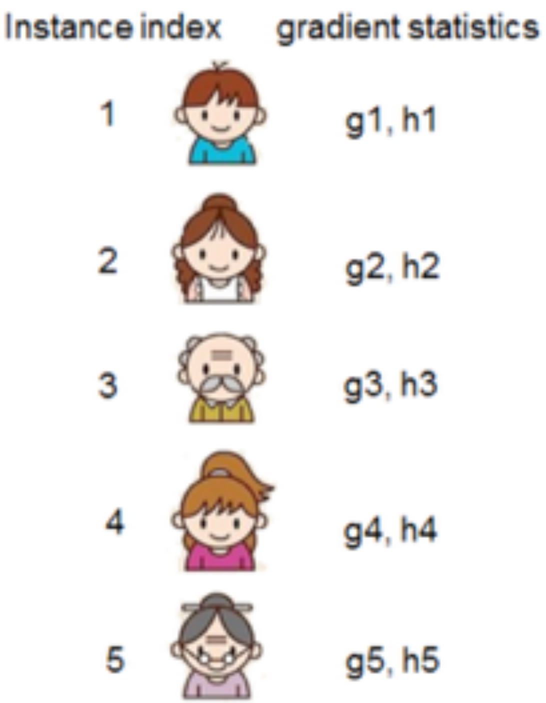
Resource Management

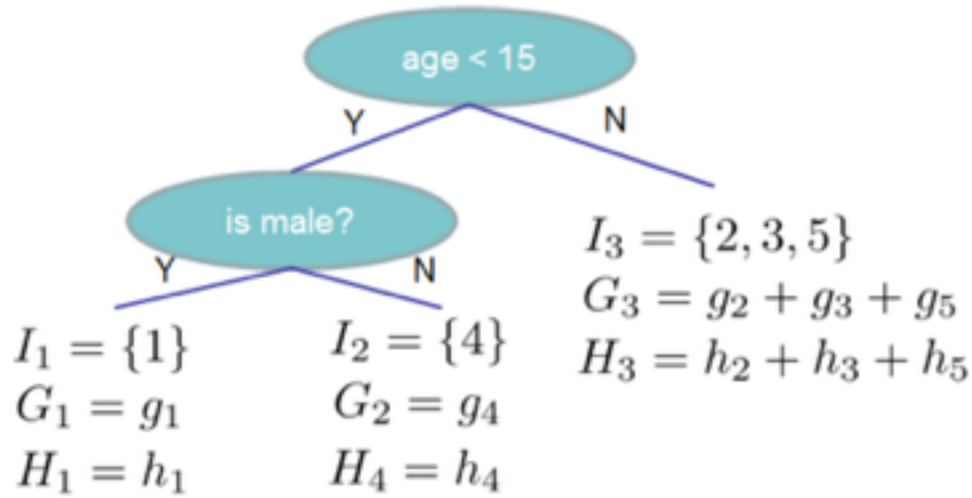
- CPU & GPU allocation on cluster
- What if there’s no enough resources in cluster?




```
% create executor for each GPU
execs = [symbol.bind(mx.gpu(i)) for i in range(ngpu)]
% w -= learning_rate * grad
kvstore.set_updater(…)
% iterating on data
for dbatch in train_iter:
    % iterating on GPUs
    for i in range(ngpu):
        % read a data partition
        copy_data_slice(dbatch, execs[i])
        % pull the parameters
        for key in update_keys:
            kvstore.pull(key, execs[i].weight_array[key])
        % compute the gradient
        execs[i].forward(is_train=True)
        execs[i].backward()
        % push the gradient
        for key in update_keys:
            kvstore.push(key, execs[i].grad_array[key])
```

XGBoost

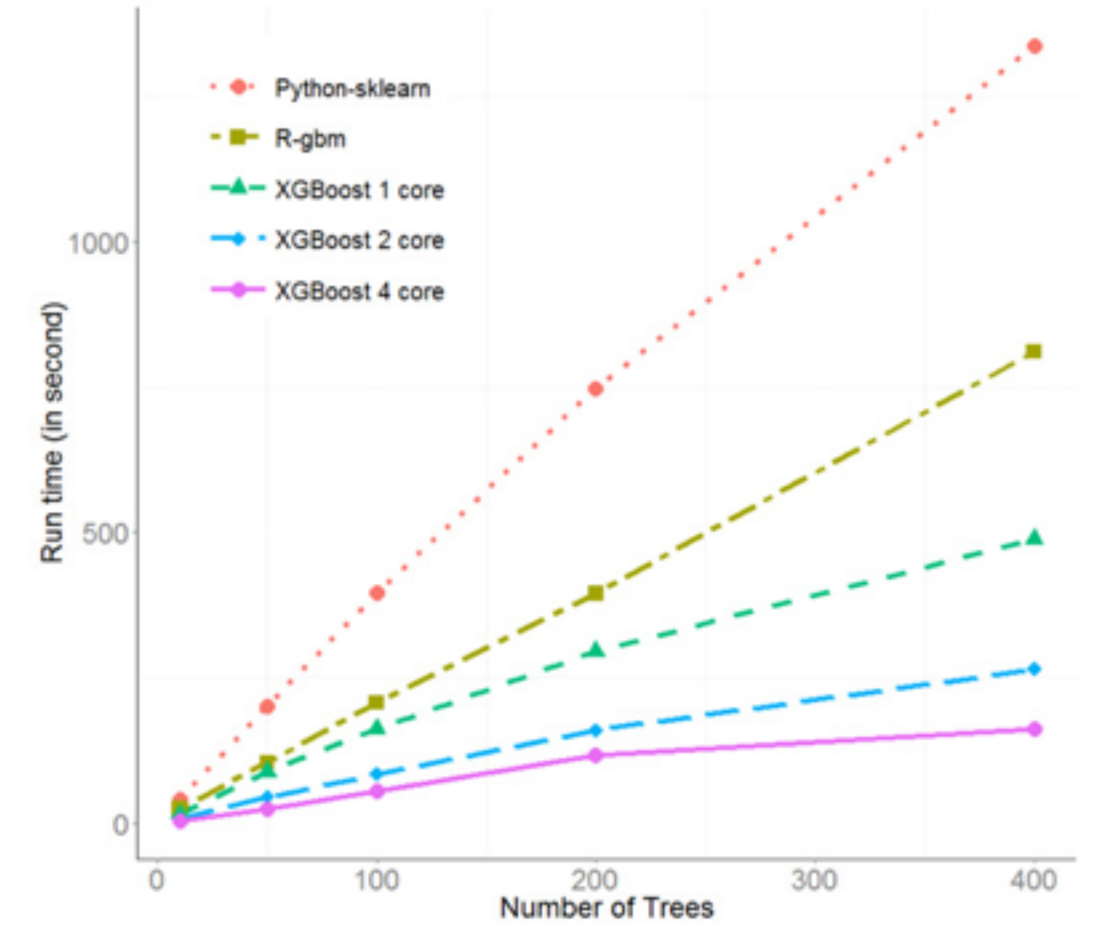




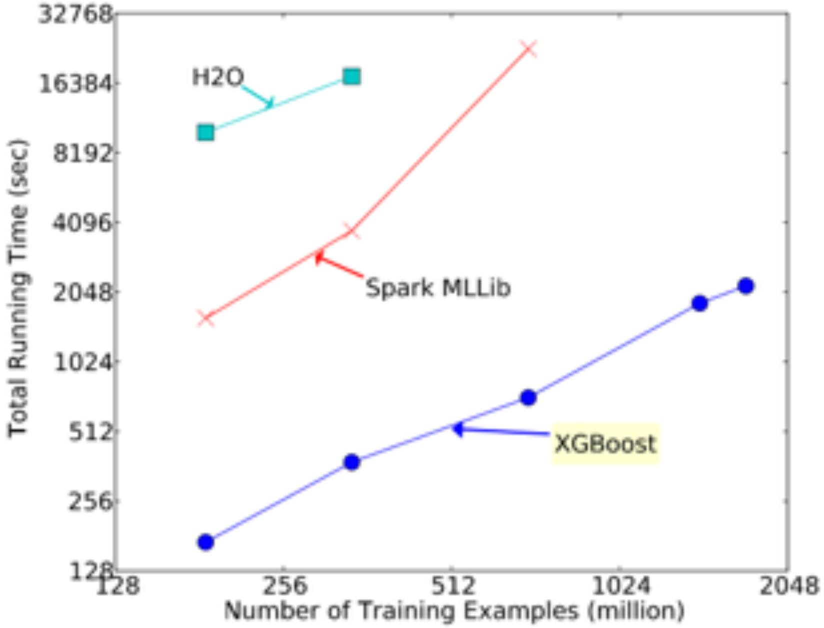
$$Obj = -\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

The smaller the score is, the better the structure is

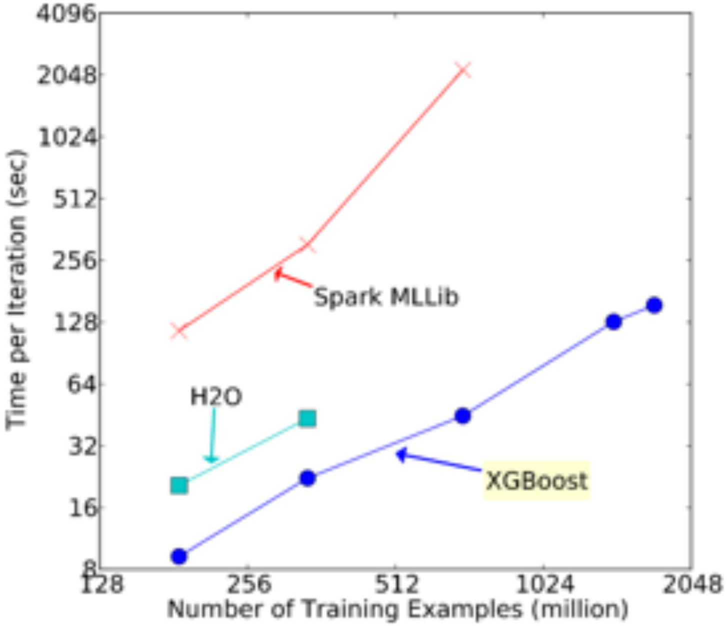
XGBoost Single Machine Performance



XGBoost Distributed Training Performance

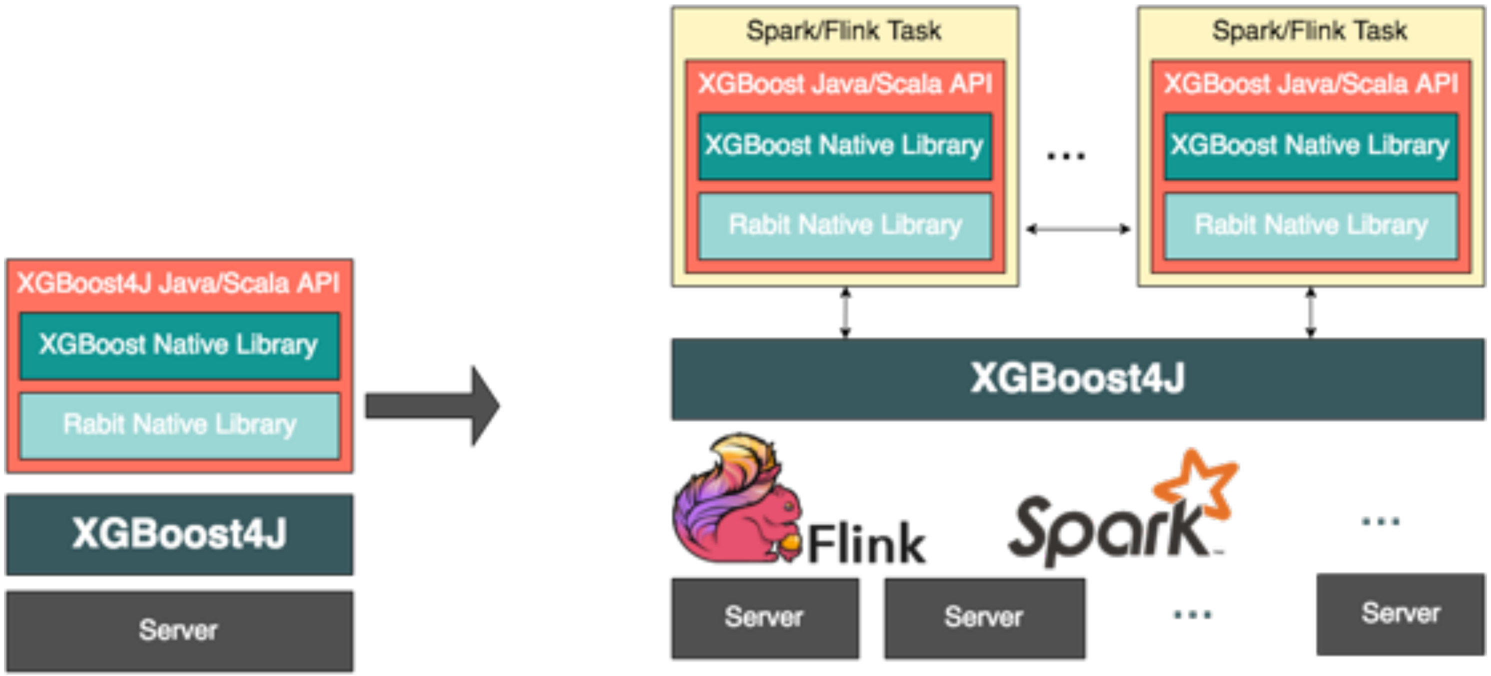


(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

XGBoost on Spark



XGBoost Training on Single Machine

```
val params = new mutable.HashMap[String, Any]()
params += "eta" -> 1.0
params += "max_depth" -> 2
params += "silent" -> 1
params += "objective" -> "binary:logistic"

val watches = new mutable.HashMap[String, DMatrix]
watches += "train" -> trainMax
watches += "test" -> testMax

val round = 2
// train a model
val booster = XGBoost.train(trainMax, params.toMap, round, watches.toMap)

val predicts = booster.predict(testMax)
```

XGBoost Training on Spark

parse libsvm and wrap it as RDD

or, DataSet in Flink

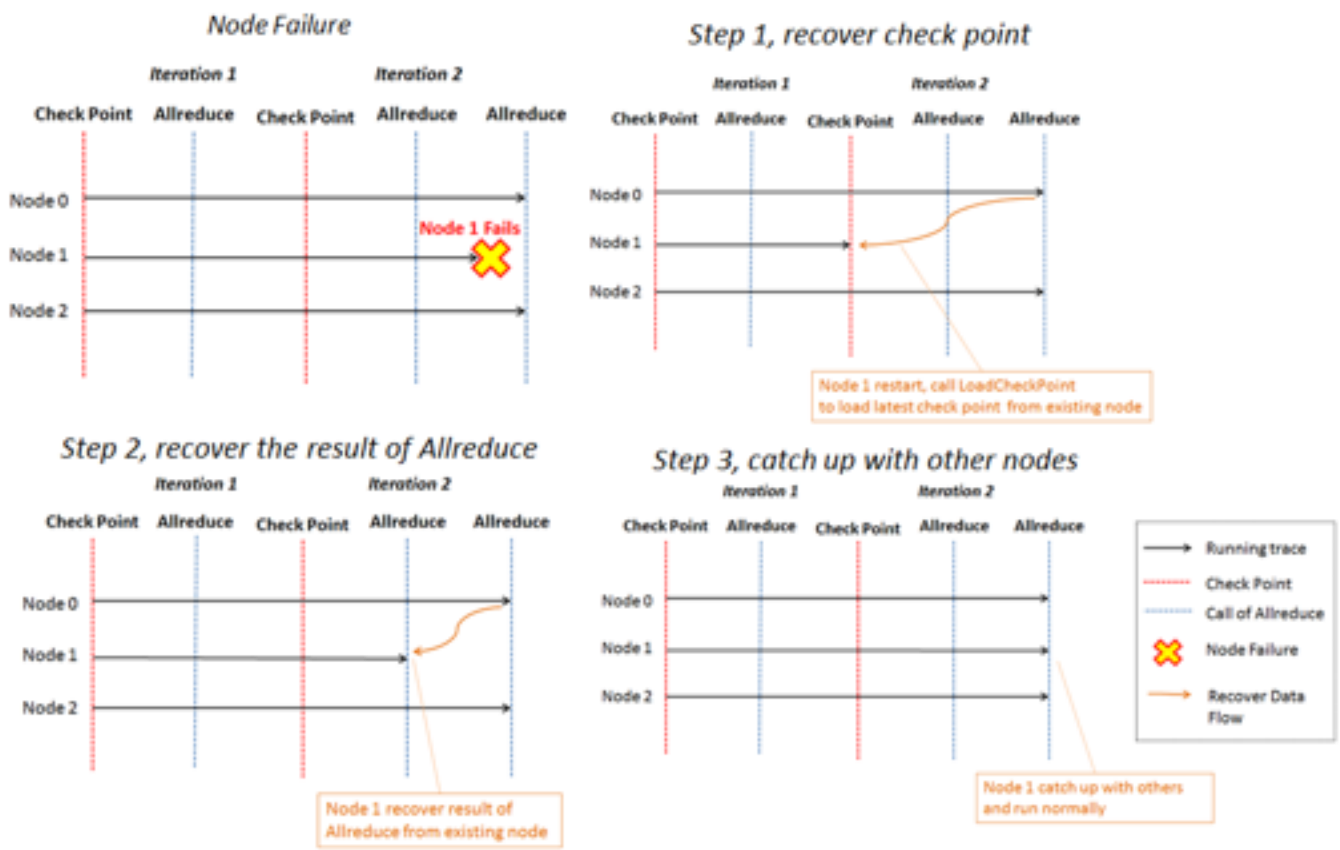
```
val trainRDD
  = MLUtils.loadLibSVMFile(sc, inputTrainPath).repartition(args(1).toInt)
val xgboostModel = XGBoost.train(trainRDD, paramMap, numRound, numWorkers)

// testSet is an RDD containing testset data represented as
// org.apache.spark.mllib.regression.LabeledPoint
val testSet = MLUtils.loadLibSVMFile(sc, inputTestPath)

// local prediction
// import methods in DataUtils to convert
Iterator[org.apache.spark.mllib.regression.LabeledPoint]
// to Iterator[ml.dmlc.xgboost4j.LabeledPoint] in automatic
import DataUtils._
xgboostModel.predict(new DMatrix(testSet.collect().iterator)

// distributed prediction
xgboostModel.predict(testSet)
```

predict locally

Rabit Allreduce

- Portable: rabit is light weight and runs everywhere.
  - Rabit is a library instead of a framework.
  - Rabit only relies on a mechanism to start program, which was provided by most framework.
- Scalable and Flexible: rabit runs fast
  - Rabit program use Allreduce to communicate, and do not suffer the cost between iterations of MapReduce abstraction.
  - Programs can call rabit functions in any order, as opposed to frameworks where callbacks are offered and called by the framework.
  - Programs persist over all the iterations, unless they fail and recover.
- Reliable: rabit dig burrows to avoid disasters
  - Rabit programs can recover the model and results using synchronous function calls.

Rabit Failover



Conclusion

- DMLC is committed to build efficient, flexible and portable machine learning systems.
- Our solution takes advantage of the flexible parallel training approaches and GPU support with DMLC's underlying modules, and the fast data processing pipeline with Spark. We combine the strengths of both side to scale Boosting Trees to larger dataset and faster converge rate, and bring large-scale distributed deep learning to Spark.
- Our JVM stack solution for XGBoost/MXNet is universal, it also works for other data processing systems, which means users can integrate it into their product pipeline easily.

Roadmap

- Parameter server failure tolerance
- Run multiple tasks in one process
- Resource management on cluster
- Improve input data iterators
- Cross validation and parameter selection
- More convenient APIs
- Deploy to Maven repositories
- … …

Acknowledge

- Tianqi Chen: initiator of XGBoost and MXNet.
- Mu Li: guy behind the ps-lite and kvstore.
- Nan Zhu: creator of XGBoost on Spark.
- Zixuan Huang: contributor of Java/Scala package for XGBoost and MXNet.
- Yuan Tang: contributor of Scala package for MXNet.
- Hundreds of contributors:
    - https://github.com/dmlc/mxnet/blob/master/CONTRIBUTORS.md
    - https://github.com/dmlc/xgboost/blob/master/CONTRIBUTORS.md

Looking for contributors

https://github.com/dmlc

Thank you