基于Spark的面向十亿级别特征的大规模机器学习

Yanbo Liang

(Apache Spark committer @ Hortonworks)

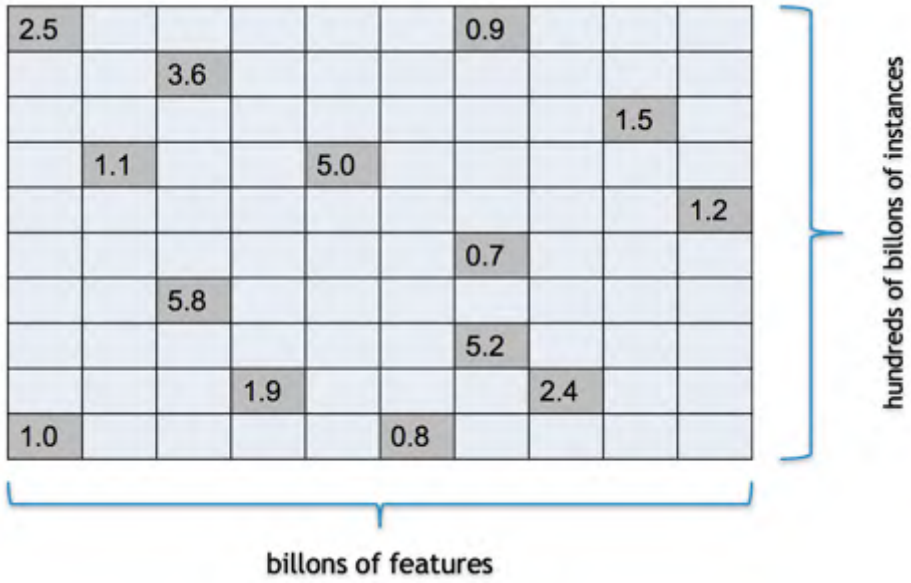
Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work

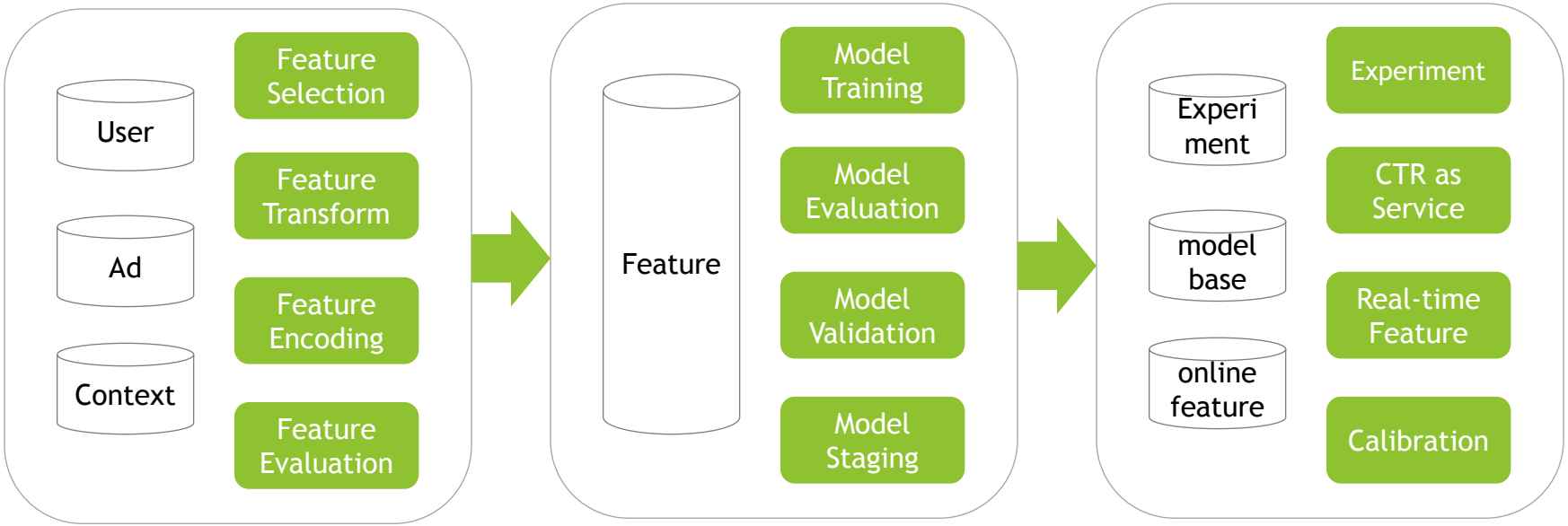
Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work

Big data + big models = better accuracies



CTR Pipeline



CTR Model

- Logistic regression (LR)
  - LR on SGD/LBFGS – batch
  - Follow the regularized leader (FTRL) - online
- Factorization machine (FM/FMM)
- Gradient boosting tree (GBT)
- Deep neural networks (DNN)
- Ensemble

State-of-the-art solution

- GBDT + LR
- DNN + LR

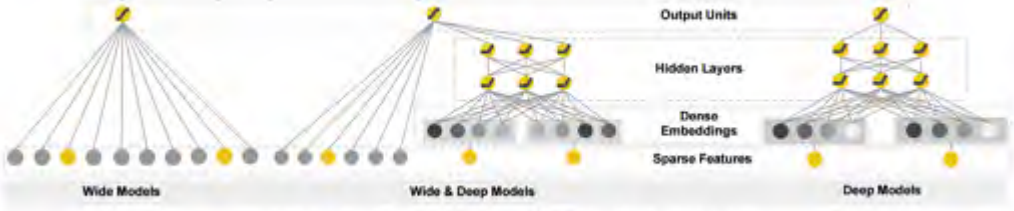


Figure 1: The spectrum of Wide & Deep models.

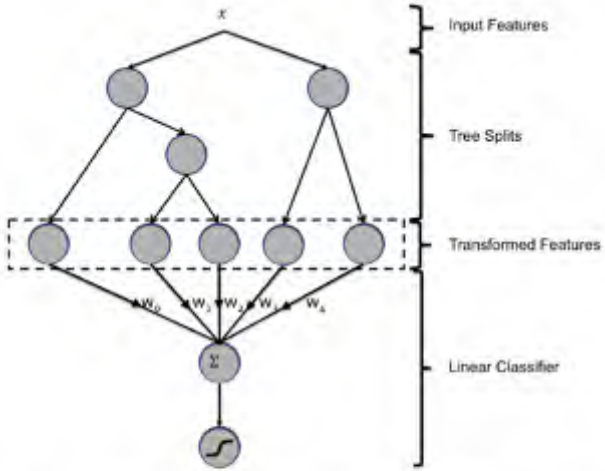


Figure 1: Hybrid model structure. Input features are transformed by means of boosted decision trees. The output of each individual tree is treated as a categorical input feature to a sparse linear classifier. Boosted decision trees prove to be very powerful feature transforms.

Large-scale optimization

- MPI - Sync
- Spark - Sync
- Parameter Server - Async

Spark: a unified platform

"Hidden Technical Debt in Machine Learning Systems", Google

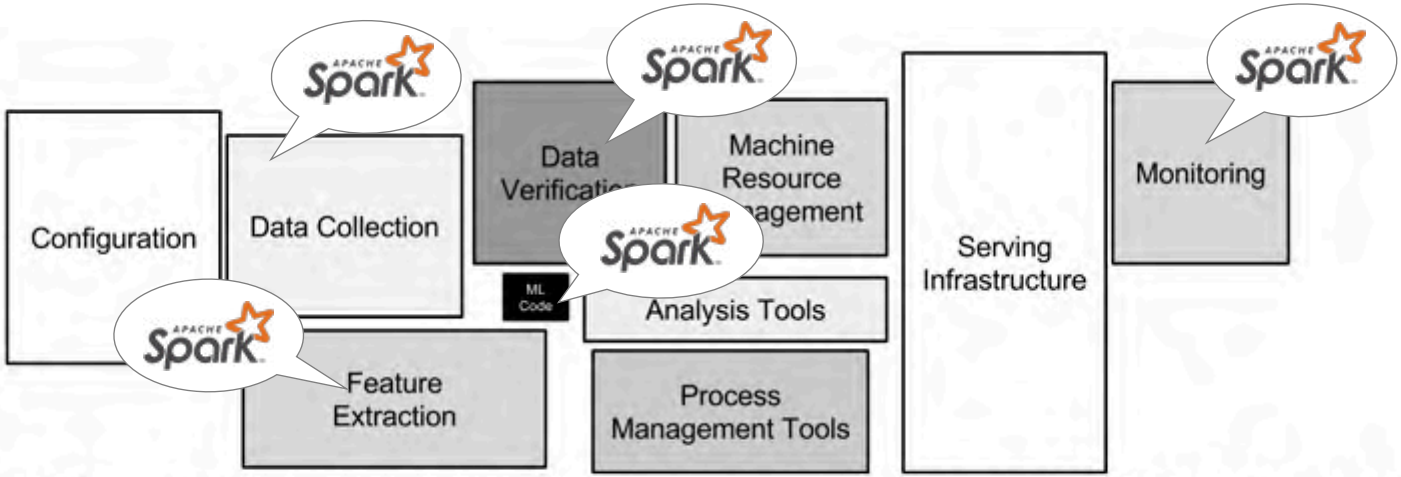
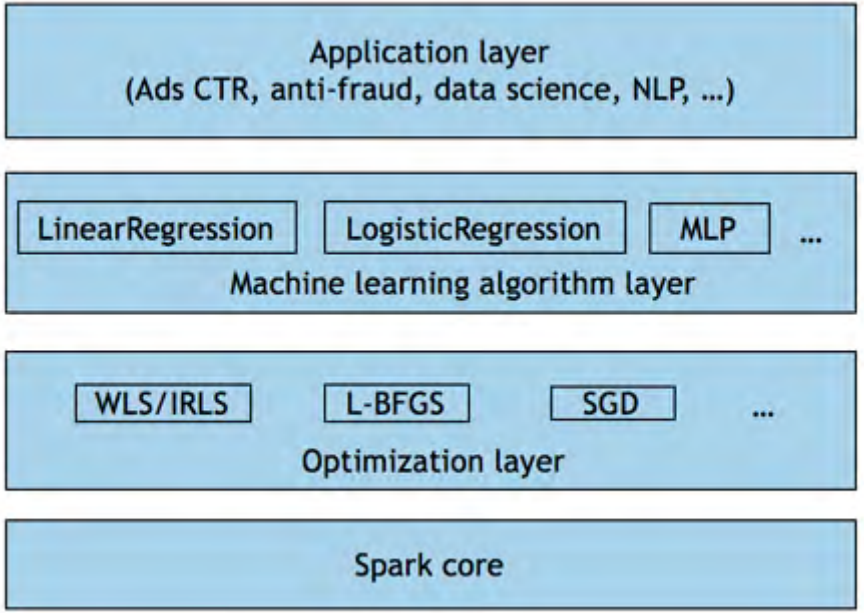


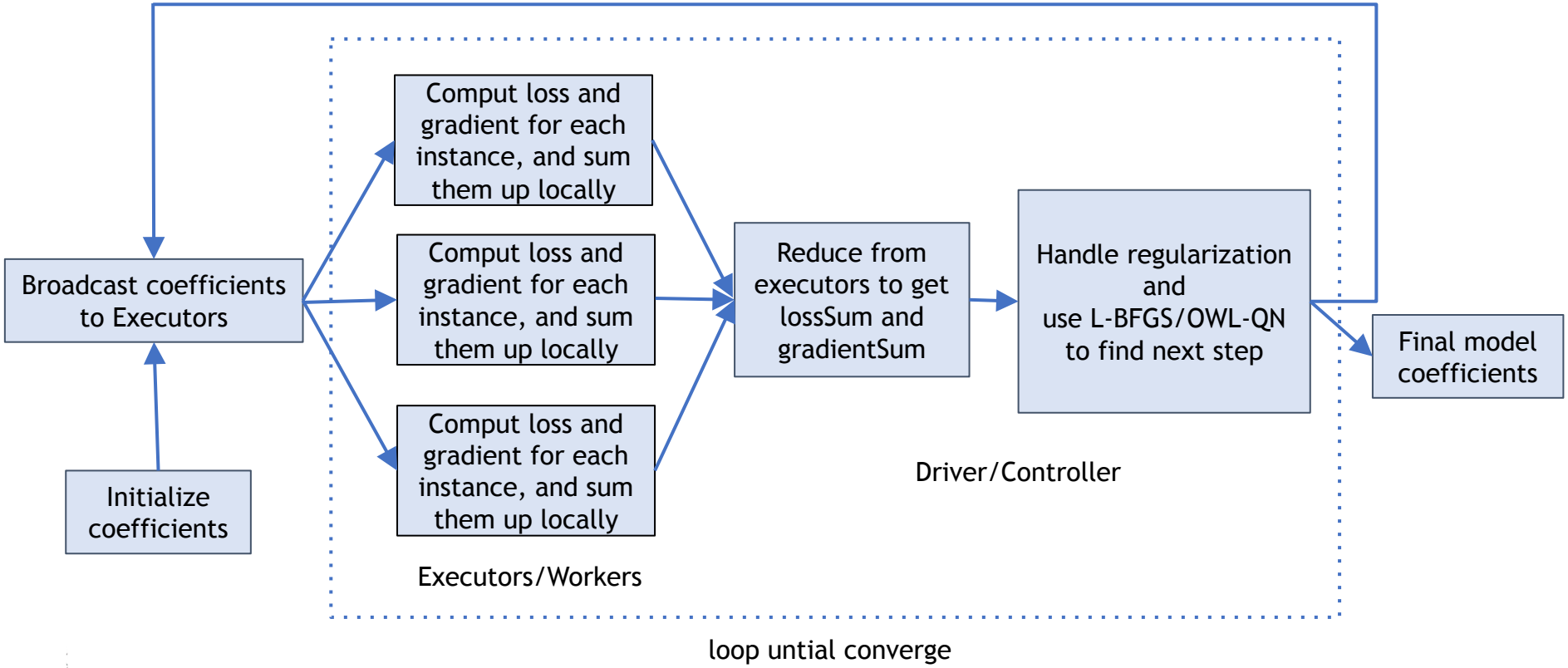
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Optimization

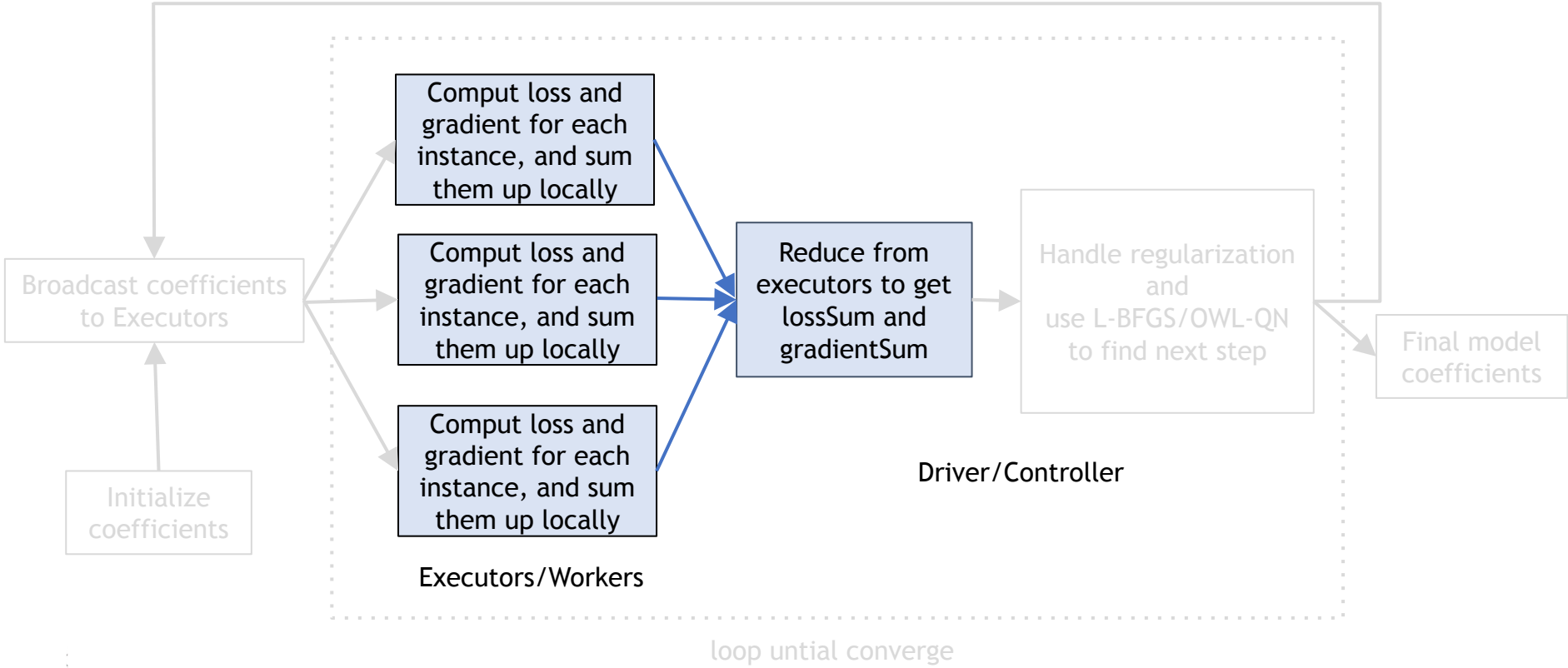
- L-BFGS
- SGD
- WLS



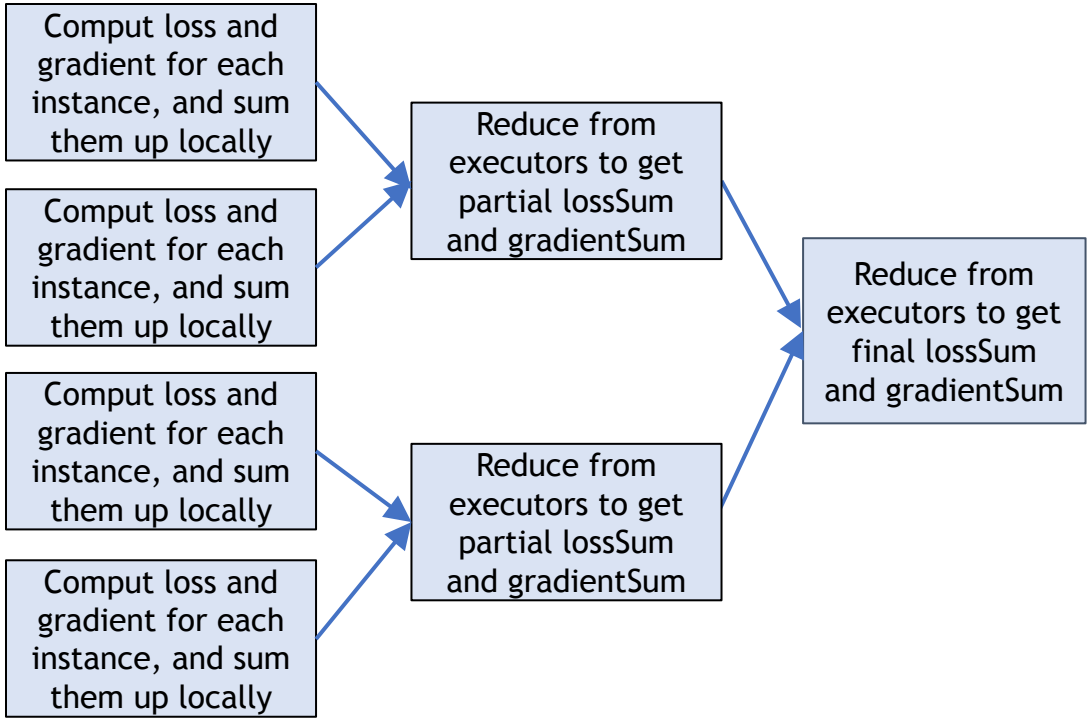
MLlib logistic regression



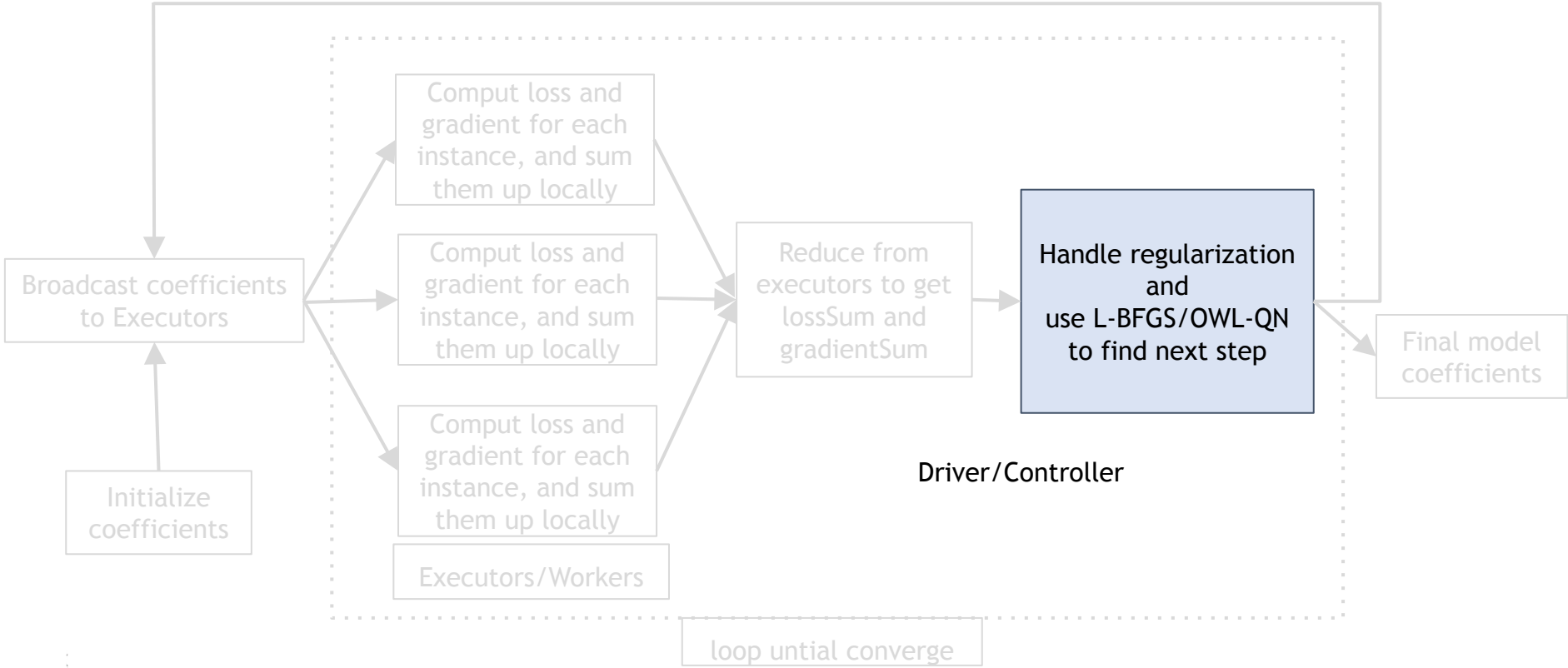
Bottleneck 1



Solution 1

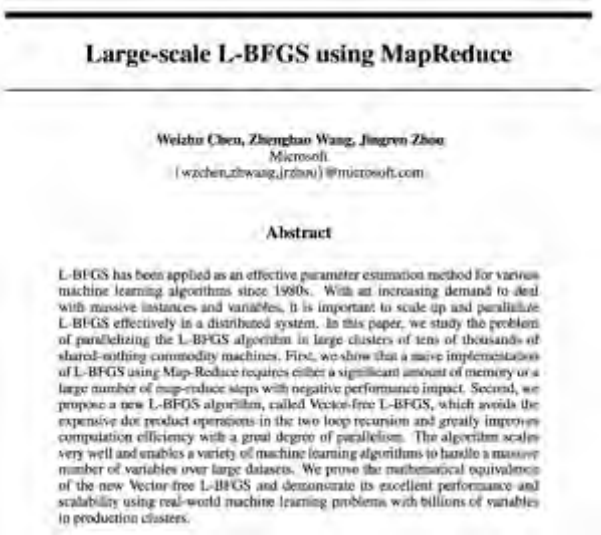

Comput loss and gradient for each instance, and sum them up locally
Comput loss and gradient for each instance, and sum them up locally
Comput loss and gradient for each instance, and sum them up locally
Comput loss and gradient for each instance, and sum them up locally
Reduce from executors to get partial lossSum and gradientSum
Reduce from executors to get partial lossSum and gradientSum
Reduce from executors to get final lossSum and gradientSum

Bottleneck 2



Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work



Large-scale L-BFGS using MapReduce

Weizhu Chen, Zhenghao Wang, Jingren Zhou
Microsoft
{wzchen,zhwang,jrzhou}@microsoft.com

Abstract

L-BFGS has been applied as an effective parameter estimation method for various machine learning algorithms since 1980s. With an increasing demand to deal with massive instances and variables, it is important to scale up and parallelize L-BFGS effectively in a distributed system. In this paper, we study the problem of parallelizing the L-BFGS algorithm in large clusters of tens of thousands of shared-nothing commodity machines. First, we show that a naive implementation of L-BFGS using Map-Reduce requires either a significant amount of memory or a large number of map-reduce steps with negative performance impact. Second, we propose a new L-BFGS algorithm, called Vector-free L-BFGS, which avoids the expensive dot product operations in the two loop recursion and greatly improves computation efficiency with a great degree of parallelism. The algorithm scales very well and enables a variety of machine learning algorithms to handle a massive number of variables over large datasets. We prove the mathematical equivalence of the new Vector-free L-BFGS and demonstrate its excellent performance and scalability using real-world machine learning problems with billions of variables in production clusters.

Distributed Vector

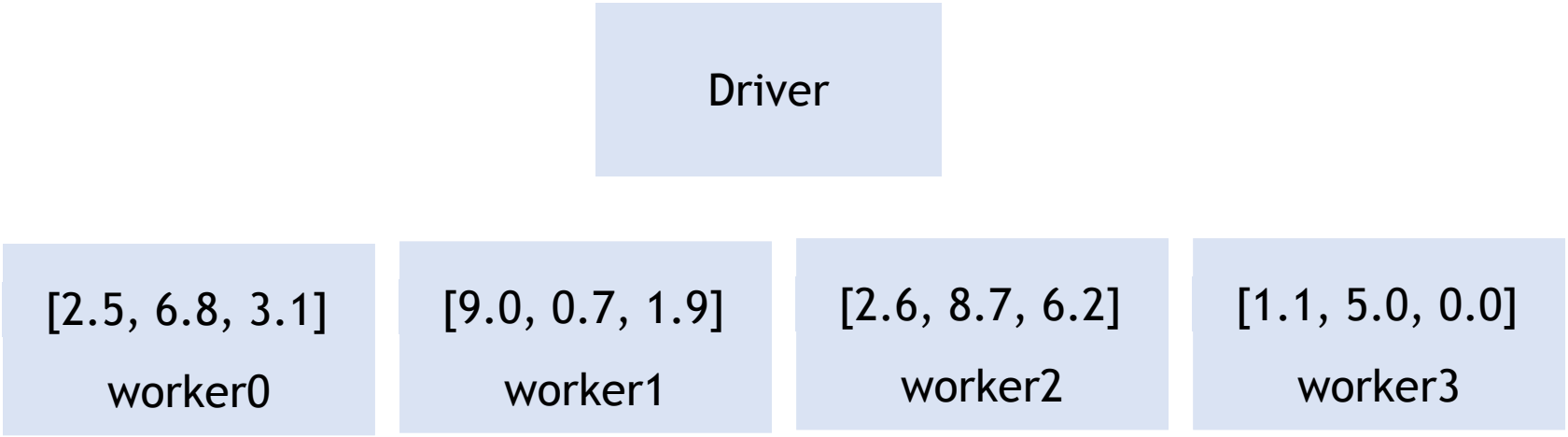
[2.5, 6.8, 3.1, 9.0, 0.7, 1.9, 2.6, 8.7, 6.2, 1.1, 5.0, 0.0]

Driver

worker0 | worker1 | worker2 | worker3

Distributed Vector

[2.5, 6.8, 3.1, 9.0, 0.7, 1.9, 2.6, 8.7, 6.2, 1.1, 5.0, 0.0]



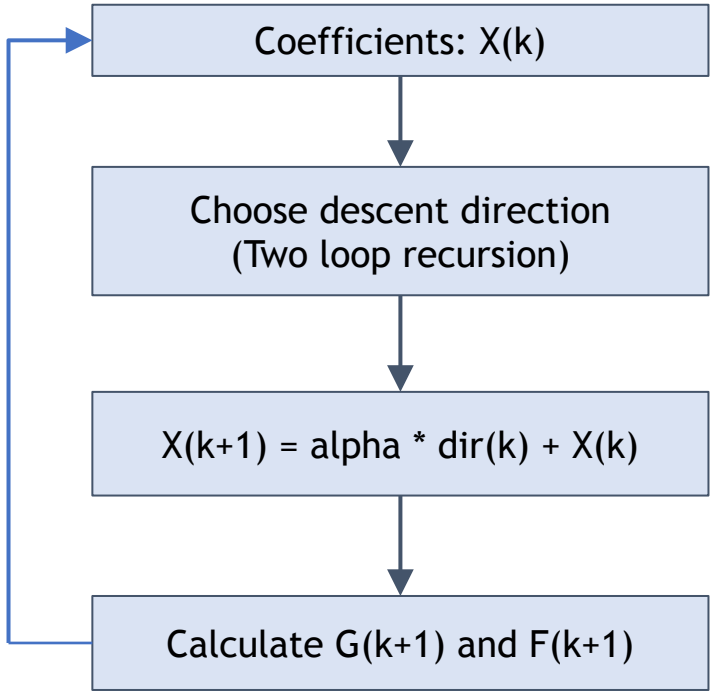
Distributed Vector

- Definition:

```
class DistribuedVector(
        val values: RDD[Vector],
        val sizePerPart: Int,
        val numPartitions: Int,
        val size: Long)
```

- Linear algebra operations:
  - a * x + y
  - a * x + b * y + c * z + …
  - x.dot(y)
  - x.norm
  - …

L-BFGS



Algorithm 1: L-BFGS Algorithm Outline

Input: starting point x_0, integer history size $m > 0$, k=1;
Output: the position x with a minimal objective function

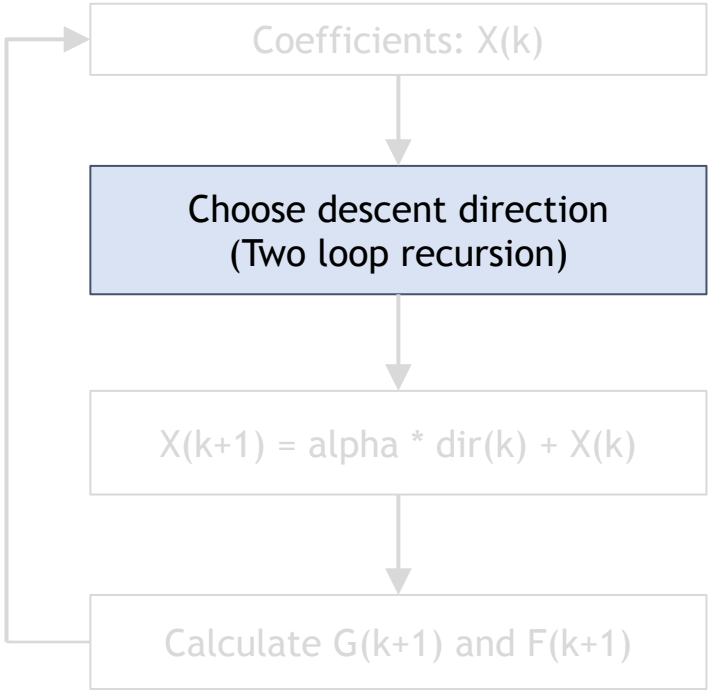
while *no converge* **do**
  Calculate gradient $\nabla f(x_k)$ at position x_k ;
  Compute direction p_k using Algorithm 2 ;
  Compute $x_{k+1} = x_k + \alpha_k p_k$ where α_k is chosen to satisfy Wolfe conditions;
  if $k > m$ **then**
    Discard vector pair s_{k-m}, y_{k-m} from memory storage;;
  end
  Update $s_k = x_{k+1} - x_k, y_k = \nabla f(x_{k+1}) - \nabla f(x_k), k = k + 1$;
end

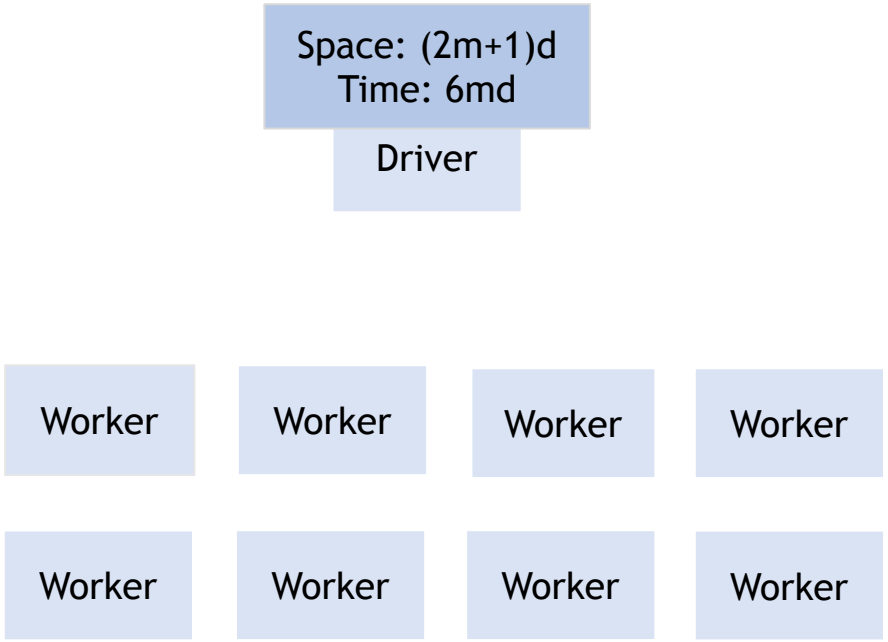
Algorithm 2: L-BFGS two-loop recursion

Input: $\nabla f(x_k)$, s_i, y_i where $i = k - m, ..., k - 1$
Output: new direction p

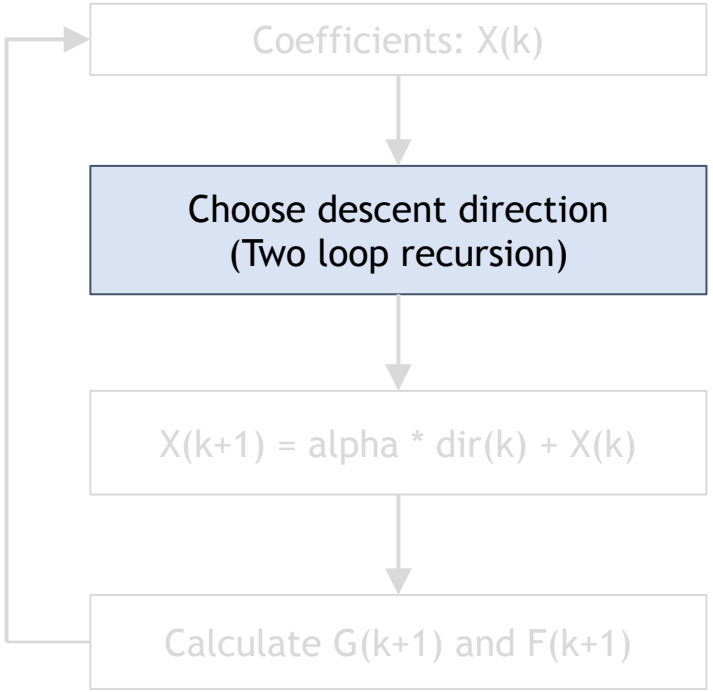
$p = -\nabla f(x_k)$;
for $i \leftarrow k - 1$ **to** $k - m$ **do**
  $\alpha_i \leftarrow \frac{s_i \cdot p}{s_i \cdot y_i}$;
  $p = p - \alpha_i \cdot y_i$;
end
$p = (\frac{s_{k-1} \cdot y_{k-1}}{y_{k-1} \cdot y_{k-1}})p$
for $i \leftarrow k - m$ **to** $k - 1$ **do**
  $\beta = \frac{y_i \cdot p}{s_i \cdot y_i}$;
  $p = p + (\alpha_i - \beta) \cdot s_i$;
end

L-BFGS


Coefficients: X(k)
Choose descent direction
(Two loop recursion)
X(k+1) = alpha * dir(k) + X(k)
Calculate G(k+1) and F(k+1)


Space: (2m+1)d
Time: 6md
Driver
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker

Vector-free L-BFGS

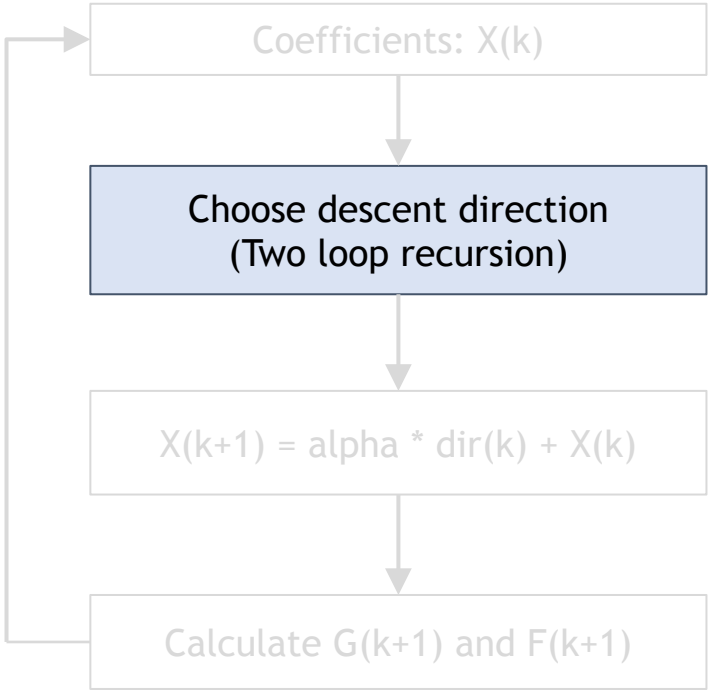


Algorithm 3: Vector-free L-BFGS two-loop recursion

Input: $(2m+1) * (2m+1)$ dot product matrix between b_i
Output: The coefficients δ_i where $i = 1, 2, ...2m+1$

for $i \leftarrow 1$ **to** $2m+1$ **do**
  $\delta_i = i \leq 2m \; ? \; 0 : -1$
end
for $i = k-1$ **to** $k-m$ **do**
  $j = i - (k-m) + 1$;
  $\alpha_i \leftarrow \frac{s_i \cdot p}{s_i \cdot y_i} = \frac{b_j \cdot p}{b_j \cdot b_{m+j}} = \frac{\sum_{l=1}^{2m+1} \delta_l b_l \cdot b_j}{b_j \cdot b_{m+j}}$;
  $\delta_{m+j} = \delta_{m+j} - \alpha_i$;
end
for $i \leftarrow 1$ **to** $2m+1$ **do**
  $\delta_i = (\frac{b_m \cdot b_{2m}}{b_{2m} \cdot b_{2m}})\delta_i$
end
for $i \leftarrow k-m$ **to** $k-1$ **do**
  $j = i - (k-m) + 1$;
  $\beta = \frac{b_{m+j} \cdot p}{b_j \cdot b_{m+j}} = \frac{\sum_{l=1}^{2m+1} \delta_l b_{m+j} \cdot b_l}{b_j \cdot b_{m+j}}$;
  $\delta_j = \delta_j + (\alpha_i - \beta)$
end

Vector-free L-BFGS



Space: (2m+1)^2
Time: 8m^2

Driver

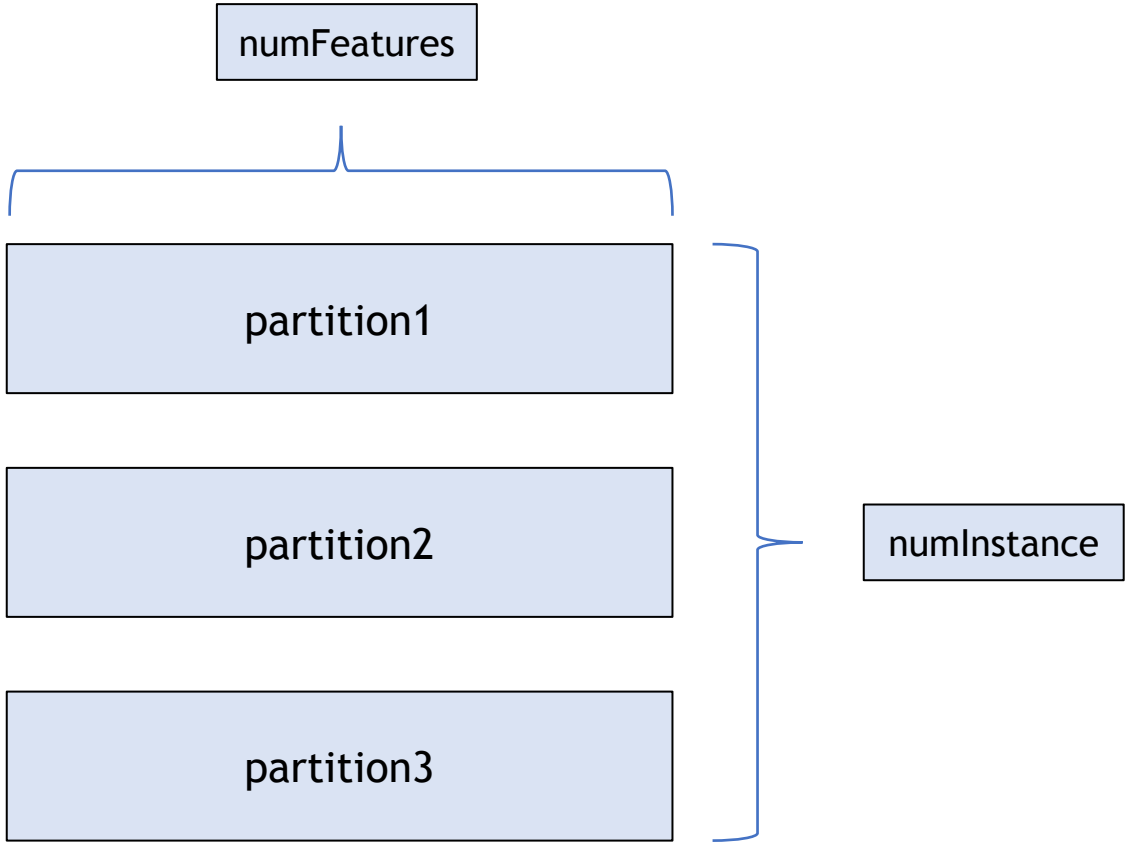
Worker Worker Worker Worker

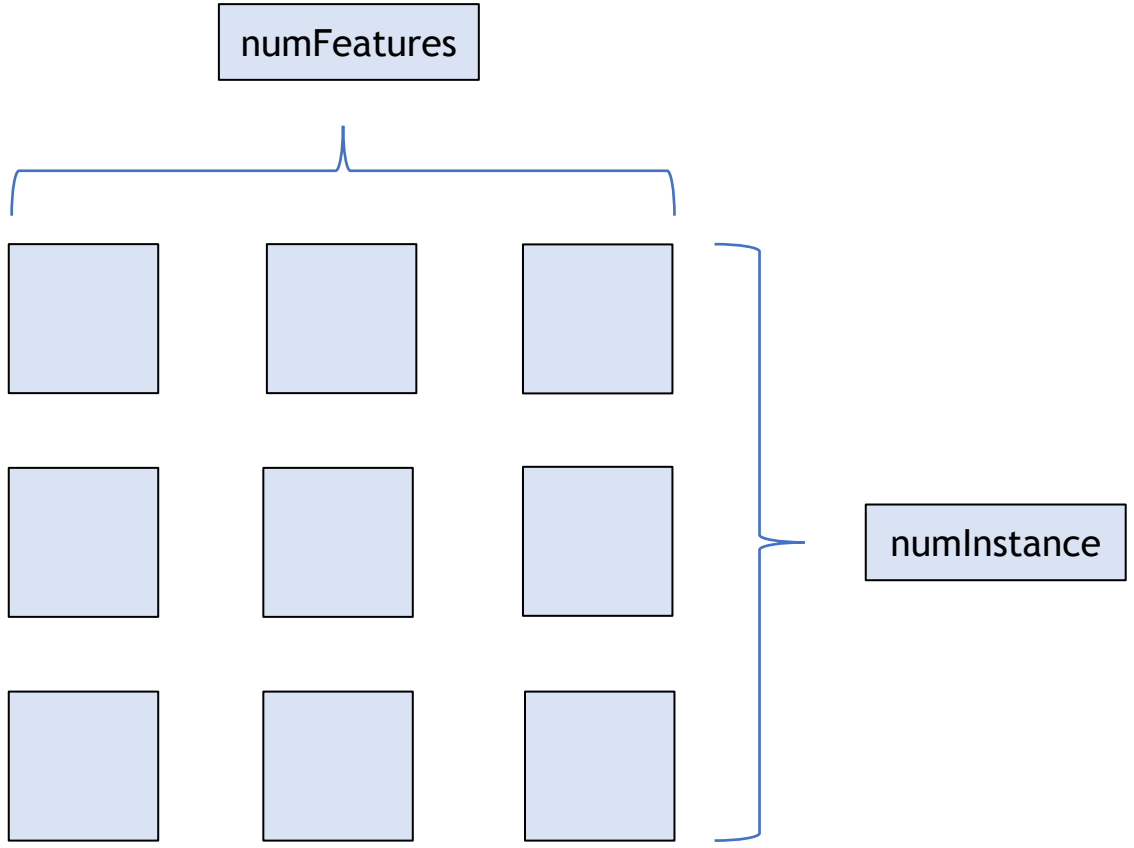
Space: (2m+1)*d
Time: 6md

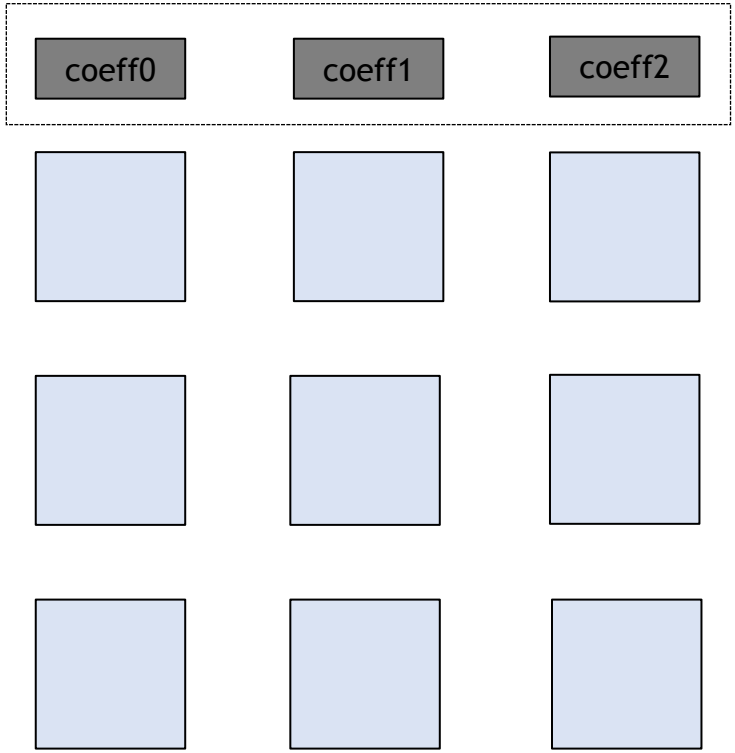
Worker Worker Worker Worker

Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work


numFeatures
partition1
partition2
partition3
numInstance

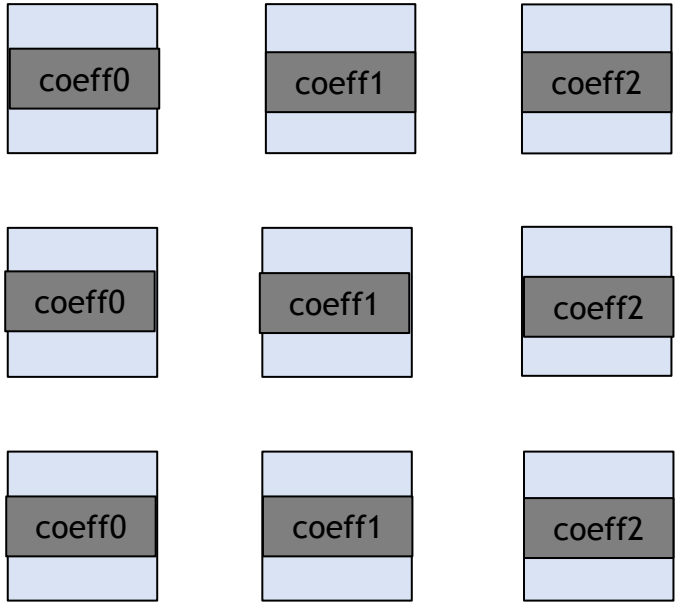

numFeatures
numInstance

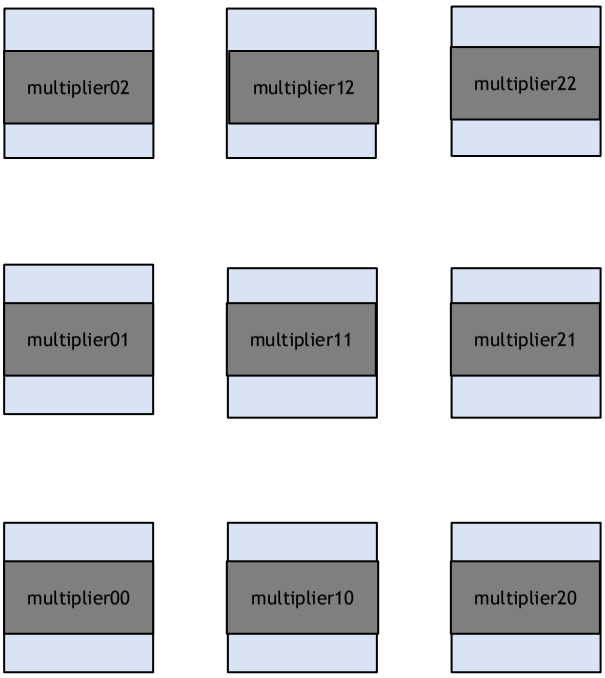


coefficients: DistributedVector

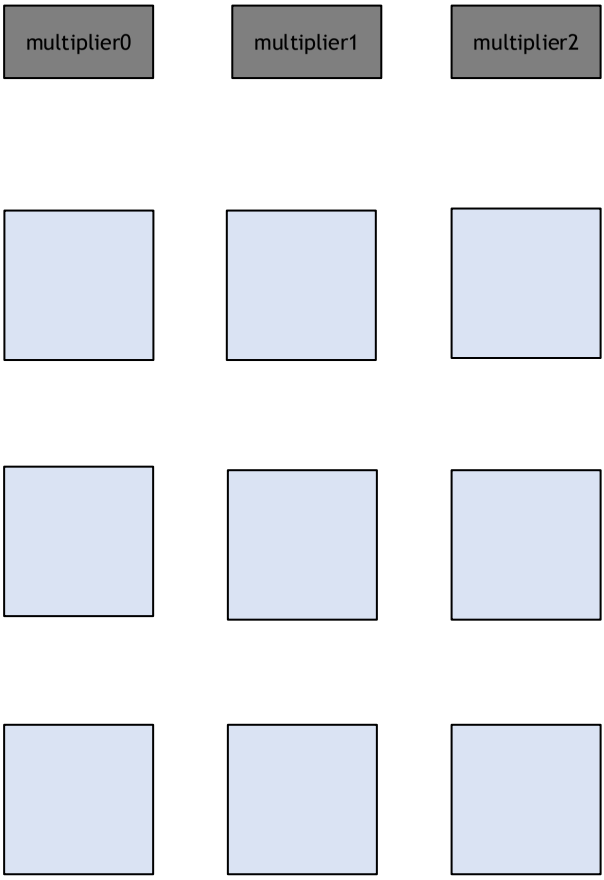
$$margin = -W^T X$$

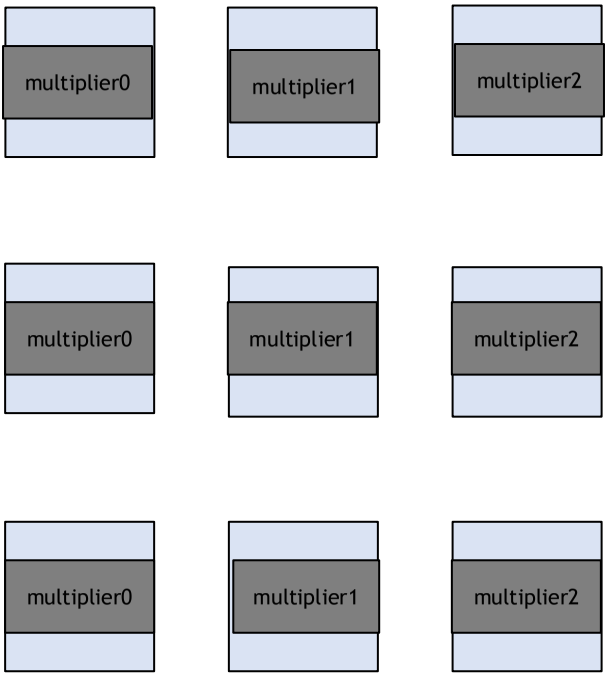
$$multiplier = \frac{1.0}{(1.0 + e^{margin})} - y$$

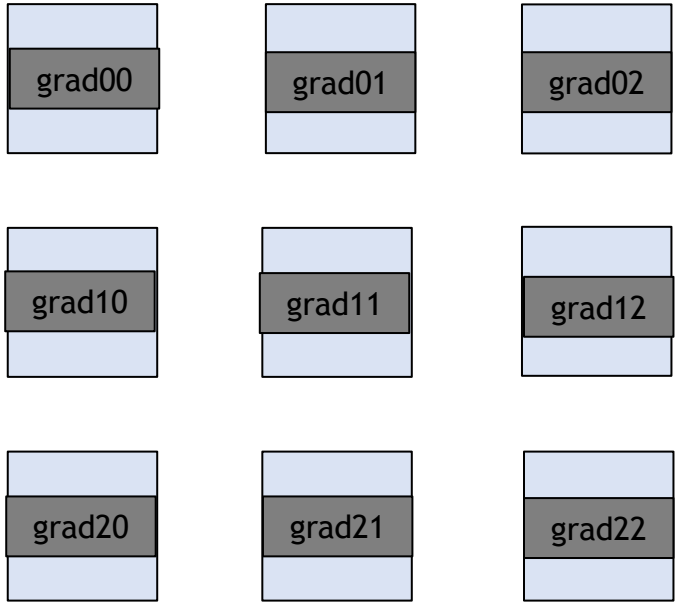

coeff0
coeff1
coeff2
coeff0
coeff1
coeff2
coeff0
coeff1
coeff2

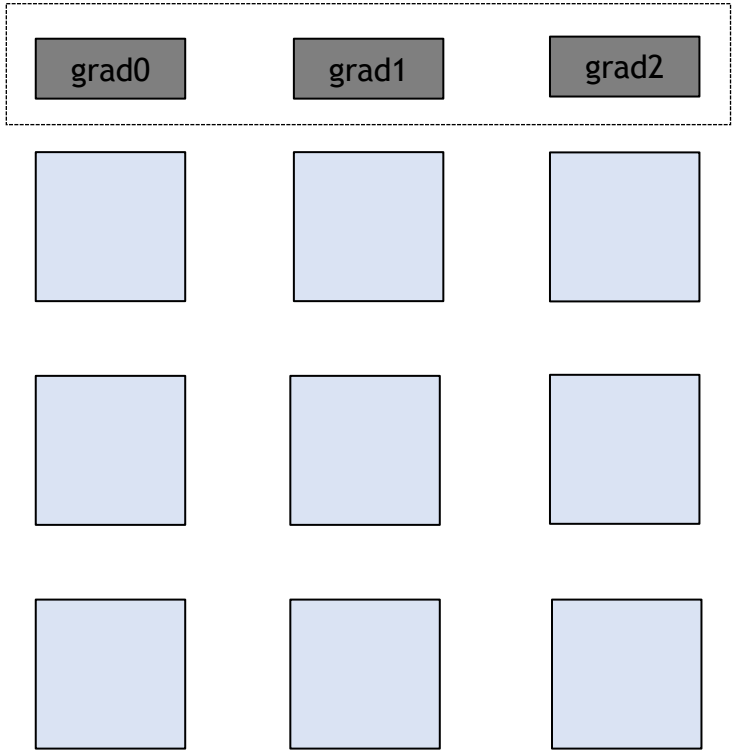

multiplier02
multiplier12
multiplier22
multiplier01
multiplier11
multiplier21
multiplier00
multiplier10
multiplier20


label0
label1
label2
multiplier02
multiplier12
multiplier22
multiplier01
multiplier11
multiplier21
multiplier00
multiplier10
multiplier20


multiplier0
multiplier1
multiplier2


multiplier0
multiplier1
multiplier2
multiplier0
multiplier1
multiplier2
multiplier0
multiplier1
multiplier2


grad00
grad01
grad02
grad10
grad11
grad12
grad20
grad21
grad22

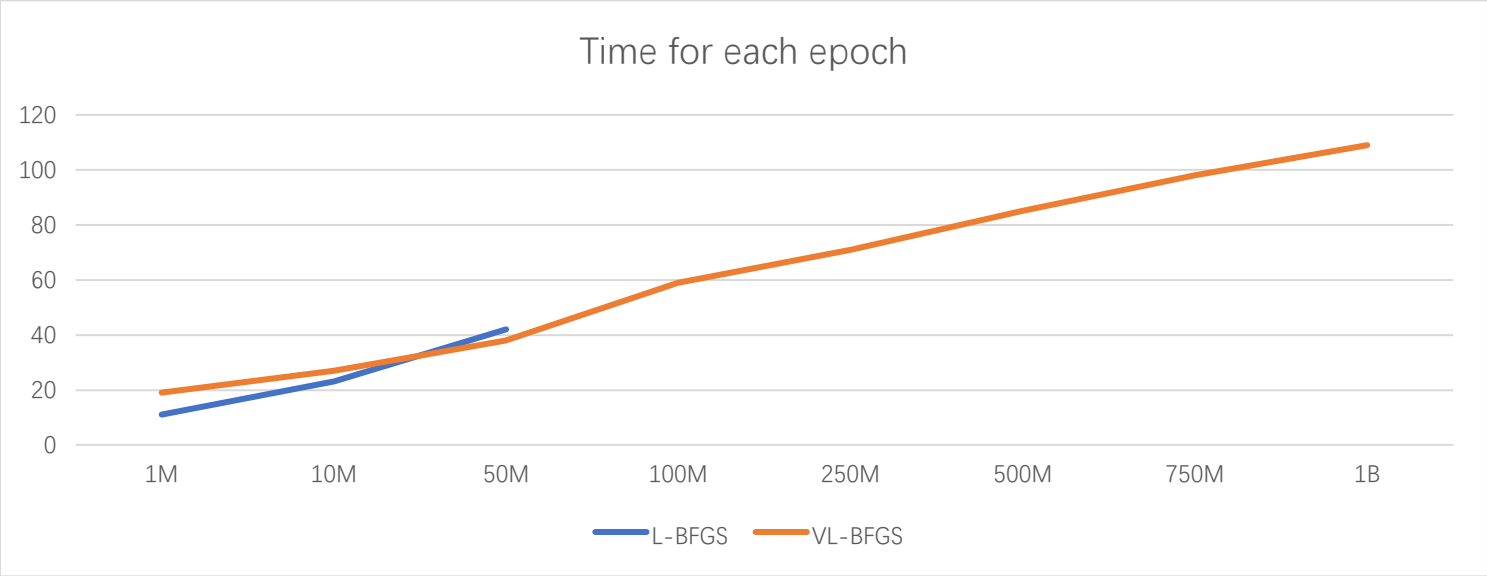


gradient: DistributedVector

Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work

Performance(WIP)



Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work

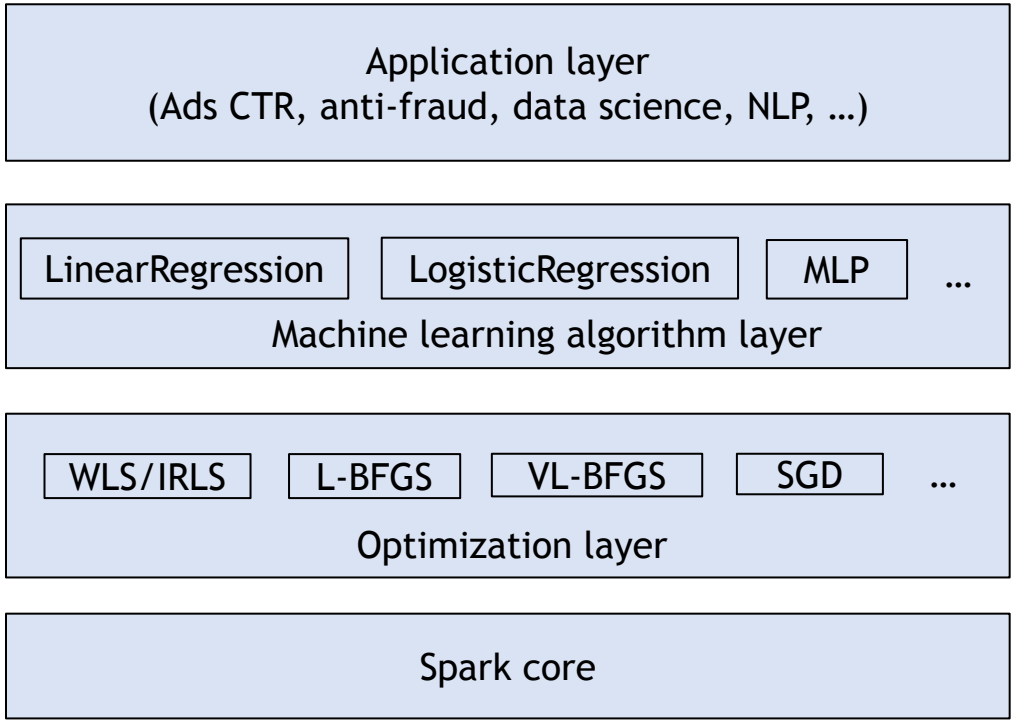
APIs

```
val dataset: Dataset[_] =
      spark.read.format("libsvm").load("data/a9a")
val trainer = new VLogisticRegression()
      .setColsPerBlock(100)
      .setRowsPerBlock(10)
      .setColPartitions(3)
      .setRowPartitions(3)
      .setRegParam(0.5)
val model = trainer.fit(dataset)
println(s"Vector-free logistic regression coefficients:
      ${model.coefficients}")
```

Adaptive logistic regression

Number of features	Optimization method
less than 4096	WLS/IRLS
more than 4096, but less than 10 million	L-BFGS
more than 10 million	VL-BFGS

Whole picture of MLlib



APIs

```
val dataset: Dataset[_] =
      spark.read.format("libsvm").load("data/a9a")
val trainer = new LogisticRegression()
      .setColsPerBlock(100)
      .setRowsPerBlock(10)
      .setColPartitions(3)
      .setRowPartitions(3)
      .setRegParam(0.5)
      .setSolver(“vl-bfgs”)
val model = trainer.fit(dataset)
println(s"Vector-free logistic regression coefficients:
      ${model.coefficients}")
```

Outline

- Background
- Vector-free L-BFGS on Spark
- Logistic regression on vector-free L-BFGS
- Performance
- Integrate with existing MLlib
- Future work

Future work

- Reduce stages for each iteration.
- Performance improvements.
- Implement LinearRegression, SoftmaxRegression and MultilayerPerceptronClassifier base on vector-free L-BFGS/OWL-QN.
- Real word use case for Ads CTR prediction with billions parameters and will share experiences and lessons we learned:
  - https://github.com/yanboliang/spark-vlbfgs

Key takeaways

- Full distributed calculation, full distributed model, can run successfully without OOM.
- VL-BFGS API is consistent with breeze L-BFGS.
- VL-BFGS output exactly the same solution as breeze L-BFGS.
- Does not require special components such as parameter servers.
- Pure library and can be deployed and used easily.
- Can benefit from the Spark cluster operation and development experience.
- Use the same platform as the data size increasing.

Reference

- https://papers.nips.cc/paper/5333-large-scale-l-bfgs-using-mapreduce
- https://github.com/mengxr/spark-vl-bfgs
- https://spark-summit.org/2015/events/large-scale-lasso-and-elastic-net-regularized-generalized-linear-models/

Thank You.