Embrace sparsity at web scale: Apache Spark* MLlib algorithms optimization for sparse data

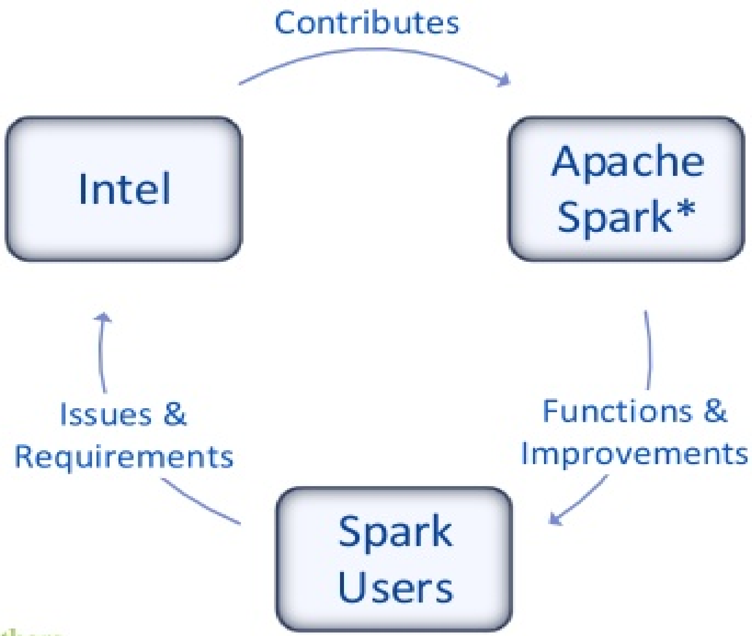
Yuhao Yang / Ding Ding

Big Data Technology, Intel

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Intel & Big Data

- Contribution to big data community
  - Consistently and actively
  - Enthusiastic engineering team
  - https://software.intel.com/en-us/bigdata
- Wide cooperation and partnership
  - Consultations and co-development
  - Send to open source projects.



Sparse data is almost everywhere

- Data Source:
  - Movie ratings
  - Purchase history

- Feature engineering:
  - NLP: CountVectorizer, HashingTF
  - Categorical: OneHotEncoder
  - Image, video

Sparse data support in MLlib

```
new DenseVector(
  values = Array(1.0, 0.0, 0.0, 100.0))

new SparseVector(
  size = 4,
  indices = Array(0, 3),
  values = Array(1.0, 100.0))
```

First Tip: Anther option

- HashVector: a sparse vector backed by a hash array.
  - Mutable Sparse Vector
  - O(1) random access
  - O(nnz) axpy, dot

- Available in Breeze and our package

Sparse data support in MLlib

- Supporting Sparse data since v1.0
  - Load / Save, Sparse Vector, LIBSVM
  - Supporting sparse vector is one of the primary review focus.
  - Xiangrui's talk in Spark Summit 2014: Sparse data support in MLlib
  - https://spark-summit.org/2014/wp-content/uploads/2014/07/sparse_data_support_in_mllib1.pdf

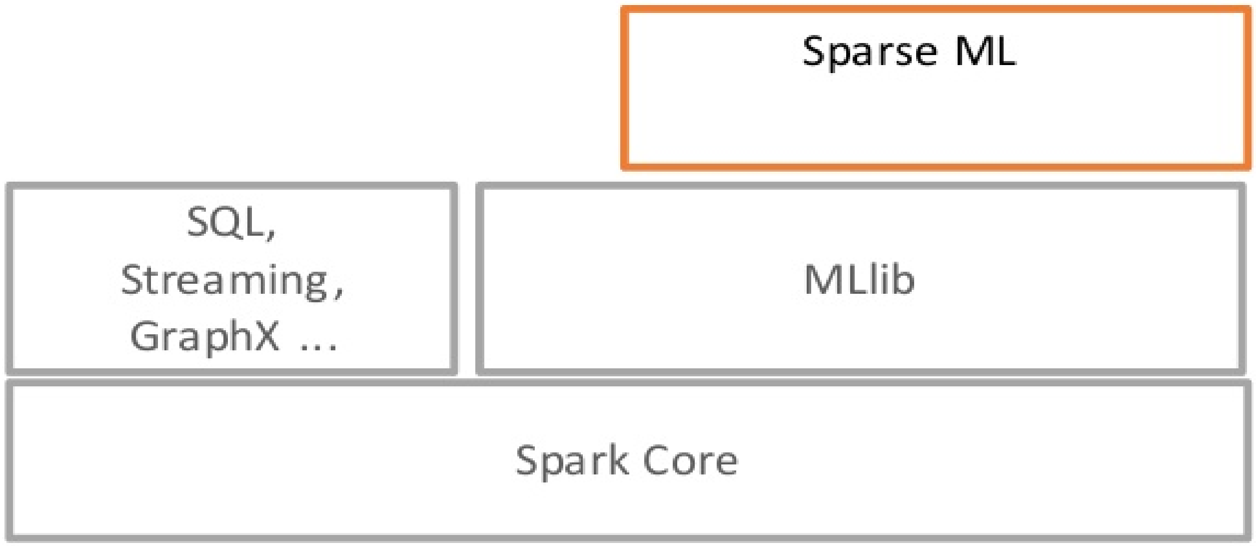
Gaps with some industry scenarios

- Hi, I need
  - LR with 1 billion dimension
  - clustering with 10M dimension
  - Large scale documents classification/clustering
  - My data is quite sparse

- Yet with MLlib
  - OOM…
  - Can you help?

Sparse ML for Apache Spark*

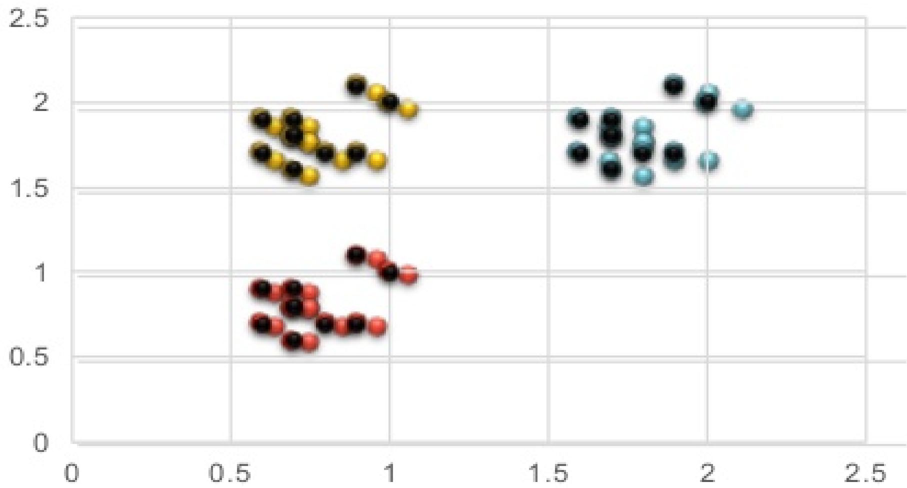
- A Spark package containing algorithm optimization to support the sparse data at large scope



Sparse ML for Apache Spark*

- KMeans
- Linear methods (logistic regression, linear SVM, etc)
- HashVector
- MaxAbsScaler
- NaiveBayes
- Neural Network (WIP)

KMeans



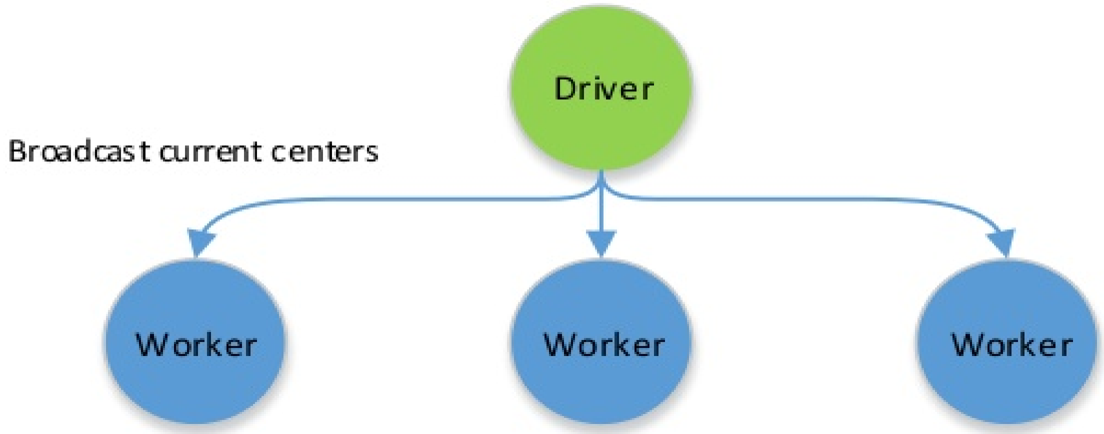
- Pick initial cluster centers
  - Random
  - KMeans||
- Iterative training
  - Points clustering, find nearest center for each point
  - Re-compute center in each cluster (avg.)
- Cluster centers are vectors with the same dimension of data

KMeans scenario: e-commerce

- Cluster customers into 200 clusters according to purchase history:
  - 20M customers
  - 10M different products (feature dimension)
  - 200 clusters
  - Avg. sparsity 1e-6

MLlib iteration

1. Broadcast current centers (all dense vectors, 200 * 10M * 8 = 16G), to all the workers

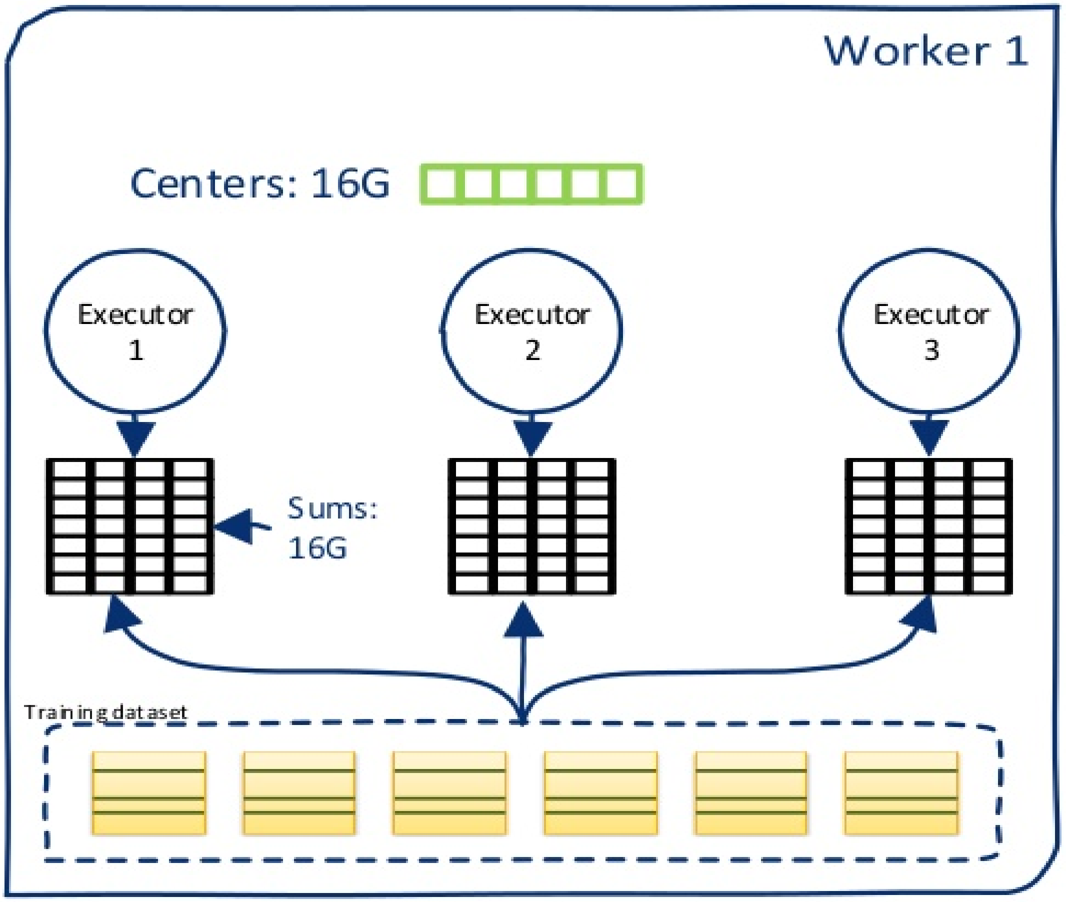


MLlib iteration

2. Compute a sum table

for each partition of data

```
val sum = new Array[Vector](k)
for (each point in the partition) {
    val bestCenter = traverse()
    sum(bestCenter) += point
}
```

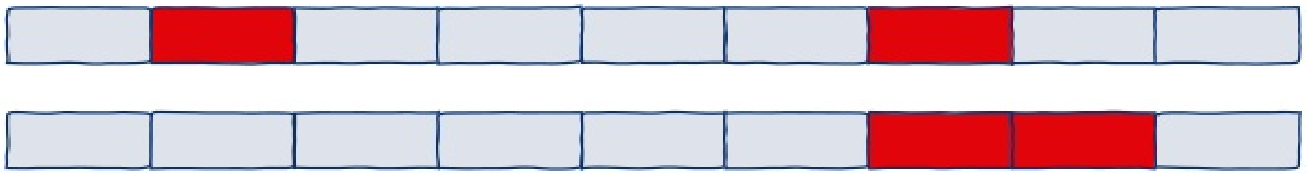


Analysis: Data

- Are the cluster centers dense?

- Let's assume all the records have no overlapping features:
  - 20M records / 200 clusters = 0.1M records per cluster
  - 0.1M * 10 = 1M non-zero in their sum/center
  - 1M / 10M = 0.1 center sparsity at most

Analysis: operations



- Core linear algebra operation:

Operations		Sparse friendly
axpy	Y += A * X	No if Y is sparse, yet X + Y is sparse-friendly
dot	X dot Y	Yes
Sqdist	Square distance	Yes, sparse faster

SparseKMeans

- Represent clustering centers with SparseVector
  - Reduce memory and time consumption

Cluster centers

- What a center goes through in each iteration
  - Broadcast
  - Compute distance with all the points (sqdist , dot)
  - Discard (New centers are generated)

- Cluster centers can always use SparseVector
  - Without extra cost during computation

Advanced: Sum table

- Use SparseVectors to hold the sum for each cluster
  - Reduce max memory requirement;

- Isn't it slower to compute with Sparse vectors?
  - SparseVector can not support axpy, but it supports x + y
  - Modern JVM handles small objects efficiently
  - Automatically converts to DenseVector (sparseThrehold)

Scalable KMeans

- What if you cluster centers are dense
  - Reduce max memory consumption
  - Break the constraint imposed by centers and sums

- Can we make the centers distributed?
  - Array[Center] => RDD[Center]
  - Each point vs. each cluster center.
  - That sounds like a join

Scalable KMeans

```
val pointWithCenter = data.cartesian(centers).map { case (point, center) =>
  (point, (center, ScalableKMeans.fastSquaredDistance(point, center)))
}.reduceByKey { case((c1, d1), (c2, d2)) =>
  if(d1 < d2) (c1, d1) else (c2, d2)
}

val sumByCenter = pointWithCenter.map { case (point, (center, dist)) =>
  (center, (point.vector, 1L))
}.reduceByKey(mergeContribs)
```

Scalable KMeans

- Scalable
  - No broadcast, no sum table
  - 200G -> 20G * 10
  - Remove memory constraint on single node
- Not only for Sparse data

KMeans

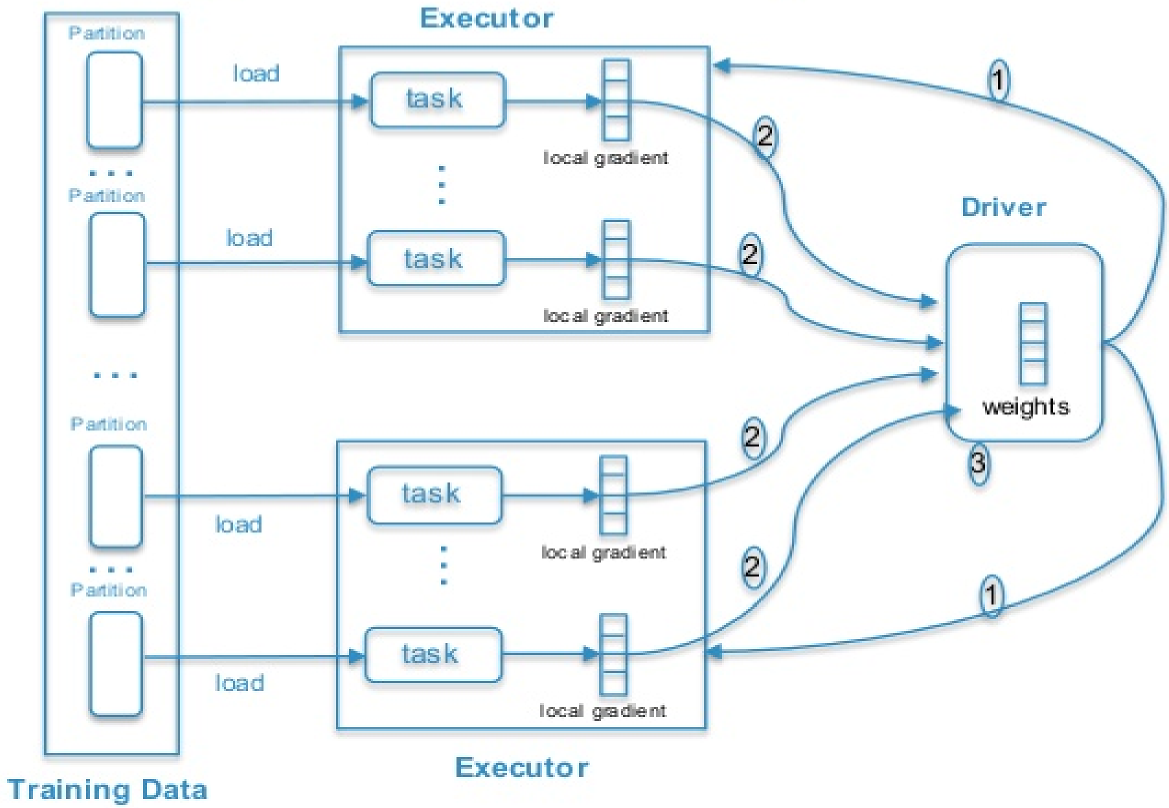
- Sparse KMeans:
  - Cluster centers can be sparse:
- Scalable KMeans
  - Cluster centers can be distributed

Tip2: MaxAbsScaler for feature engineering

- MinMaxScaler destroys data sparsity

- StandardScaler does not support SparseVector withMean

Logistic Regression on Spark

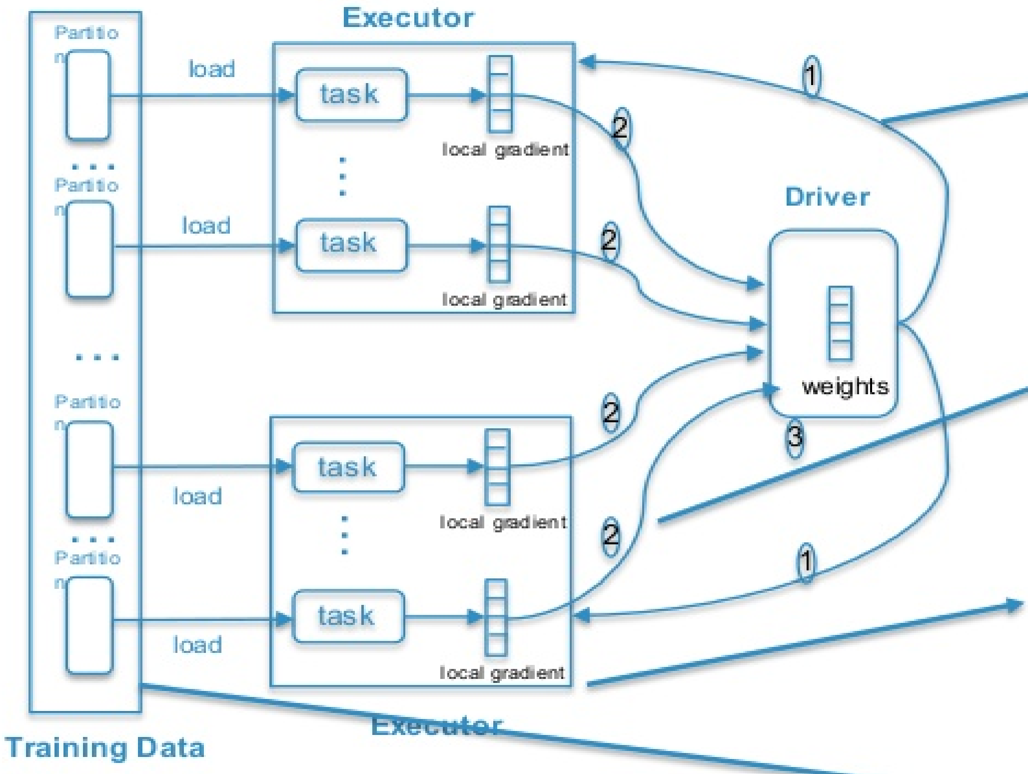


Large Scale Logistic Regression

Customer's training set:

- Number of features : 200s million
- Billions ~ trillions training samples
- Each sample has 100s – 1000 non-zero elements

Challenges: big data and big model



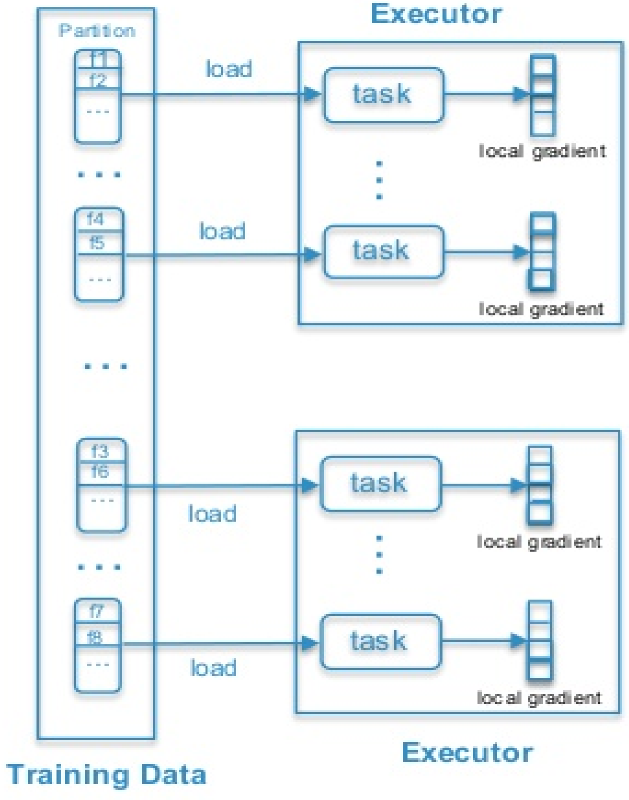
Driver broadcast weights(>2G) to each executor

Each task sends local gradient(>2G) for aggregation

Gradients in each executor (>>2G)

Training data cached in executor

Exploiting sparsity in gradients



$$g(w; x, y) = f(x^T w; y) \cdot x$$

The gradient is sparse as the feature vector is sparse

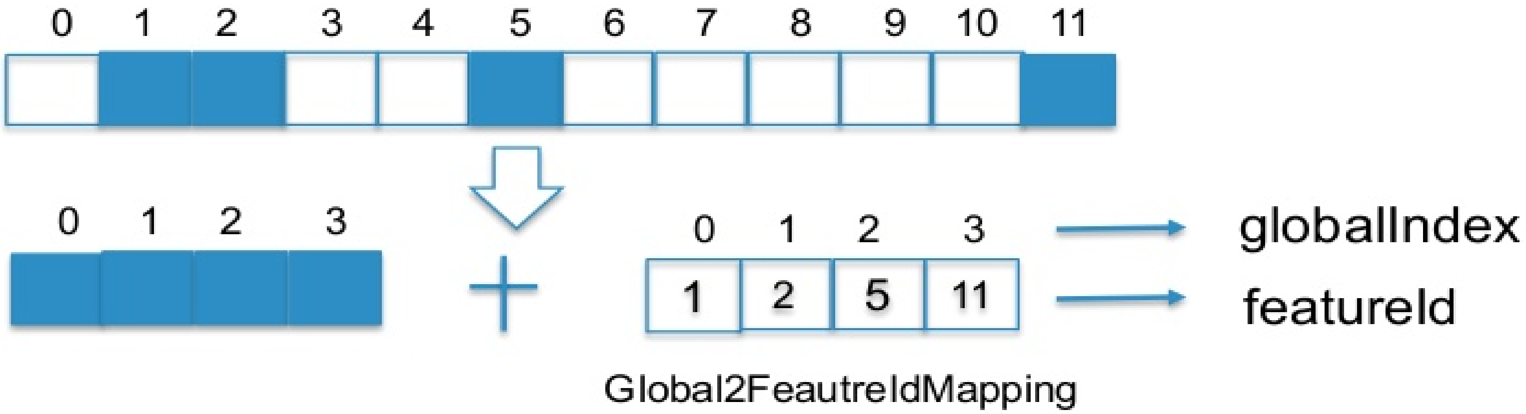
Switch to sparse gradients

- g = points.map(p => grad(w, p)).reduce(_ + _)
- Gradients: hashSparseVector
- Adds gradients to an initial hashSparseVector :
  - ✓ Fast random access: O(1)
  - ✓ Memory friendly:

    Executor: 10G -> ~200M

Exploiting sparsity in weights

- Weights is with great sparsity
  - Waste memory on meaningless 0
  - Use dense vector with non zero elements



Prune weights

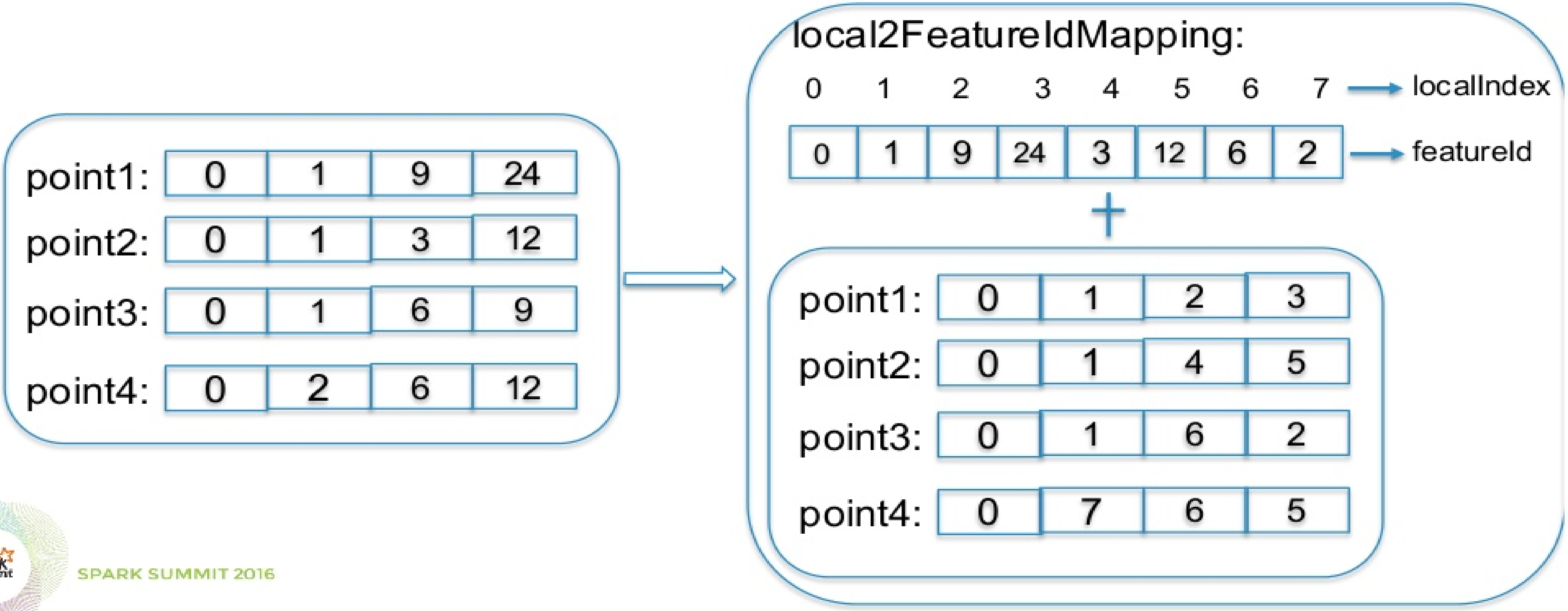
- Implementation:

```
val global2FeatureIdMapping =
  points.mappartition {p => p.mapping}.collect().flatMap(t => t).distinct
```

- GlobalIndex is used during traing
- Convert back to featureId after training

Optimize cached training data

- Use localIndex as sparse vector indices



Optimize cached training data

- Encode localIndex

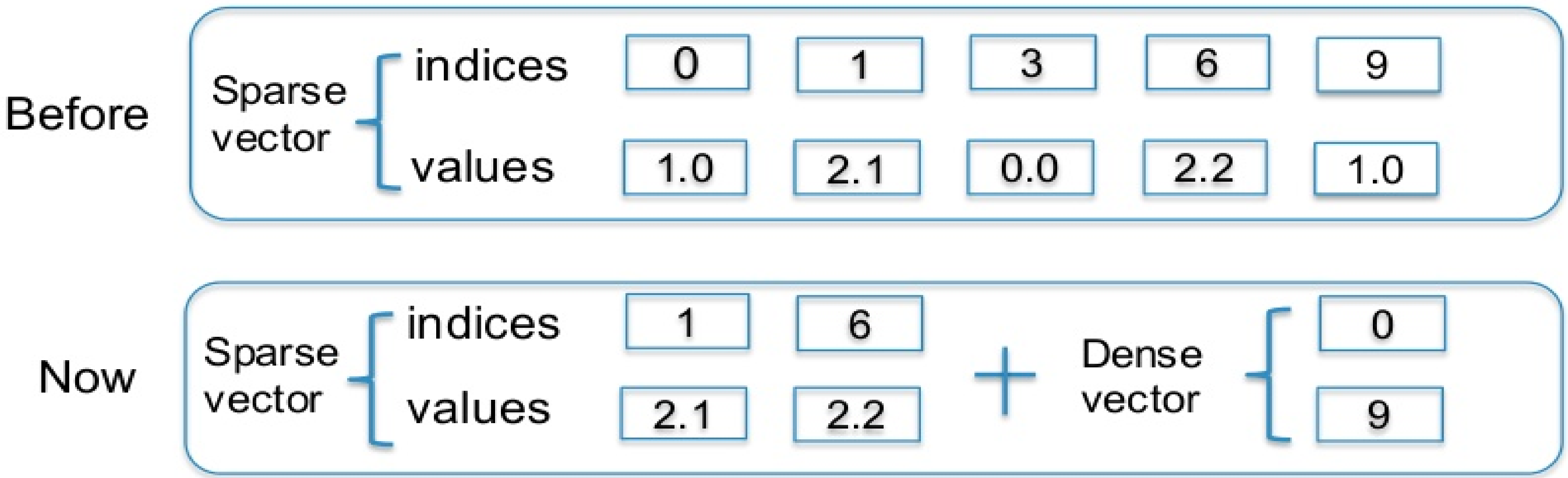
  featureId: 0 – 200 millions   localIndex: 0 – ~2 millions

  Use 1-3 bytes to store localIndex

  - indices: Array[Int] -> ~~Array[Array[Byte]]~~ -> Array[Byte]
  - use first bit to identify if the following byte is a new localIndex

Optimize cached training data

- Support for binary(0 or 1) values



Sparse Logistic Regression Performance

- Enviroment (12 executors with 8g memory in each)
  - ❑ Spark LR: OOM
  - ❑ Sparse LR: 90 seconds per epoch

Hardware： Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz, 128GB DRAM
Software： Spark on yarn (Spark ver1.6.0 , Hadoop ver2.6.0)

How to use SparseSpark

- https://github.com/intel-analytics/SparseSpark
- Consistent interface with MLlib
- Compile with application code.

THANK YOU.

Yuhao.yang@intel.com
Ding.ding@intel.com

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Legal Disclaimer

Software