Scaling Machine Learning To Billions of Parameters

Badri Bhaskar, Erik Ordentlich

(joint with Andy Feng, Lee Yang, Peter Cnudde)

Yahoo, Inc.

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

Outline

- Large scale machine learning (ML)
- Spark + Parameter Server
  – Architecture
  – Implementation
- Examples:
  - Distributed L-BFGS (Batch)
  - Distributed Word2vec (Sequential)
- Spark + Parameter Server on Hadoop Cluster

LARGE SCALE ML

Web Scale ML

Big Model

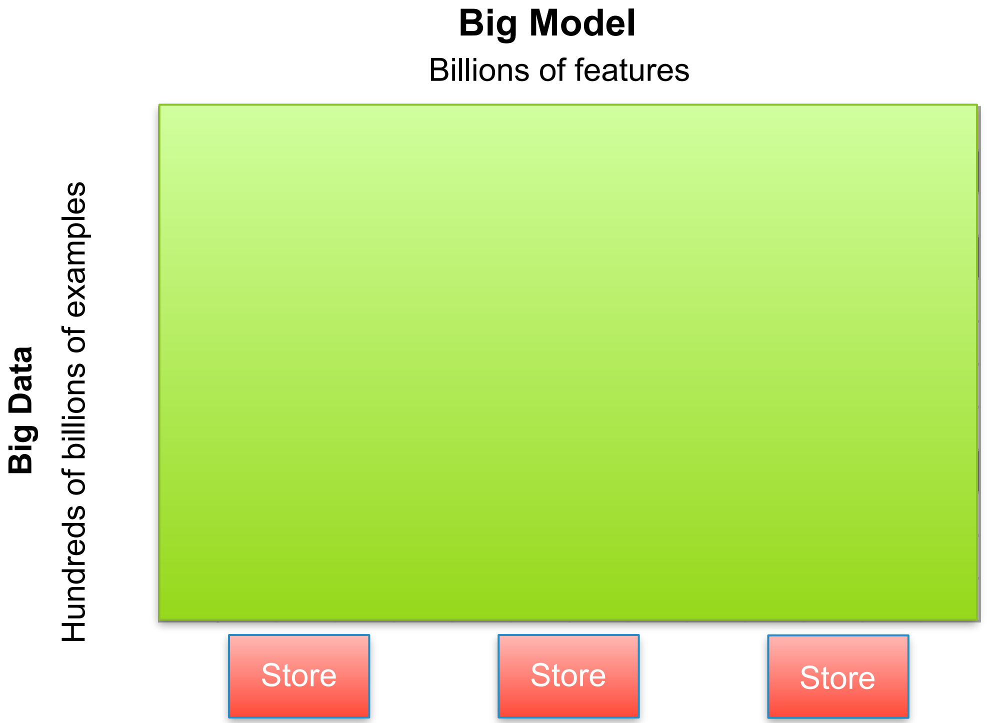
Billions of features

Big Data

Hundreds of billions of examples

Ex: Yahoo word2vec - 120 billion parameters and 500 billion samples

Web Scale ML

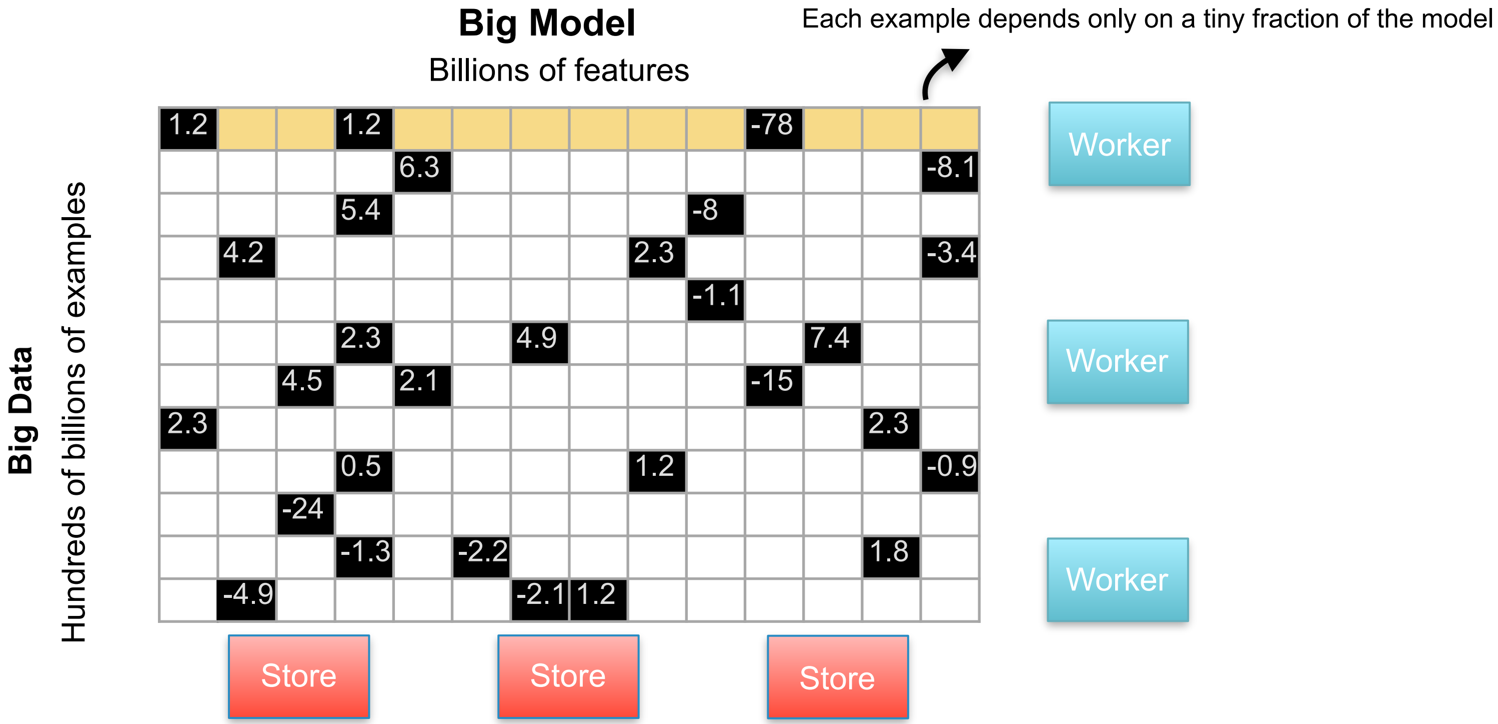


Ex: Yahoo word2vec - 120 billion parameters and 500 billion samples

Web Scale ML

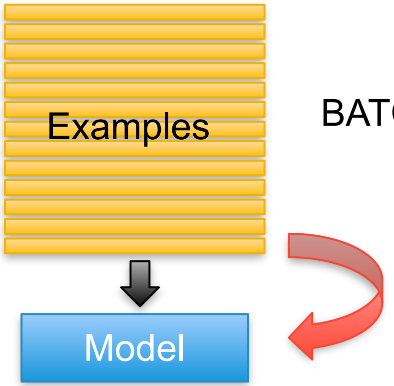
Ex: Yahoo word2vec - 120 billion parameters and 500 billion samples

Web Scale ML



Ex: Yahoo word2vec - 120 billion parameters and 500 billion samples

Two Optimization Strategies



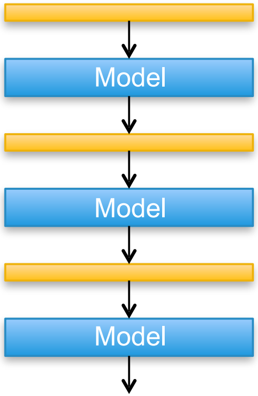
BATCH

Multiple epochs…

Example: Gradient Descent, L-BFGS

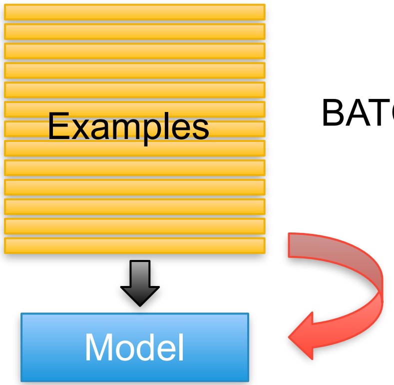
SEQUENTIAL

Multiple random samples…



Example: (Minibatch) stochastic gradient method, perceptron

Two Optimization Strategies

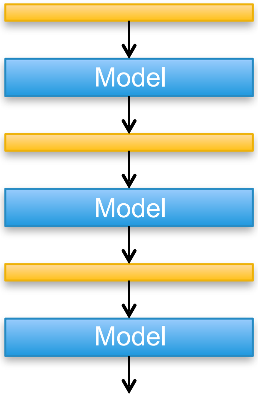


BATCH

Multiple epochs…

Example: Gradient Descent, L-BFGS

- Small number of model updates
- Accurate
- Each epoch may be expensive.
- *Easy to parallelize.*

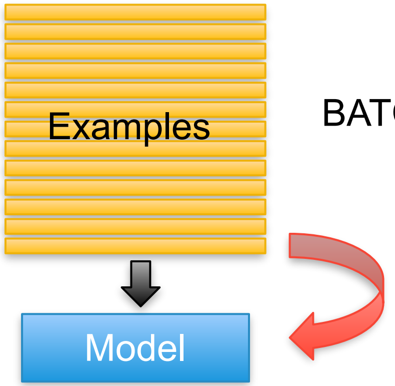


SEQUENTIAL

Multiple random samples…

Example: (Minibatch) stochastic gradient method, perceptron

Two Optimization Strategies



BATCH

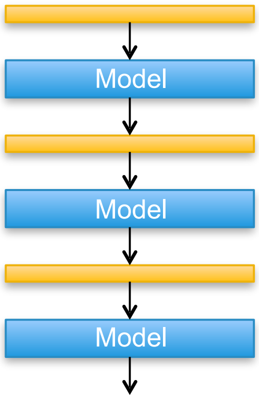
Multiple epochs…

Example: Gradient Descent, L-BFGS

- Small number of model updates
- Accurate
- Each epoch may be expensive.
- *Easy to parallelize.*

SEQUENTIAL

Multiple random samples…



Example: (Minibatch) stochastic gradient method, perceptron

- Requires lots of model updates.
- Not as accurate, but often good enough
- A lot of progress in one pass* for big data.
- *Not trivial to parallelize.*

*also optimal in terms of generalization error (often with a lot of tuning)

Requirements

Requirements

✓ Support both **batch** and **sequential** optimization

Requirements

✓ Support both **batch** and **sequential** optimization

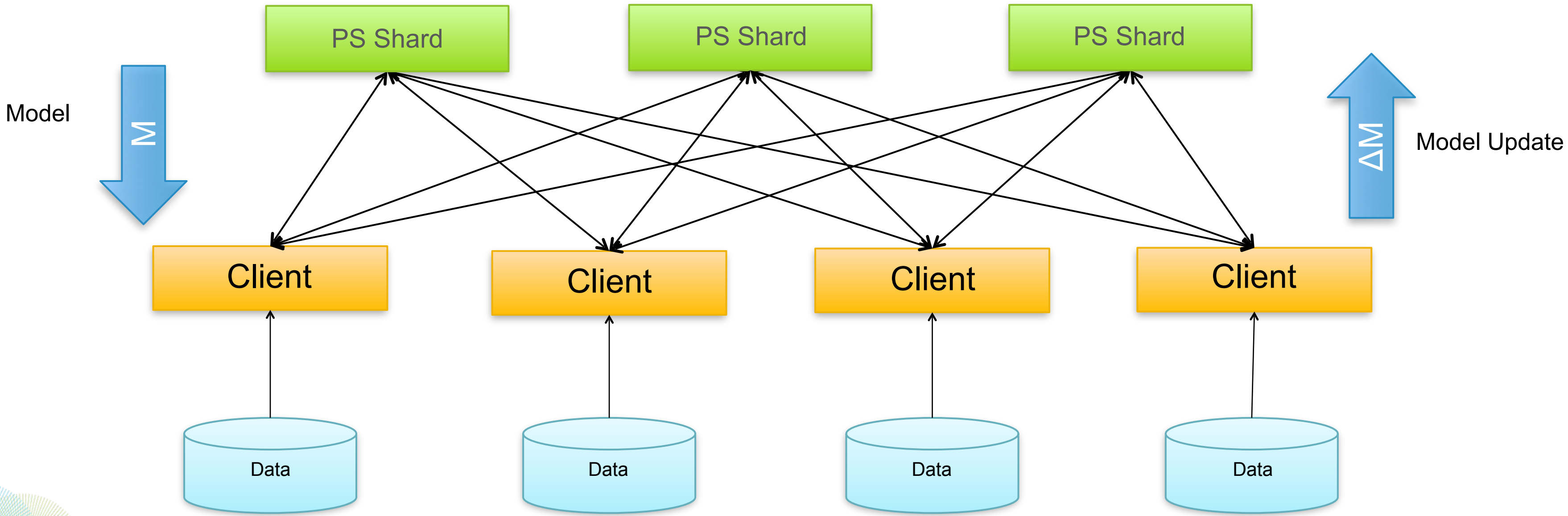
✓ **Sequential training**: Handle frequent updates to the model

Requirements

✓ Support both **batch** and **sequential** optimization
✓ **Sequential training**: Handle <u>frequent updates</u> to the model
✓ **Batch training**: 100+ passes ⇒ each pass must be <u>fast</u>**.**

Parameter Server (PS)

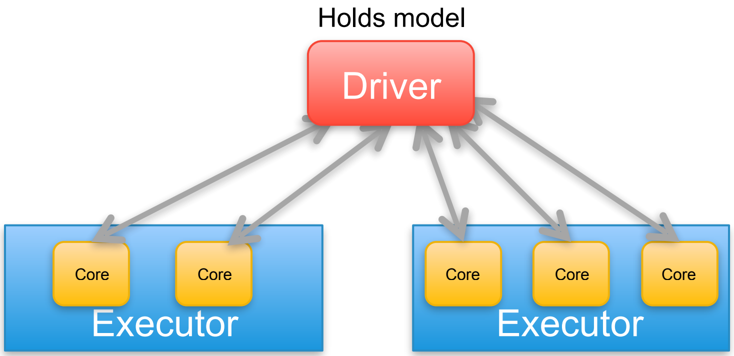
Training state stored in PS shards, asynchronous updates



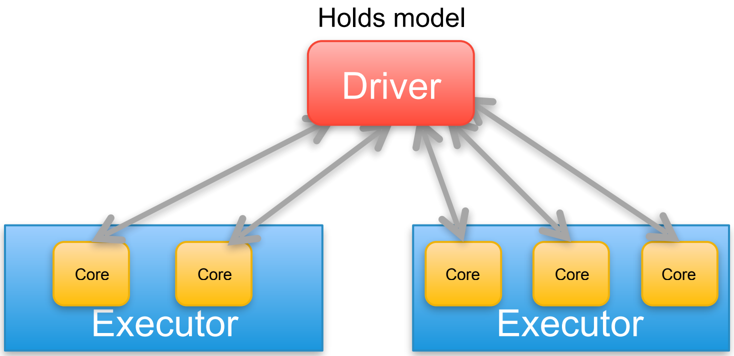
Early work: Yahoo LDA by Smola and Narayanamurthy based on memcached (2010), Introduced in Google's Distbelief (2012), refined in Petuum / Bösen (2013), Mu Li et al (2014)

SPARK + PARAMETER SERVER

ML in Spark alone



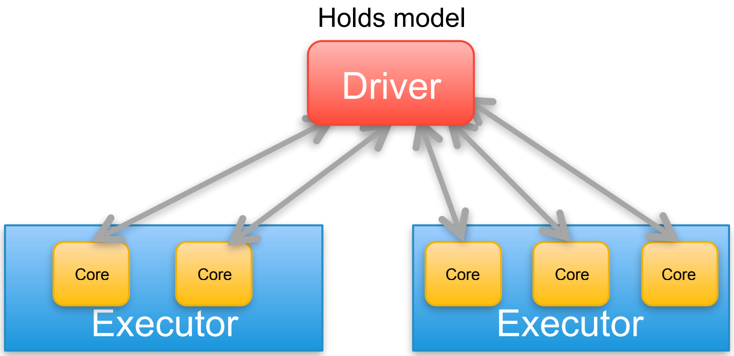
ML in Spark alone




```
def train(data: RDD[Example]) = {
    while (not_converged) {
        broadcast(model)
        val cumGradient = data.sample().treeAggregate(...)
        model.update(cumGradient)
    }
}
```

MLlib optimization

ML in Spark alone

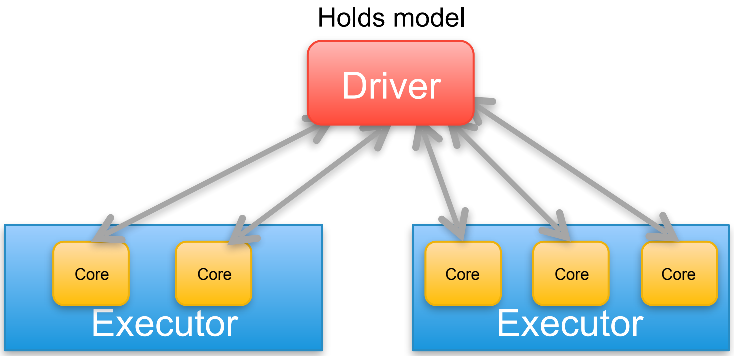



```
def train(data: RDD[Example]) = {
    while (not_converged) {
        broadcast(model)
        val cumGradient = data.sample().treeAggregate(...)
        model.update(cumGradient)
    }
}
```

MLlib optimization

- Sequential:
  - Driver-based communication limits frequency of model updates.
  - Large minibatch size limits model update frequency, convergence suffers.

ML in Spark alone



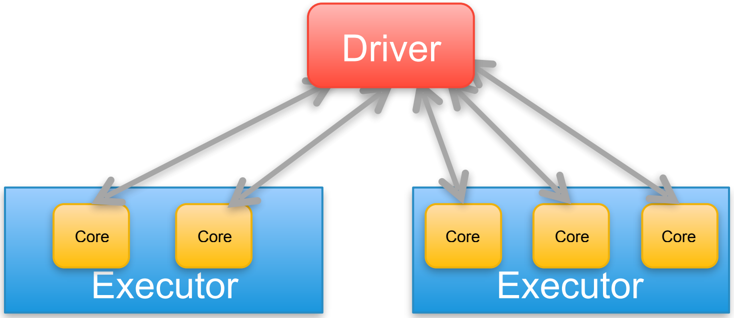

```
def train(data: RDD[Example]) = {
    while (not_converged) {
        broadcast(model)
        val cumGradient = data.sample().treeAggregate(...)
        model.update(cumGradient)
    }
}
```

MLlib optimization

- Sequential:
  - Driver-based communication limits frequency of model updates.
  - Large minibatch size limits model update frequency, convergence suffers.
- Batch:
  - Driver bandwidth can be a bottleneck
  - Synchronous stage wise processing limits throughput.

ML in Spark alone

Holds model




```scala
def train(data: RDD[Example]) = {
    while (not_converged) {
        broadcast(model)
        val cumGradient = data.sample().treeAggregate(...)
        model.update(cumGradient)
    }
}
```

MLlib optimization

- Sequential:
  - Driver-based communication limits frequency of model updates.
  - Large minibatch size limits model update frequency, convergence suffers.
- Batch:
  - Driver bandwidth can be a bottleneck
  - Synchronous stage wise processing limits throughput.

PS Architecture circumvents both limitations…

Spark + Parameter Server

Spark + Parameter Server

- Leverage Spark for HDFS I/O, distributed processing, fine-grained load balancing, failure recovery, in-memory operations

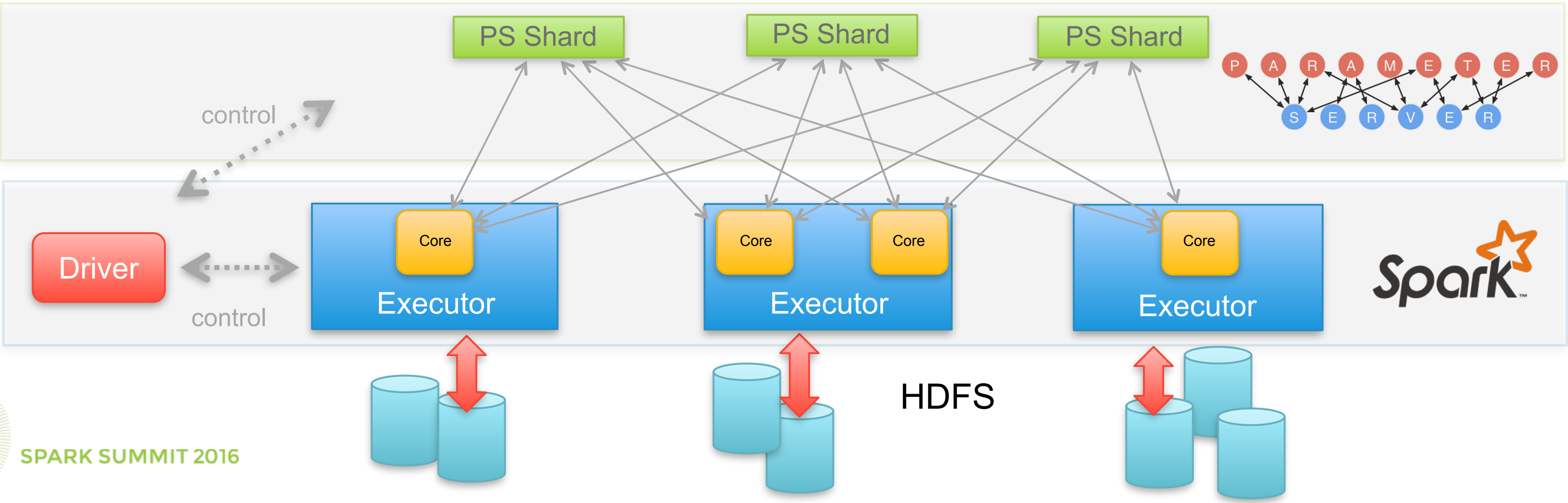
Spark + Parameter Server

- Leverage Spark for HDFS I/O, distributed processing, fine-grained load balancing, failure recovery, in-memory operations
- Use PS to sync models, incremental updates during training, or sometimes even some vector math.

Spark + Parameter Server

- Leverage Spark for HDFS I/O, distributed processing, fine-grained load balancing, failure recovery, in-memory operations
- Use PS to sync models, incremental updates during training, or sometimes even some vector math.

Training state stored in PS shards



Yahoo PS

Yahoo PS

Server

Client API

Yahoo PS

Java Netty | Server

Scala | Client API

~~GC~~ | Preallocated arrays

Yahoo PS

Java Netty | **Server**

Scala | **Client API**

~~GC~~ | Preallocated arrays

K-V

- In-memory
- Lock per key / Lock-free
- Sync / Async

Yahoo PS

Java Netty | **Server**

Scala | **Client API**

~~GC~~ | Preallocated arrays

K-V

- In-memory
- Lock per key / Lock-free
- Sync / Async

$$\begin{bmatrix} 9 & 13 & 5 & 2 \\ 1 & 11 & 7 & 6 \\ 3 & 7 & 4 & 1 \\ 6 & 0 & 7 & 10 \end{bmatrix}$$

- Column-partitioned
- Supports BLAS

Yahoo PS

Java Netty — **Server**

Scala — **Client API**

~~GC~~ — Preallocated arrays

K-V

- In-memory
- Lock per key / Lock-free
- Sync / Async

$$\begin{bmatrix} 9 & 13 & 5 & 2 \\ 1 & 11 & 7 & 6 \\ 3 & 7 & 4 & 1 \\ 6 & 0 & 7 & 10 \end{bmatrix}$$

- Column-partitioned
- Supports BLAS

HDFS

- Export Model
- Checkpoint

Yahoo PS

Java Netty | **Server**

Scala | **Client API**

~~GC~~ | Preallocated arrays

K-V

- In-memory
- Lock per key / Lock-free
- Sync / Async

$$\begin{bmatrix} 9 & 13 & 5 & 2 \\ 1 & 11 & 7 & 6 \\ 3 & 7 & 4 & 1 \\ 6 & 0 & 7 & 10 \end{bmatrix}$$

- Column-partitioned
- Supports BLAS

HDFS

- Export Model
- Checkpoint

UDF

- Client supplied aggregation
- Custom shard operations

Map PS API

```scala
trait MapClient[K,V] {
  def get(key: K) : Future[V]
  def put(key: K, value: V) : Future[Unit]

  def multiGet(keys: Seq[K]) : Future[Map[K,V]]
  def multiPut(keyValue: Seq[(K, V)]) : Future[Int]

  def mapReduce[T,U](zero: U, mapFunc: T => U, reduceFunc: (U,U) => U) : Future[U]
}
```

- Distributed key-value store abstraction
- Supports batched operations in addition to usual get and put
- Many operations return a future – you can operate asynchronously or block

Matrix PS API

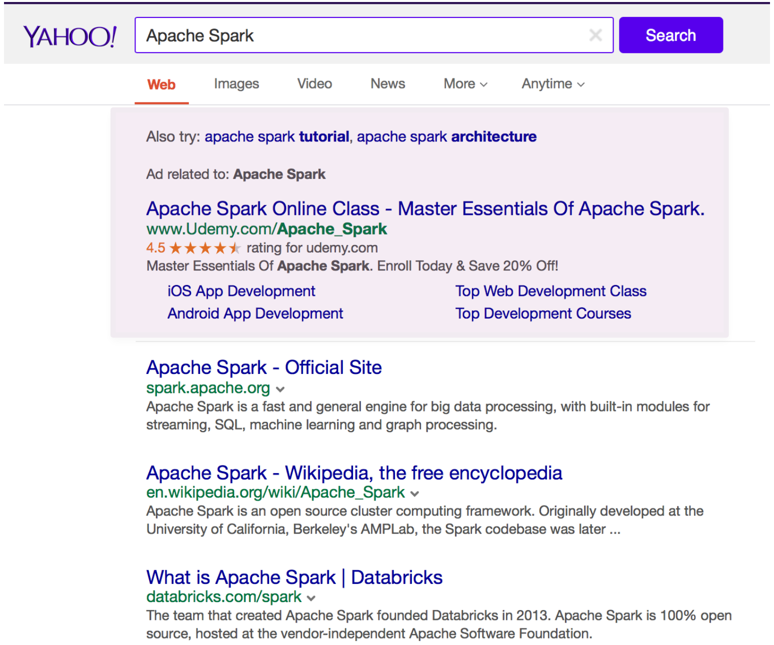
```
trait MatrixClient extends MapClient[Int, Array[Float]] {
  def dot(x: Int, y: Int): Float
  def scal(row: Int, factor: Float) : Future[Unit]
  def axpy(a: Float, x: Int, y: Int) : Future[Unit]
  def copy(to: Int, from: Int) : Future[Unit]
  ...

  def increment(x: Int, indices: Array[Int], values: Array[Int]) : Future[Unit]
  def fetch(x: Int, indices: Array[Int]) : Array[Float]
}
```

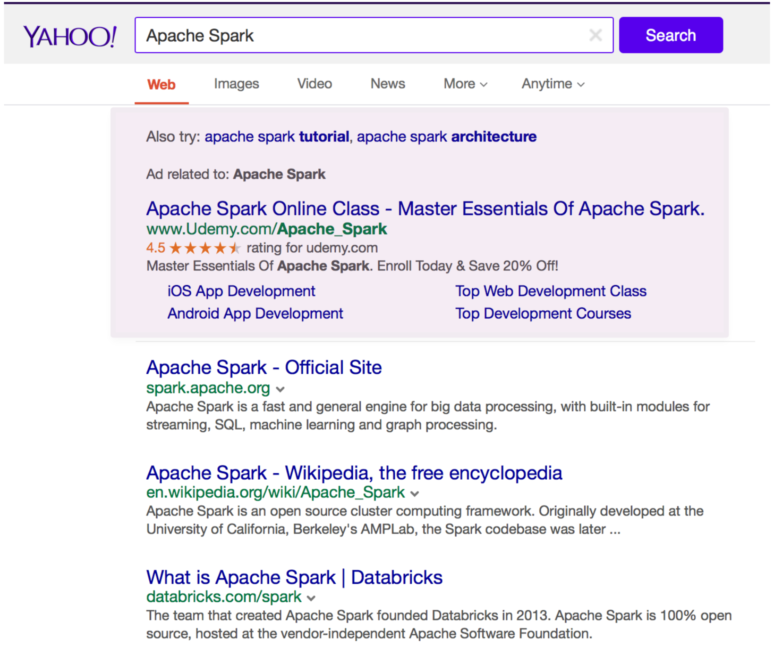
- Vector math (BLAS style operations), in addition to everything Map API provides
- Increment and fetch sparse vectors (e.g., for gradient aggregation)
- We use other custom operations on shard (API not shown)

EXAMPLES

Sponsored Search Advertising

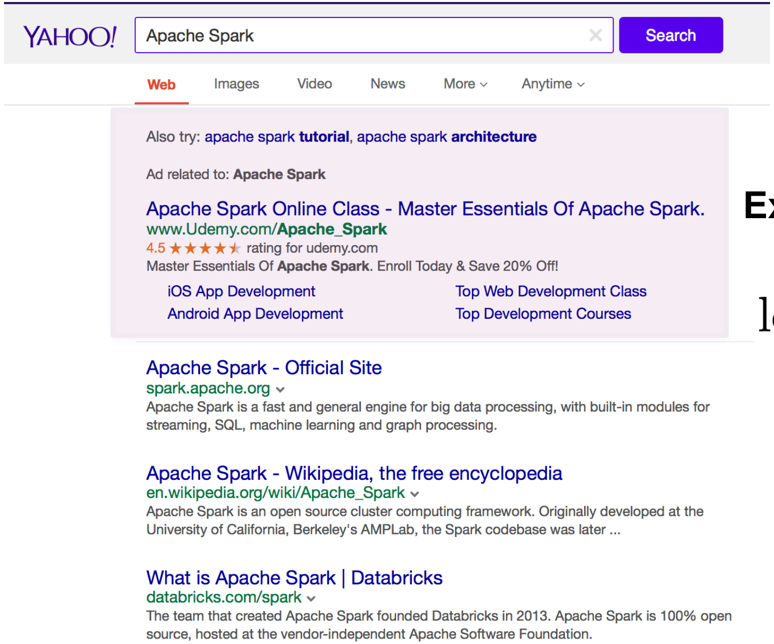


Sponsored Search Advertising



$$\text{cost per impression} = \text{advertiser bid} \times \text{click probability}$$

Sponsored Search Advertising



Example Click Model: (Logistic Regression)

Features

$$\log\left(\frac{p(y=1|q,u,a,C)}{p(y=0|q,u,a,C)}\right) = \boxed{w}^{\top}\boxed{f(q,u,a,C)}$$

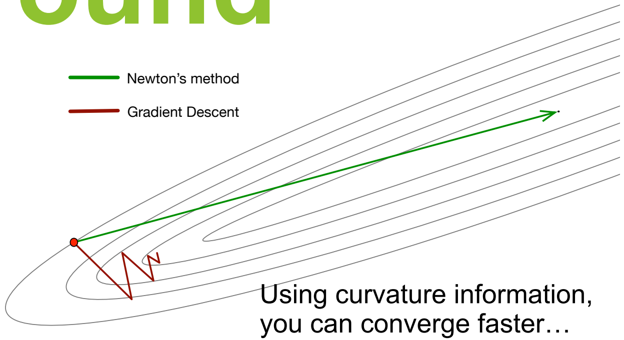
Model

query user ad context

$$\text{cost per impression} = \text{advertiser bid} \times \boxed{\text{click probability}}$$

L-BFGS Background

L-BFGS Background



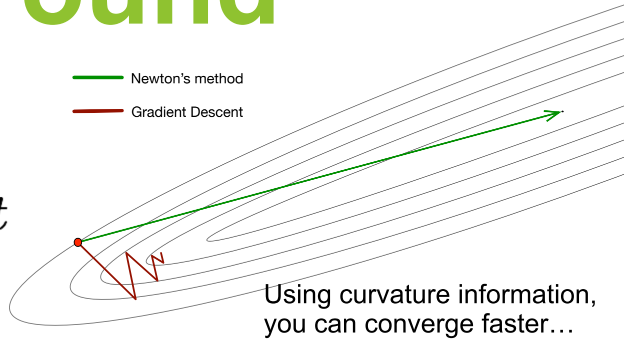
Using curvature information,
you can converge faster…

L-BFGS Background

in $\mathbb{R}^d$

Exact, impractical

$$w_{t+1} \leftarrow w_t - \boxed{H_t^{-1}} g_t$$



Using curvature information, you can converge faster…

$d \times d$ matrix of partial derivatives

$\mathcal{O}\left(d^3\right)$ to invert!

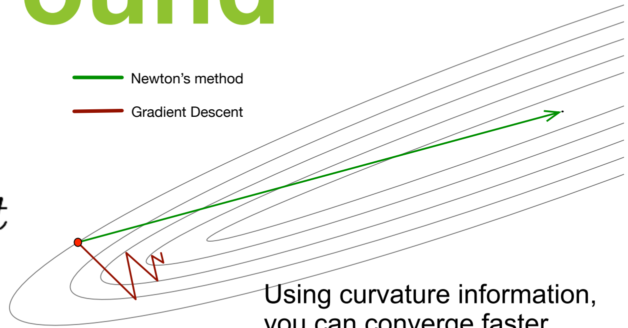
L-BFGS Background

Exact, impractical

$$w_{t+1} \leftarrow \underbrace{w_t}_{\text{in } \mathbb{R}^d} - \boxed{H_t^{-1}} g_t$$

$d \times d$ matrix of partial derivatives

$\mathcal{O}\left(d^3\right)$ to invert!



Using curvature information, you can converge faster…

Approximate, practical

$$w_{t+1} \leftarrow w_t - \gamma_t \tilde{H}_{\text{inv}}\left(g_t\right)$$

Step Size computation

- Needs to satisfy some technical (Wolfe) conditions
- Adaptively determined from data

Inverse Hessian Approximation
(based on history of L-previous gradients and model deltas)

L-BFGS Ba

REQUIRE: State vectors $M = (\{s_i\}_{i=t-1}^{t-m}, \{y_i\}_{i=t-1}^{t-m}$

OUTPUT: Proposed search direction

function $H_{\text{inv}}(g_t)$

$q \leftarrow g_t$

for $i = t-1, t-2, \ldots, t-m$ **do**

$\alpha_i \leftarrow \rho_i s_i^\top q$

$q \leftarrow q - \alpha_i y_i$

end for

$\gamma_t \leftarrow s_{t-1}^\top y_{t-1} / y_{t-1}^\top y_t$

$r \leftarrow \gamma_t q$

for $i = t-m, t-m+1, \ldots, t-1$ **do**

$\beta \leftarrow \rho_i y_i^\top r$

$r \leftarrow r + s_i(\alpha_i - \beta)$

end for

in

Exact, impractical

$$w_{t+1} \leftarrow w_t -$$

Approximate, practical

$$w_{t+1} \leftarrow w_t - \boxed{\gamma_t} \boxed{\tilde{H}_{\text{inv}}(g_t)}$$

Step Size computation

- Needs to satisfy some technical (Wolfe) conditions
- Adaptively determined from data

Inverse Hessian Approximation

(based on history of L-previous gradients and model deltas)

L-BFGS Ba

in

Exact, impractical $w_{t+1} \leftarrow w_t$ -

REQUIRE: State vectors $M = (\{s_i\}_{i=t-1}^{t-m}, \{y_i\}_{i=t-1}^{t-m})$

OUTPUT: Proposed search direction

function $H_{\text{inv}}(g_t)$

$q \leftarrow g_t$

for $i = t-1, t-2, \ldots, t-m$ **do**

$\alpha_i \leftarrow \rho_i s_i^\top q$

$q \leftarrow q - \alpha_i y_i$

end for

$\gamma_t \leftarrow s_{t-1}^\top y_{t-1} / y_{t-1}^\top y_t$

$r \leftarrow \gamma_t q$

for $i = t-m, t-m+1, \ldots, t-1$ **do**

$\beta \leftarrow \rho_i y_i^\top r$

$r \leftarrow r + s_i(\alpha_i - \beta)$

end for

Approximate, practical $w_{t+1} \leftarrow w_t - \gamma_t \tilde{H}_{\text{inv}}(g_t)$

Step Size computation

- Needs to satisfy some technical (Wolfe) conditions
- Adaptively determined from data

Inverse Hessian Approximation
(based on history of L-previous gradients and model deltas)

L-BFGS Ba

Exact, impractical

$$w_{t+1} \leftarrow w_t -$$

REQUIRE: $\boxed{\text{State vectors } M = (\{s_i\}_{i=t-1}^{t-m}, \{y_i\}_{i=t-1}^{t-m}}$

OUTPUT: Proposed search direction

	Vector Math
function $H_{\text{inv}}(g_t)$	
$\quad q \leftarrow g_t$	`copy`
$\quad$ **for** $i = t-1, t-2, \ldots, t-m$ **do**	
$\quad\quad \alpha_i \leftarrow \rho_i s_i^\top q$	`dotprod`
$\quad\quad q \leftarrow q - \alpha_i y_i$	`axpy (y ← ax + y)`
$\quad$ **end for**	
$\quad \gamma_t \leftarrow s_{t-1}^\top y_{t-1} / y_{t-1}^\top y_t$	`dotprod`
$\quad r \leftarrow \gamma_t q$	`scal`
$\quad$ **for** $i = t-m, t-m+1, \ldots, t-1$ **do**	
$\quad\quad \beta \leftarrow \rho_i y_i^\top r$	`axpy`
$\quad\quad r \leftarrow r + s_i(\alpha_i - \beta)$	`scal`
$\quad$ **end for**	

Approximate, practical

$$w_{t+1} \leftarrow w_t - \boxed{\gamma_t} \boxed{\tilde{H}_{\text{inv}}(g_t)}$$

Step Size computation

- Needs to satisfy some technical (Wolfe) conditions
- Adaptively determined from data

Inverse Hessian Approximation

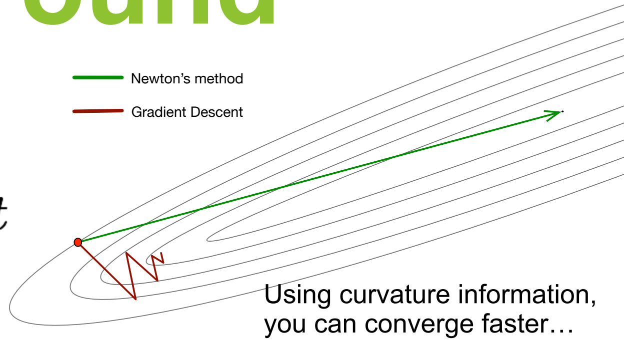
(based on history of L-previous gradients and model deltas)

L-BFGS Background

in $\mathbb{R}^d$

Exact, impractical

$$w_{t+1} \leftarrow w_t - \boxed{H_t^{-1}} g_t$$



Using curvature information, you can converge faster…

$d \times d$ matrix of partial derivatives

$\mathcal{O}\left(d^3\right)$ to invert!

Approximate, practical

$$w_{t+1} \leftarrow w_t - \gamma_t \tilde{H}_{\text{inv}}\left(g_t\right)$$

Step Size computation

- Needs to satisfy some technical (Wolfe) conditions
- Adaptively determined from data

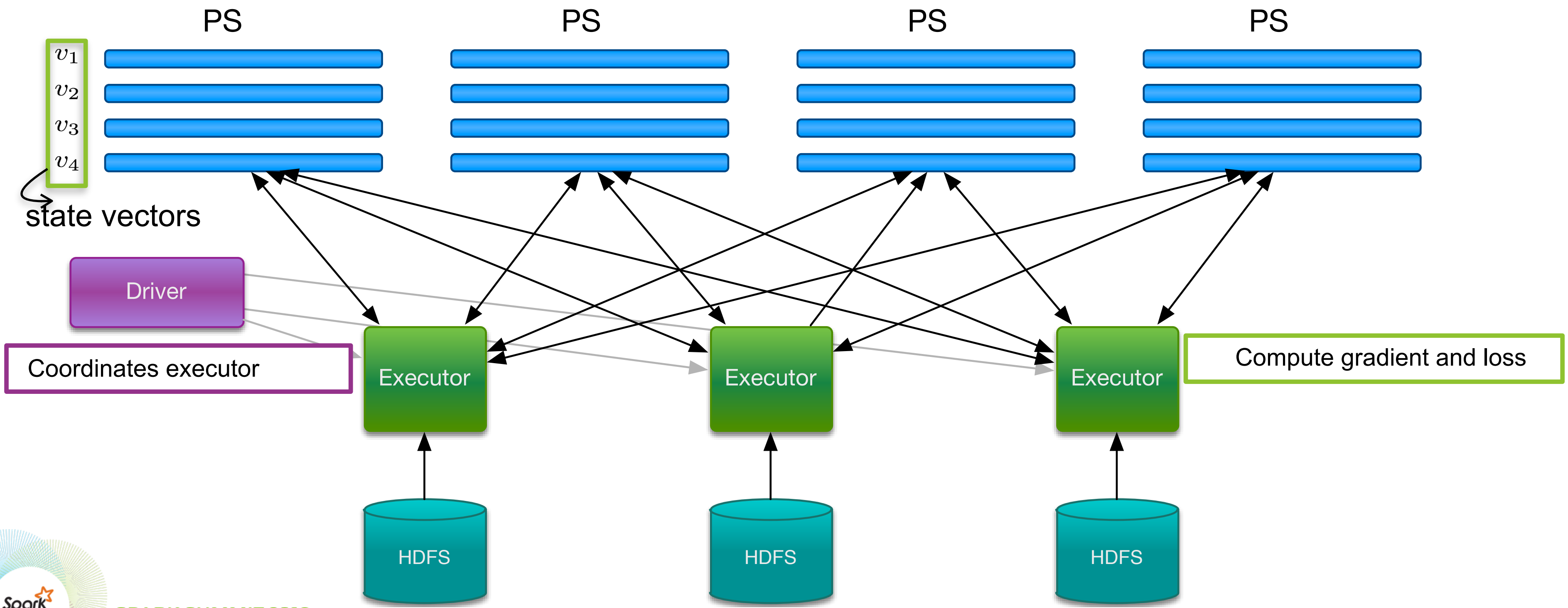
Inverse Hessian Approximation

(based on history of L-previous gradients and model deltas)

Distributed LBFGS*

Step 1: Compute and update Gradient

1. Incremental sparse gradient update
2. Fetch sparse portions of model

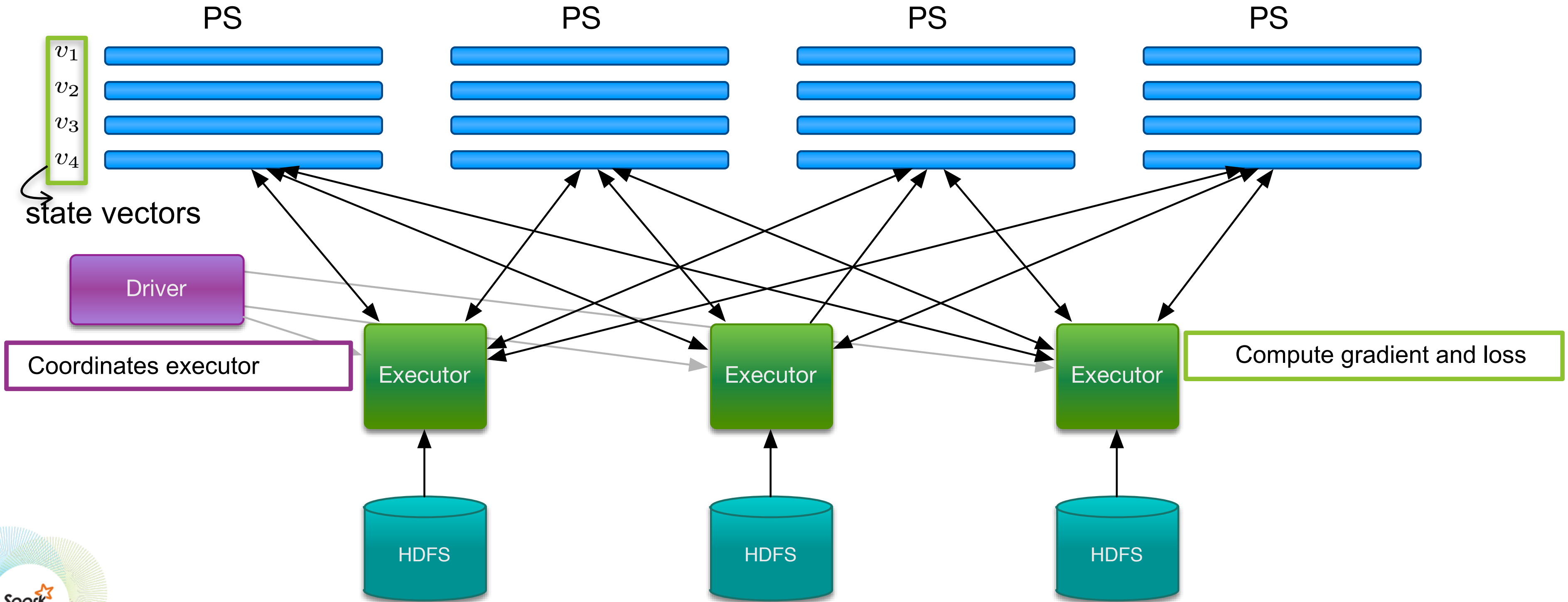


*Our design is very similar to *Sandblaster* L-BFGS, Jeff Dean et al, Large Scale Distributed Deep Networks (2012)

Distributed LBFGS*

Step 1: Compute and update Gradient

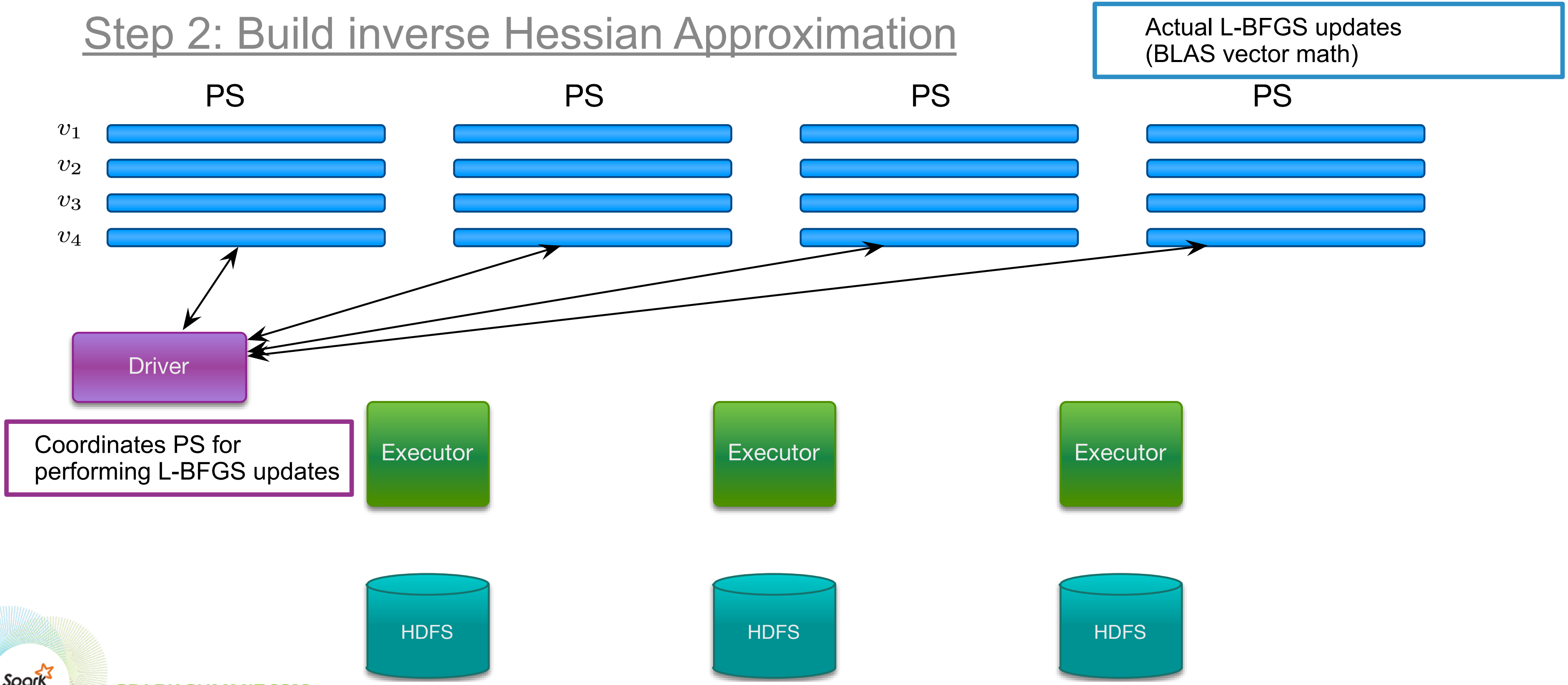
1. Incremental sparse gradient update
2. Fetch sparse portions of model



*Our design is very similar to *Sandblaster* L-BFGS, Jeff Dean et al, Large Scale Distributed Deep Networks (2012)

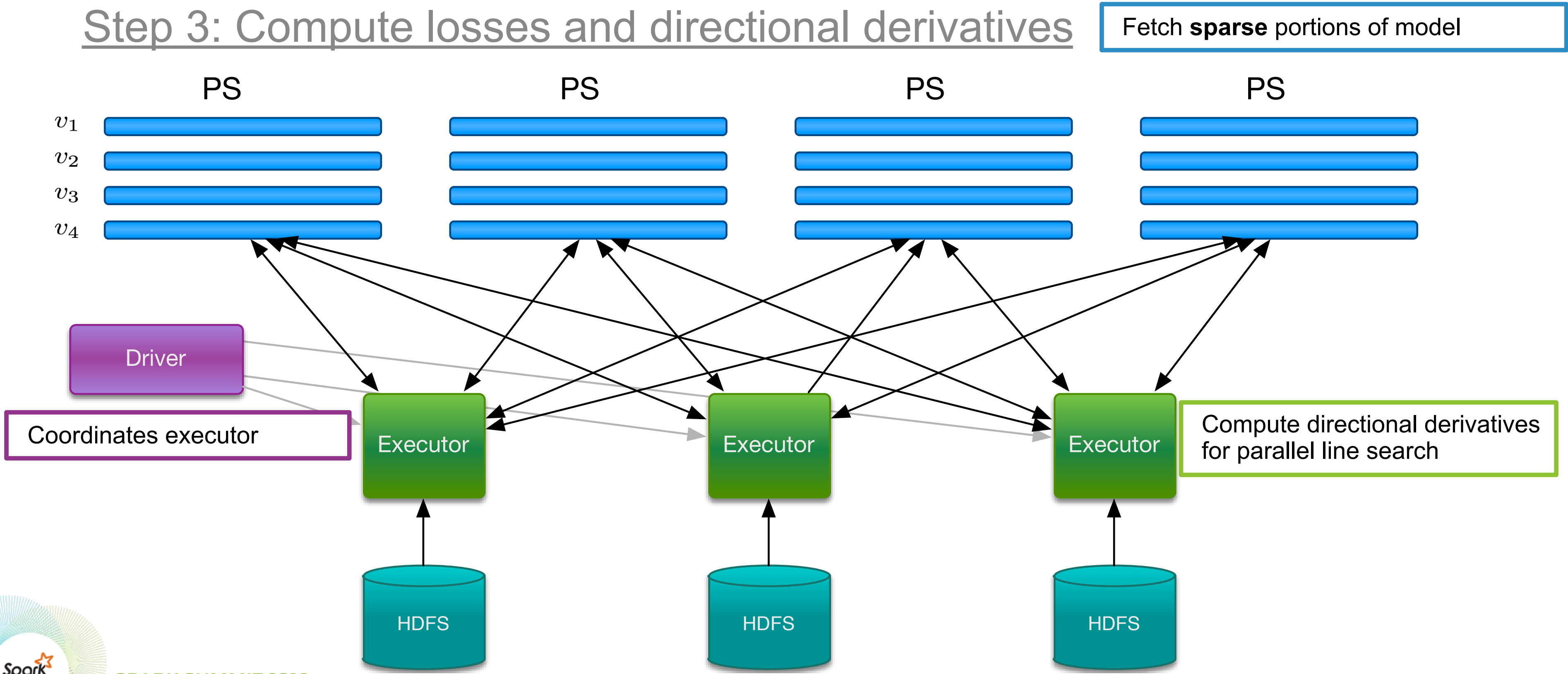
Distributed LBFGS

Step 2: Build inverse Hessian Approximation



Distributed LBFGS

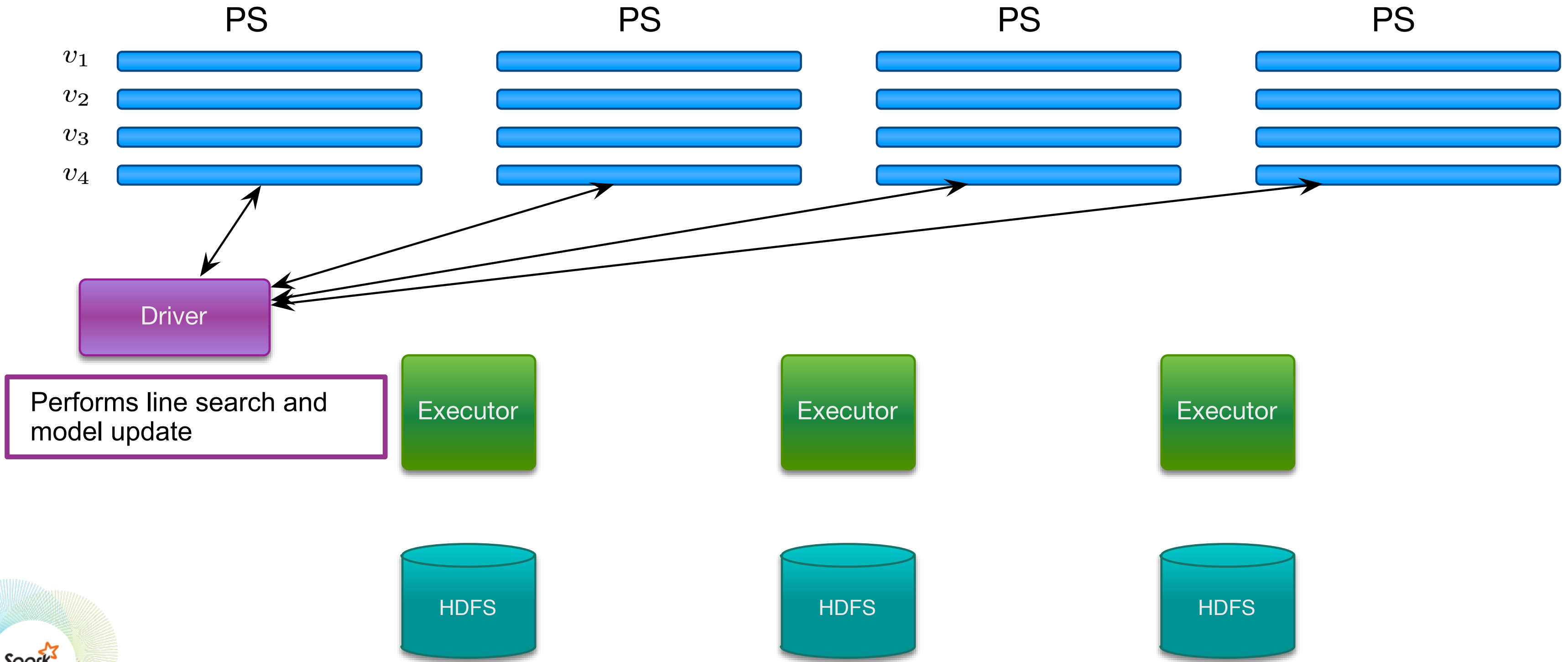
Step 3: Compute losses and directional derivatives



Distributed LBFGS

Step 4: Line search and model update

Model update (BLAS vector math)



Speedup tricks

Speedup tricks

- Intersperse communication and computation

Speedup tricks

- Intersperse communication and computation
- Quicker convergence
  - Parallel line search for step size
  - Curvature for initial Hessian approximation*

*borrowed from vowpal wabbit

Speedup tricks

- Intersperse communication and computation
- Quicker convergence
  - Parallel line search for step size
  - Curvature for initial Hessian approximation*
- Network bandwidth reduction
  - Compressed integer arrays
  - Only store indices for binary data

*borrowed from vowpal wabbit

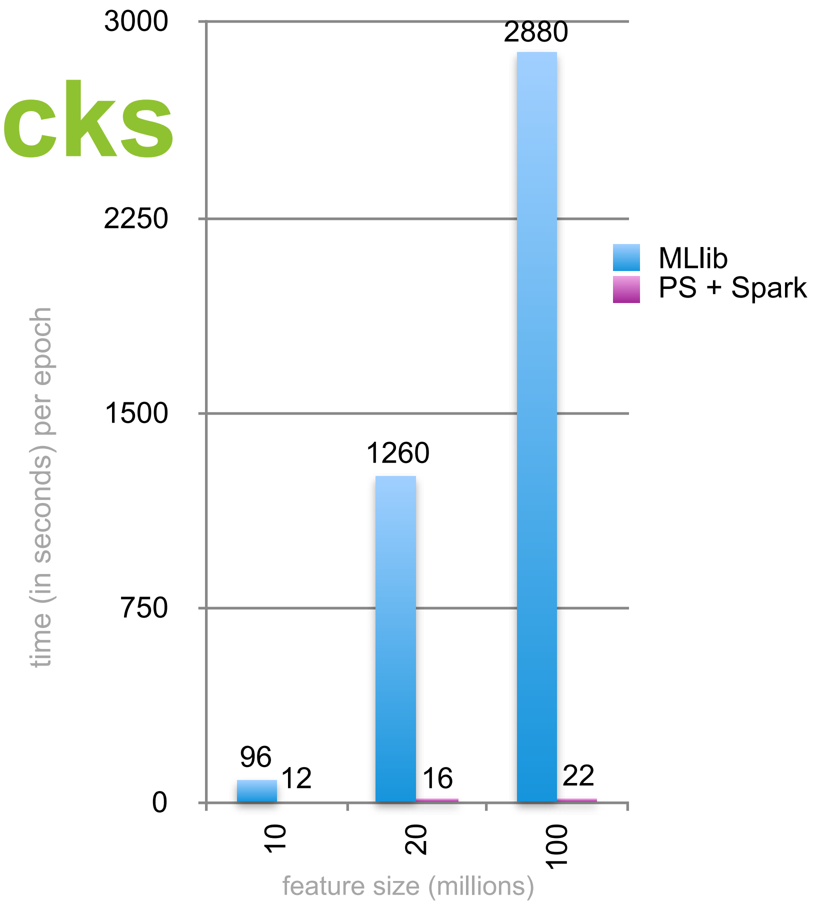
Speedup tricks

- Intersperse communication and computation
- Quicker convergence
  - Parallel line search for step size
  - Curvature for initial Hessian approximation*
- Network bandwidth reduction
  - Compressed integer arrays
  - Only store indices for binary data
- Matrix math on minibatch

*borrowed from vowpal wabbit

Speedup tricks

- Intersperse communication and computation
- Quicker convergence
  - Parallel line search for step size
  - Curvature for initial Hessian approximation*
- Network bandwidth reduction
  - Compressed integer arrays
  - Only store indices for binary data
- Matrix math on minibatch



1.6 x 10^8 examples, 100 executors, 10 cores

*borrowed from vowpal wabbit

Word Embeddings

Word Embeddings

WIKIPEDIA
The Free Encyclopedia

Word Embeddings

WIKIPEDIA
The Free Encyclopedia

v(paris) = [0.13, -0.4, 0.22, …., -0.45]
v(lion) = [-0.23, -0.1, 0.98, …., 0.65]
v(quark) = [1.4, 0.32, -0.01, …, 0.023]

⋮

Word2vec

Word2vec

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Word2vec

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

- new techniques to compute vector representations of words from corpus

Word2vec

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

- new techniques to compute vector representations of words from corpus
- geometry of vectors captures word semantics

Word2vec

Word2vec

- Skipgram with negative sampling:

Word2vec

- Skipgram with negative sampling:
  - training set includes pairs of words and neighbors in corpus, along with randomly selected words for each neighbor

Word2vec

- Skipgram with negative sampling:
  - training set includes pairs of words and neighbors in corpus, along with randomly selected words for each neighbor
  - determine $w \rightarrow \mathbf{u}(w),\mathbf{v}(w)$ so that $sigmoid(\mathbf{u}(w)\bullet\mathbf{v}(w'))$ is close to (minimizes log loss) the probability that w' is a neighbor of w as opposed to a randomly selected word.

Word2vec

- Skipgram with negative sampling:
  - training set includes pairs of words and neighbors in corpus, along with randomly selected words for each neighbor
  - determine $w \to \mathbf{u}(w),\mathbf{v}(w)$ so that $sigmoid(\mathbf{u}(w)\bullet\mathbf{v}(w'))$ is close to (minimizes log loss) the probability that w' is a neighbor of w as opposed to a randomly selected word.
  - SGD involves computing many vector dot products e.g., $\mathbf{u}(w)\bullet\mathbf{v}(w')$ and vector linear combinations e.g., $\mathbf{u}(w) \mathrel{+}= \alpha\, \mathbf{v}(w')$.

Word2vec Application at Yahoo

- Example training data:

gas_cap_replacement_for_car
slc_679f037df54f5d9c41cab05bfae0926
gas_door_replacement_for_car
slc_466145af16a40717c84683db3f899d0a fuel_door_covers
adid_c_28540527225_285898621262
slc_348709d73214fdeb9782f8b71aff7b6e autozone_auto_parts
adid_b_3318310706_280452370893 auoto_zone
slc_8dcdab5d20a2caa02b8b1d1c8ccbd36b
slc_58f979b6deb6f40c640f7ca8a177af2d

[Grbovic, et. al. SIGIR 2015 and SIGIR 2016 (to appear)]

Distributed Word2vec

Distributed Word2vec

- Needed system to train 200 million 300 dimensional word2vec model using minibatch SGD

Distributed Word2vec

- Needed system to train 200 million 300 dimensional word2vec model using minibatch SGD
- Achieved in a high throughput and network efficient way using our matrix based PS server:

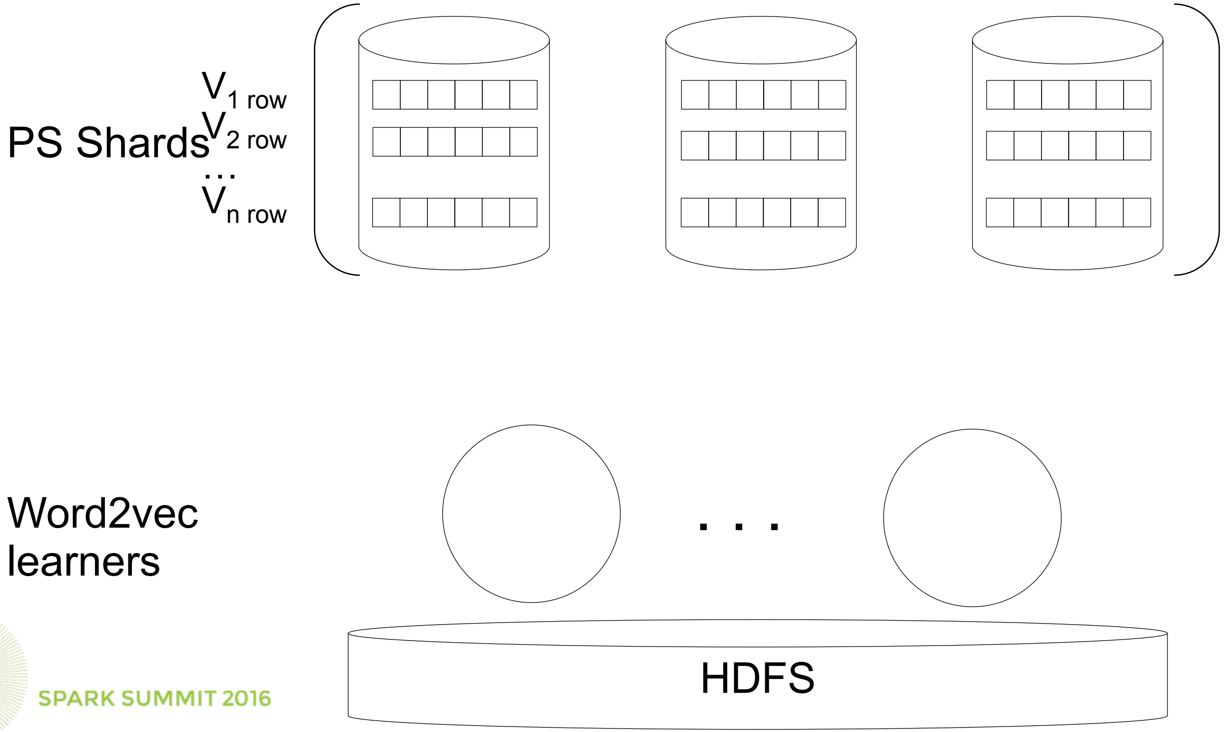
Distributed Word2vec

- Needed system to train 200 million 300 dimensional word2vec model using minibatch SGD
- Achieved in a high throughput and network efficient way using our matrix based PS server:
  - Vectors don't go over network.

Distributed Word2vec

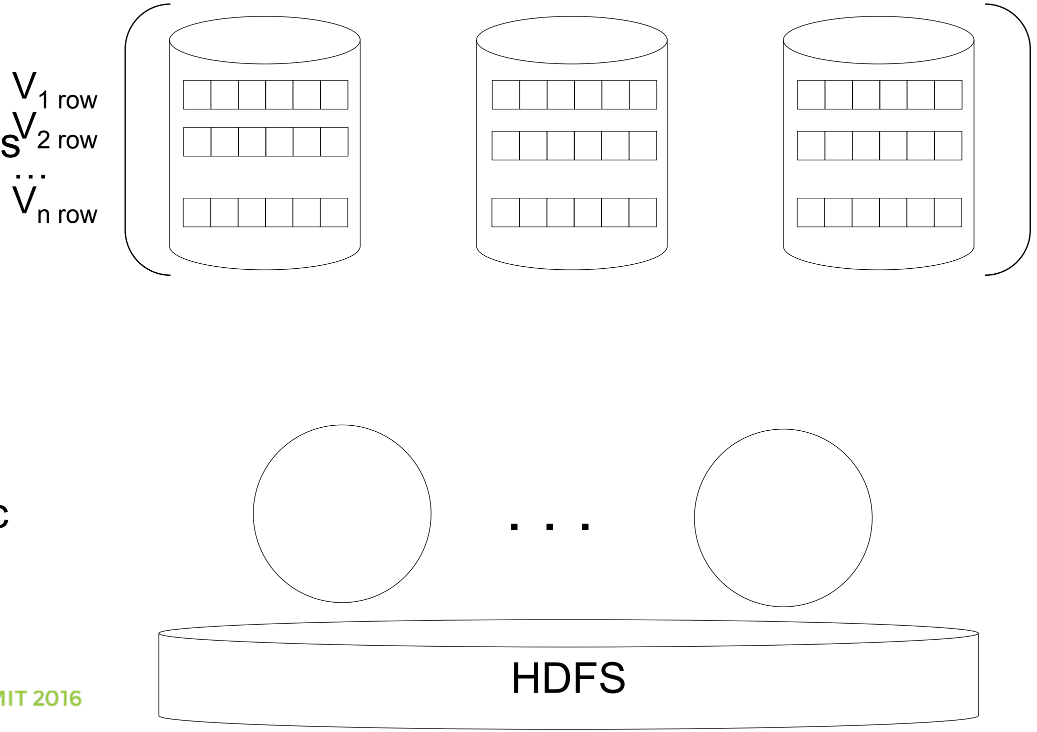
- Needed system to train 200 million 300 dimensional word2vec model using minibatch SGD
- Achieved in a high throughput and network efficient way using our matrix based PS server:
  - Vectors don't go over network.
  - Most compute on PS servers, with clients aggregating partial results from shards.

Distributed Word2vec



Distributed Word2vec

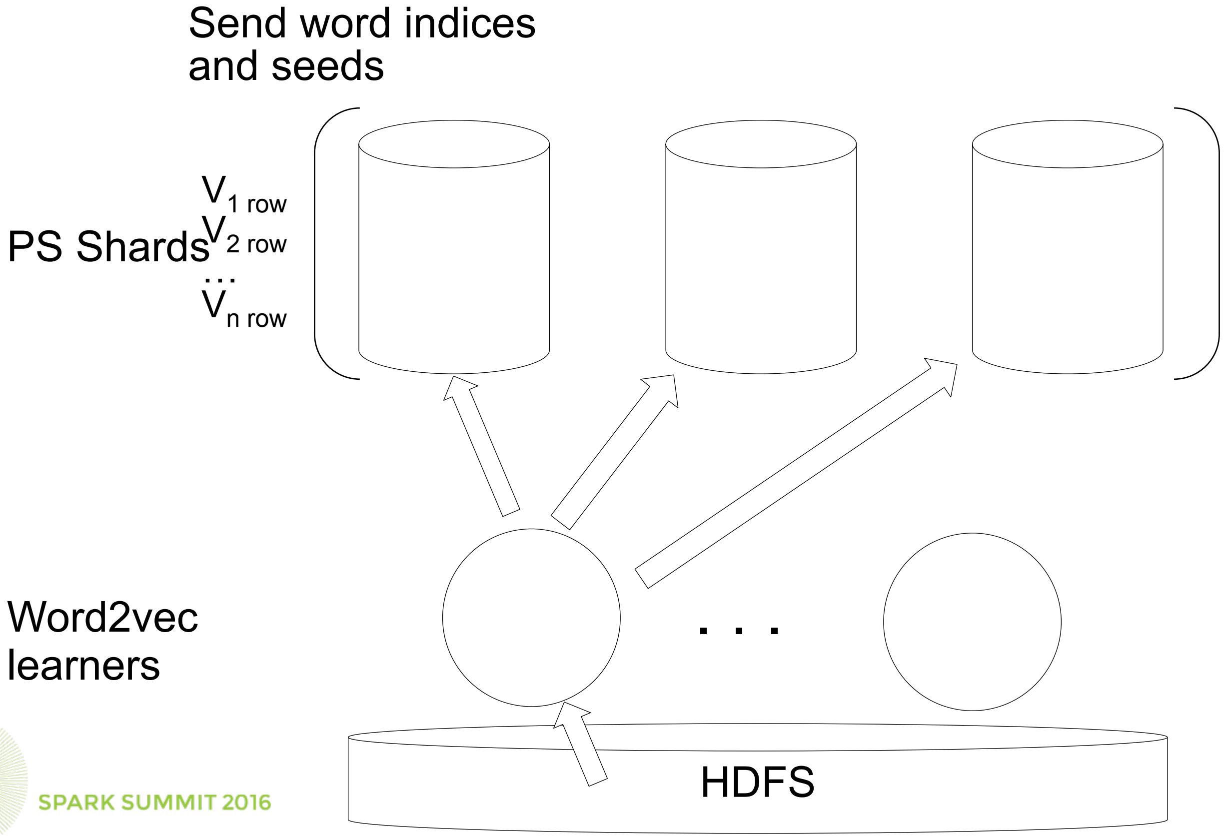
PS Shards



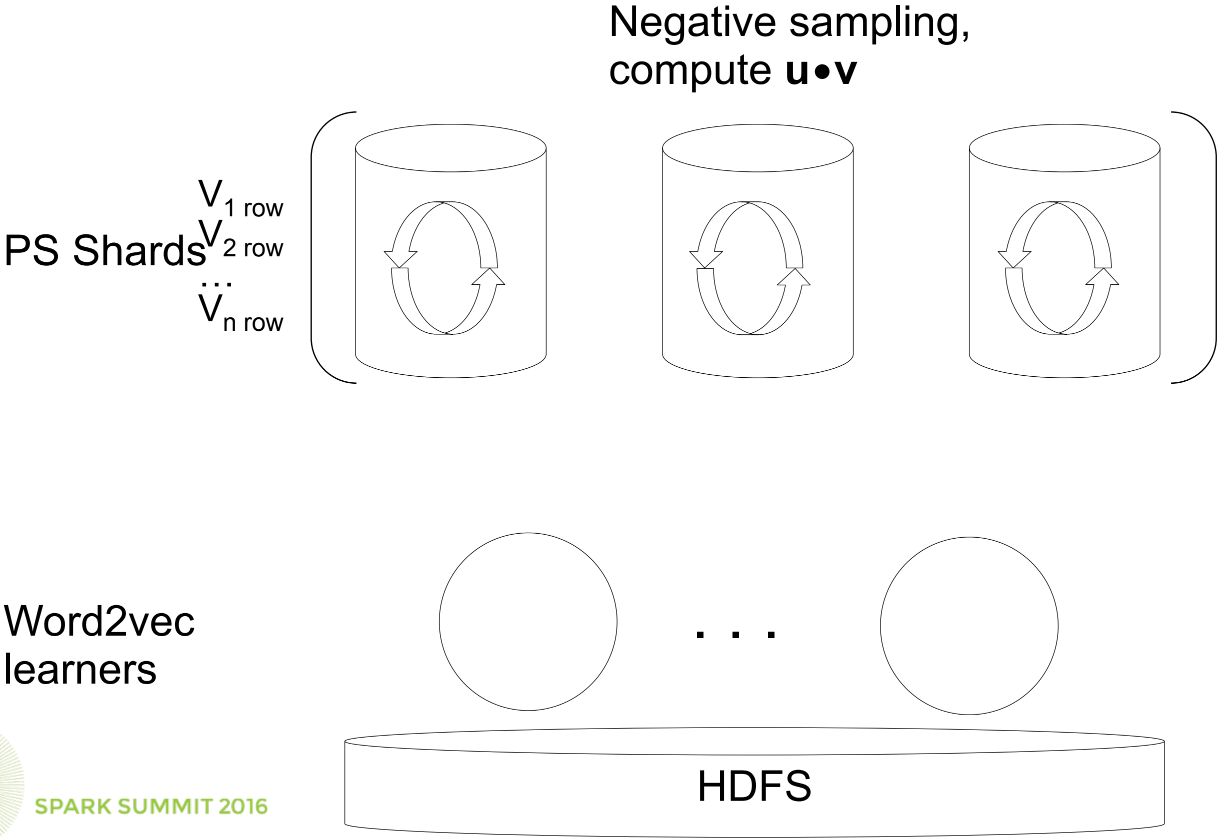
Each shard stores
a **part** of **every** vector

Word2vec
learners

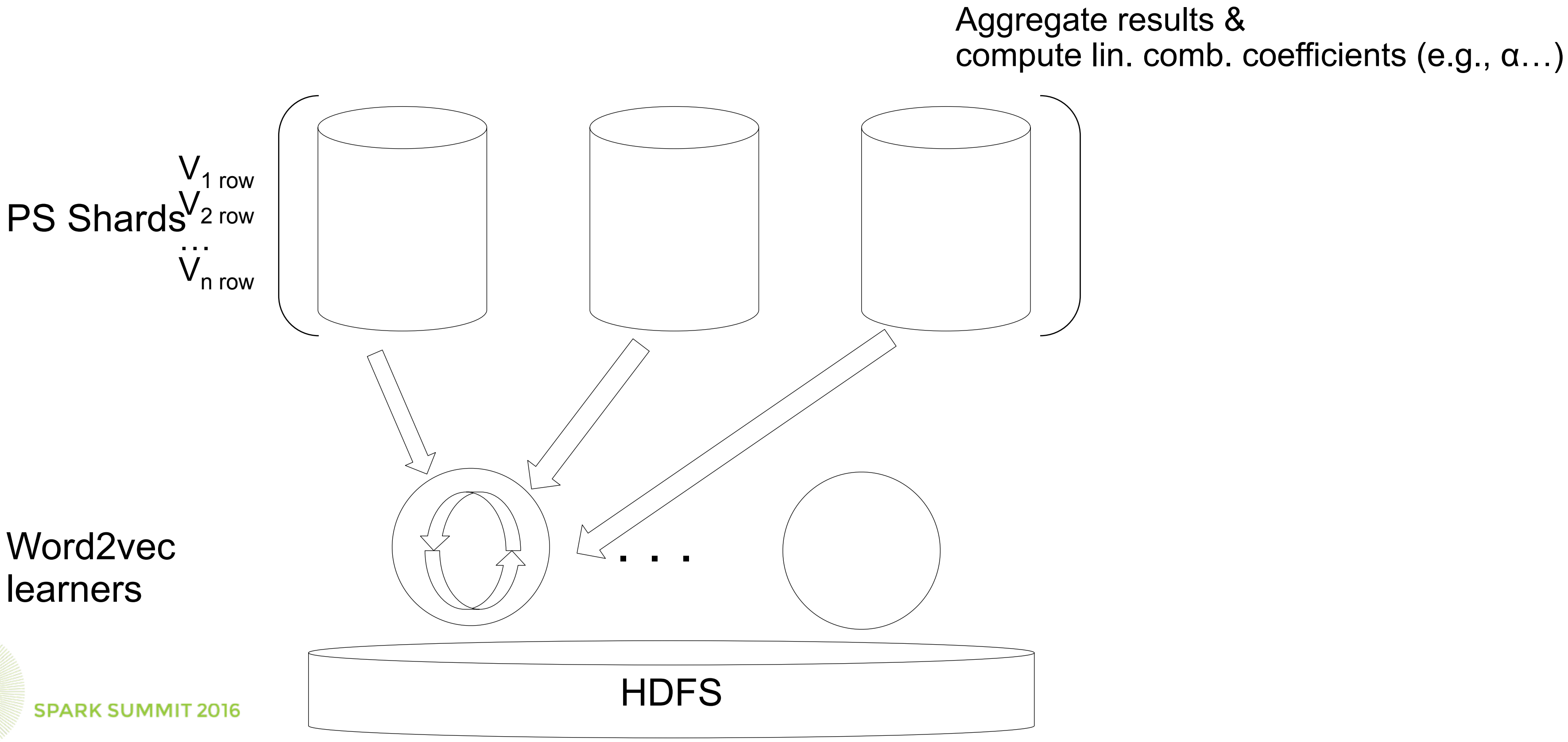
Distributed Word2vec



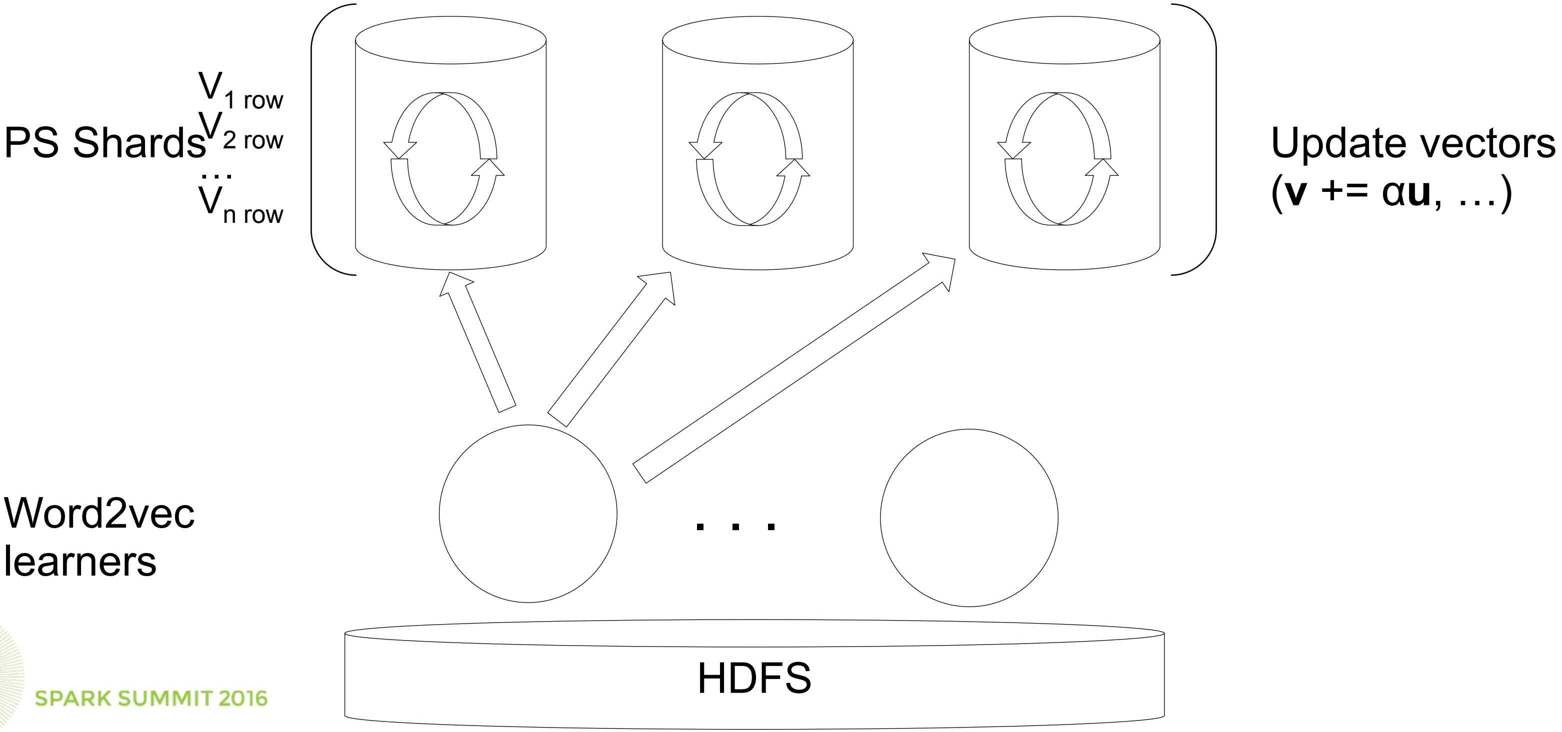
Distributed Word2vec



Distributed Word2vec



Distributed Word2vec



Distributed Word2vec

Distributed Word2vec

- Network lower by factor of #shards/dimension compared to conventional PS based system (1/20 to 1/100 for useful scenarios).

Distributed Word2vec

- Network lower by factor of #shards/dimension compared to conventional PS based system (1/20 to 1/100 for useful scenarios).
- Trains 200 million vocab, 55 billion word search session in 2.5 days.

Distributed Word2vec

- Network lower by factor of #shards/dimension compared to conventional PS based system (1/20 to 1/100 for useful scenarios).
- Trains 200 million vocab, 55 billion word search session in 2.5 days.
- In production for regular training in Yahoo search ad serving system.

Other Projects using Spark + PS

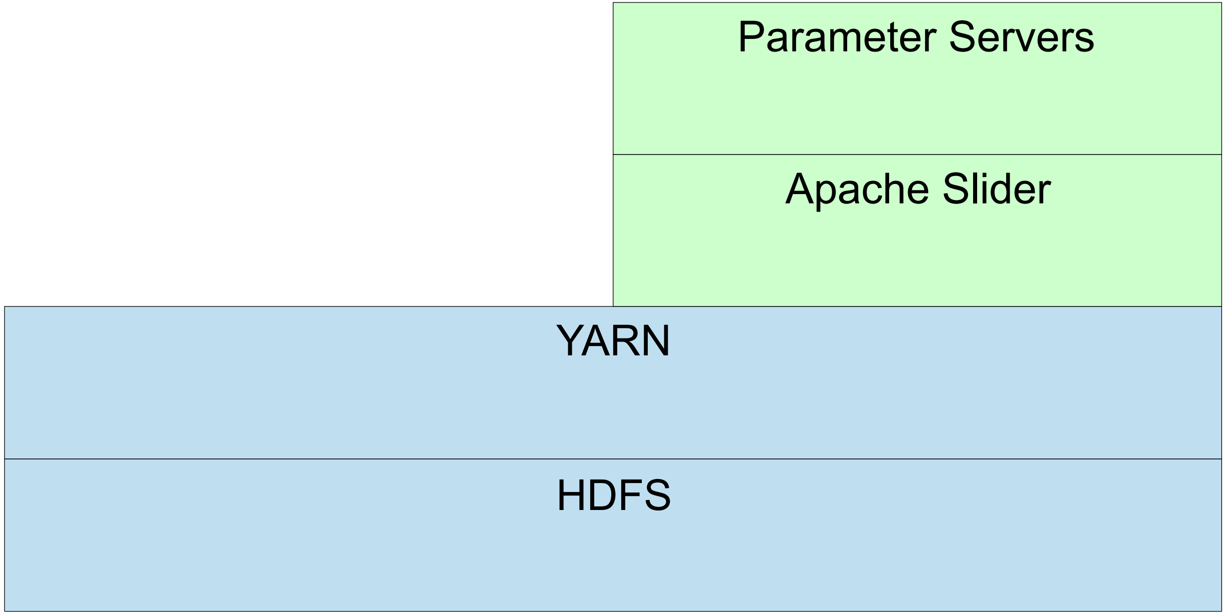
- Online learning on PS
  - Personalization as a Service
  - Sponsored Search
- Factorization Machines
  - Large scale user profiling

SPARK+PS ON HADOOP CLUSTER

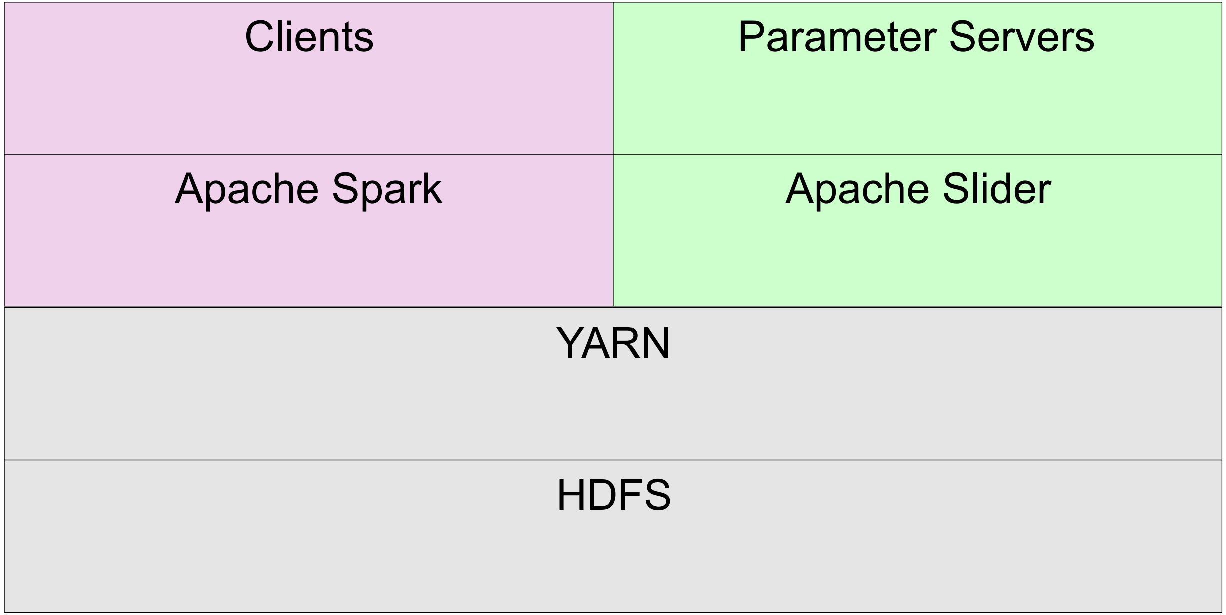
Training Data on HDFS

HDFS

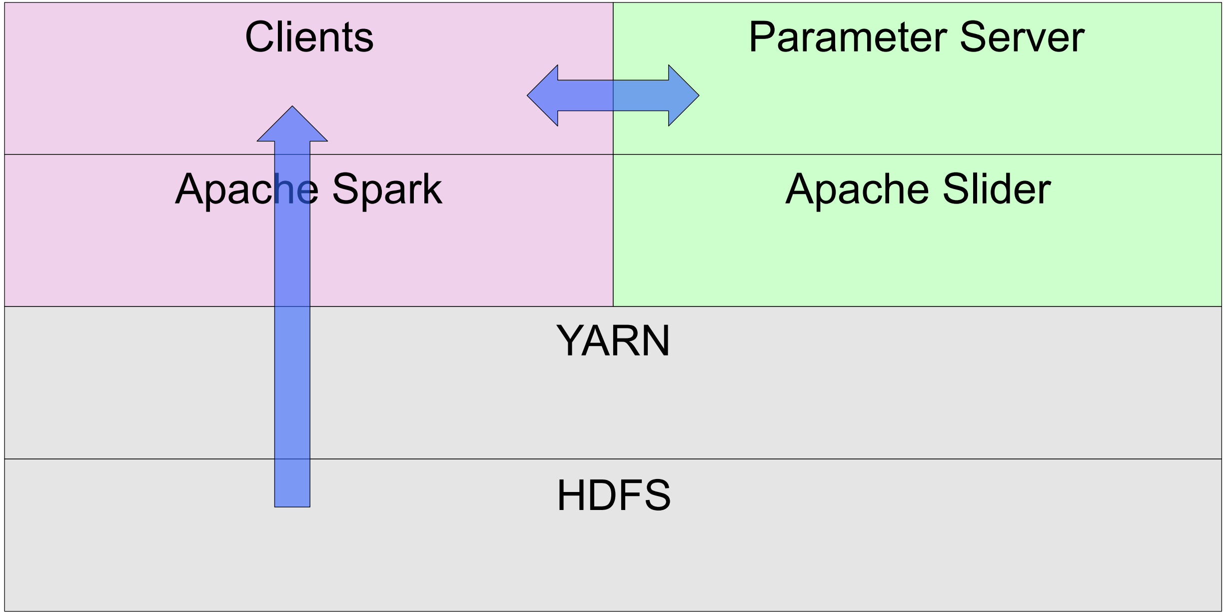
Launch PS Using Apache Slider on YARN



Launch Clients using Spark or Hadoop Streaming API

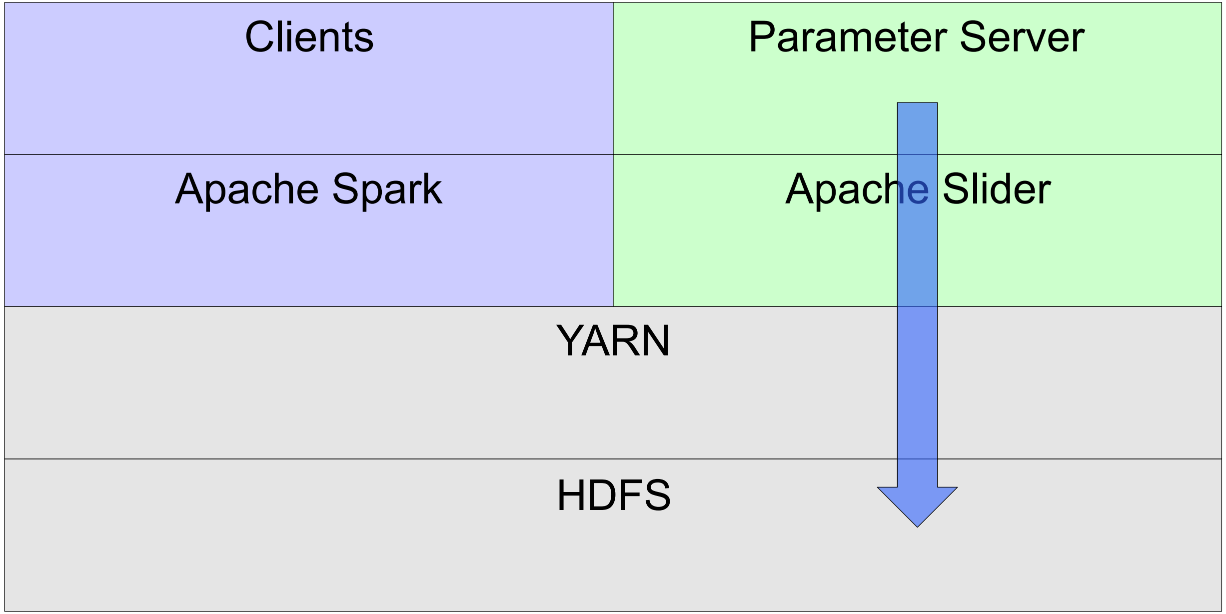


Training

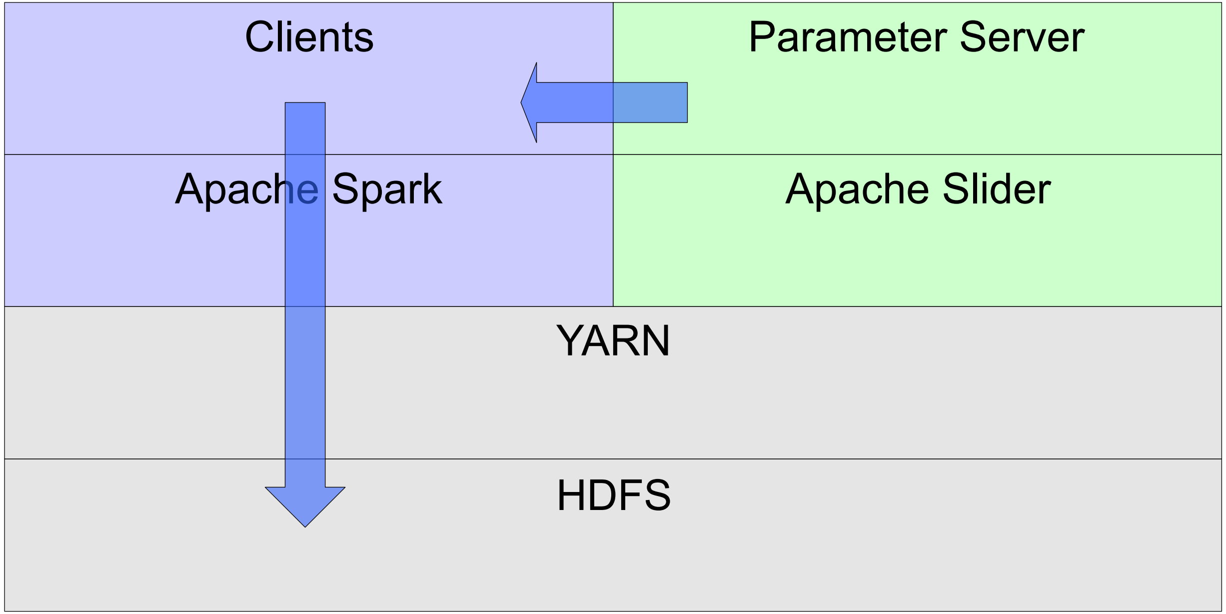


Clients
Parameter Server
Apache Spark
Apache Slider
YARN
HDFS

Model Export



Model Export



Summary

- Parameter server indispensable for big models
- Spark + Parameter Server has proved to be very flexible platform for our large scale computing needs
- Direct computation on the parameter servers accelerate training for our use-cases

Thank you!

For more, contact bigdata@yahoo-inc.com.