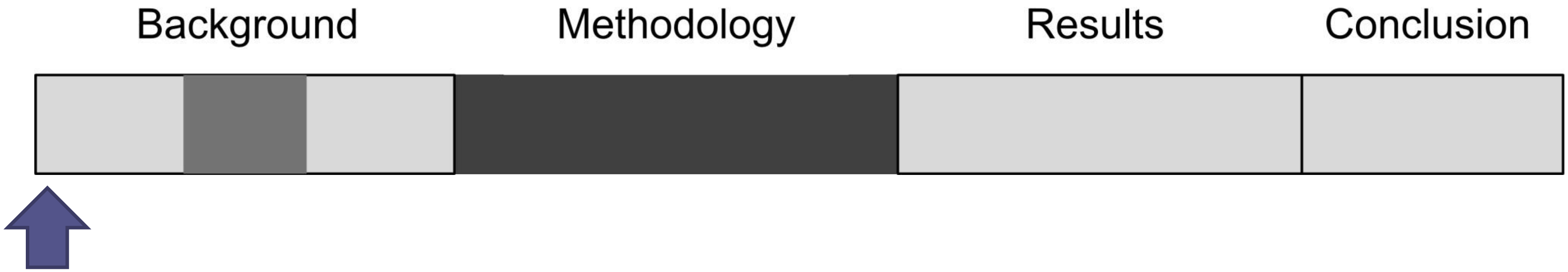
PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Luyu Wang
SciCom Group, UW

UNIVERSITY OF WATERLOO

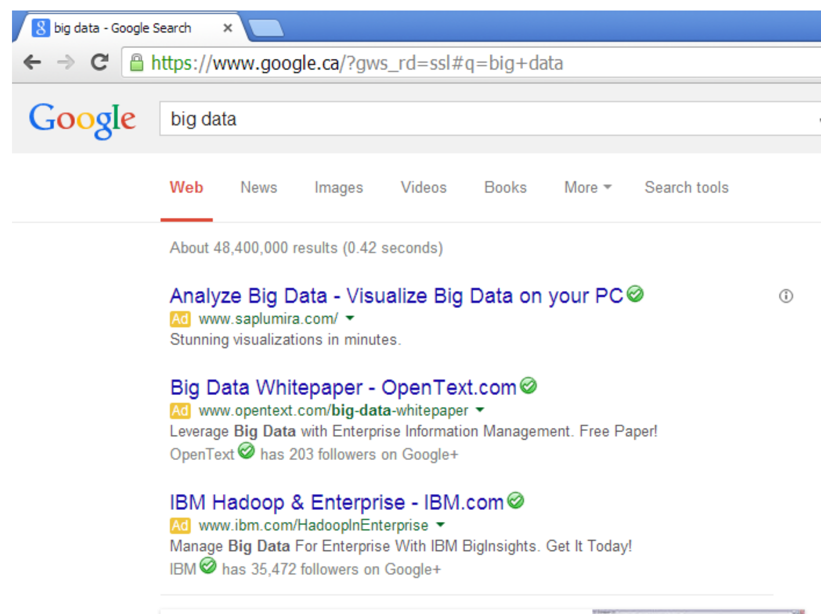
Content

- Background
- Methodology
- Results
- Conclusion



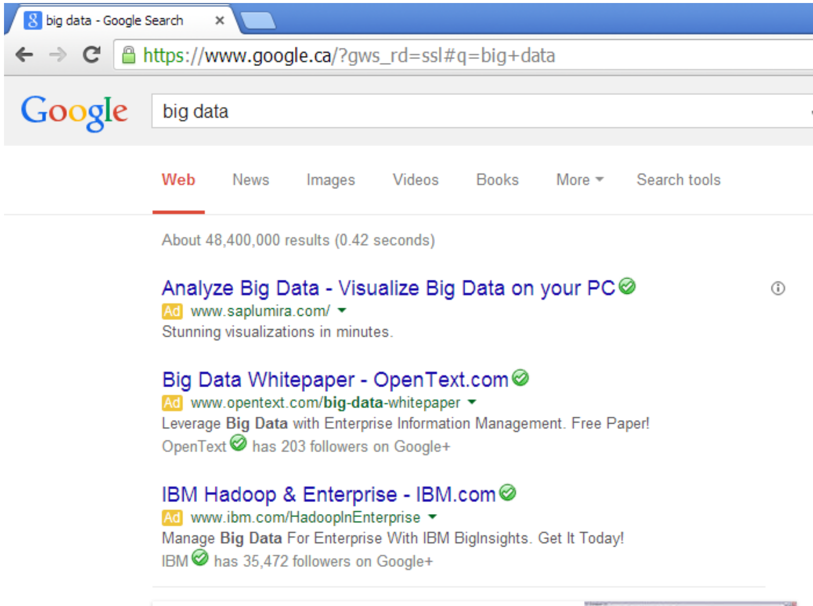
Google’s Bounce Rate Prediction Problem

- 'Bounce'



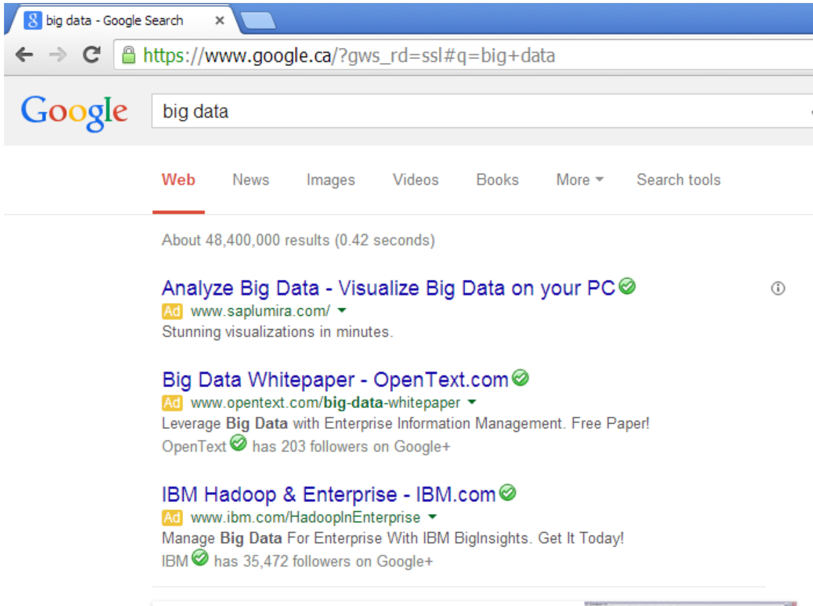
Google's Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience



Google's Bounce Rate Prediction Problem

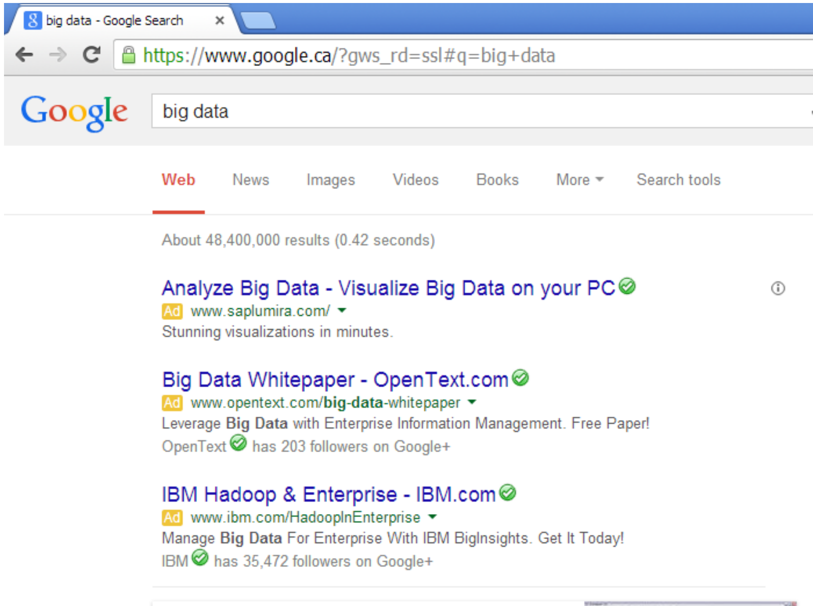
- 'Bounce'
- High bounce rate = poor user experience
- Task: to predict bounce rate with data on hand



Google's Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
- Task: to predict bounce rate with data on hand

COMPUTATIONAL ADVERTISING



The Data Mining Tasks

- Discovering patterns in large data sets (knowledge)

- Data mining vs machine learning?
  - Lines are blurred

The Data Mining Tasks

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
  - Clustering
  - Compression
  - Outlier detection
- Reinforce learning

  ……

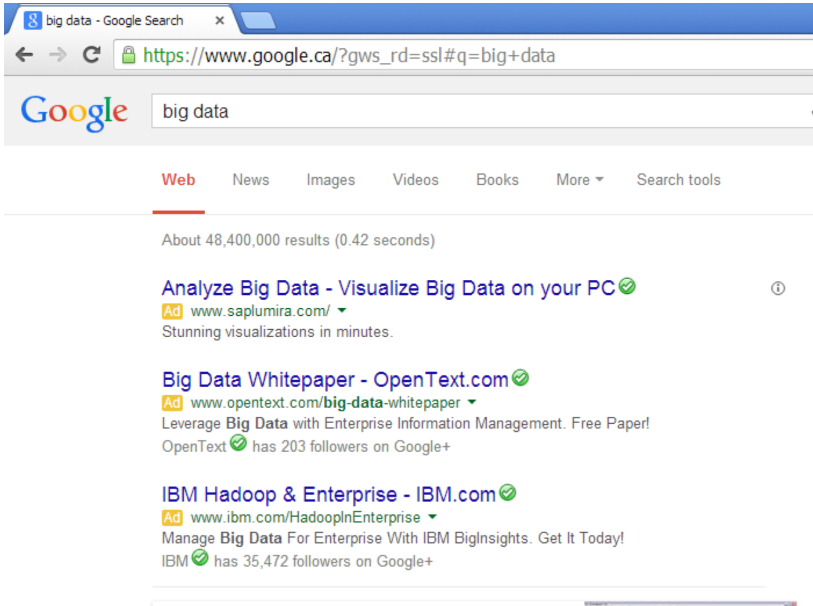
The Data Mining Tasks

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
  - Clustering
  - Compression
  - Outlier detection
- Reinforce learning

  ……

Google's Bounce Rate Prediction Problem

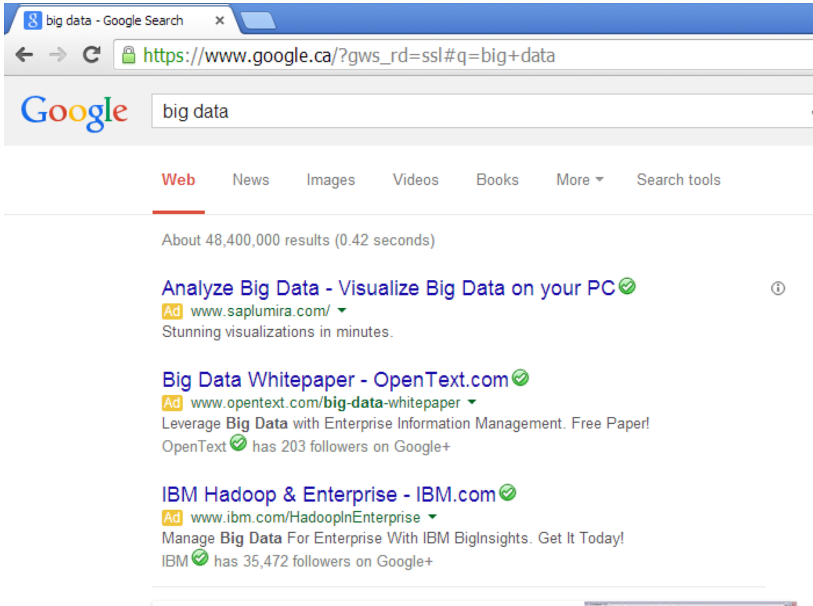
- Go back to 'bounce'



Google's Bounce Rate Prediction Problem

- Go back to 'bounce'

- One Click:
  - 6 attributes
  - 1 label



Google's Bounce Rate Prediction Problem

- 6 attributes
  - search query of the click
  - advertiser chosen keyword
  - ad text
  - estimated clickthrough rate of the ad click
  - numeric similarity score
  - whether the ad matches the query
- 1 label
  - bounce or not

Supervised Learning - Data Model

- Set of attributes

$$\chi = \{X_1, X_2, ..., X_N\}$$

- Output

$$Y$$

- Training data set (the i[th] vector)

$$D^* = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathrm{D}_{x_1} \times \mathrm{D}_{x_2} \times ... \times \mathrm{D}_{x_N}\}$$

Supervised Learning - Task

- Given the training dataset

$$D^* = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathrm{D}_{x_1} \times \mathrm{D}_{x_2} \times \ldots \times \mathrm{D}_{x_N}\}$$

- Goal: to learn a mapping model

$$F : \mathrm{D}_{x_1} \times \mathrm{D}_{x_2} \times \ldots \times \mathrm{D}_{x_N} \rightarrow \mathrm{D}_Y$$

Google's Bounce Rate Prediction Problem

- Given 6 attributes
  - search query of the click
  - advertiser chosen keyword
  - ad text
  - estimated clickthrough rate of the ad click
  - numeric similarity score
  - whether the ad matches the query

- Want to know if it is going to be a bounce?

Supervised Learning - Task

- Given the training dataset

$$D^{*} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathrm{D}_{x_1} \times \mathrm{D}_{x_2} \times \ldots \times \mathrm{D}_{x_N}\}$$

- Goal: to learn a mapping model

$$F : \mathrm{D}_{x_1} \times \mathrm{D}_{x_2} \times \ldots \times \mathrm{D}_{x_N} \rightarrow \mathrm{D}_{Y}$$

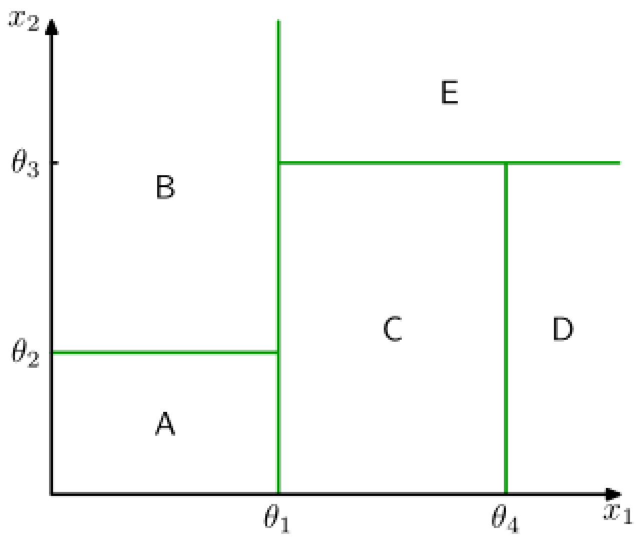
- Tree models
  - Capable of modeling complex tasks

Tree Model

- Goal: to learn a mapping model

$$F : D_{x_1} \times D_{x_2} \times \ldots \times D_{x_N} \rightarrow D_Y$$

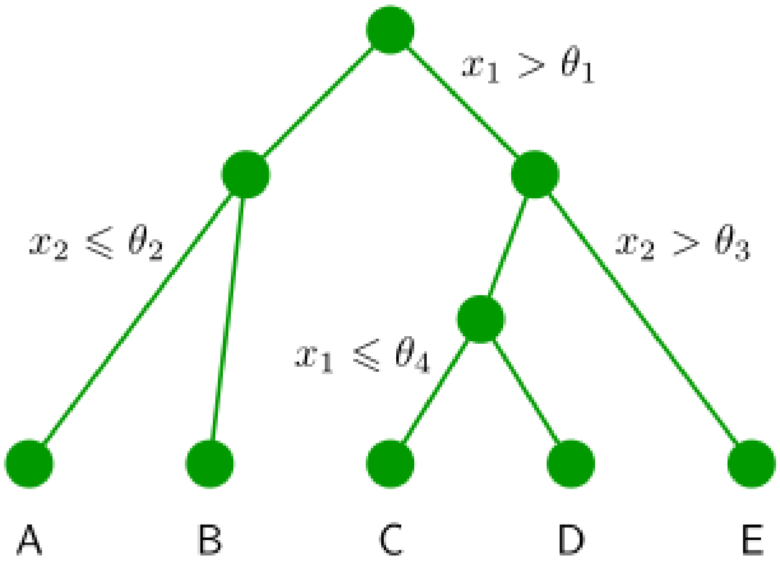
- Recursively partitioning the input data space into non-overlapping regions



- Simple model each region
  - constant
  - simple function

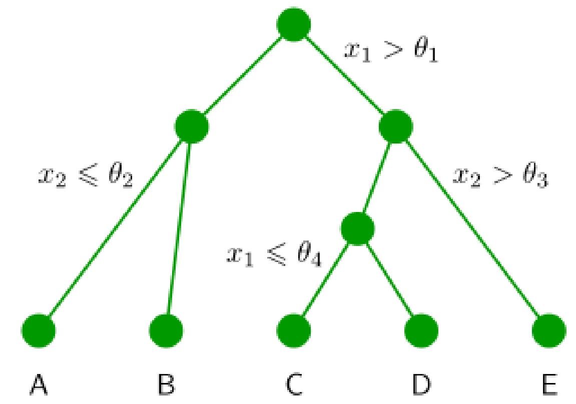
Tree Model

- Easy to interpret; thus popular



Bishop, Christopher M. *Pattern recognition and machine learning*. Vol. 1. New York: springer, 2006.

Learning Tree Model



- Greedy learning algorithm

Input: node *n*, training dataset *D*

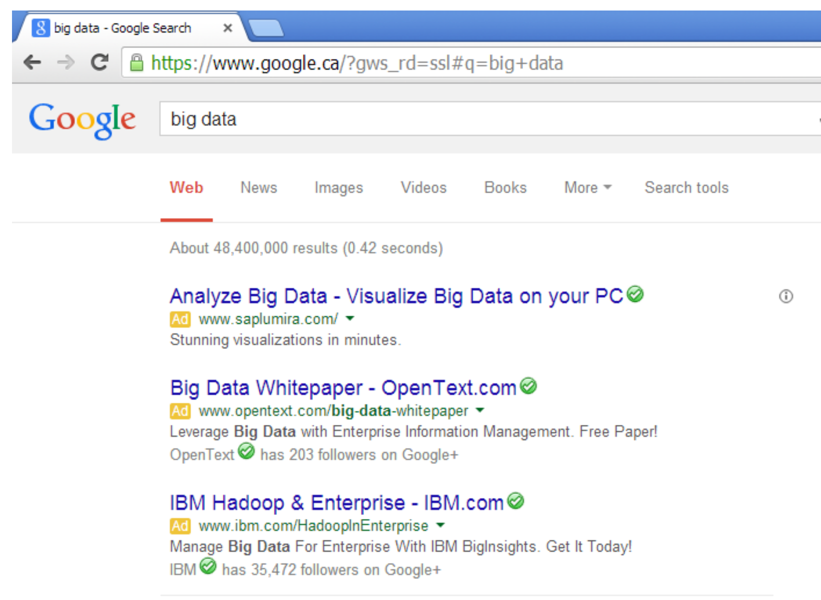
1) fully scan *D*, find the *best split*, by maximizing ‘purity’

$$|D| \times Var(D) - (|D_L| \times Var(D_L) + |D_R| \times Var(D_R))$$

2) for either branch
- if stopping criteria satisfied: *pure* region
- else: advance a level

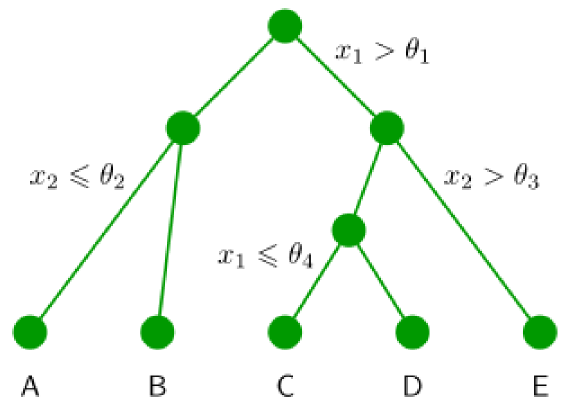
Google's Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
- Task: predicting bounce rate with data on hand



What if big data?

Learning Tree Model



- Greedy learning algorithm

Input: node n, training dataset D

1) fully scan D, find the *best split*

- out of memory
- hard disk slow

2) for either branch
   - if stopping criteria satisfied: *pure* region
   - else: build a higher-level node

Solution – Scaling Up Tree Learning

- Fully scan *D*, find the *best split*
  - out of memory
  - hard disk slow

- By Google Research, 2009
  - Computer Cluster
  - MapReduce
  - Tree learning

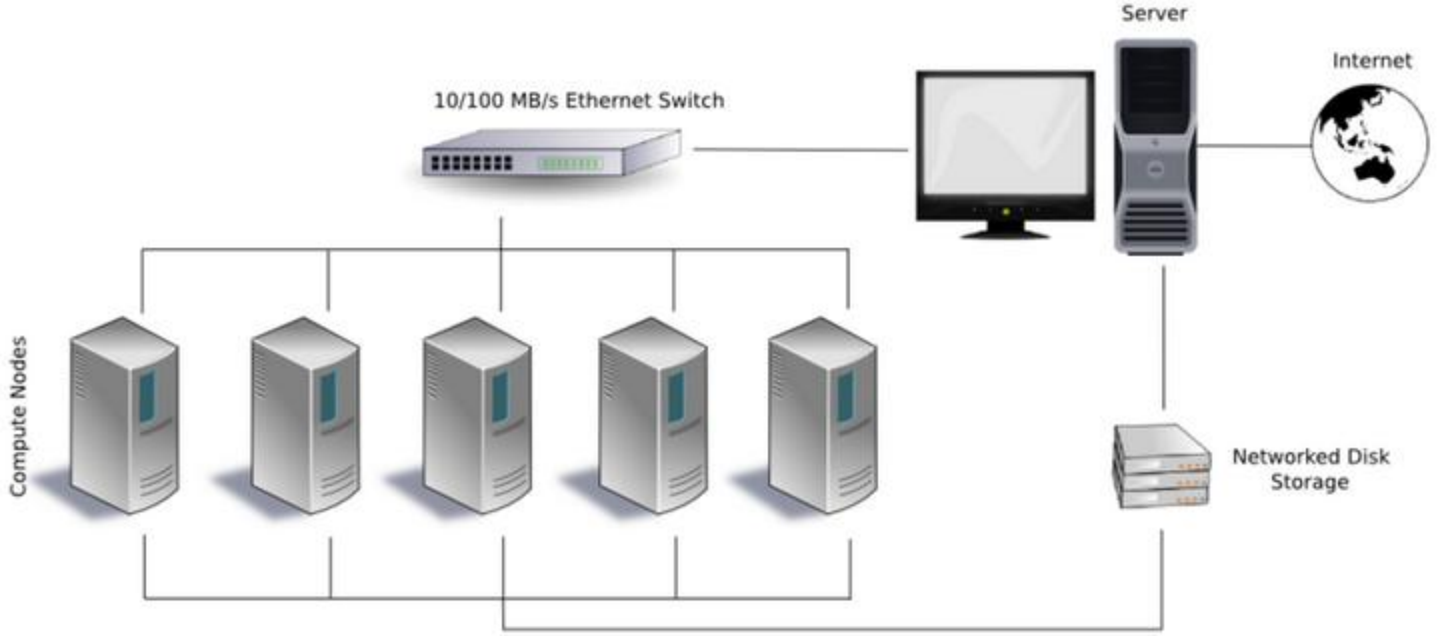
Panda, Biswanath, et al. "Planet: massively parallel learning of tree ensembles with mapreduce." *Proceedings of the VLDB Endowment* 2.2 (2009): 1426-1437.

Content

- Background
- Methodology
- Results
- Conclusion

Computer Cluster

- Controller and workers



Source: Wikepedia

MapReduce Framework

- Objective: to easily handle data too large to fit in memory

MapReduce Framework

- Objective: to easily handle data too large to fit in memory

- It does all the dirty work:
  - distribute the data
  - parallelize the computation
  - handle failures

MapReduce Framework

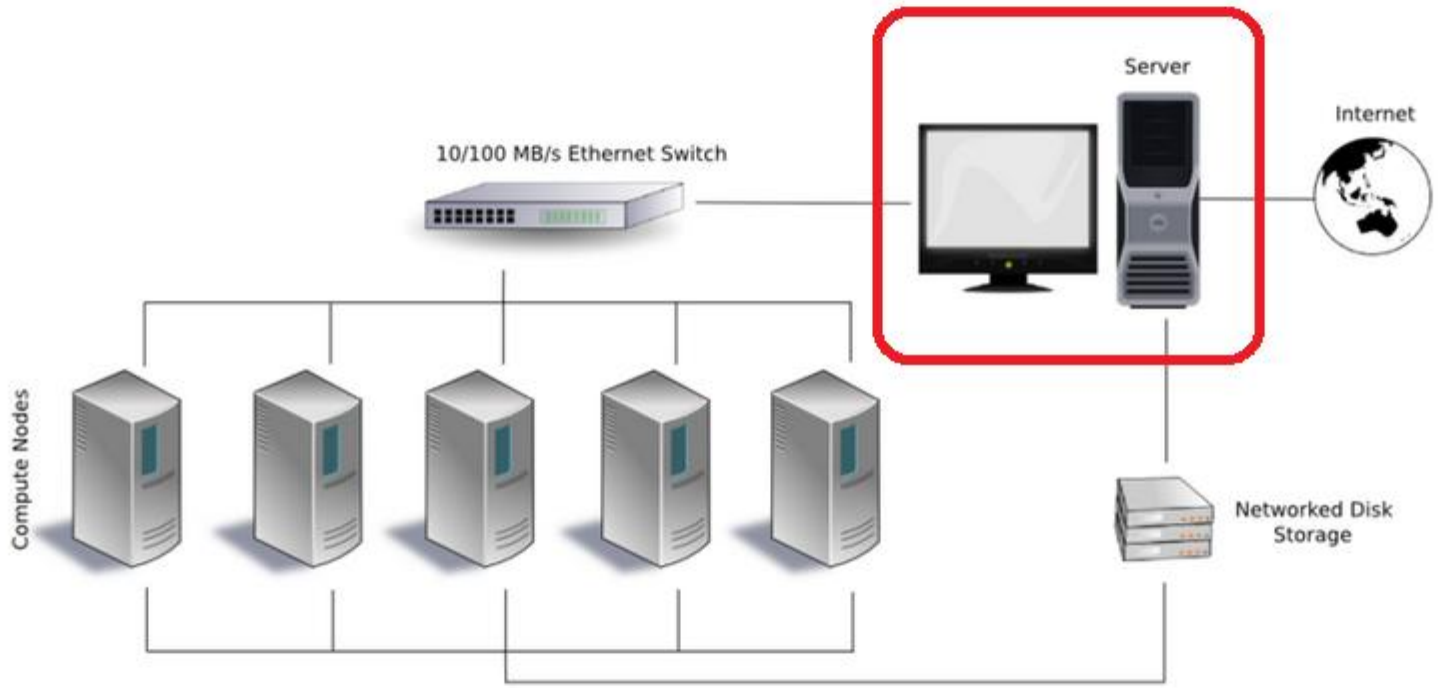
- Objective: to easily handle data too large to fit in memory

- It does all the dirty work:
  - distribute the data
  - parallelize the computation
  - handle failures

- User simply writes Map and Reduce functions

Computer Cluster

- Core: Controller



Source: Wikepedia

Job of Controller

- Keeps model file (M), containing the entire tree constructed so far
- Partitions the whole training dataset, across a set of mappers

Job of Controller

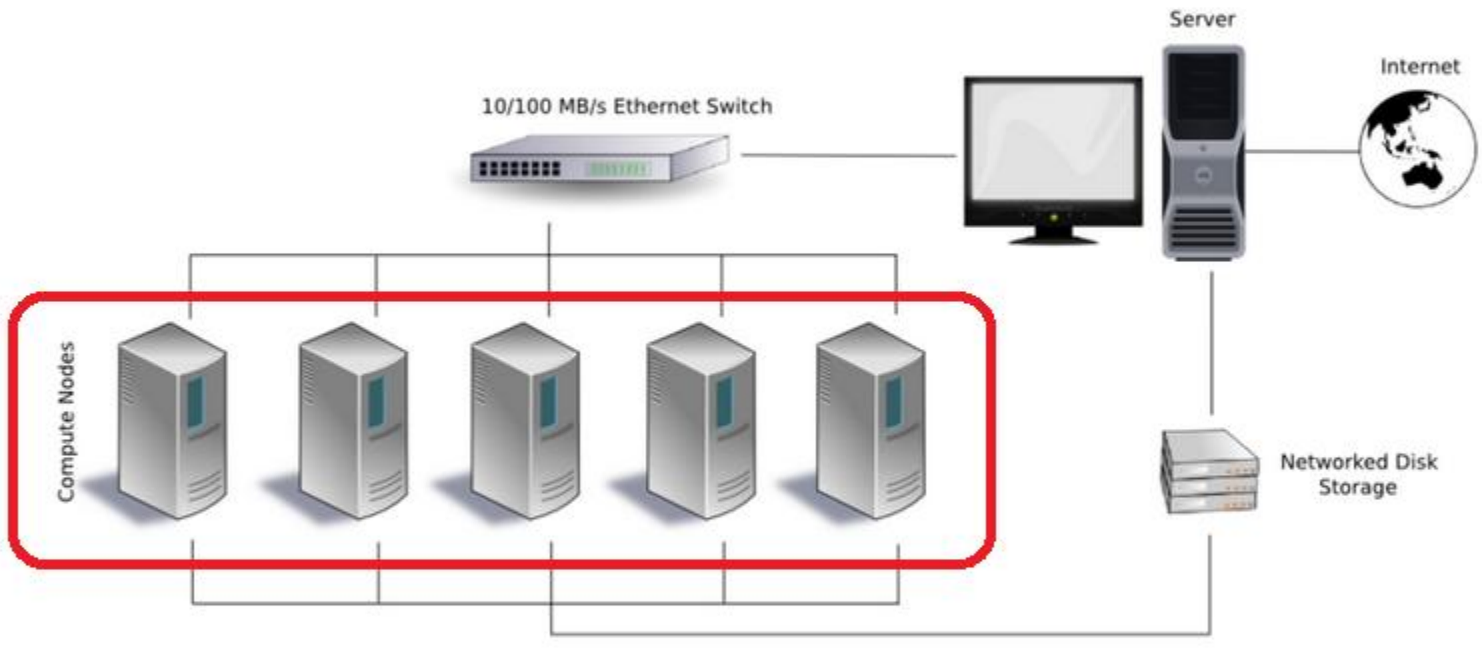
- Each tree node, detects size of data set

```
if  single machine ok?
   -> push to 'SmallData Queue'
else
   -> push to 'LargeData Queue'
```

- Schedules jobs in both queues for workers

Job of Workers

- Map and Reduce functions



Source: Wikepedia

MapReduce Work - SmallData Queue

- Map function

  - input:

    partitioned training set D_k

    node n

    Model file M

  - check if an instance input to n -> emits

  - output (list):

    key = node n

    value = subset of D_k input to n

MapReduce Work - SmallData Queue

- Reduce function

  - input:

    key = node n

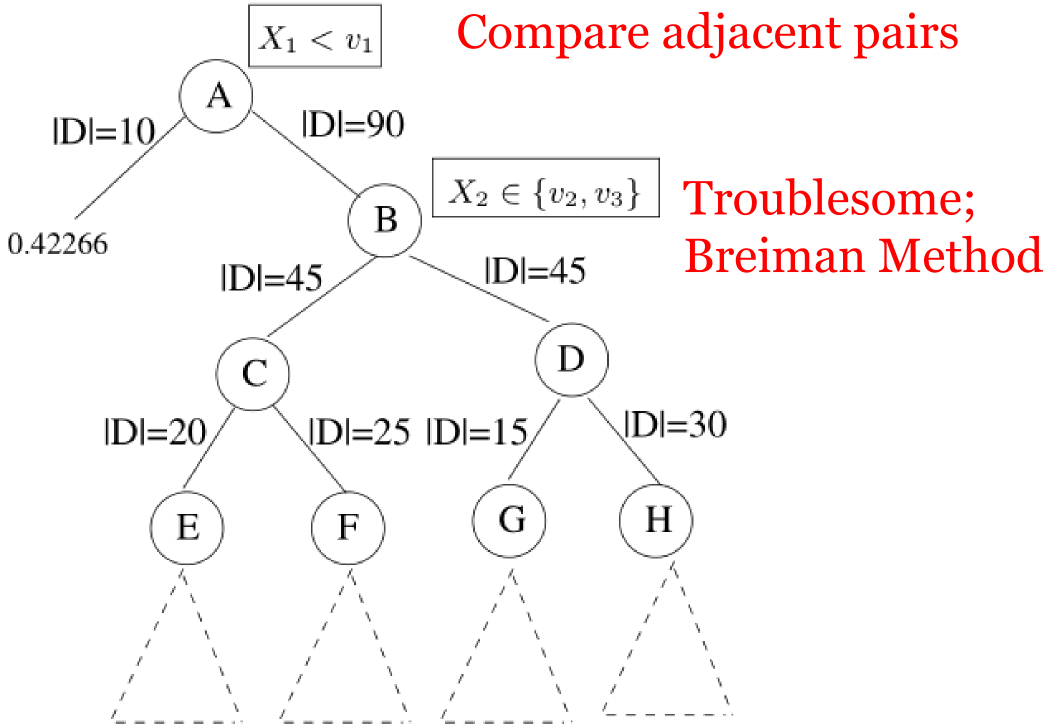
    value = subset of D_k input to *n*

  - loads training records in memory
  - single-machine algorithm to find the *split*

In this way, cluster can process many nodes in parallel to grow the tree

MapReduce Work – LargeData Queue

- Ordered attribute vs. Unordered



MapReduce Work – LargeData Queue

- Map function

Algorithm 2 MR_ExpandNodes::Map

Require: NodeSet N, ModelFile M, Training record $(\mathbf{x}, y) \in D^*$

n = TraverseTree(M, $\mathbf{x}$)
if $n \in N$ **then**
  agg_tup$_n \leftarrow y$
  for all $X \in \mathcal{X}$ **do**
    v = Value on X in $\mathbf{x}$
    if X is ordered **then**
      for all Split point s of X s.t. $s < v$ **do**
        $T_{n,X}[s] \leftarrow y$
    else
      $T_{n,X}[v] \leftarrow y$

Algorithm 3 MR_ExpandNodes::Map_Finalize

Require: NodeSet N

for all $n \in N$ **do**
  Output to all reducers(n, agg_tup$_n$)
  for all $X \in \mathcal{X}$ **do**
    if X is ordered **then**
      for all Split point s of X **do**
        Output($(n, X, s), T_{n,X}[s]$)
    else
      for all $v \in T_{n,X}$ **do**
        Output($(n, X), (v, T_{n,X}[v])$)

MapReduce Work – LargeData Queue

- Reduce function

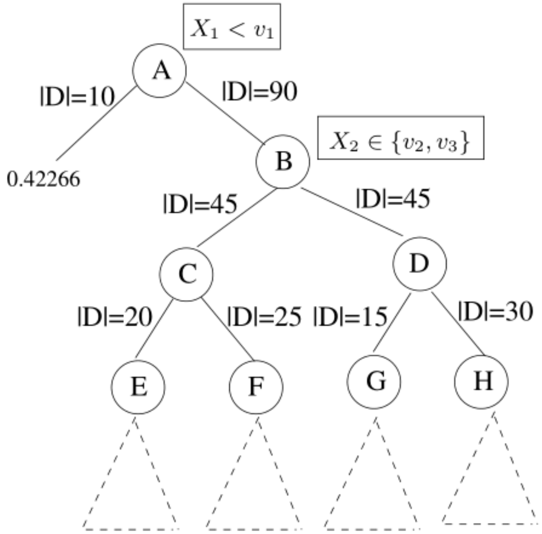
Algorithm 4 MR_ExpandNodes::Reduce

Require: Key k,Value Set V

```
if k == n then
    {Aggregate agg_tup_n's from mappers}
    agg_tup_n = Aggregate(V)
else if k == n, X, s then
    {Split on ordered attribute}
    agg_tup_left = Aggregate(V)
    agg_tup_right = agg_tup_n - agg_tup_left
    UpdateBestSplit(S[n],X,s,agg_tup_left, agg_tup_right)
else if k == n, X then
    {Split on unordered attribute}
    for all v,agg_tup ∈ V do
        T[v] ← agg_tup
    UpdateBestSplit(S[n],BreimanSplit(X,T,agg_tup_n))
```

Walkthrough

- Training set *D**
  - 100 instances
- Memory constraint
  - 25 instances
- Stopping criteria
  - instances <= 10

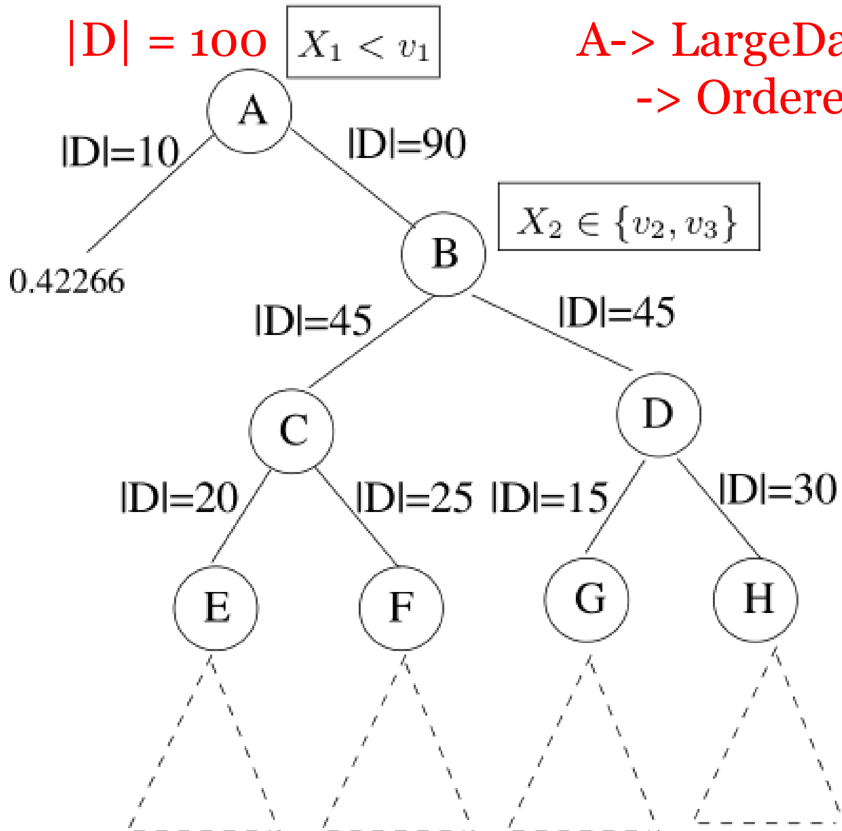


Walkthrough

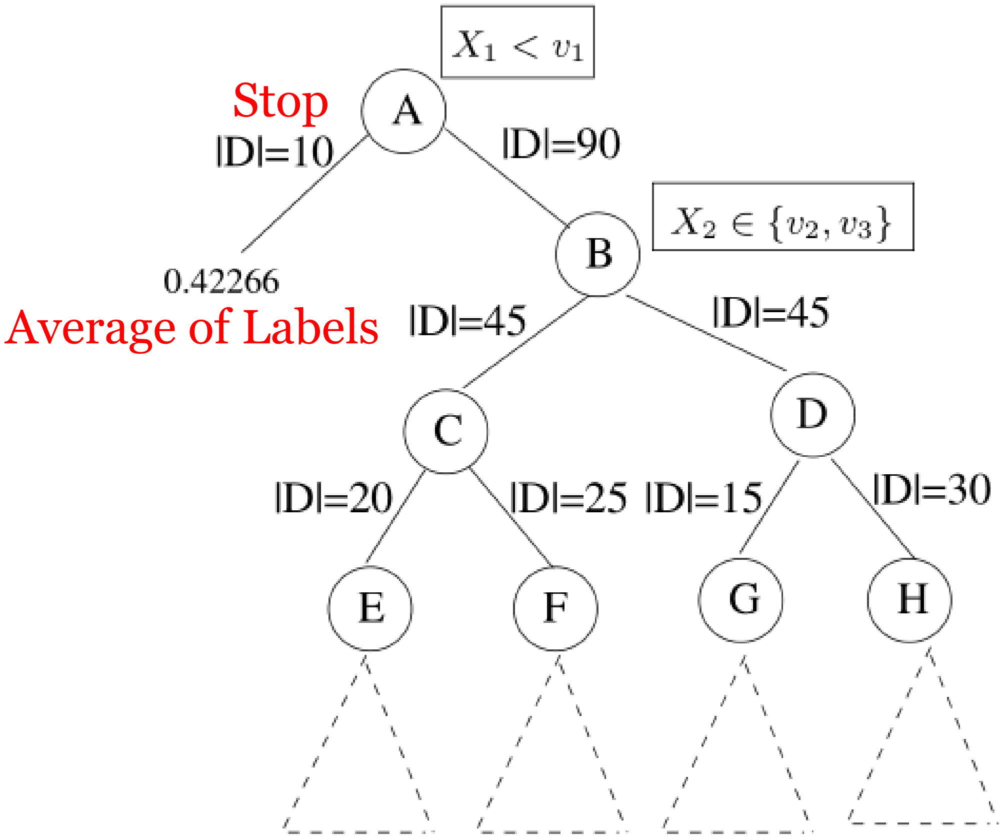
Ordered attribute

|D| = 100

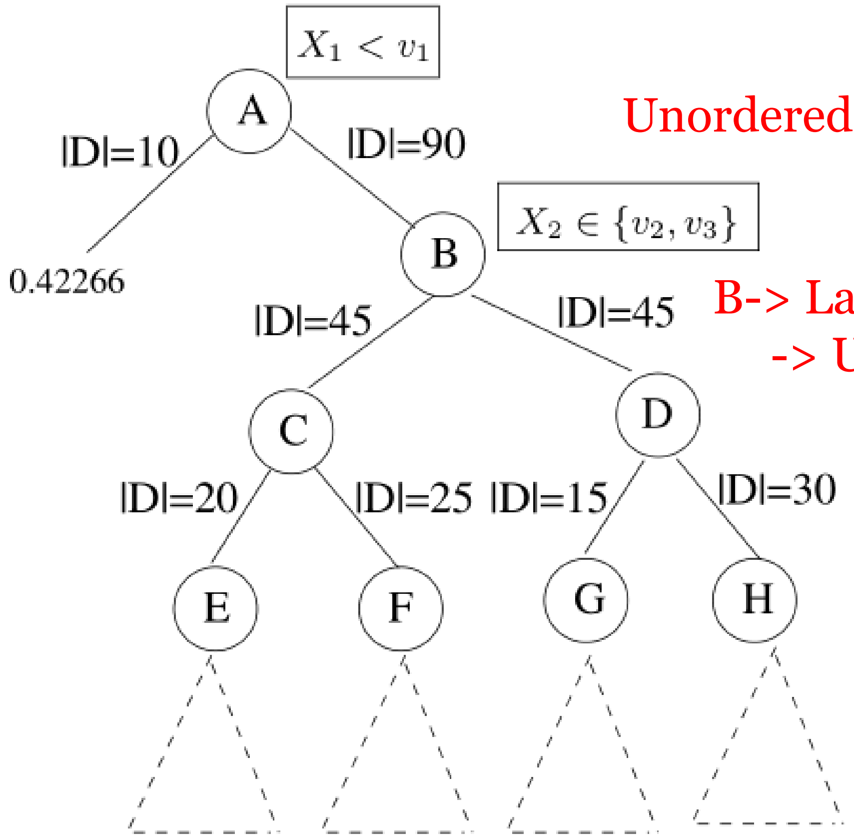
A-> LargeData Queue
-> Ordered node splitting



Walkthrough



Walkthrough

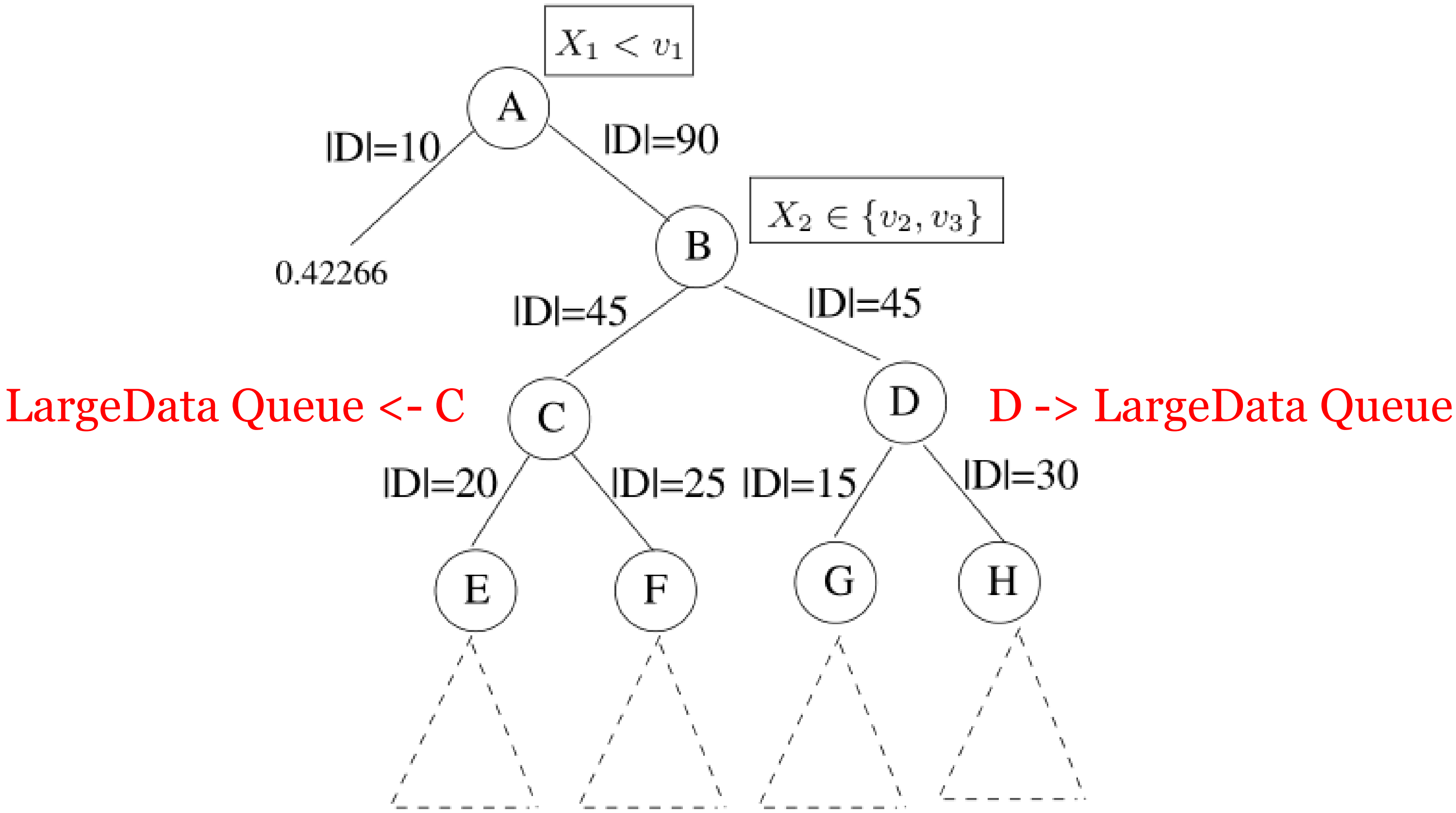


Unordered attribute

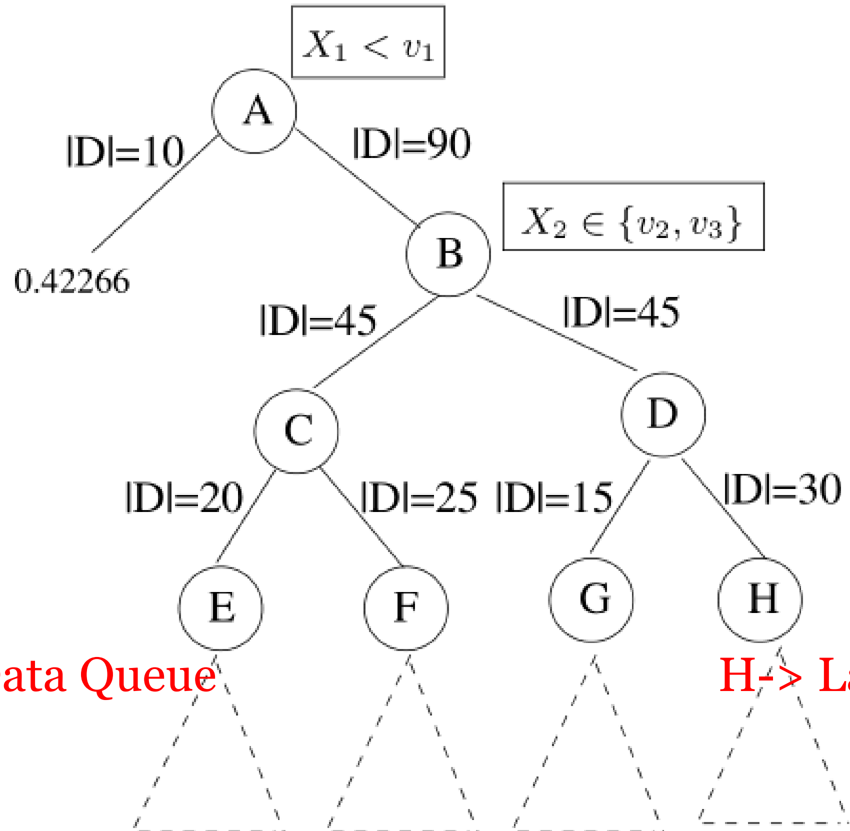
B-> LargeData Queue

-> Unordered node splitting

Walkthrough



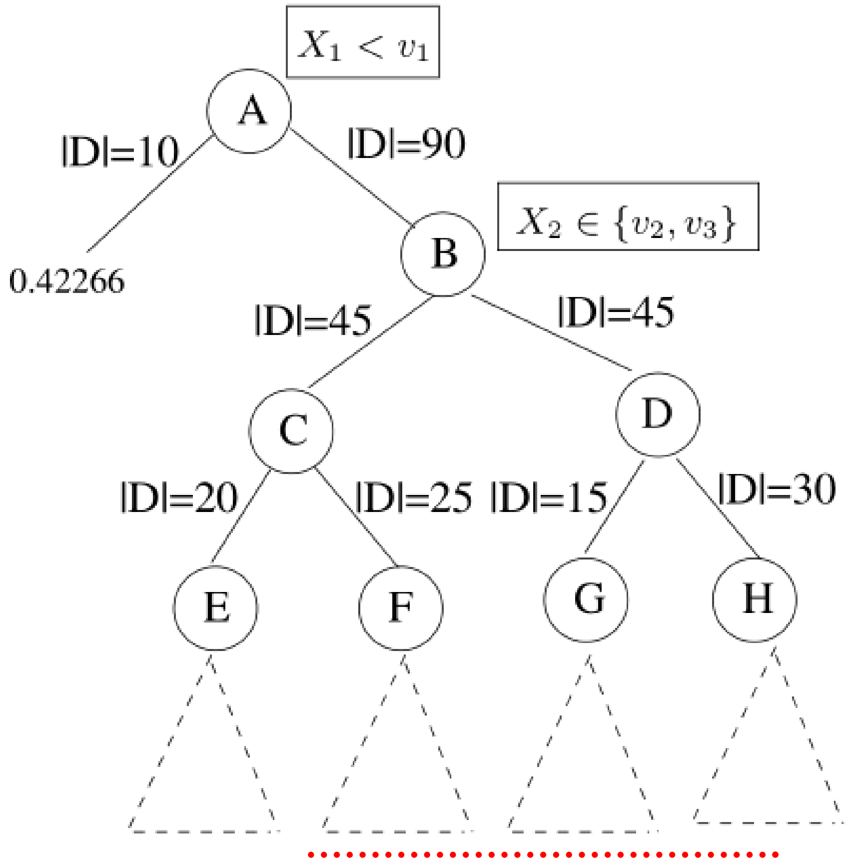
Walkthrough



E, F, G -> SmallData Queue

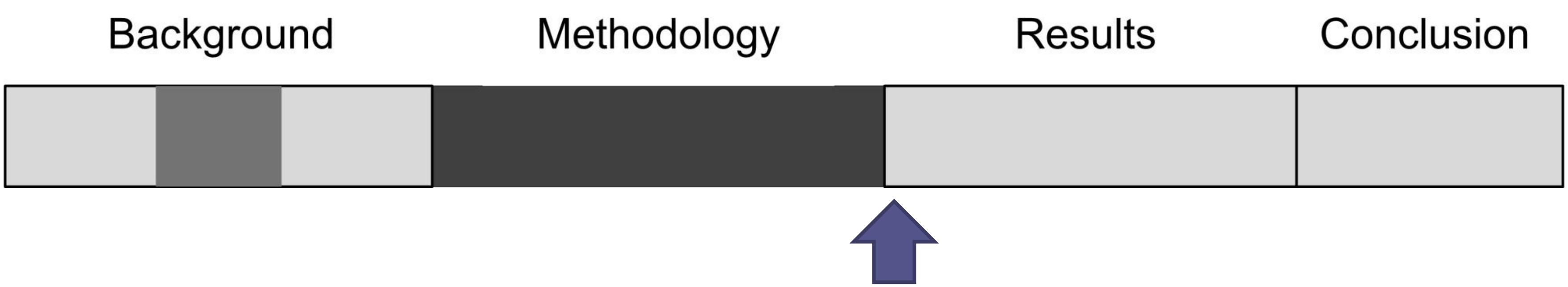
H-> LargeData Queue

Walkthrough



Content

- Background
- Methodology
- Results
- Conclusion



Setup

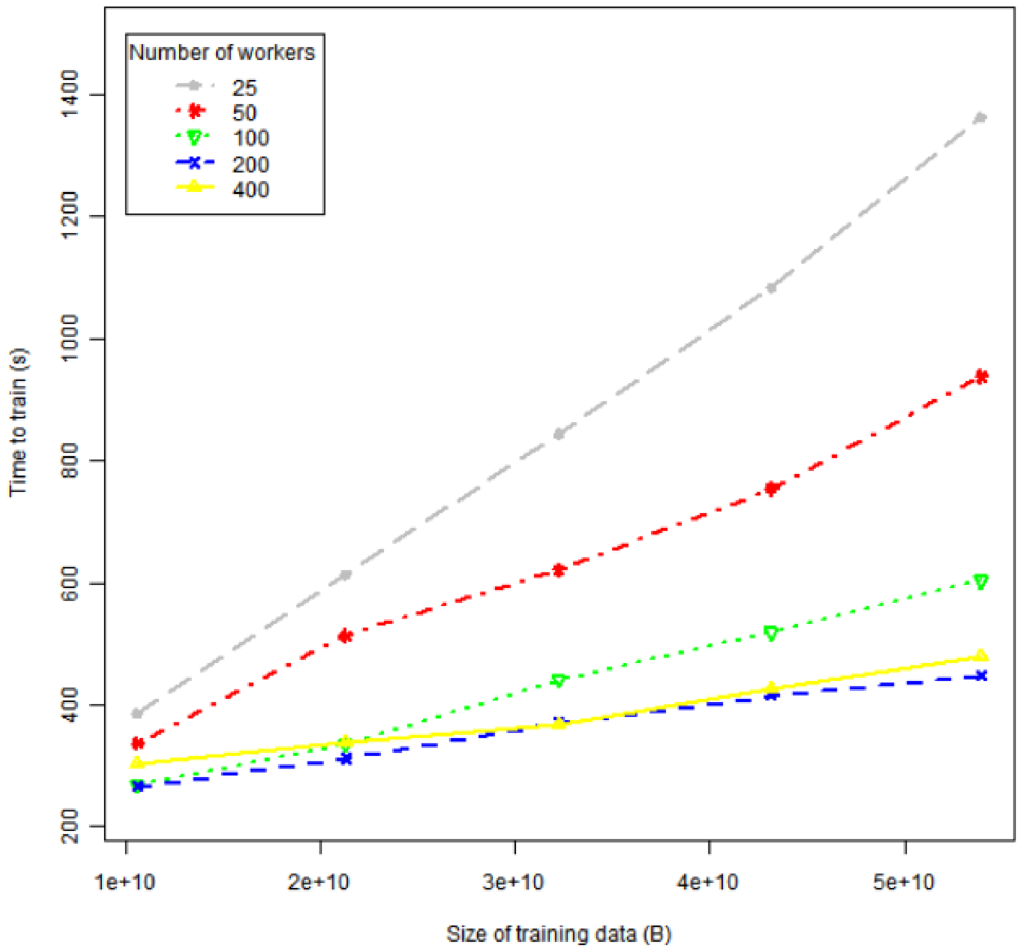
- Bounce rate prediction problem

- 314 million records
  - 10 features
  - 1 label

- Each machine 768MB memory

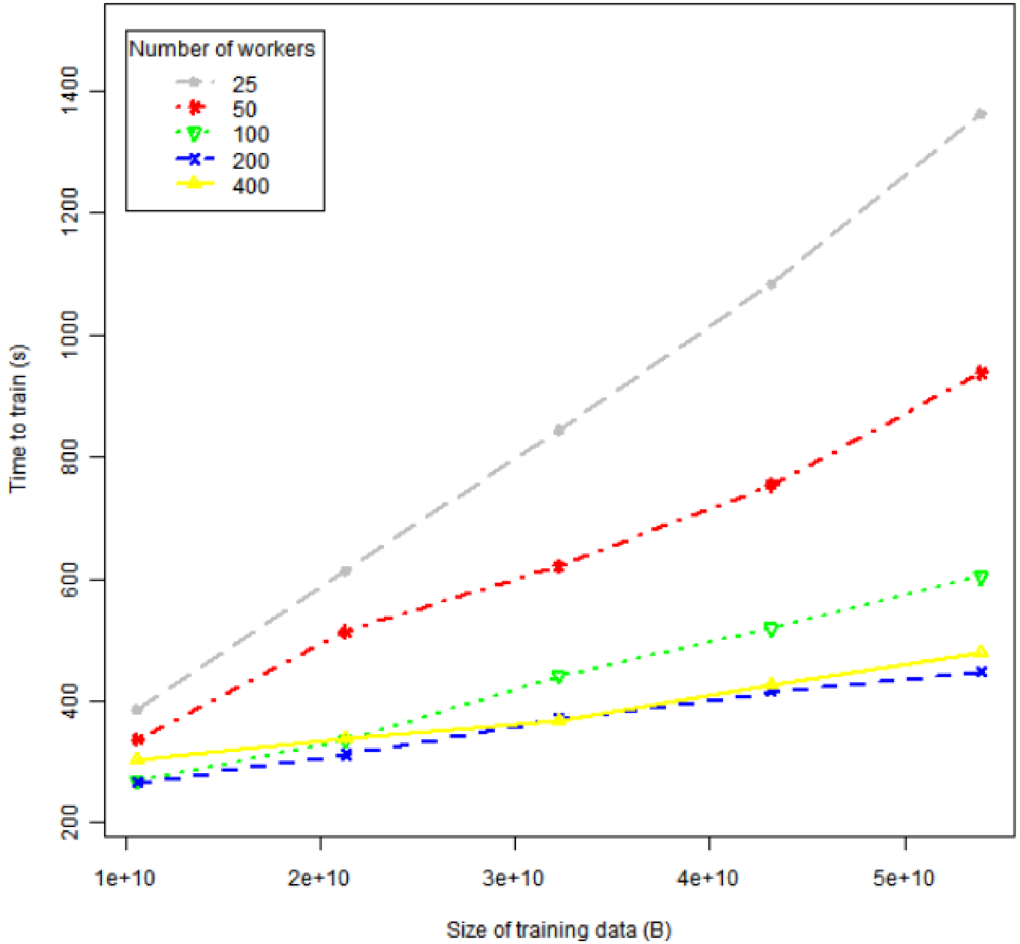
Time to Train vs. Data Size

- Works well
  - 25 nodes



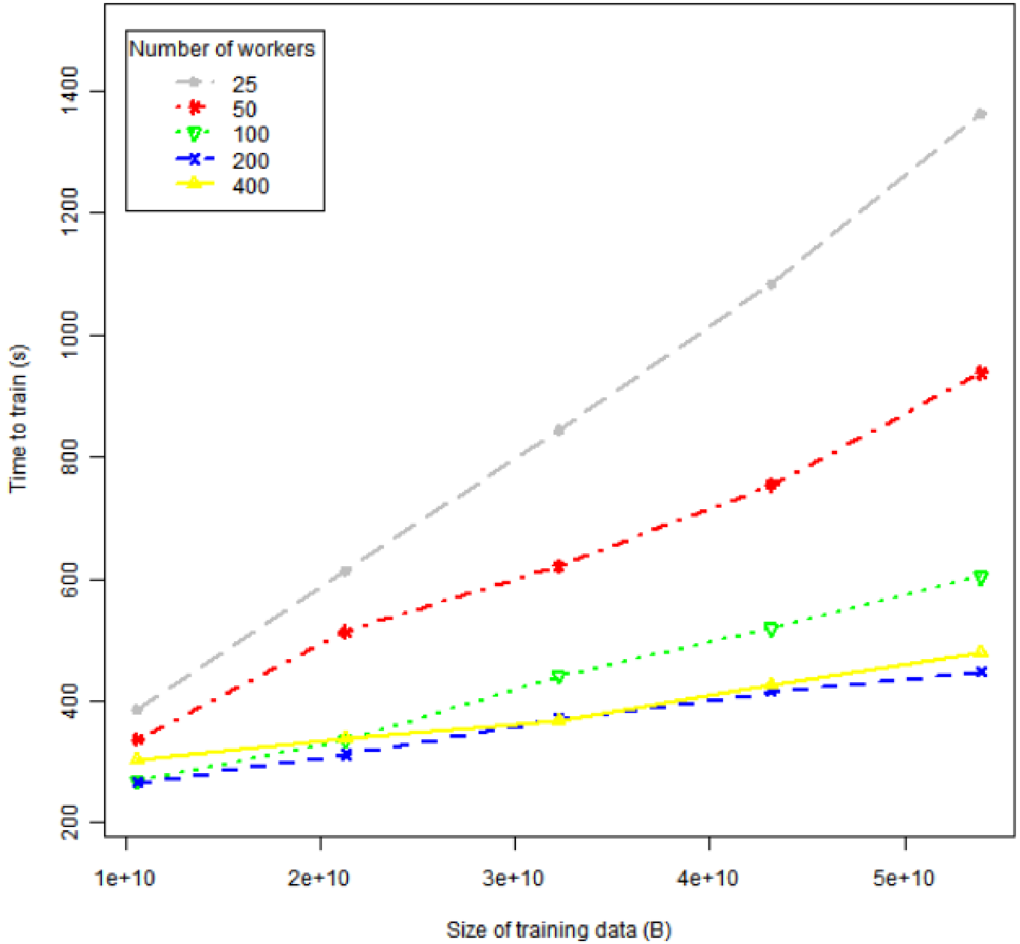
Time to Train vs. Data Size

- Works well
  - 25 nodes
  - 50



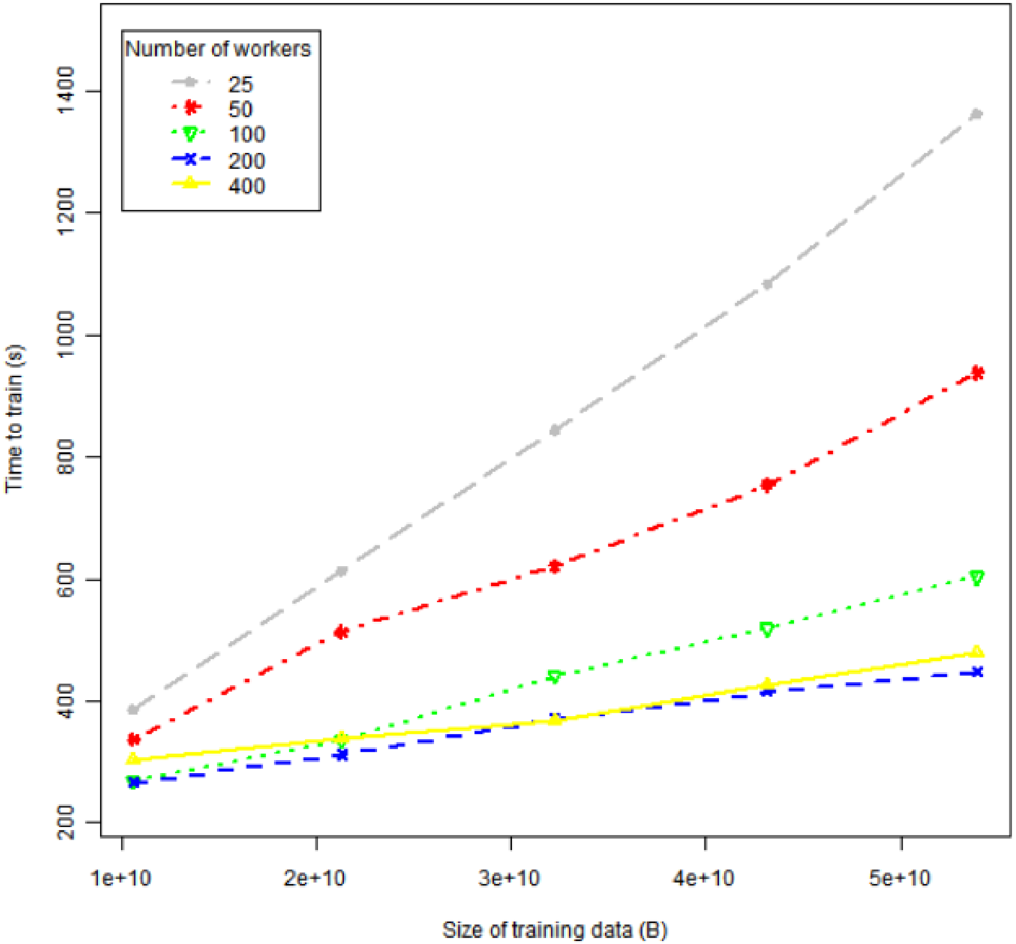
Time to Train vs. Data Size

- Works well
  - 25 nodes
  - 50
  - 100
  - 200



Time to Train vs. Data Size

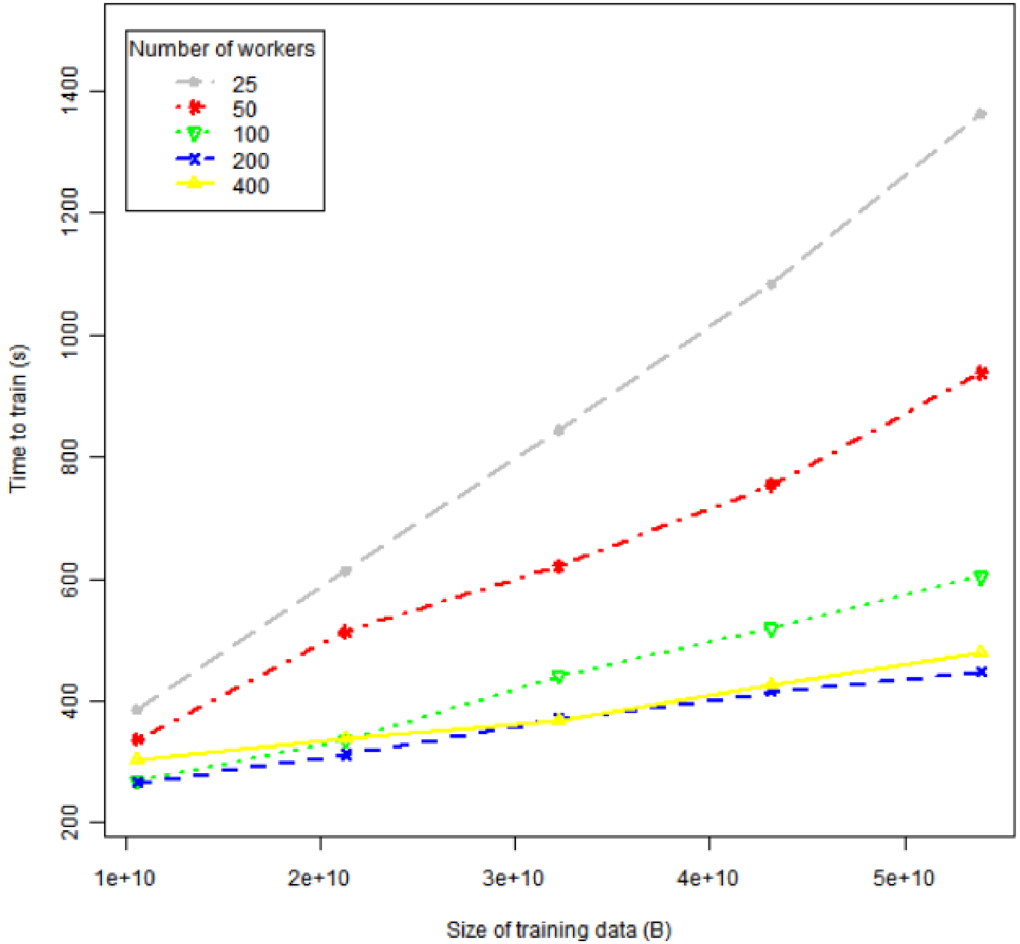
- Works well
  - 25 nodes
  - 50
  - 100
  - 200
  - 400?



Time to Train vs. Data Size

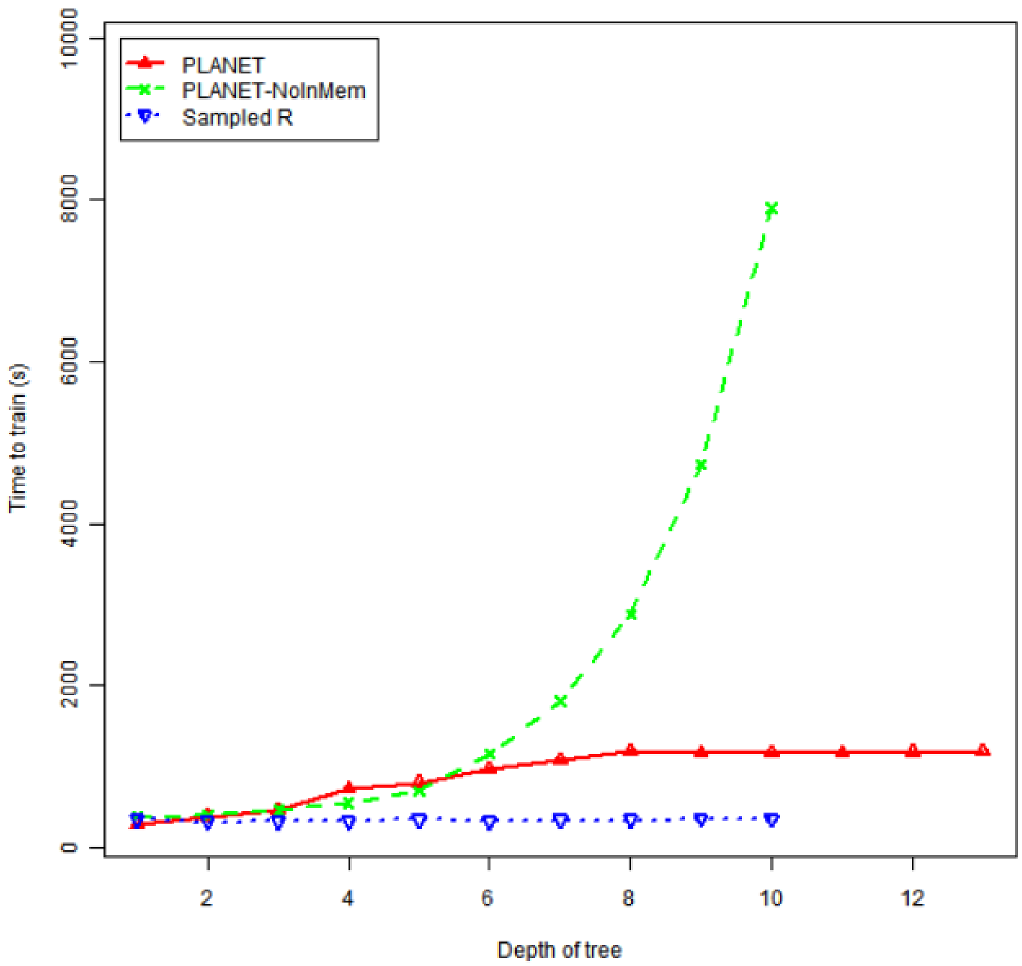
- 400 workers worse than 200?
- Cluster Management

- network overhead
- failure watching
- schedule backups
- data distribution & collection



Time to Train vs. Tree Depth

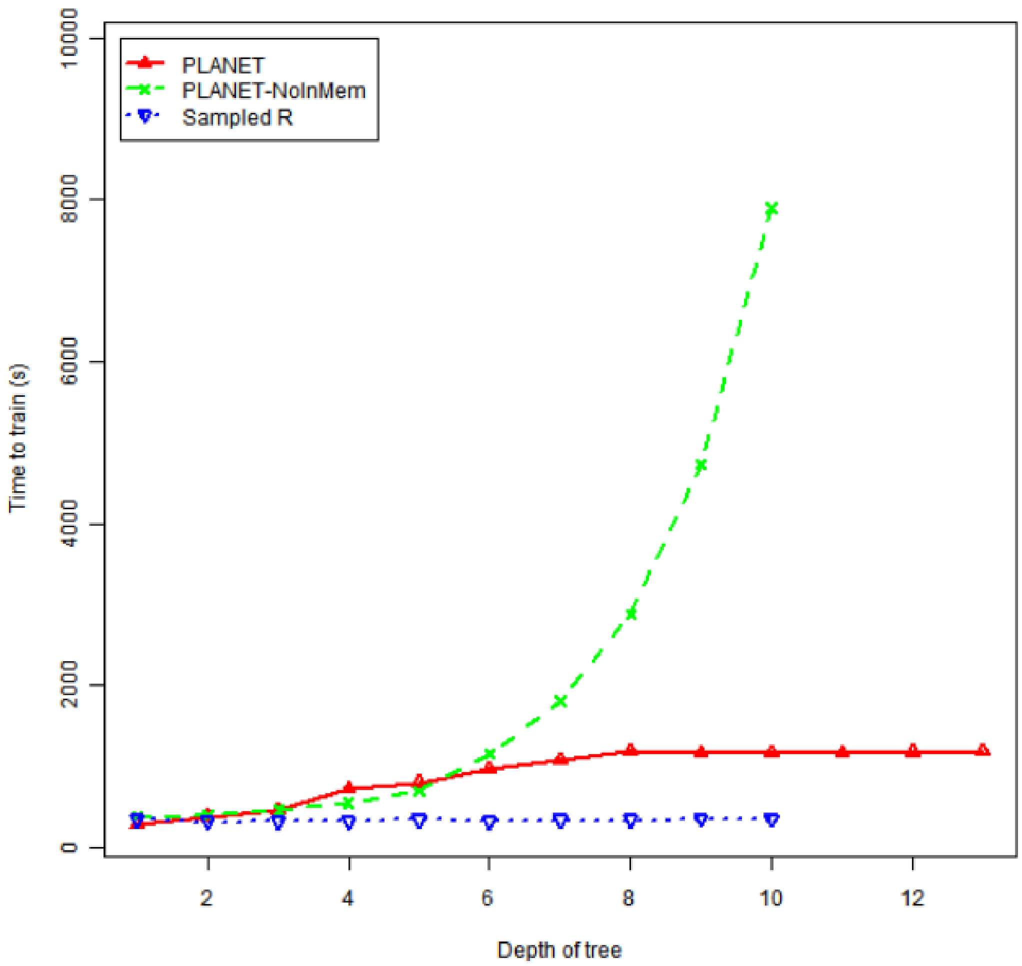
- With/without 'SmallData Queue'



Time to Train vs. Tree Depth

- With/without 'SmallData Queue'

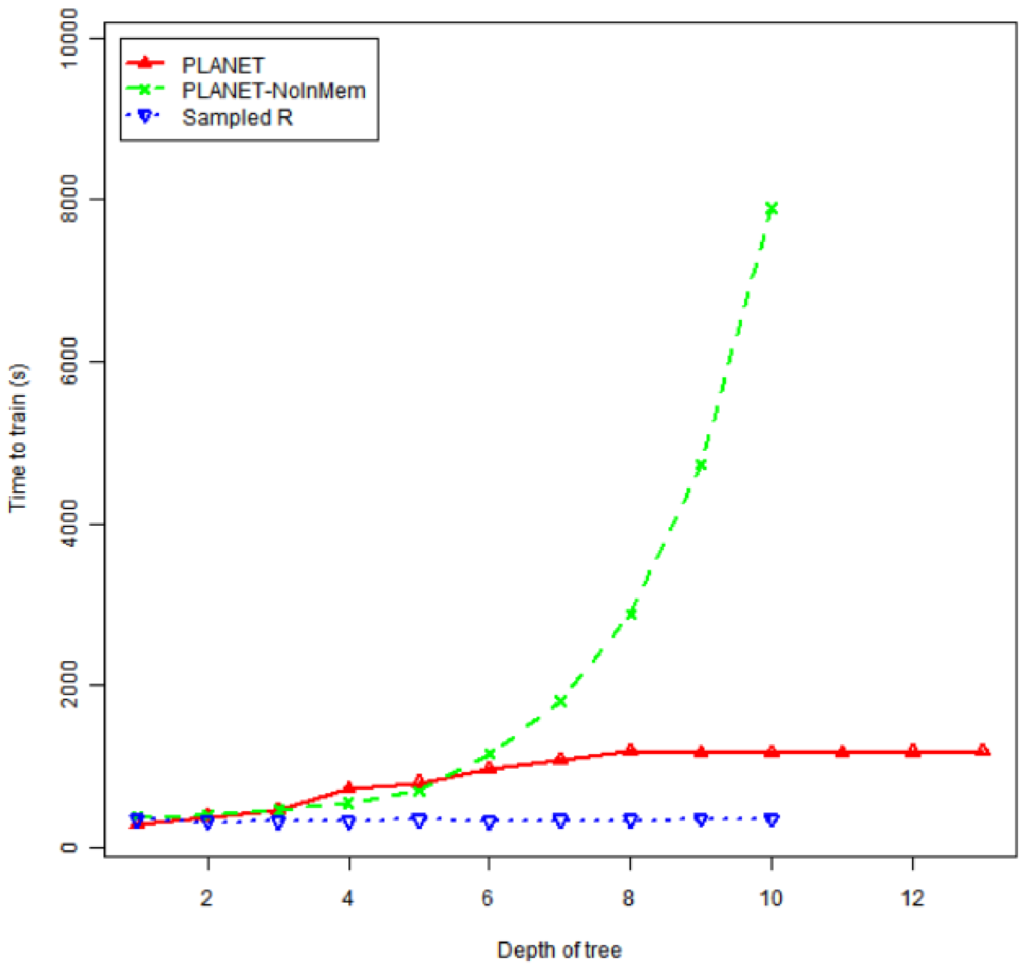
  - overhead of cluster management



Time to Train vs. Tree Depth

- With/without 'SmallData Queue'

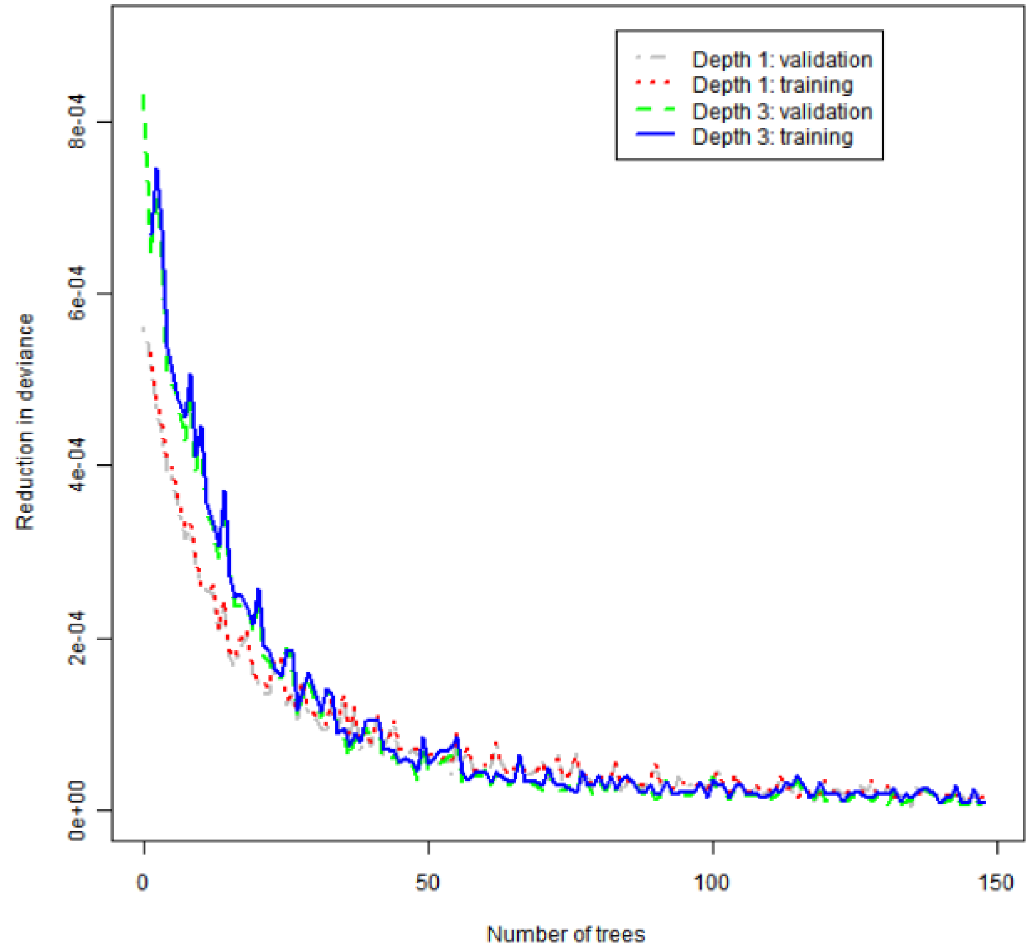
  - overhead of cluster management
  - sampling based method on single machine



Error Reduction vs Num of Trees

- Boosted tree model

  - a bundle of weighted weak learners: better performance

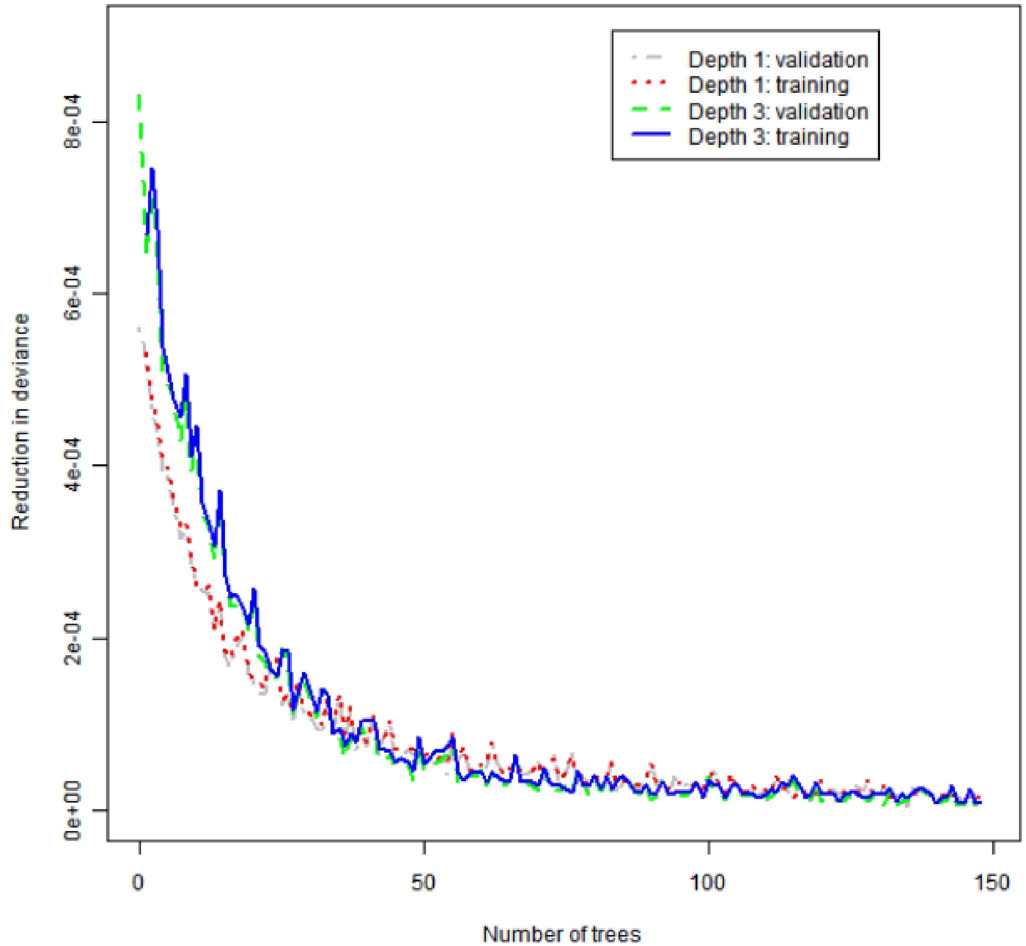


Error Reduction vs Num of Trees

- Boosted tree model

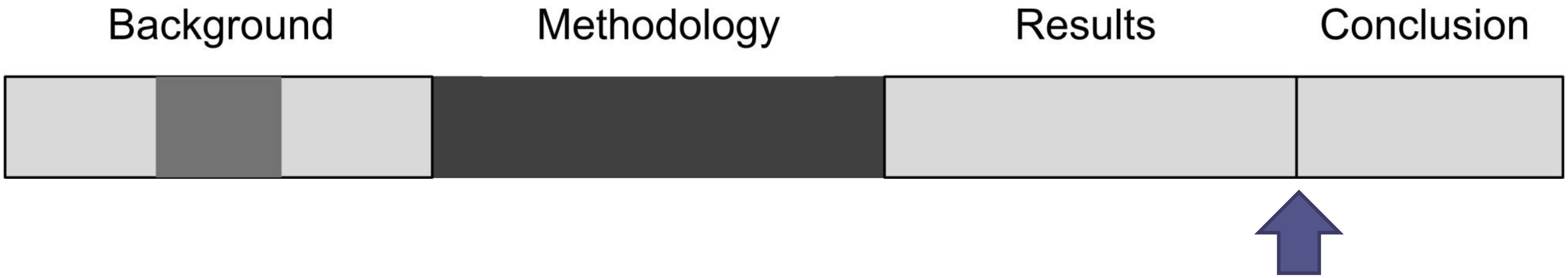
  - a bundle of weighted weak learners: better performance

  - better weak learners faster error reduction



Content

- Background
- Methodology
- Results
- Conclusion



Conclusion

- Successfully scales up tree learning with MapReduce
- Performs well
- Pioneered large-scale machine learning

Bekkerman, Ron, Mikhail Bilenko, and John Langford, eds. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

Related Work – Survey

Learning Setting	Algorithm	Cluster Nodes	Parallelization on Framework	Speedup
Regression	Decision Tree	200	MapReduce	100
Classification	SVM	500	MPI	100
Ranking	LambdaMART	32	MPI	10
Inference	Loopy belief propagation	40	MPI	23
Inference	MCMC	1024	MPI	1000
Clustering	Spectral clustering	256	MapReduce, MPI	256
Clustering	Information-theoretic clustering	400	MPI	100

Go on. I dare you.

Source:
Linkedin
sharing

References

1. Bishop, Christopher M. *Pattern recognition and machine learning*. Vol. 1. New York: springer, 2006.
2. Panda, Biswanath, et al. "Planet: massively parallel learning of tree ensembles with mapreduce." *Proceedings of the VLDB Endowment* 2.2 (2009): 1426-1437.
3. Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.
4. Bekkerman, Ron, Mikhail Bilenko, and John Langford, eds. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.
5. Garcia-Molina, Hector. *Database systems: the complete book*. Pearson Education India, 2008.