PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Authors: B. Panda, J. S. Herbach, S. Basu, R. J. Bayardo.

VLDB 2009

CS 422

Decision Trees: Main Components

- Find Best Split
  - Choose split point that minimizes weighted impurity (e.g., variance (regression) and information gain (classification) are common)
- Stopping Criteria (common ones)
  - Maximum Depth
  - Minimum number of data points
  - Pure data points (e.g., all have the same Y label)

PLANET uses an impurity measure based on variance (regression trees). The higher the variance in Y values of a node → the greater the node's impurity.

InMemoryBuildNode: Greedy Top-Down Algorithm

Algorithm 1 InMemoryBuildNode

Require: Node n, Data $D \subseteq D^*$

1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$
2: **if** StoppingCriteria(D_L) **then**
3: $\quad n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$
4: **else**
5: $\quad \text{InMemoryBuildNode}(n \rightarrow \text{left}, D_L)$
6: **if** StoppingCriteria(D_R) **then**
7: $\quad n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: **else**
9: $\quad \text{InMemoryBuildNode}(n \rightarrow \text{right}, D_R)$

Most important step

This algorithm does not scale well for large data sets

FindBestSplit

- Continuous attributes:
  - Treating a point as a boundary (e.g., <5.0 or >=5.0)
- Categorical attributes
  - Membership in a set of values (e.g., is the attribute value one of {Ford, Toyota, Volvo}?)

FindBestSplit(D): Variance

- Let **Var(D)** be the variance of the output attribute **Y** measured over all records in **D** (D refers to the records that are input to a given node). At each step the tree learning algorithm picks a split which maximizes:

$$|D| \times Var(D) - (|D_L| \times Var(D_L) + |D_R| \times Var(D_R)), \quad (1)$$

- Var(D) is the variance of the output attribute Y measured over all records in D
- D_L and D_R are the training records in the left and right subtree after splitting D by a predicate

SoppingCriteria(D)

- A node in the tree is not expanded if the number of records in D falls below a threshold.

FindPrediction(D)

- The prediction at a leaf is simply the average of all the Y values in D.

Problem Motivation

- Large data sets
  - Full scan over the input data is slow (required by FindBestSplit)
  - Large inputs that do not fit in memory (cost of scanning data from a secondary storage)
  - Finding the best split on high dimensional data sets is slow (2^M possible splits for categorical attribute with M categories)

Previous Work

- Two previous approaches for scaling up tree learning algorithms:
  - Centralized algorithms for large data sets on disk
  - Parallel algorithms on specific parallel computing architectures
- PLANET is based on the MapReduce platform that uses commodity hardware for massive-scale parallel computing

How does PLANET work?

- Controller
  - Monitors and controls everything
- MapReduce initialization task
  - Identifies all feature values that need to be considered for splits
- **MapReduce FindBestSplit task (the main part)**
  - MapReduce job to find best split when there's too much data to fit in memory
- MapReduce inMemoryGrow task
  - Task to grow an entire subtree once the data for it fits in memory (in memory MapReduce job)
- Model File
  - A file describing the state of the model

PLANET MapReduce

- Map phase:
  - The system partitions D* (the entire training data) into a set of disjoint units which are assigned to workers, known as mappers
  - In parallel, each mapper scans through its assigned data and applies a user-specified map function to each record. The output is a set of <key, value> pairs which are collected for the Reduce phase.
- Reduce phase:
  - The <key, value> pairs are grouped by key and distributed among workers called reducers
  - Each reducer applies a user-specified reduce function and outputs a final value for a key.

Different MapReduce jobs build different parts of the tree

Main Components: The Controller

- Controls the entire process
- Periodically checkpoints system **Important in real production systems**
- Determine the state of the tree and grows it
  - Decides if nodes should be split
  - If there's relatively little data entering a node, launch an InMemory MapReduce job to grow the entire subtree
  - For larger nodes, launches MapReduce to find candidates for best split
  - Collects results from MapReduce jobs and chooses the best split for a node
  - Updates model

Main Components: The Model File (M)

- **Model File (M)**
  - The controller constructs a tree using a set of MapReduce jobs that are working on different parts of the tree. At any point, **the model file contains the entire tree constructed so far**.

The controller checks with the model file the nodes at which split predicates can be computed next. For example, if M has nodes A and B, then the controller can compute splits for C and D.

Tree Construction using PLANET: Example

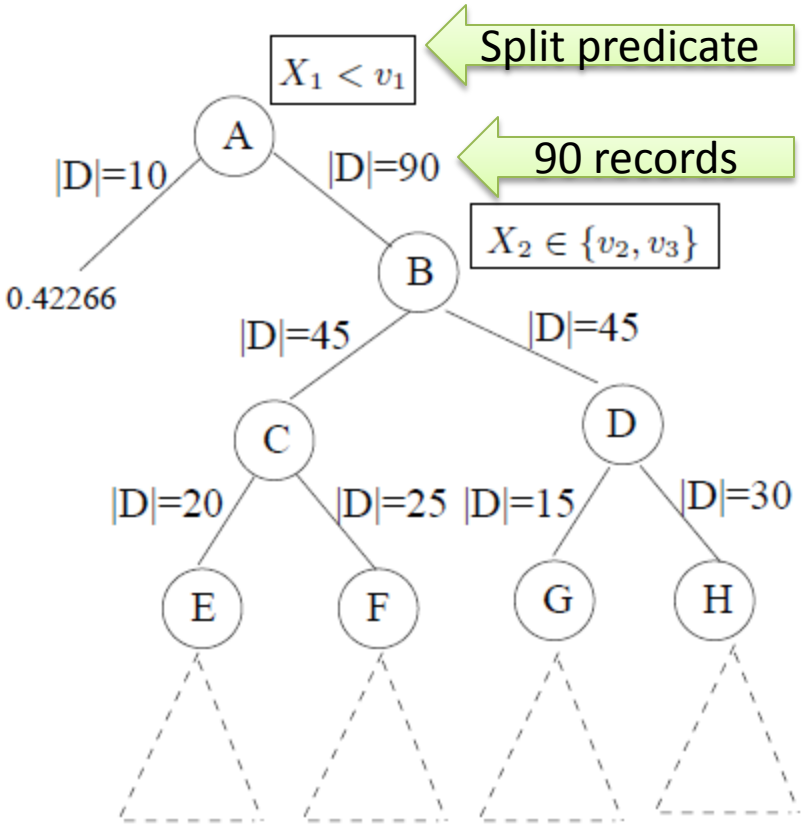
Using PLANET to construct the tree:

Assumptions:

(1) D* with 100 records;

(2) Tree induction stops once the # of training records at a node falls below 10;

(3) A memory constraint limiting the algorithm to operating on inputs of size 25 records or less.



Two Node Queues

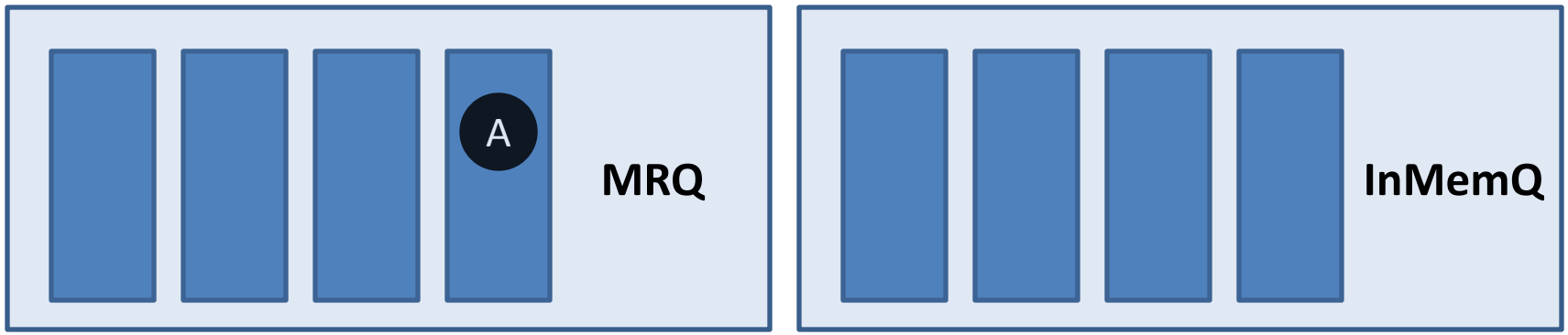
- **MapReduceQueue (MRQ)**
  - Contains nodes for which D is too large to fit in memory (i.e., >25 in our example)
- **InMemoryQueue (InMemQ)**
  - Contains nodes for which D fits in memory

Two MapReduce Jobs

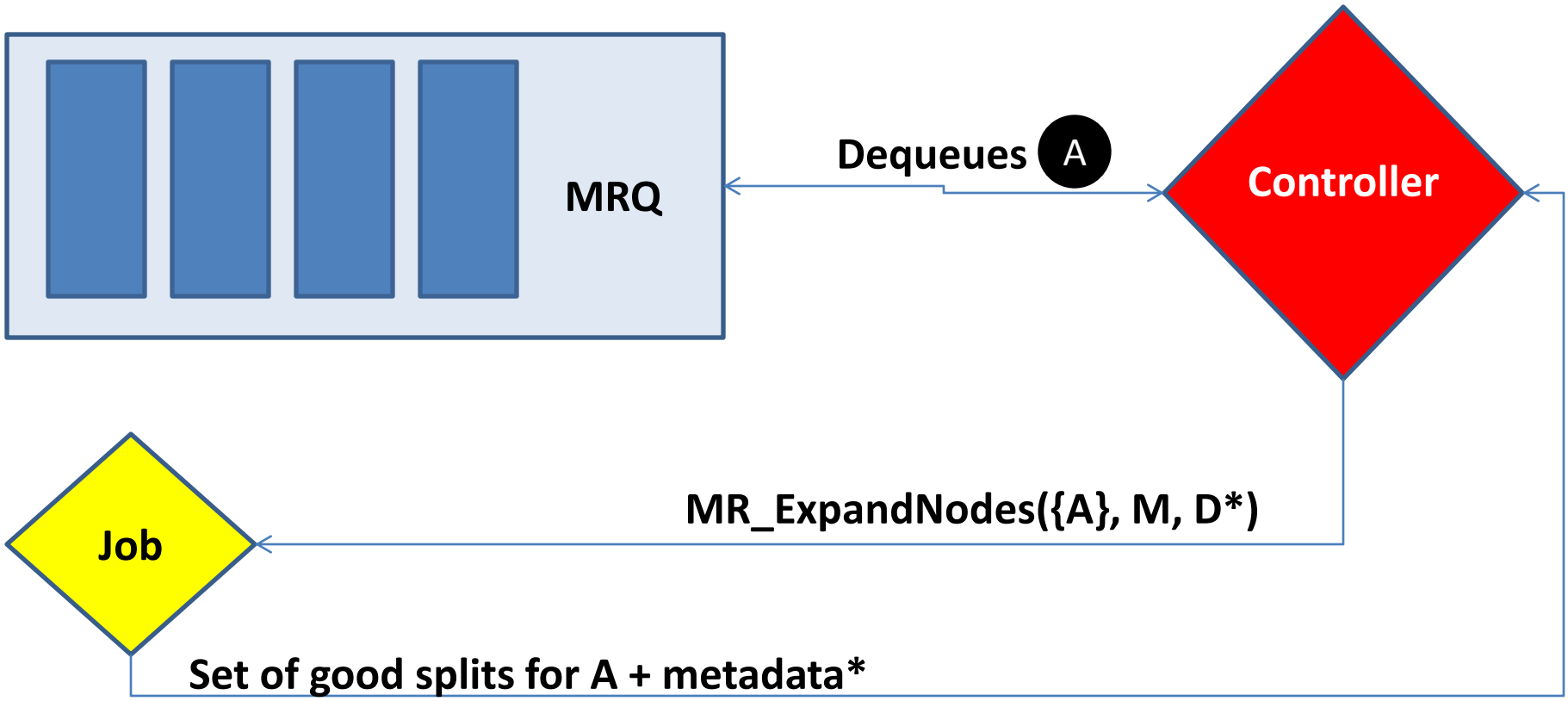
- **MR_ExpandsNodes**
  - Process nodes from the MRQ. For a given set of nodes N, computes a candidate of good split predicate for each node in N.
- **MR_InMemory**
  - Process nodes from the InMemQ. For a given set of nodes N, completes tree induction at nodes in N using the InMemoryBuildNode algorithm.

Walkthrough

1. Initially: M, MRQ, and InMemQ are empty.
   - The controller can only expand the root.
   - Finding the split for the root requires the entire training set D* of 100 records (does not fit in memory).
2. A is pushed onto MRQ; InMemQ stays empty.

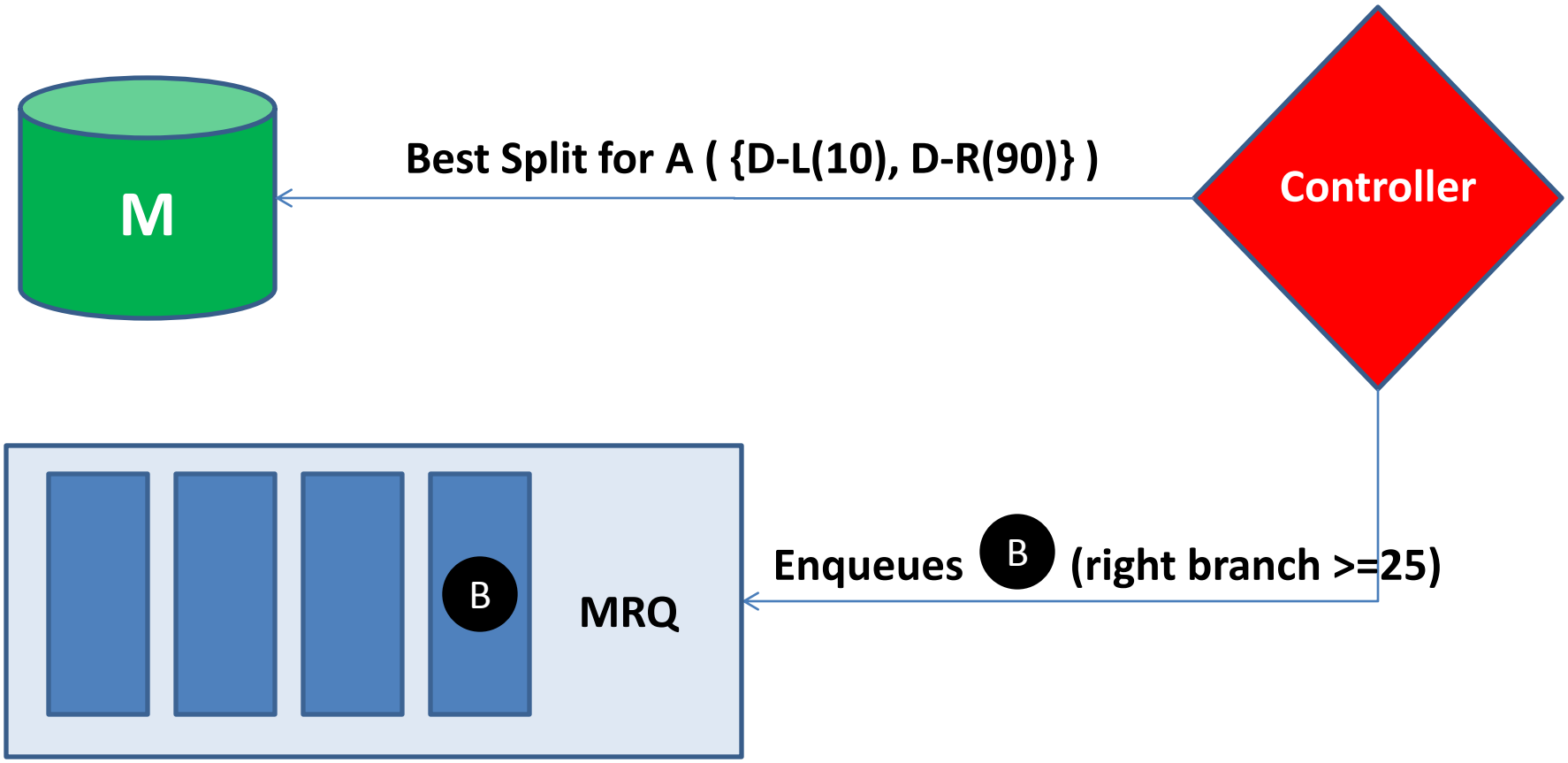


Scheduling the First MapReduce Job



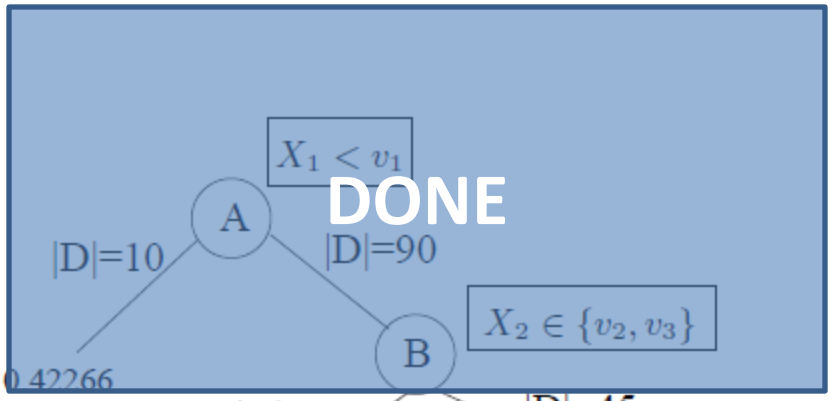
*** (1) Quality of the split; (2) The predictions in left and right branches); and (3) The # of training records in the left and right branches.**

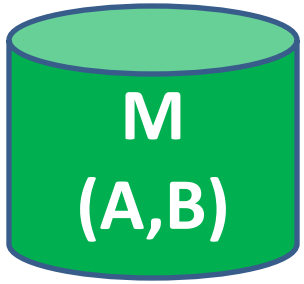
The Controller Selects the Best Split

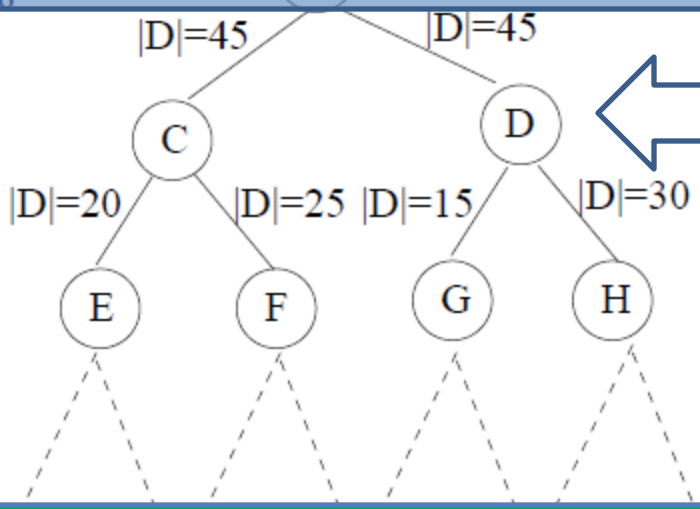


No new nodes are added to the left branch since it matches the stopping criteria (10 records)

Expanding C and D





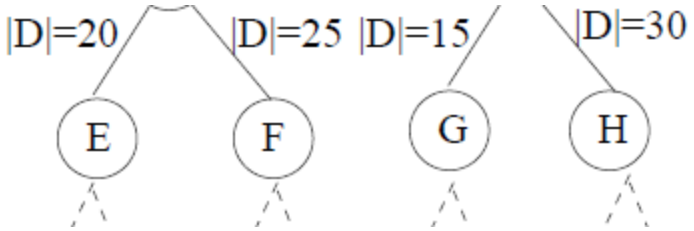


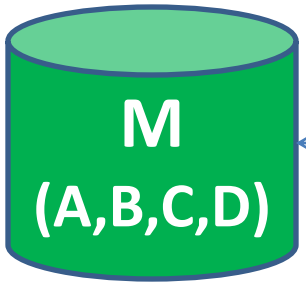
MR_ExpandNodes({C, D}, M, D*}

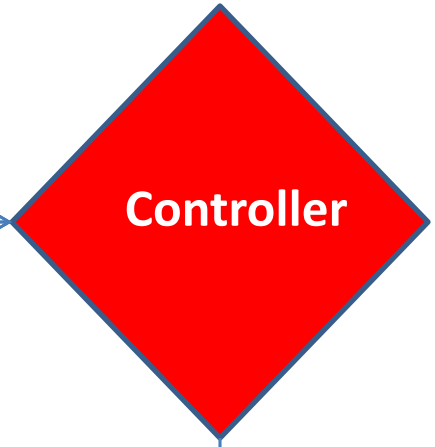
Breadth First Expanding

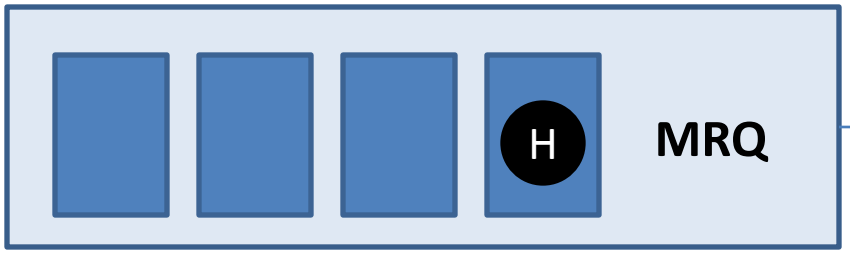
The controller schedules a single MR_ExpandNodes for C and D that is done in a single step → PLANET expands trees breadth first as opposed to the depth first process used by the inMemory algorithm.

Scheduling Jobs Simultaneously

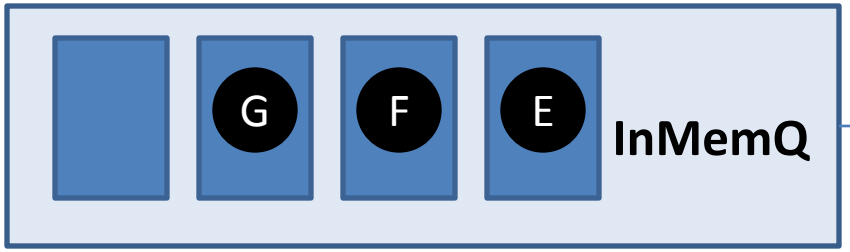








Dequeues H



Dequeues E,F, and G

MR_ExpandNodes({H}),M, D*)

MR_InMemory({E, F, G}),M, D*)

Once both jobs return, the **subtrees rooted at G, F, and E are complete**. The controller continues the process with the children of H only.

Main Controller Thread

Algorithm 8 MainControllerThread

```
Require: Model M = ∅, MRQ=∅, InMemQ=∅
 1: MRQ.append(root)
 2: while true do
 3:    while MRQ not empty do
 4:       if TryReserveClusterResources then
 5:          jobs_running ++
 6:          NewThread(ScheduleMR_ExpandNode(⊆MRQ,M))
 7:    while InMemQ not empty do
 8:       if TryReserveClusterResources then
 9:          jobs_running ++
10:          NewThread(ScheduleMR_InMemory(⊆InMemQ,M))
11:    if jobs_running==0 && MRQ empty && InMemQ
       empty then
12:       Exit
```

1. The thread schedules jobs off of its queues until no jobs are running and queues are empty
2. Scheduled MapReduce jobs are launched in separate threads so the controller can send those in parallel
3. When a MR_ExpandNode job returns, the queues are updated with the new nodes to expand (next algorithm)

Scheduling

Algorithm 6 Schedule_MR_ExpandNode

Require: NodeSet N,Current Model M

1: CandidateGoodSplits = MR_ExpandNodes(N,M,D^*)
2: **for all** $n \in N$ **do**
3: $n \rightarrow$split,$n \rightarrow$l_pred, $n \rightarrow$r_pred,$|D_L|$,$|D_R|$ = FindBestSplit(n, CandidateGoodSplits)
4: UpdateQueues($|D_L|$,$n \rightarrow$left)
5: UpdateQueues($|D_R|$,$n \rightarrow$right)
6: jobs_running - -

1. Schedules an MR_ExpandNodes job to find the best split for the nodes that were just dequeued
2. Update the queues with the new nodes and number of relevant records

Algorithm 7 Schedule_MR_InMemory

Require: NodeSet N,Current Model M

1: MR_InMemory(N,M,D)
2: jobs_running - -

1. MR_InMemory completes the construction of the subtree rooted at every node in M

Updating Queues

Algorithm 5 UpdateQueues

Require: DataSetSize $|D|$, Node n

```
1: if not StoppingCriteria(|D|) then
2:     if |D| < in_memory_threshold then
3:         InMemQ.append(n)
4:     else
5:         MRQ.append(n)
```

1. If the number of records is smaller than the stopping threshold, then the branch is done
2. Otherwise, updates the appropriate queue (based on the size of D) with the node that needs to be scheduled for a new MapReduce job (expanding)

MR_ExpandNodes

- **Map phase**:
  - D* is partitioned across a set of mappers
  - Each mapper loads into memory the current model M and the input nodes N
  - Every MapReduce job scans the entire training set applying a Map function to every record

MR_ExpandNodes: Map

Algorithm 2 MR_ExpandNodes::Map

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

```
1: n = TraverseTree(M, x)
2: if n ∈ N then
3:    agg_tup_n ← y
4:    for all X ∈ 𝒳 do
5:       v = Value on X in x
6:       if X is ordered then
7:          for all Split point s of X s.t. s < v do
8:             T_{n,X}[s] ← y
9:       else
10:         T_{n,X}[v] ← y
```

Determines if the record is relevant to N

Next is to evaluate possible splits for a node

Iterating over the attributes X

Ordered attributes: the reduction in variance is computed for every adjacent pair of values

Unordered attributes: keys correspond to unique values of X

T is a tables of key-value pairs.
Keys: the split points to be considered for X
Values: tuples of the form $\{\sum y, \sum y^2, \sum 1\}$ -- used by the reducers to evaluate the split

MR_ExpandNodes

- **Reduce phase:**
  - Performs aggregations and computes the quality of each split being considered for nodes in N
  - Each reducer maintains a table S indexed by nodes. S[n] contains the best split seen by the reducer for node n

MR_ExpandNodes: Reduce

Algorithm 4 MR_ExpandNodes::Reduce

Require: Key k,Value Set V

1: **if** $k == n$ **then**
2: {Aggregate agg_tup$_n$'s from mappers}
3: agg_tup$_n$ = Aggregate(V)
4: **else if** $k == n, X, s$ **then**
5: {Split on ordered attribute}
6: agg_tup$_{left}$ = Aggregate(V)
7: agg_tup$_{right}$ = agg_tup$_n$ - agg_tup$_{left}$
8: UpdateBestSplit($S[n]$,X,s,agg_tup$_{left}$, agg_tup$_{right}$)
9: **else if** $k == n, X$ **then**
10: {Split on unordered attribute}
11: **for all** v,agg_tup $\in$ V **do**
12: $T[v] \leftarrow$ agg_tup
13: UpdateBestSplit($S[n]$,BreimanSplit(X,T,agg_tup$_n$))

1. Iterating over all splits
2. For each node, update the best split found together with the relevant statistics collected

InMemoryGrow MapReduce

- Task to grow the entire subtree once the data for it fits in memory
- Map
  - Initialize by loading current model file
  - For each record identify the node and id the node needs to be grown, output <Node_id, Record>
- Reduce
  - Initialize by attribute file from Initialization task
  - For each <Node_id, List<Record>> run the basic tree growing algorithm on the records
  - Output the best split for each node in the subtree

References

- B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. PLANET: Massively parallel learning of tree ensembles with MapReduce. In Proceedings of the 35th International Conference on Very Large Data Base (VLDB 2009), pages 1426{1437, Lyon, France, 2009.
- Josh Herbach: PLANET, MapReduce, and Decision Trees. The Association for Computing Machinery.
- T. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.
- G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In International Conference on ACM Special Interest Group on Management of Data (SIGMOD), pages 251–262, 1999.