

Parallel and Distributed GBT

9/18/2018

Bo Peng

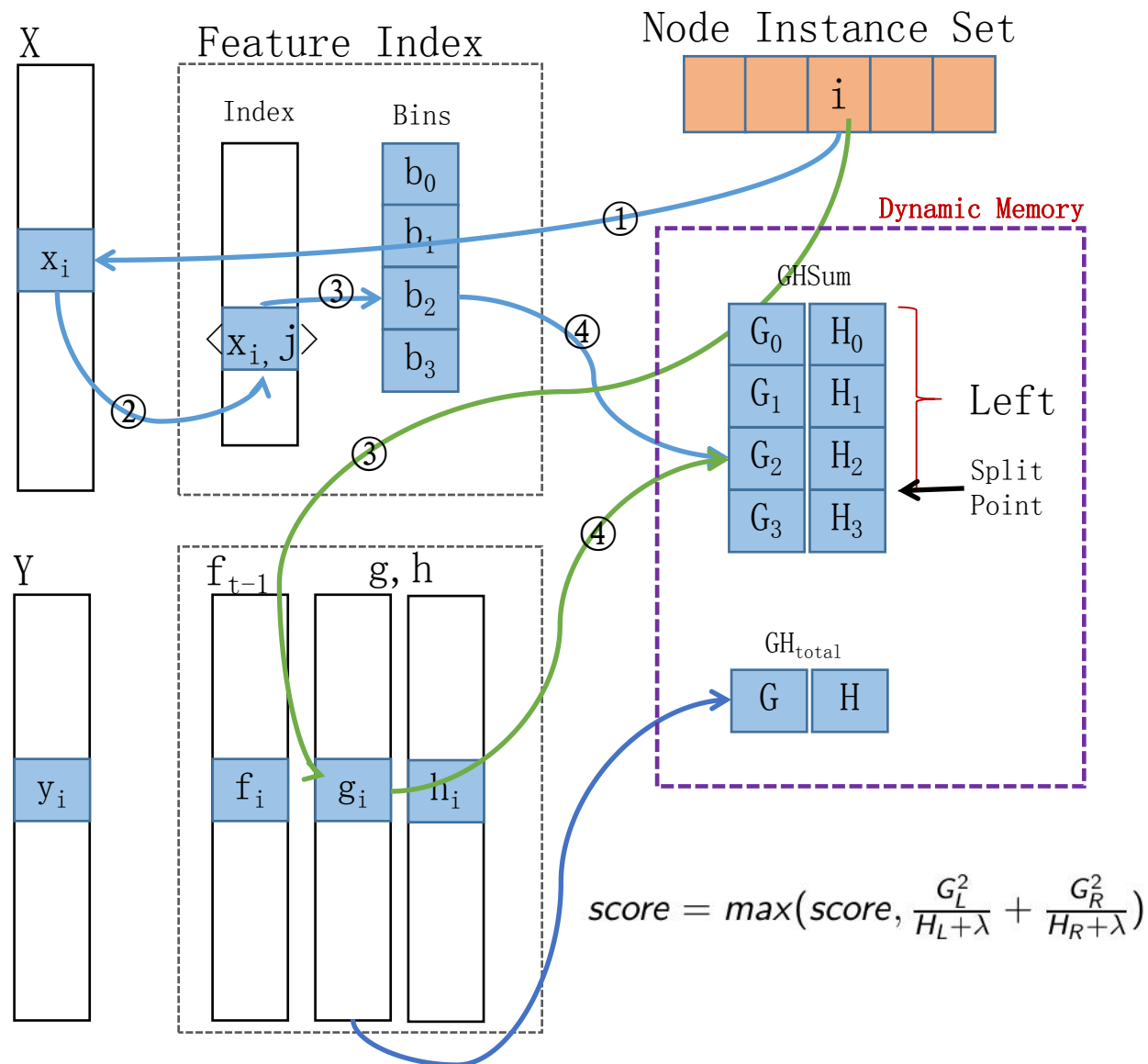
Digital Science Center

Indiana University

Outline

- Problem
- Related work
- Proposal

Single Feature

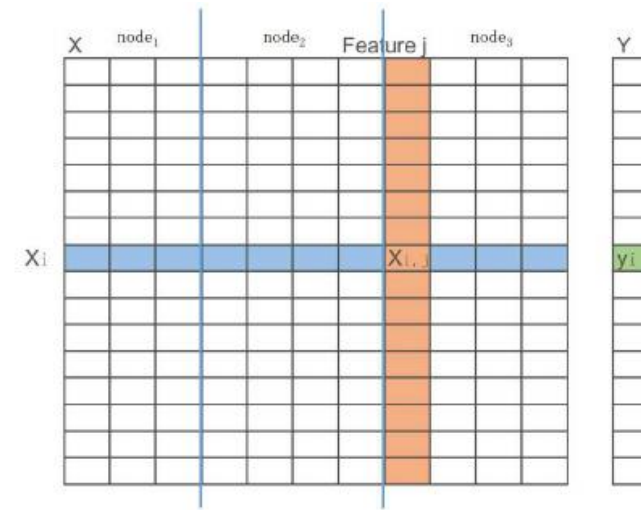


- `findBestSplit` is the bottleneck

- **GHSum** is the core data structure

- for each feature
- bins, fixed split points
- g,h summation on instances whose value fall into the bin

Distributed Split



- Partition by rows (samples)
 - need global communication to build **Bins**, once if use global static Bins.
 - need global communication to build **GHSum** for each feature in `findBestSplit()` → `allreduce(GHSum)`
- Partition by columns (features)
 - **Bins** and **GHSum** are all local, no communication
 - need global communication to select the best feature in `findBestSplit()` → `allreduce(maxscore)`, also need to `broadcast(Split Instance Set)`

Issues

- irregular memory access
 - instances in tree nodes are dynamic sets
 - non-continuous memory access to g,h
- sparsity
 - missing values
 - frequent zero entries
 - artifacts of feature engineering such as one-hot-encoding
- high dimensionality
 - $|\text{features}|$

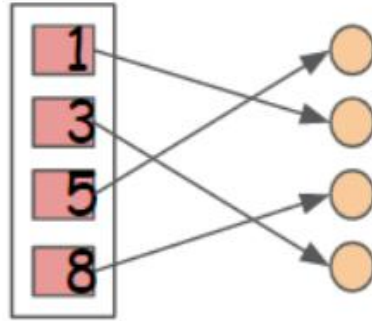
Outline

- Problem
- Related work
- Proposal

XGBoost^[1]

- Standard baseline
- Algorithm
 - approximate algorithm with both local and global proposal methods (**Bins**)
 - sparsity-aware, only collect statistics of non-missing entries
- System
 - **Column Block**
 - each block is a subset of rows
 - in each block, data stored in compressed column(CSC) format, with each column **sorted** by corresponding feature value
 - support feature level parallelism
 - linear scan to find best split
 - Cache-aware access
 - choose correct block size to get gradient statistics(g,h) fit into the CPU cache. (2^{16})

Block Structure



Instructions

$G = G + g[\text{ptr}[i]]$

$H = H + h[\text{ptr}[i]]$

calculate score....

$G = G + g[\text{ptr}[i]]$

$H = H + h[\text{ptr}[i]]$



Short range
dependency,
with **non-contiguous**
access to g

Figure 8: Short range data dependency pattern that can cause stall due to cache miss.

LightGBM^{[2][3]}

- high communication costs $\sim O(|\text{features}| * \text{binSize})$
 - PV-Tree (Parallel Voting Decision Tree)
 - local voting: select top-k features based on local data
 - global voting: select top-2k features by votings from local candidates
 - collect full-grained histograms of the globally top-2k, and findBestSplit
- high dimensionality
 - Gradient-based One-side sampling
 - exclude a significant proportion of data instances with small gradients in estimate the score
 - better than SGB(Stochastic GB) with the same sampling ratio
- Sparsity
 - Exclusive feature bundling
 - bundle mutually exclusive features(never nonzero values simultaneously) into a single feature
 - letting exclusive features reside in different bins (adding offsets to original values)

DimBoost^{[4][5]}

- high dimensionality (industry application with 330K features)
- Optimize Communication
 - parameter server
 - claims to be one communication step and take less time (?)
 - two-phase-split finding
 - server-side split
 - round-robin task scheduler
 - schedule the splitting tasks among the workers
 - low-precision gradient histogram
 - each item q in a histogram, encode to a d -bit integer $q' = \text{floor}(q/|c| * 2^d)$
 - $d=8$ often enough to obtain no loss on final accuracy
- Optimize Computation
 - parallel batch construction
 - divides a range into **batches** for the big nodes in the first few layers
 - sparse histogram construction
 - only non-zero entries

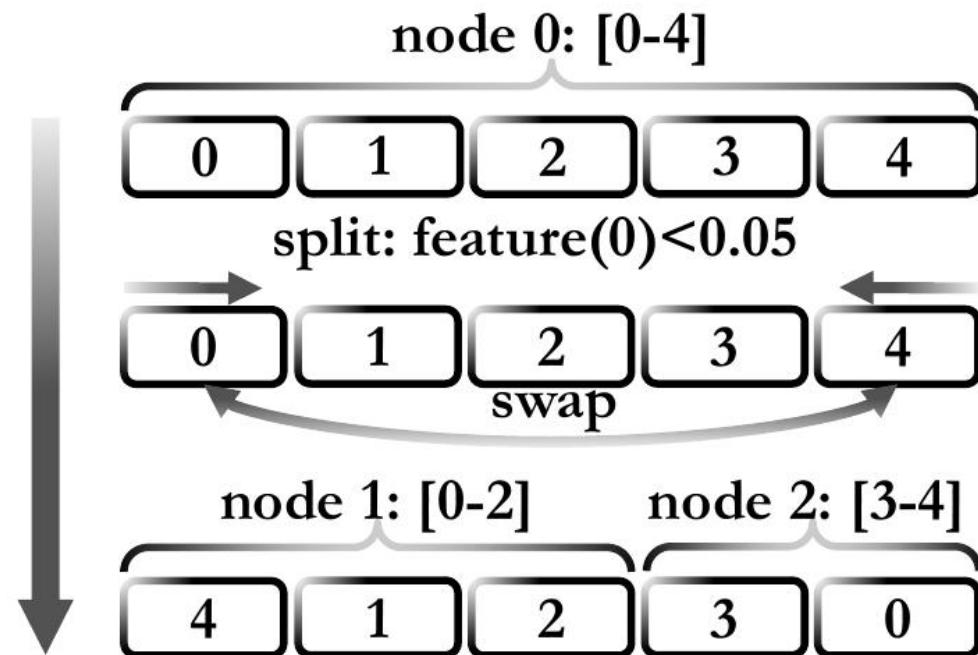
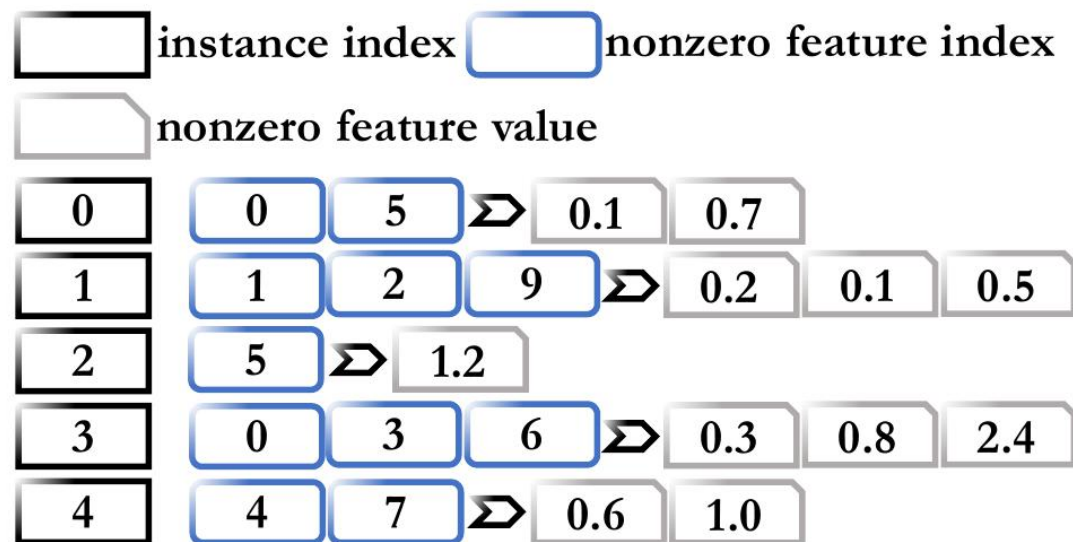


Figure 9: Node-to-instance Index.

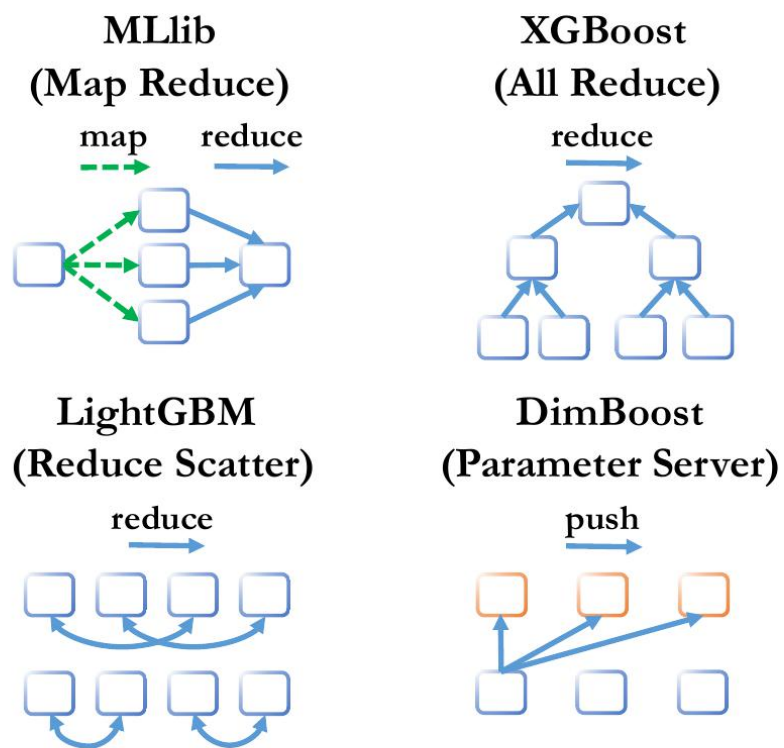


Figure 3: Existing Model Aggregation Methods.

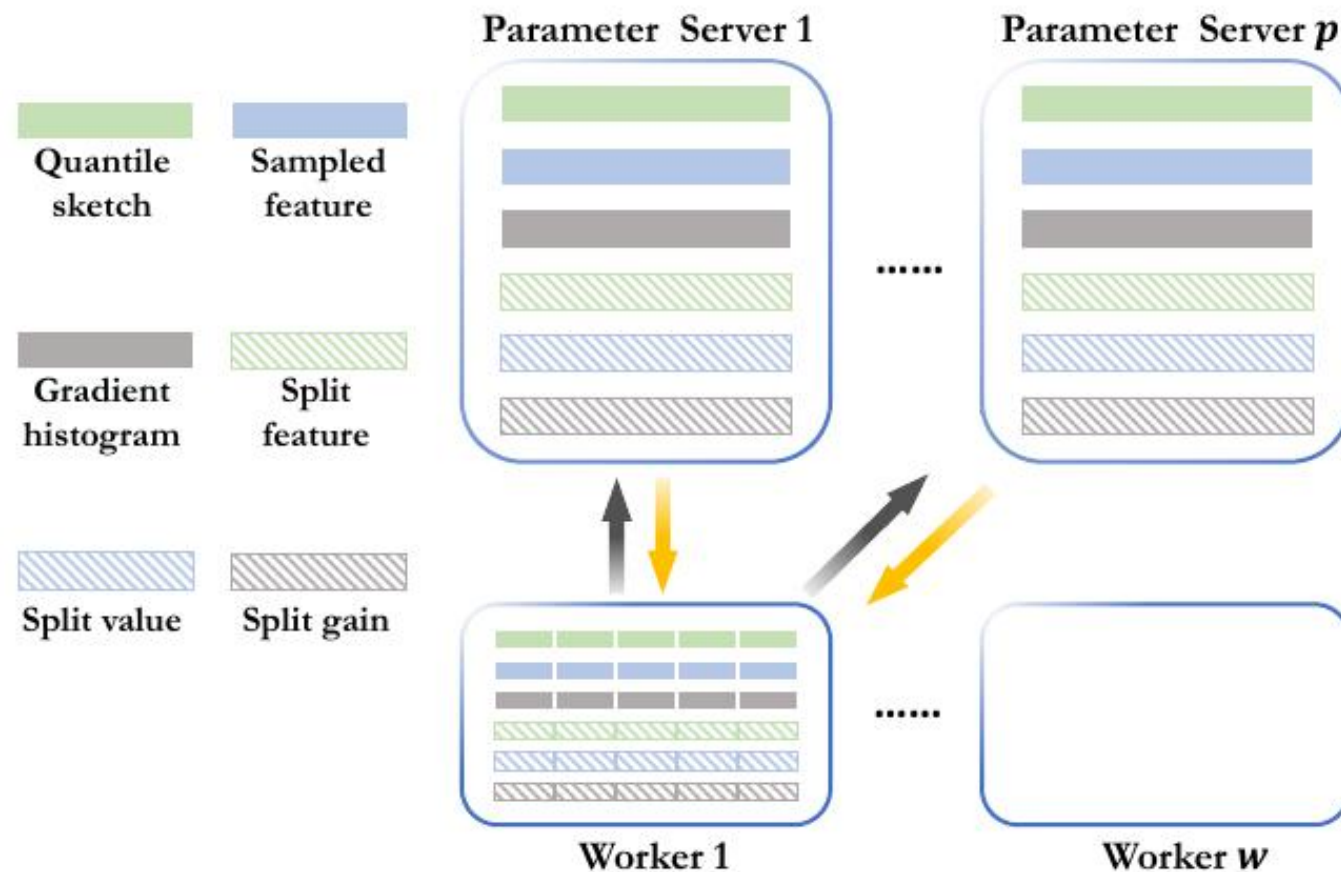


Figure 6: Parameter layout.

GPU-GBDT^[6]

- sparse representation
 - similar to Column Block
- **fine-grain** parallelism
 - parallelizing the score computation of each split point: segmented prefix sum
--> GHSum
 - select best split point: segmented reduction
 - splitting a node: maintain values of each feature in the new node in sorted order
- Run-length encoding compression
 - sorted by feature values
 - repeated values can be compressed
 - directly splitting RLE elements
- thread/block workload dynamic allocation
 - under a memory constrain

TABLE I

DENSE AND SPARSE DATA REPRESENTATION

	Dense	Sparse
\mathbf{x}_1	$\langle 0.0, 0.0, 0.1, 0.0 \rangle$	$(a_3 : 0.1)$
\mathbf{x}_2	$\langle 1.2, 0.0, 0.1, 0.6 \rangle$	$(a_1 : 1.2); (a_3 : 0.1); (a_4 : 0.6)$
\mathbf{x}_3	$\langle 0.5, 1.0, 0.0, 0.0 \rangle$	$(a_1 : 0.5); (a_2 : 1.0)$
\mathbf{x}_4	$\langle 1.2, 0.0, 2.0, 0.0 \rangle$	$(a_1 : 1.2); (a_3 : 2.0)$

$$a_1 = (\mathbf{x}_2 : 1.2); (\mathbf{x}_4 : 1.2); (\mathbf{x}_3 : 0.5)$$

$$a_2 = (\mathbf{x}_3 : 1.0)$$

$$a_3 = (\mathbf{x}_4 : 2.0); (\mathbf{x}_2 : 0.1); (\mathbf{x}_1 : 0.1)$$

$$a_4 = (\mathbf{x}_2 : 0.6)$$

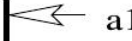


						
Instance Id	x1	x2	x4	x1	x3	x2
Gradient	0.7	-0.3	0.7	0.7	0.6	-0.3
<p>Segmented prefix sum</p> 						
Prefix sum	0.7	0.4	0.7	1.4	2	1.7

Fig. 1. Example results of segmented prefix sum

Outline

- Problem
- Related work
- Proposal

Assumption

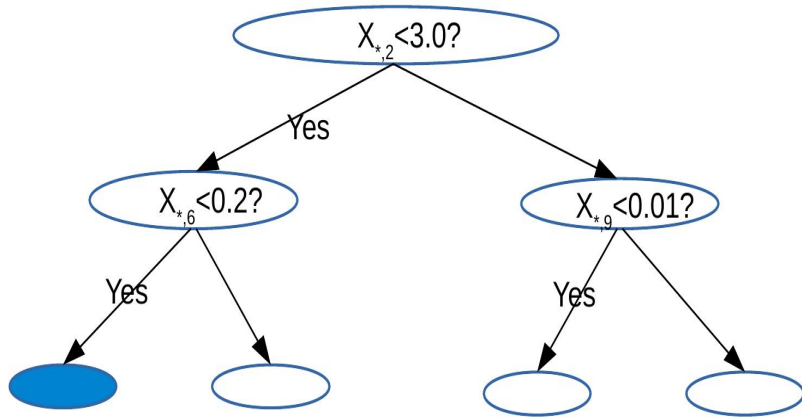
- Training Data N samples, M Features, build tree with L levels
- Model GHSum with bin size B
 - at each node, size: $B*M$
 - at each level, size: $2^{L-1}B*M$
- Higgs, for example:
 - $N = 10m$, $M = 28$, $L = 6$, $B = 256$
 - model size at $L=6$, $32*256*28 \ll 10m$
- Most cases
 - dense: $M < 1K$, $L < 8$
 - sparse can be an issue
 - deep tree can be an issue

Cache Friendly Data Organization



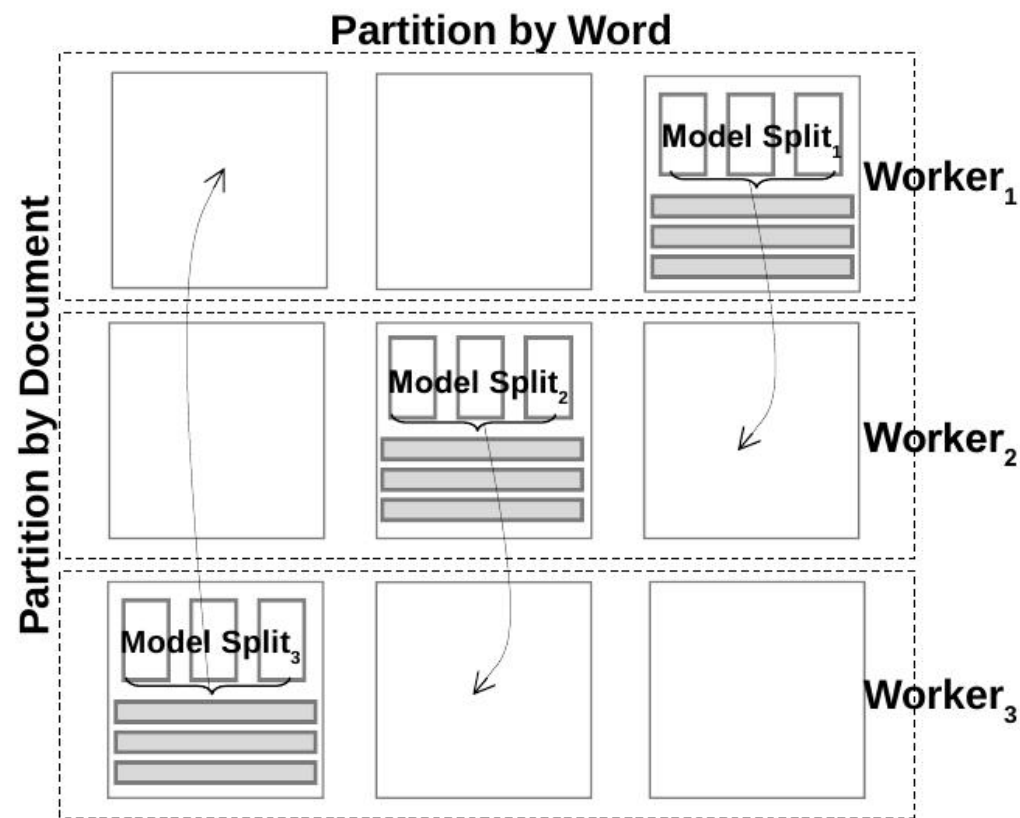
- Column-wised orgnization
- Issue: non-continuous memory access
 - sorted by feature values
 - dynamic instance set
- Proposal
 - multi-columns as a **block** to fit one **cache line**

Training data access



- In case $N \gg \text{ModelSize}$
 - data parallelism: training data statically partitioned among nodes
 - model parallelism: optional and preferred when modelsize is big
- Proposal
 - Layer-by-layer
 - **Single pass** on train data for each level. (half when reuse $\text{GHSum}_{\text{parent}}$)
 - Keep GHSum of the same level in memory

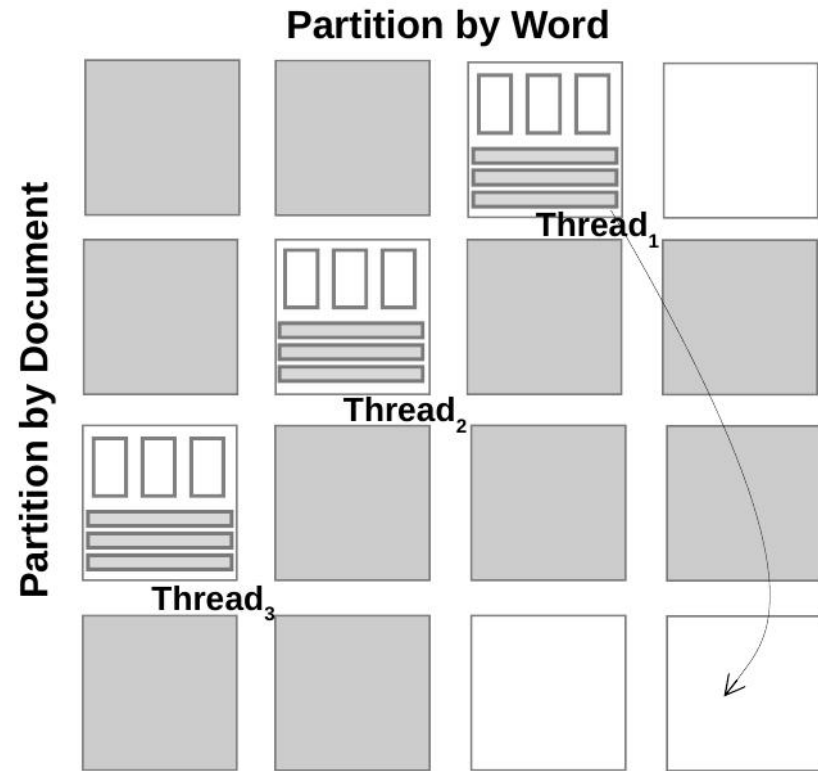
Data and Model Parallelism by Model Rotation



(a) model rotation

- Proposal
 - use computation model : **model rotation**
- Data partitioned to blocks
 - Data Parallelism:
 - Model parallelism: GHSum split by feature columns
- Scheduler to avoid update conflicts

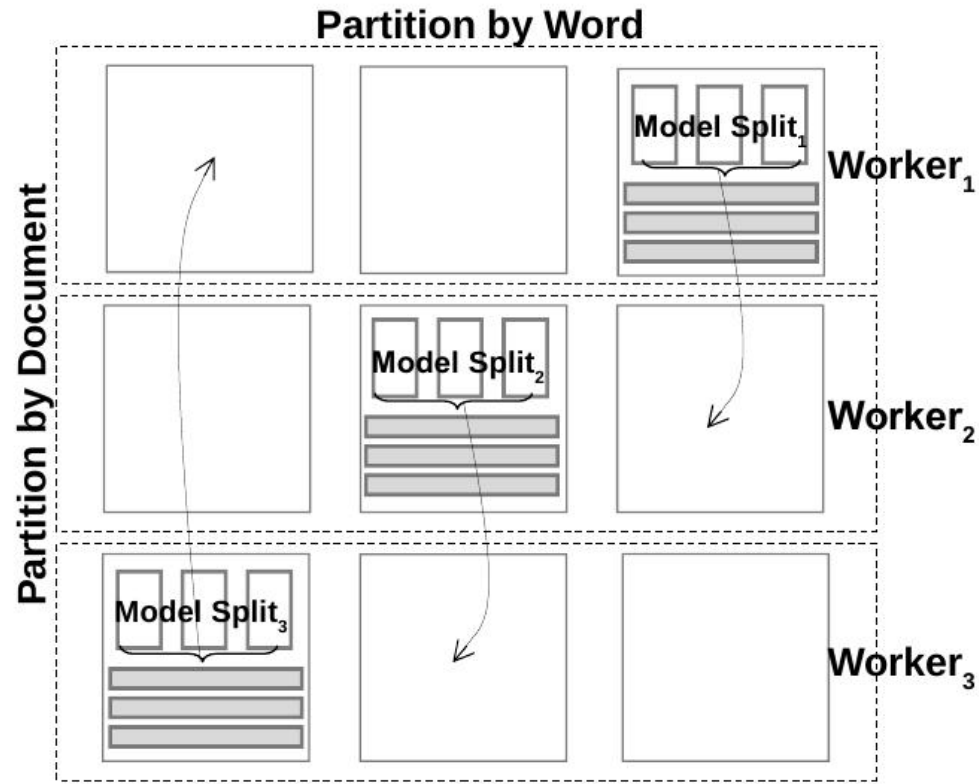
Block Dynamic Scheduling in Shared Memory systems



(b) dynamic scheduling

- a low cost solution to remove synchronization overhead
- number of partitions is larger than the number of threads,
- always 'free' rows and columns available when one thread finishes its current task.
- cache-aware block size
- sparse representation ready

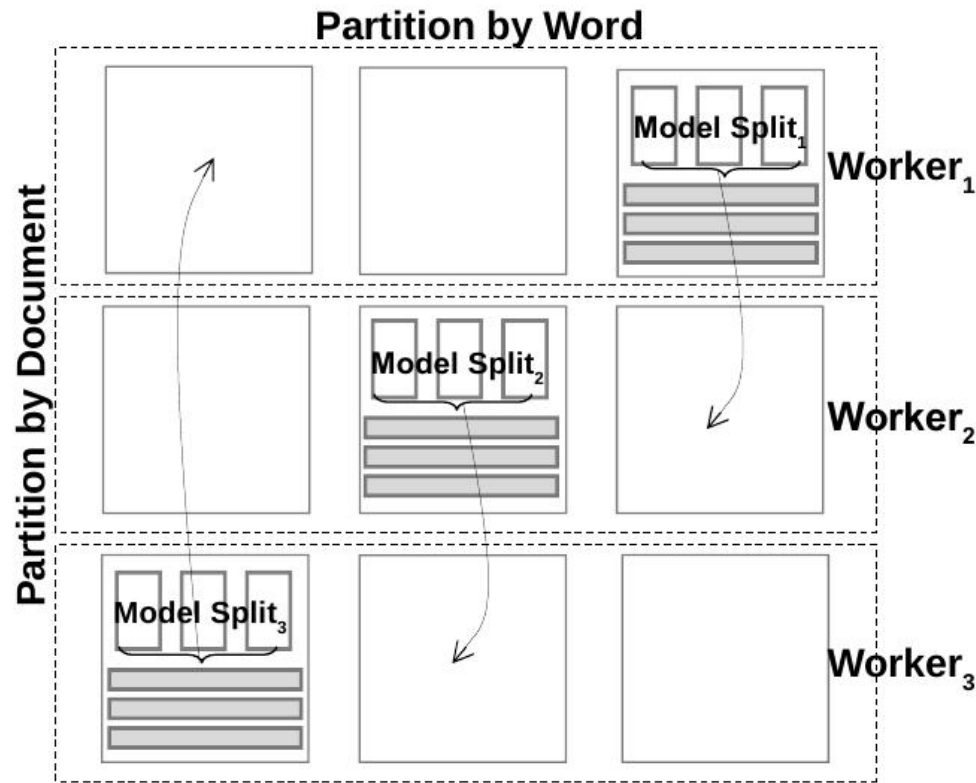
Distributed Model Rotation



(a) model rotation

- Training data randomly partitioned among nodes
- Model partitioned among nodes
 - split by feature columns
- Scheduler to avoid update conflicts
 - rotate local model partition to neighbor at each sub-step
 - finish after K sub-steps(K nodes)
 - kind of a ring-allreduce

Pipelining and Timer Control in Distributed Systems



(a) model rotation

- Each sampler only works for the same period of time and then the samplers do synchronization all together.
- They all use a **timer** to control the synchronization point rather than waiting until all the blocks to finish.
- This adjustment does not change the property of the uniform random selection of blocks.

More to consider

- Feature Bundle
 - deal with category features
- Feature Sampling
 - design scheduler with Gradient-based priority
- Data Compression
 - model compression: low-precision compression
 - training data compression: RLE

References

- [1]T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, pp. 785–794.
- [2]Q. Meng et al., “A communication-efficient parallel algorithm for decision tree,” in Advances in Neural Information Processing Systems, 2016, pp. 1279–1287.
- [3]G. Ke et al., “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in Advances in Neural Information Processing Systems, 2017, pp. 3149–3157.
- [4]J. Jiang, B. Cui, C. Zhang, and F. Fu, “DimBoost: Boosting Gradient Boosting Decision Tree to Higher Dimensions,” in Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1363–1376.
- [5]J. Jiang, J. Jiang, B. Cui, and C. Zhang, “TencentBoost: A Gradient Boosting Tree System with Parameter Server,” in 2017 IEEE 33rd International Conference on Data Engineering (ICDE), 2017, pp. 281–284.
- [6]Z. Wen, B. He, R. Kotagiri, S. Lu, and J. Shi, “Efficient Gradient Boosted Decision Tree Training on GPUs,” in 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2018, pp. 234–243.