The 13rd Chinese Workshop on Machine Learning and Applications
第十三届中国机器学习及其应用研讨会

2015年11月6~8日 南京大学 南京

Making Super-Large Scale Machine Learning Possible

Tie-Yan Liu

Principle Researcher/Research Manager

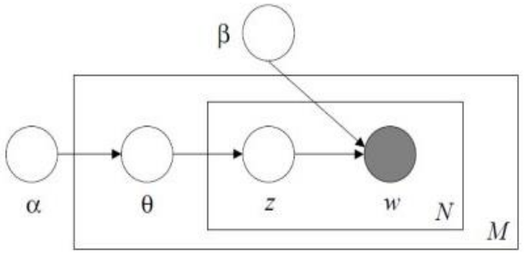
Microsoft Research Asia

Era of Big Data and Big Model

Search engine index: 10^{10} pages (10^{12} tokens)

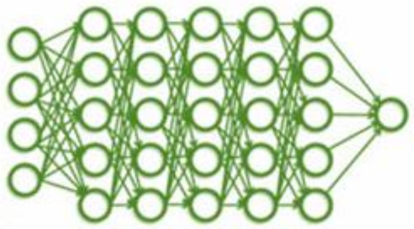
Search engine logs: 10^{12} impressions and 10^{9} clicks every year

Social networks: 10^{9} nodes and 10^{12} edges



Peacock: LDA with 10^{5} topics (10^{10} parameters); More topics → better performance in click predictions

DistBelief: DNN with 10^{10} weights; Deeper and larger networks → better performance with sufficient training data.



Human brain: 10^{11} neurons and 10^{15} connections, much larger than any existing ML model.

Existing Approach to Big Machine Learning

- Parallelization of existing machine learning algorithms using either MapReduce or Parameter Server

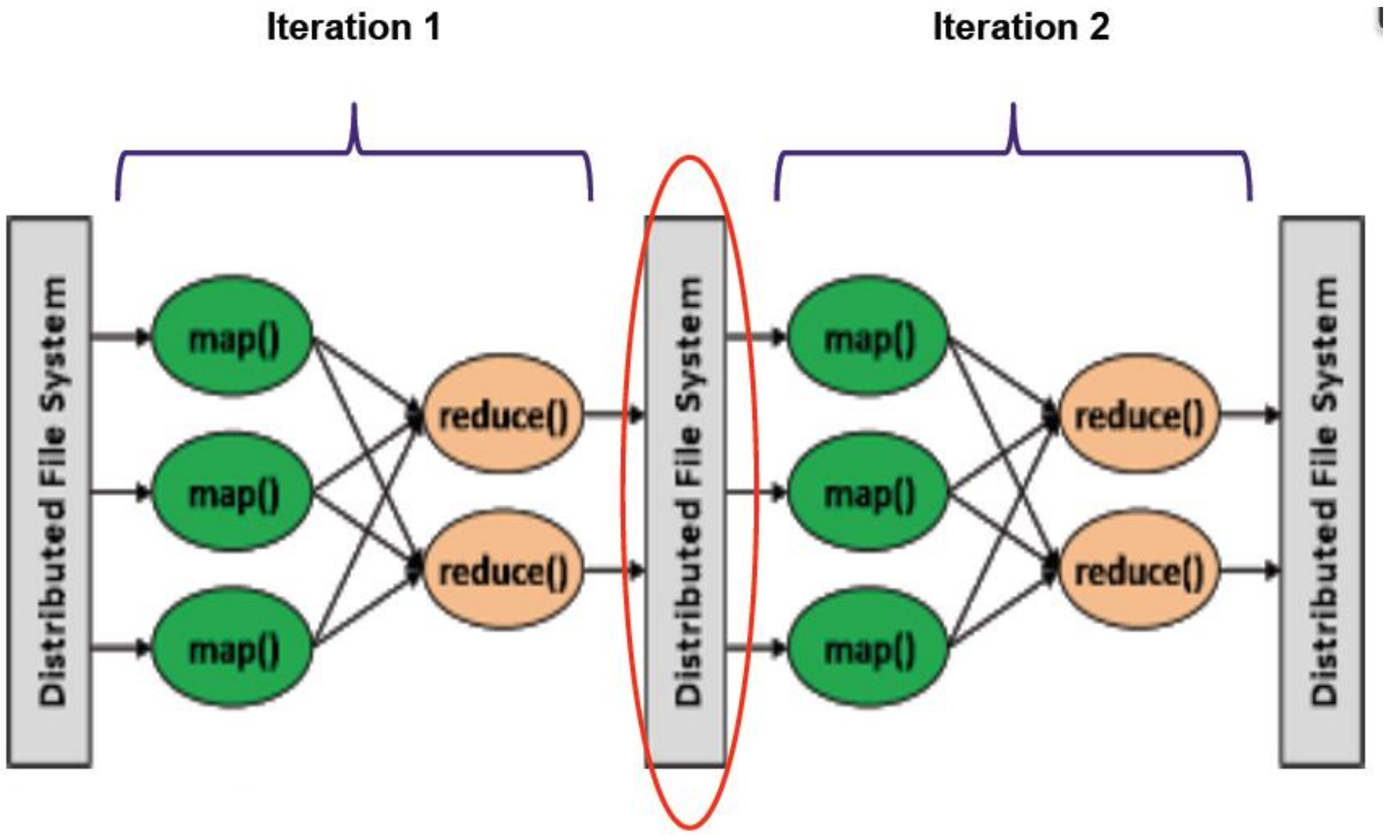
Iterative MapReduce /AllReduce

- Only synchronous updates (BSP, MA, ADMM), poor efficiency on heterogeneous clusters
- Only data parallelism, cannot handle big models

Parameter Server

- Support asynchronous updates; better efficiency on heterogeneous clusters
- Support model parallelism, but inefficient, especially on heterogeneous clusters.
- Only support fixed-structure models
- "sum", "average", and "addition" as atomic aggregation operations

Iterative MAP-Reduce



BSP, ADMM and Model Average

$$\min_{\boldsymbol{w}} \sum_{i=1}^{N} L_i(w)$$

$$w_i^t = w^t$$
$$\Delta w_i^t = -\eta_t \nabla L_i(w_i^t)$$
$$w^{t+1} = w^t + \sum_i \Delta w_i^t$$

$$\min_{\boldsymbol{w}} \sum_{i=1}^{N} L_i(w)$$

$$\text{s.t. } w_i - \boldsymbol{z} = \boldsymbol{0}, i = 1, \dots, N$$

$$\boldsymbol{w}_i^{t+1} = \arg\min_{\boldsymbol{w}_i} \{ \sum_i (L_i(\boldsymbol{w}_i) + (\boldsymbol{\lambda}_i^t)^T (w_i - \boldsymbol{z}^t) + \frac{\rho}{2} \| w_i - \boldsymbol{z}^t \|_2^2) \}$$

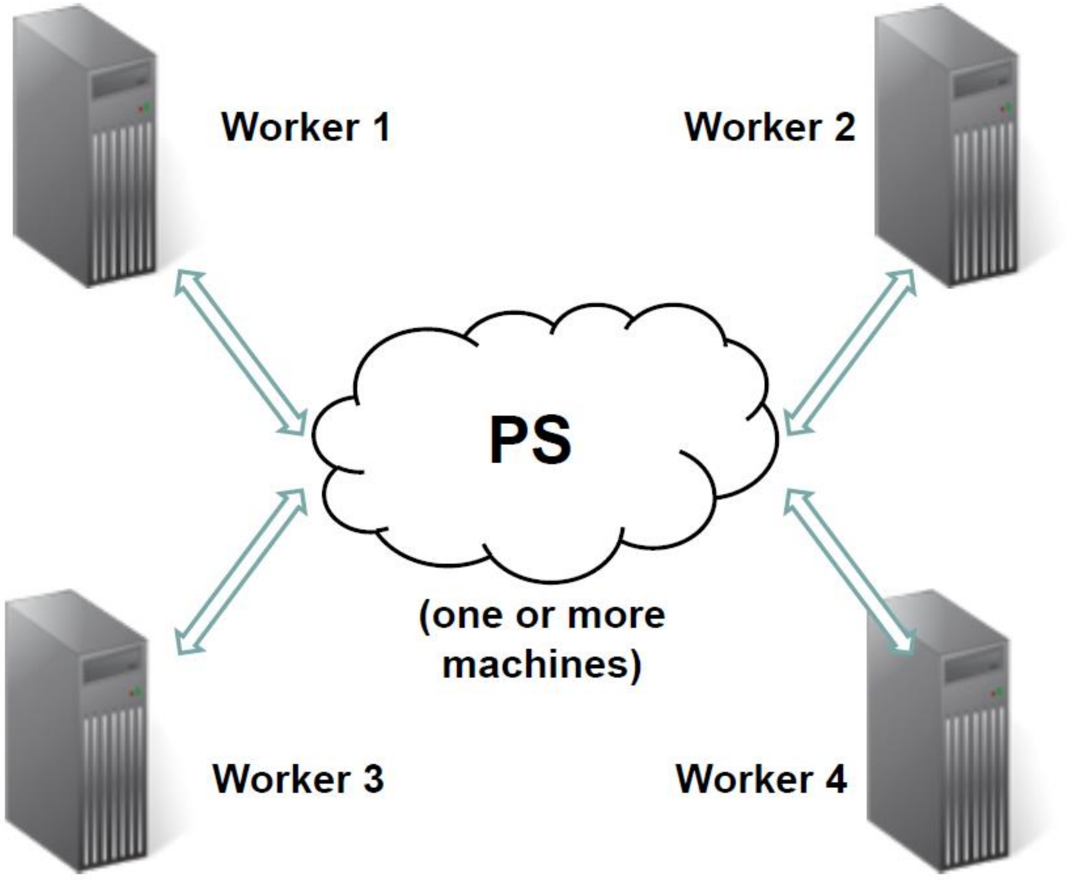
$$\boldsymbol{z}^{t+1} = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{w}_i^{t+1} + \frac{1}{\rho} \boldsymbol{\lambda}_i^t)$$

$$\boldsymbol{\lambda}_i^{t+1} = \boldsymbol{\lambda}_i^t + \rho(\boldsymbol{w}_i^{t+1} - \boldsymbol{z}^{t+1})$$

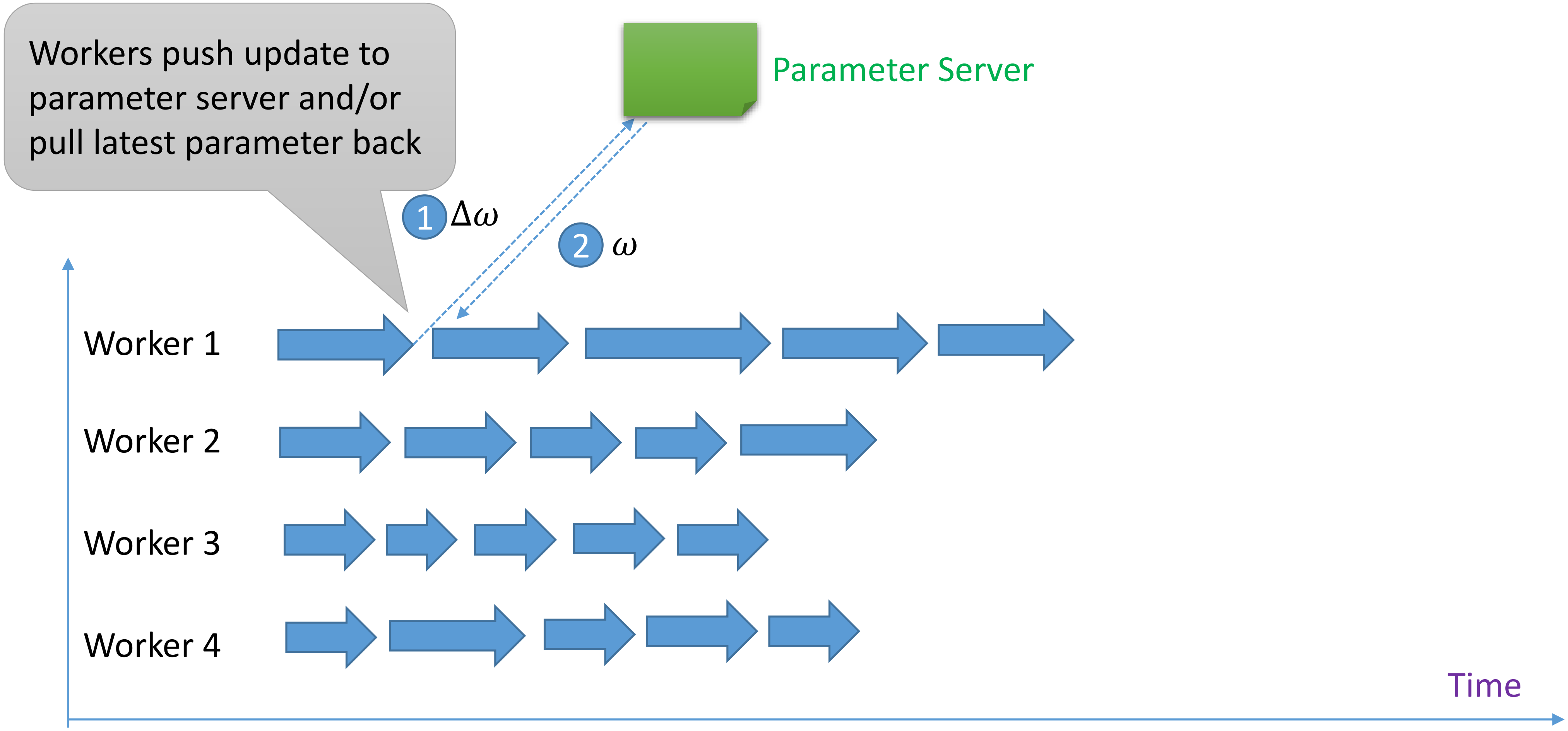
$$\boldsymbol{z}^{t+1} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{w}_i^t$$

$$\boldsymbol{w}_i^{t+1} = \boldsymbol{z}^{t+1}$$

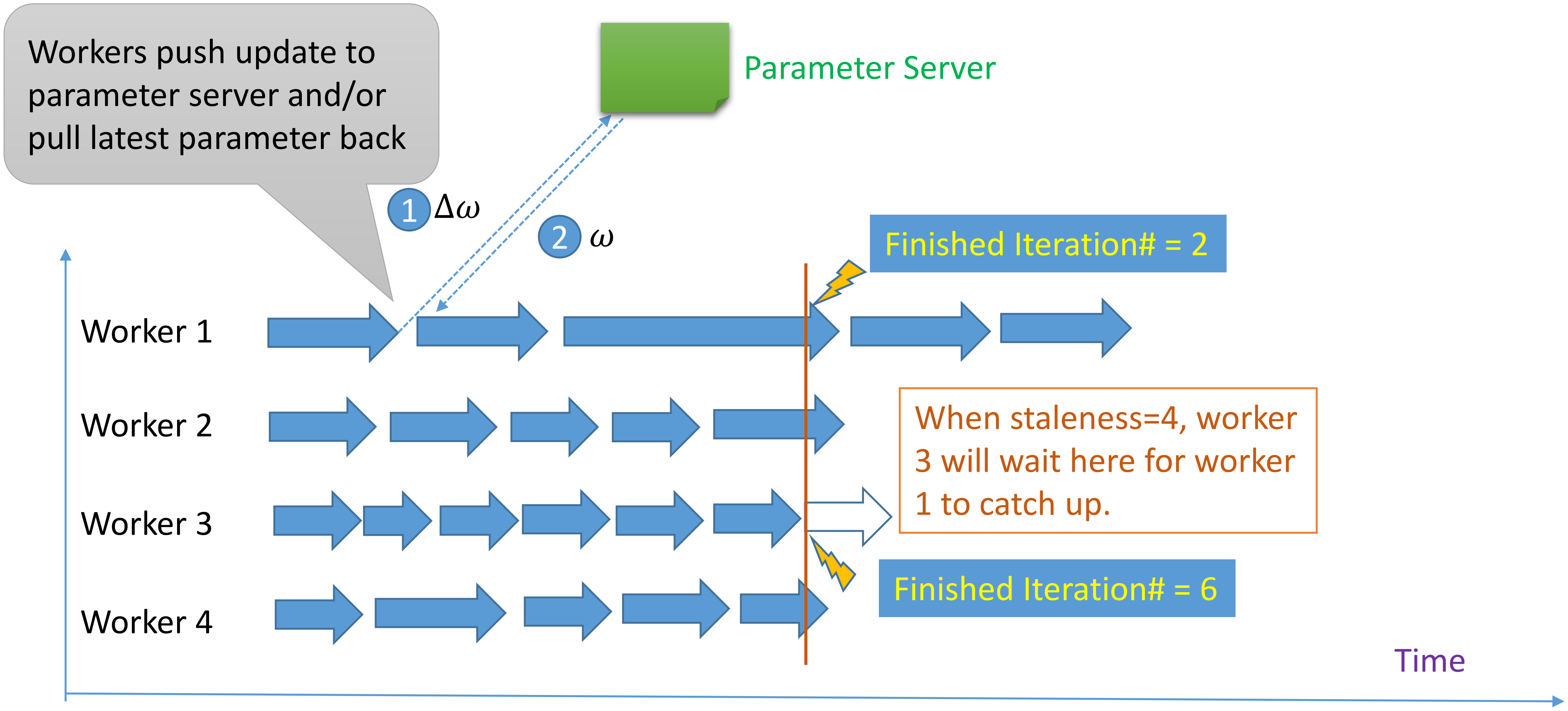
Parameter Server



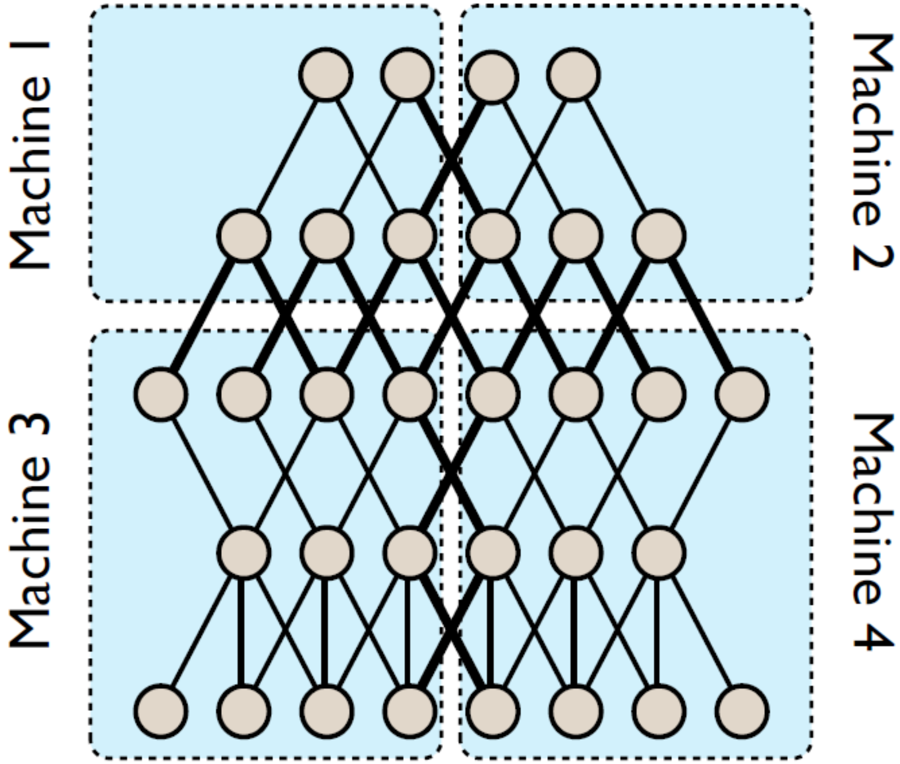
ASP: Asynchronous Parallel



SSP: Stale Synchronous Parallel



Model Parallelism



Limitations of Existing Approaches

- Scalability
  - Hard to train a topic model with millions of topics, or a DNN model with trillions of weights.
- Efficiency
  - 2+ days for 3000 CPU cores to finish the training of Peacock LDA.
  - 3 days for 16,000 CPU cores to finish the training of DistBelief DNN.
- Flexibility
  - Not many other big models beyond LDA and DNN were extensively studied in the literature.

Desirable System for Big Machine Learning

- Web data (trillions of tokens)
- Click logs (trillions of impressions)
- Social networks (trillions of edges)

Data Scalability

Model Scalability

- LDA (millions of topics)
- CNN (trillions of activations)
- DNN (trillions of edge weights)
- Word embedding (millions of words)

- Gradient boosting trees
- Decision trees / Random forest
- Ensemble models

Flexibility

Efficiency

- Almost linear speed up, even on heterogeneous clusters
- Reasonable training time even for big data and big model

How to Achieve It?

Algorithmic Innovation

- Machine learning algorithms themselves need to have sufficiently high efficiency and throughout.
- Existing design/implementation of machine learning algorithms might not have considered this request; redesign/re-implementation might be needed.

System Innovation

- One needs to leverage the full power of distributed system, and pursue almost linear scale out/speed up.
- New distributed training paradigm needs to be invented in order to revolve the bottle neck of existing distributed machine learning systems.

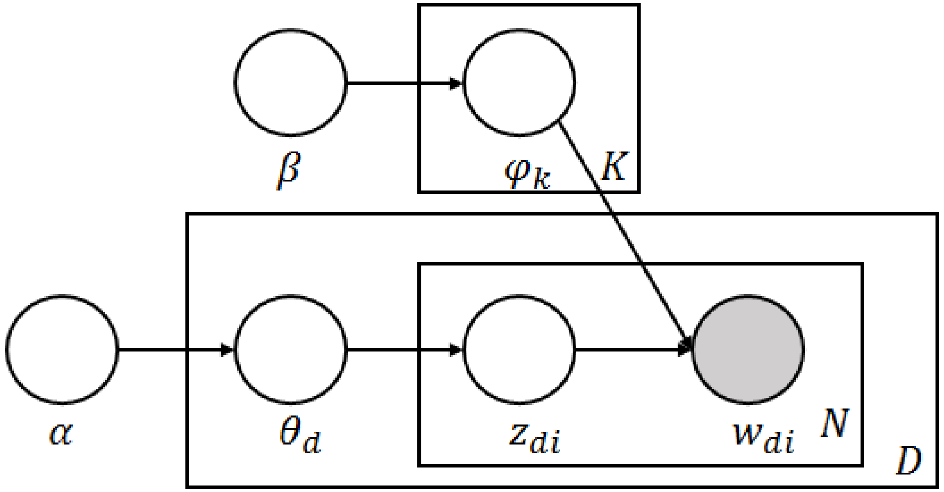
Algorithmic Innovation

Case Studies

- **LightLDA**: Highly efficient LDA algorithm (with O(1) amortized per-token sampling complexity) by using multiplicative factorization.

- **Distributed Word Embedding**: Highly scalable word embedding algorithm by using histogram-based data sampler.

Latent Dirichlet Allocation (LDA)



$$\theta_d \sim Dirichlet(\alpha); \quad \varphi_k \sim Dirichlet(\beta);$$
$$z_{di} \sim Multinomial(\theta_d); \quad w_{di} \sim Multinomial(\varphi_{z_{di}})$$

[Blei, et al. 2003]

- For document d, sample a topic distribution θ_d from a Dirichlet distribution with parameter α.
- Sample a word distribution φ_k for each topic k from a Dirichlet distribution with parameter β
- For each token i in document d
  - Sample a specific topic z_{di} from topic distribution θ_d
  - Sample a word from word distribution $\varphi_{z_{di}}$.

Collapsed Gibbs Sampling

- Sampling from a closed-form conditional probability of topics, by integrating out θ and φ:

$$p(k) = p(z_{di} = k|rest) \propto \frac{n_{kw}^{-di} + \beta_w}{n_k^{-di} + \bar{\beta}} \left(n_{kd}^{-di} + \alpha_k\right)$$

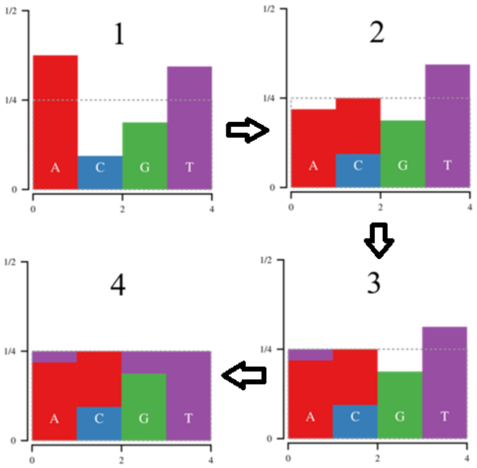
- n_k^{-di}: number of tokens assigned to topic k

Per-token sampling complexity proportional to the number of topics: $O(K)$, thus hard to scale up to large number of topics.

Reduce Complexity by Amortizing Computations

Alias Table [Walker, 1977]

- Build alias table for some terms in $p(k)$ and reuse it across many tokens (introducing approximation error)



Alias table construction: transform non-uniform distribution to uniform in O(K) time; sample from uniform distribution in O(1) time.

Metropolis Hastings [Hastings, 1970]

- Handle approximation error using a rejection procedure.
  - Given original $p(k)$ and its approximation $q(k)$
  - Sample according to $q(k)$ followed by a rejection procedure based on the difference between $q(k)$ and $p(k)$
    - $r \sim U(0,1)$, $s \xrightarrow{q(k)} t$
    - Accept t as next state if $r < \min\left\{1, \frac{p(t)q(s)}{p(s)q(t)}\right\}$.
  - Stationary distribution of the above Markov chain is exactly $p(k)$; mixing rate depends on the difference between $p(k)$ and $q(k)$.

Amortizability

Terms	n_{kd}	n_{kw}	$n_{kd} \cdot n_{kw}$
Alias table construction	For each document d, in $O(L_d)$ time	For each word, in $O(KV)$ time	For each document and word, in $O(L_d V)$ time
Reused for	Only tokens in document d	All documents	Only tokens in document d
Amortized O(1)?	Yes	Yes	No

SparseLDA [Yao, et al. 2009]

- Decompose $p(k)$ into additive terms, then sample the terms using the mixture approach

$$p(z_{di} = k|rest) \propto \boxed{\frac{\alpha_k \beta_w}{n_k^{-di} + \bar{\beta}} + \frac{n_{kd}^{-di} \beta_w}{n_k^{-di} + \bar{\beta}}} + \boxed{\frac{n_{kw}^{-di}\left(n_{kd}^{-di} + \alpha_k\right)}{n_k^{-di} + \bar{\beta}}}$$

Amortizable → O(1)

Unamortizable but sparse → $O(K_w)$

Non-zero elements in word-topic table $\{n_{kw}^{-di}\}$

Per-token complexity: $O(K_w) \ll O(K)$
K_w: number of topics word w belongs to

AliasLDA [Li, et al. 2014]

- Decompose $p(k)$ into additive terms, then sample the terms using the mixture approach

$$p(z_{di} = k|rest) \propto \boxed{\frac{n_{kd}^{-di}(n_{kw}^{-di}+\beta_w)}{n_k^{-di}+\bar{\beta}}} + \boxed{\frac{\alpha_k(n_{kw}^{-di}+\beta_w)}{n_k^{-di}+\bar{\beta}}}$$

Unamortizable but sparse → $O(K_d)$

Non-zero elements in doc-topic table $\{n_{kd}^{-di}\}$

Amortizable → O(1)

Per-token complexity: $O(K_d) \ll O(K_w) \ll O(K)$

K_d: number of topics document d contains

LightLDA

- Factorize $p(k)$ into *multiplicative* terms, instead of decomposing it into *additive* terms
  - Separate n_{kd}^{-di} and n_{kw}^{-di} into different terms, so as to avoid the issue of unamortizability.
  - All terms after factorization only contain either n_{kd}^{-di}, n_{kw}^{-di}, or constant, thus a O(1) sampling complexity can be achieved by Alias and MH methods.
- The mixture approach does not naturally work for multiplicative factorization - we use a cycling approach instead.

[Yuan, et al. 2015]

Multiplicative Factorization

$$p(z_{di} = k|rest) \propto \overset{p_1(k)}{\boxed{\frac{n_{kw}^{-di} + \beta_w}{n_k^{-di} + \bar{\beta}}}} \overset{p_2(k)}{\boxed{\left(n_{kd}^{-di} + \alpha_k\right)}}$$

Amortizable: O(1)

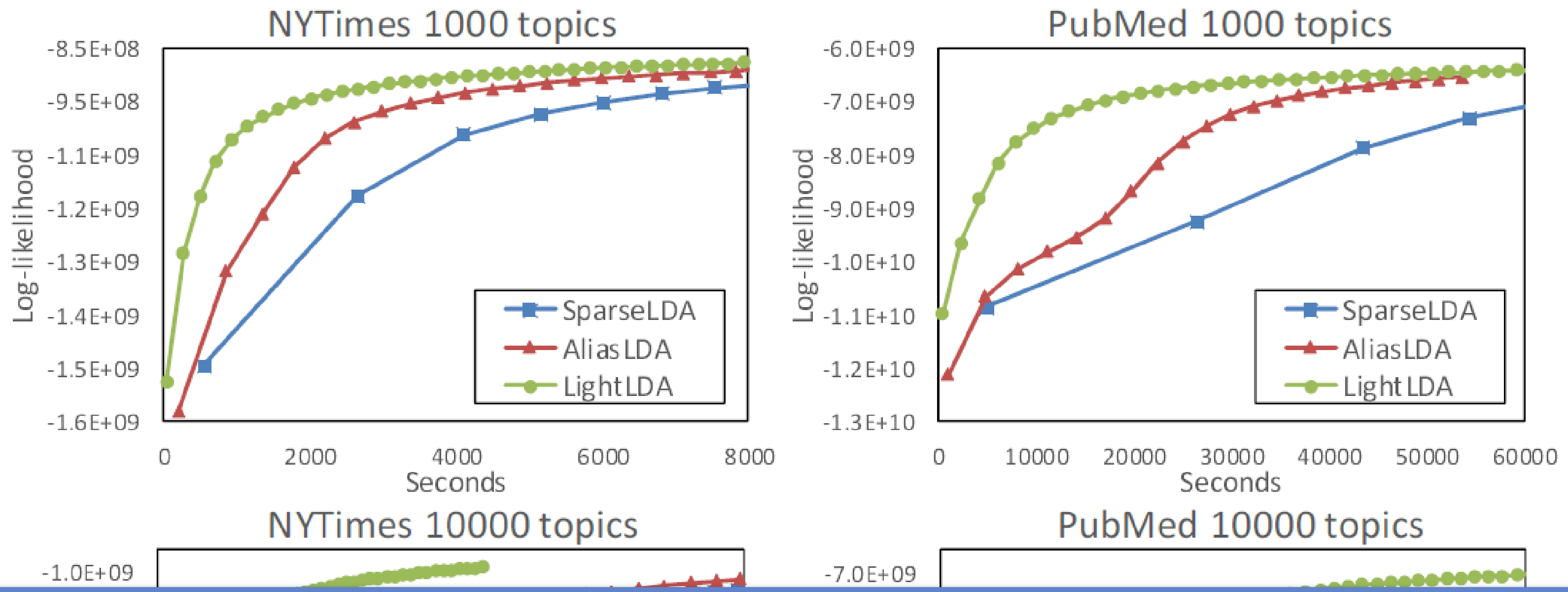
Amortizable: O(1)

Per-token complexity: $O(1)$

Other tricks: (1) sparsified alias table to further reduce the sampling complexity of $p_1(k)$; (2) fully leverage in-memory intermediate result to simply the sampling complexity of $p_2(k)$.

Microsoft

Experimental Results (Single-core)



LightLDA achieves better log-likelihood than SparseLDA and AliasLDA in much shorter time!

NYTimes Dataset

- 300K documents

With a single core only, LightLDA uses 20 hours to train 10K topics from ~1B tokens (PubMed). With a commodity machine of 20 cores, LightLDA can finish training in 2 hours. This single-machine capability is equivalent to (if not beyond) a medium-size cluster of SparseLDA or AliasLDA.

Case Studies

- **LightLDA**: Highly efficient LDA algorithm (with O(1) amortized per-token sampling complexity) by using multiplicative factorization.

- **Distributed Word Embedding**: Highly scalable word embedding algorithm by using histogram-based data sampler.

Word Embedding

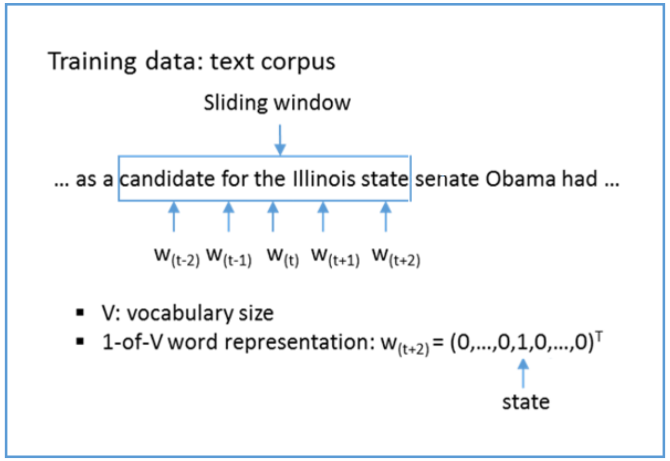
Native Discrete Representation

Word: 1-of-N vector

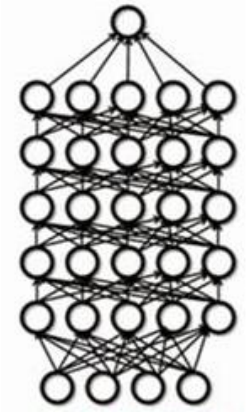
$\{w_1, \quad w_2, \quad \ldots, \quad w_i, \quad \ldots, \quad w_{N-1}, \; w_N\}$

$\langle 0, \quad 0, \quad \ldots, \quad 1, \quad \ldots, \quad 0, \quad 0 \rangle$

Natural Languages

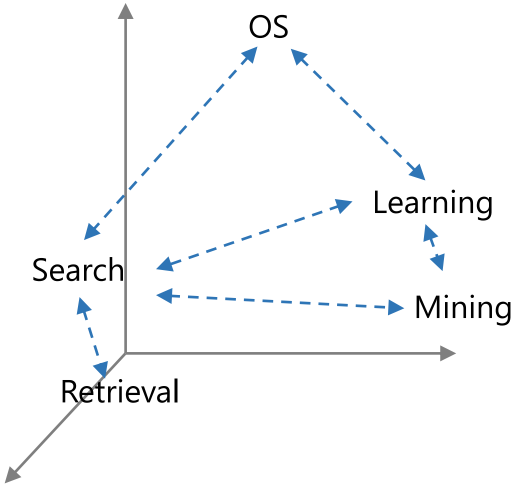


Deep Learning



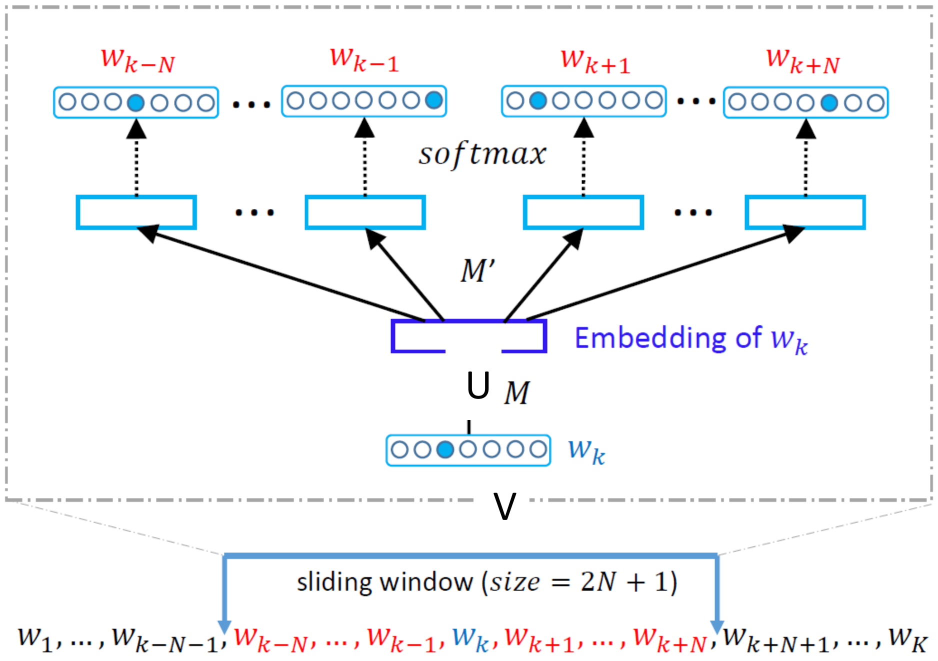
Embedding

Representations in Continuous Space



- State-of-the-art machine learning methods require data to be in a continuous space
- Continuous representation eases text understanding, inference, and reasoning

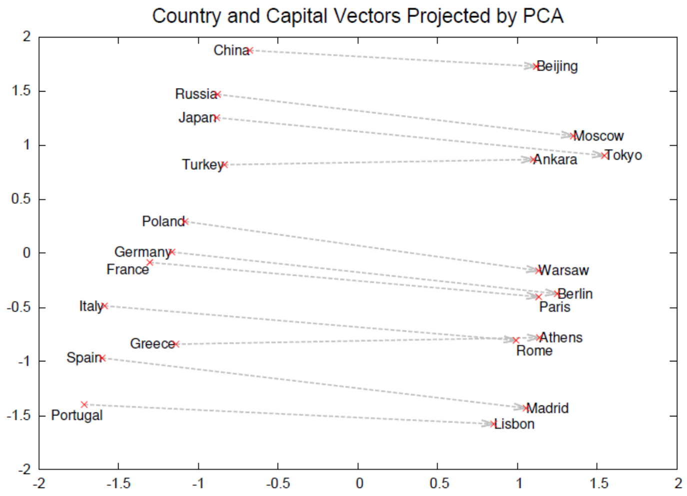
Word2Vec (Skip-Gram)



- *Training data: enwiki9*
- *Dimension of word embedding: 100*

Promising Accuracy on analogical reasoning

- Evaluate linear regularity of word embedding, e.g., the *accuracy* of [China– Beijing+ Tokyo] = [Japan]?



Dataset	#Questions	Accuracy
Mikolov	19544	31.30%

Training Word Embedding Using Entire Web

- Challenge: Web data are simply too large to copy, store, and process!

~ 1 billion words

Web Chunk

Tier-0: ~15 trillion words
(~150 trillion word pairs, ~1PB data)

~1000 machines with 1TB disk are required to store training data; and
~5000 machines with 200GB memory to support in-memory training.

Microsoft

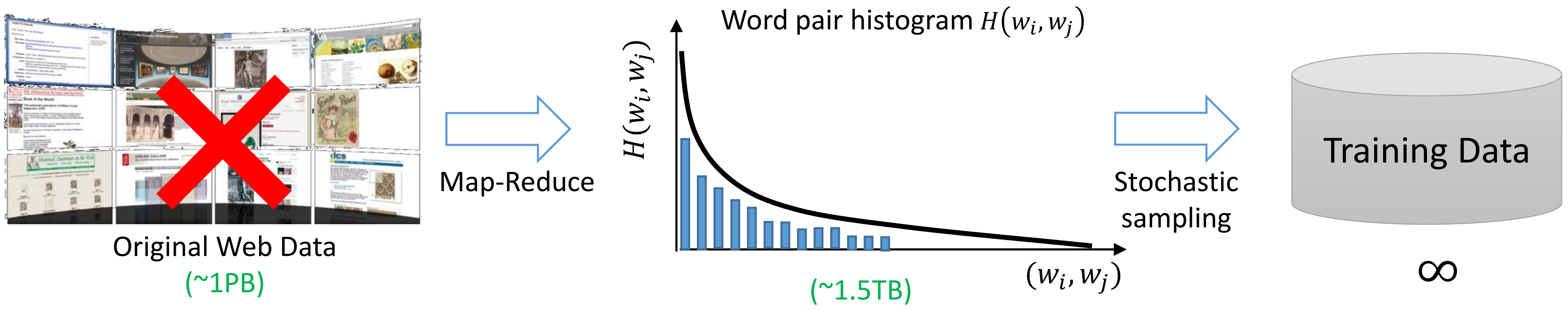
SGD Training for Word2Vec (Skip-Gram)

- Skip-gram training is based on stochastic gradient descent (SGD)
  - Read one word pair from the training corpus
  - Compute gradient for this pair, and update the model
  - Repeat this process until the model converges (after many epochs)

- SGD converges and is an unbiased estimate of gradient descent
  - When the training instances (word pairs) are i.i.d. sampled.
  - Under this assumption, only the distribution matters, but not necessarily the raw data set.

Histogram-based Sampler

- Obtain empirical distribution (word pair histogram) of the training corpus using MapReduce at the beginning of the training process.
- Train word embedding model using SGD, by sampling from the empirical distribution instead of the original text corpus, for an arbitrary number of epochs when needed.



Histogram Re-shape

- Smoothed histogram to handle truncation bias in limited number of sampling

$$\min \quad \sum_i \frac{1}{H(w_i)} |\epsilon_i|^2$$

$$s.t. \quad H(w_i) + \epsilon_i \geq T, \forall i.$$

$$\sum_i \epsilon_i = 0$$

Similar to original empirical histogram

(relative change is minimized)

Each word has at least T counts

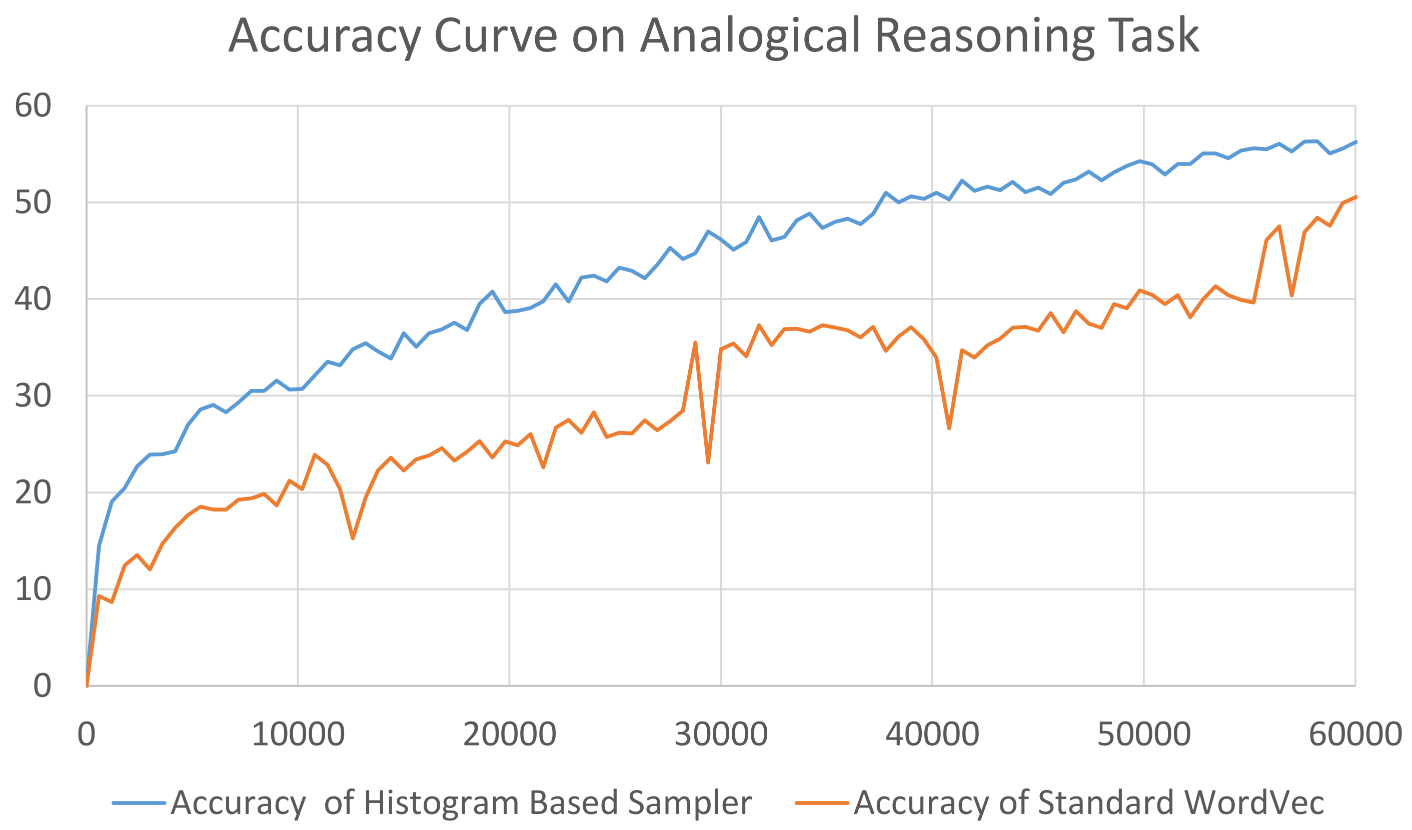
The total count remains unchanged

To satisfy hard constraint $H(w_i) + \epsilon_i \geq T$, for those pairs whose $H(w_i) < T$, the modification ϵ_i is lower bounded and the minimization of the loss function will push $\epsilon_i = T - H(w_i)$.

For those pairs whose $H(w_i) \geq T$, the optimal solution ϵ_i will be proportional to $H(w_i)$, i.e.,

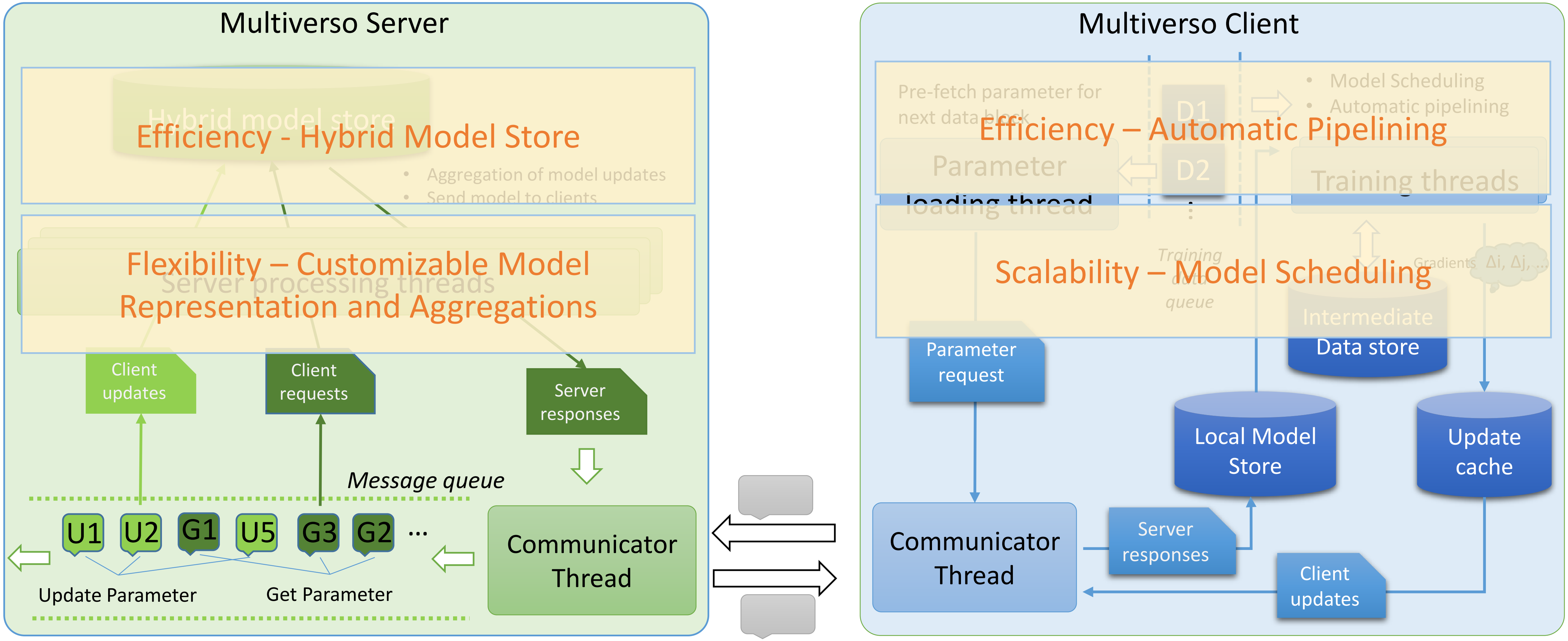
$$\epsilon_i = \frac{-H(w_i)}{\sum_{H(w_i) \geq T} H(w_i)} \sum_{H(w_i) < T} T - H(w_i)$$

Experimental Results

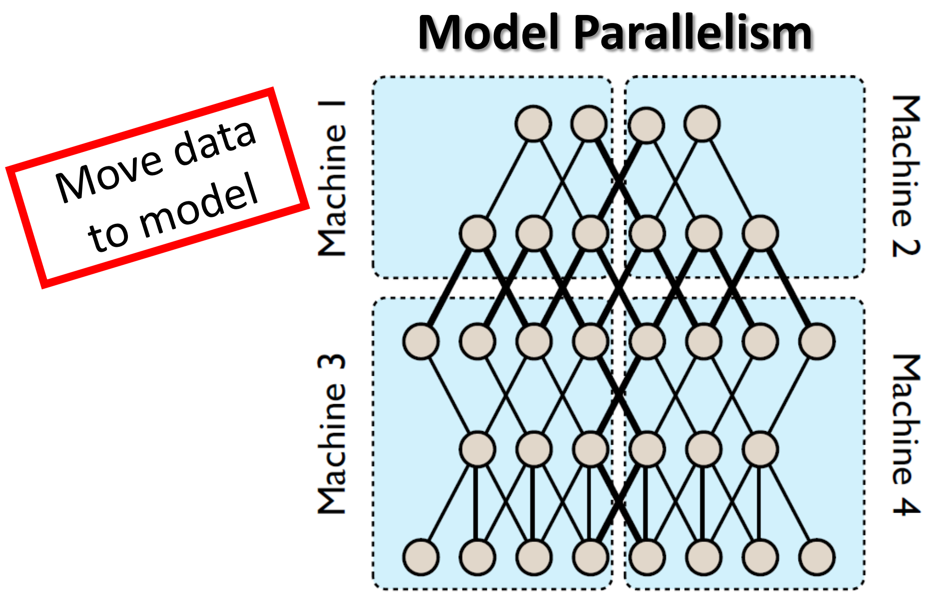


System Innovations

A New Distributed ML Framework



Scalability: Problem with Model Parallelism



- SGD-like algorithms require intermediate results for every data sample to be transferred between machines.

✗ High comm cost: huge intermediate data

- LDA: $O(10^9)$
  - 10^6 docs/data block × 10^3 tokens/doc
- CNN: $O(10^9)$
  - 10^2imgs/mini-batch × 10^5patches/img × 10 filters/patch × 10 layers

✗ Sensitive to comm delay & machine failure

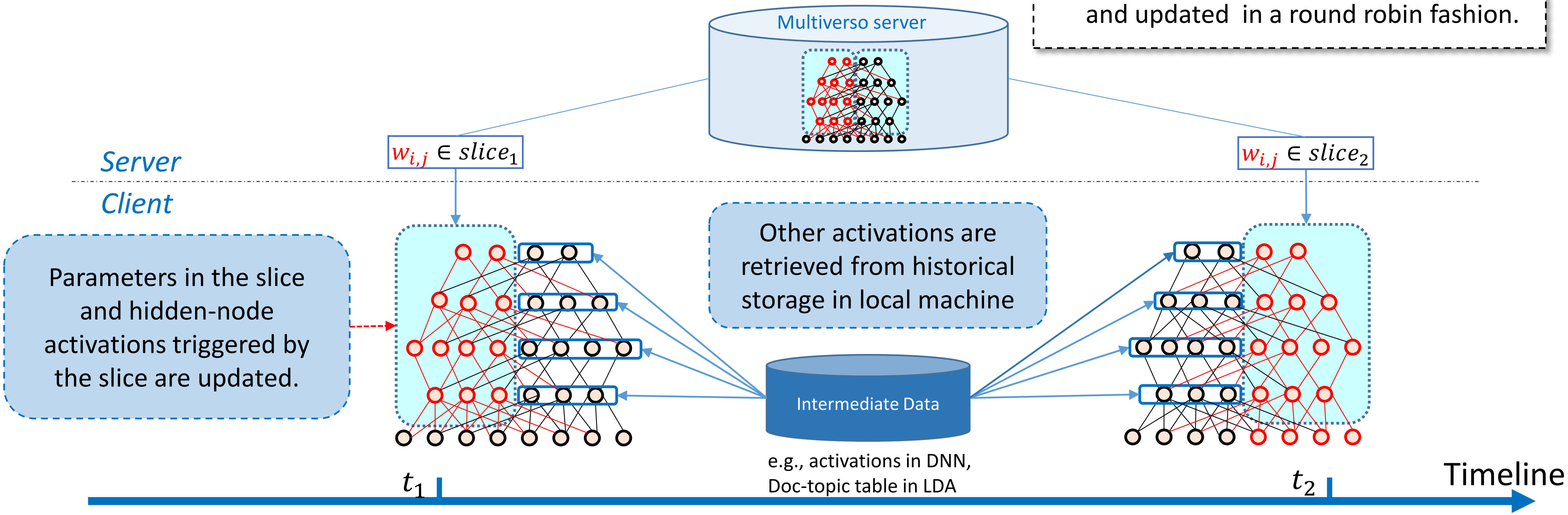
- Speed differences among machines → slow down training.
- Machine failure → break down training.

Scalability: Tackle the Challenges

- Model parallelism might be necessary from system perspective
  - Ensure the same behavior of distributed training with single machine training
- However, it is not necessary from machine learning perspective
  - Machine learning is statistical: achieving similar results (in large probability) is enough, not necessarily preserving exactly the same behaviors.
- Our proposal
  - Change gradient descent to (block) coordinate descent
  - Allow one-round communication delay

Scalability: Model Scheduling

- Model slices are pulled from server and updated in a round robin fashion.

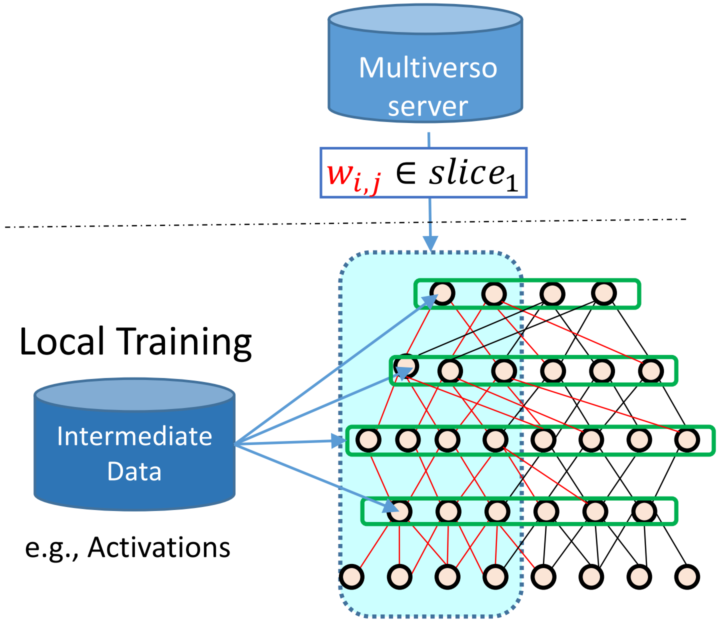


When updating $Slice_1$, previous information about $Slice_2$ is reused.

Stochastic (Block) Coordinate Descent (SCD)

when updating $Slice_2$, previous information about $Slice_1$ is reused.

Scalability: Model Scheduling



Theoretical guarantee

SCD and SGD have the similar convergence rate for λ-strongly convex problem; and both lead to local optima for non-convex problems.

Practical efficiency

- Lower comm cost (only model is transferred)

	Model Parallelism	Model Scheduling
LDA	Data ~ $O(10^9)$	Model ~ $O(10^7)$
CNN	Data ~ $O(10^9)$	Model ~ $O(10^4)$

- Robust to comm delay & machine failure

	Model Parallelism	Model Scheduling
Updates	Synchronous	Asynchronous

Efficiency: Hybrid Model Store

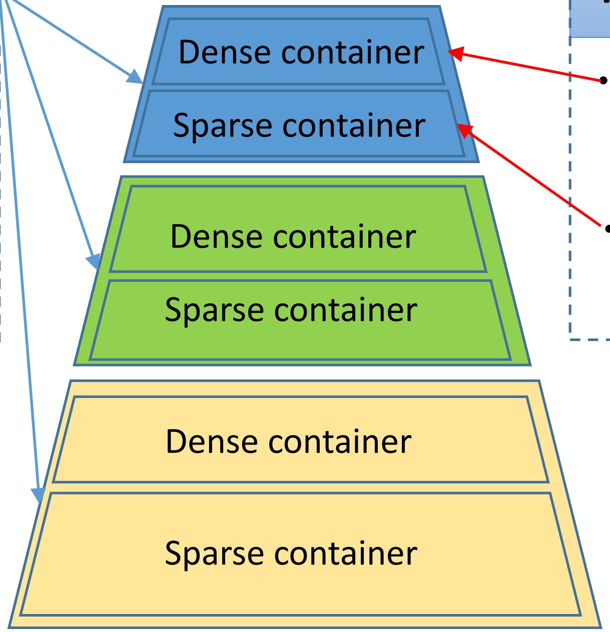
Typical scenarios

- Huge sparse model
  - Example: topic model
  - Dense format is prohibitively large and unnecessary
- Screwed model access
  - Example: word embedding
  - 0.1% terms are used in 90% training samples

Goal: High memory usage + model access speed

Multi-tier storage

- Separate storage of terms with different access frequencies
- High cache hit rate
- Balance between memory usage and access speed



Hybrid Store format

- Frequent term → topic vector is sparse → Hash table O(K)
- Rare term → topic vector is dense → Dense Array O(1).

Efficiency: Adaptive Pipelining

- Adaptively determine the optimal setting to match learning algorithms, disk speed, CPU/GPU speed, and network speed.

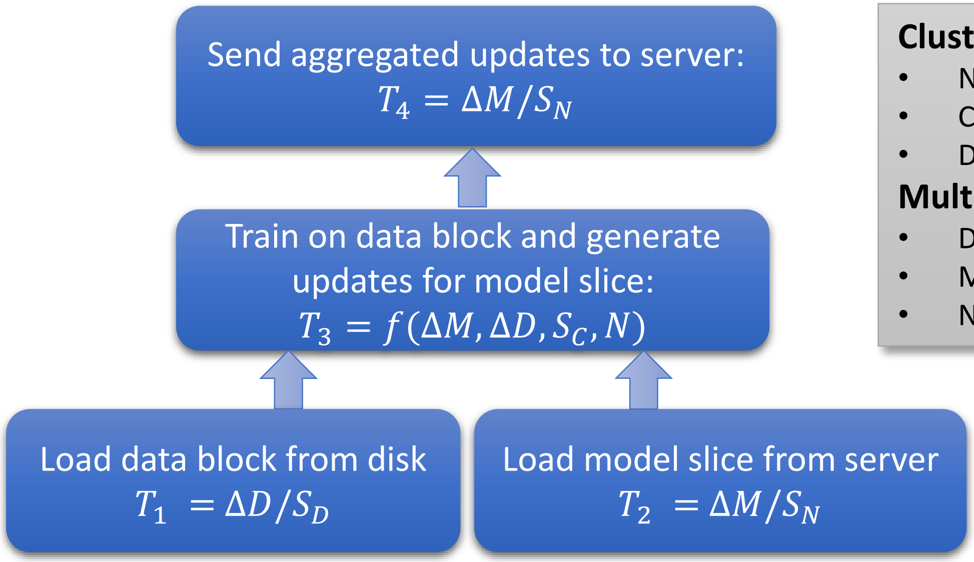
Perfect pipelining:

$$T_1 = T_2 = T_3 = T_4$$

Adaptive pipelining:

$$\min_{\Delta D, \Delta M, N} \sum_{i,j} \left| T_i - T_j \right|^2$$

- Online algorithm to adjust $\Delta D, \Delta M, N$.
- Efficient optimization since all T_i's are monotone functions w.r.t. $\Delta D, \Delta M, N$.



Cluster situations:
- Network speed: S_N
- CPU/GPU speed: S_C
- Disk speed: S_D

Multiverso settings
- Data block size: ΔD
- Model slice size: ΔM
- Number of threads: N

Flexibility: Customizable Model Representation and Aggregations

- Beyond matrix-form models and sum/average aggregation operators.

```
Interface IAggregation
{
      Public bool Aggregate(void* models, enum agg_type)
}
```

```
Class ParallelModel: IAggregation
{
       public virtual bool Aggregate(void* models,
void* inter_data, enum agg_type);
       private void* _models;//model parameters
       private void* _inter_data;//intermediate variables
}
//Pre-defined models data structure in Multiverso:
//Matrix (sparse/dense), Trees.

//Pre-defined aggregation operations:
//Weighted sum, Average, Voting, Max, Min, Histogram merge.
```

For DNN/Logistic Regression/LDA:
- `models = (sparse) matrix`
- `agg_type = Sum/Average`

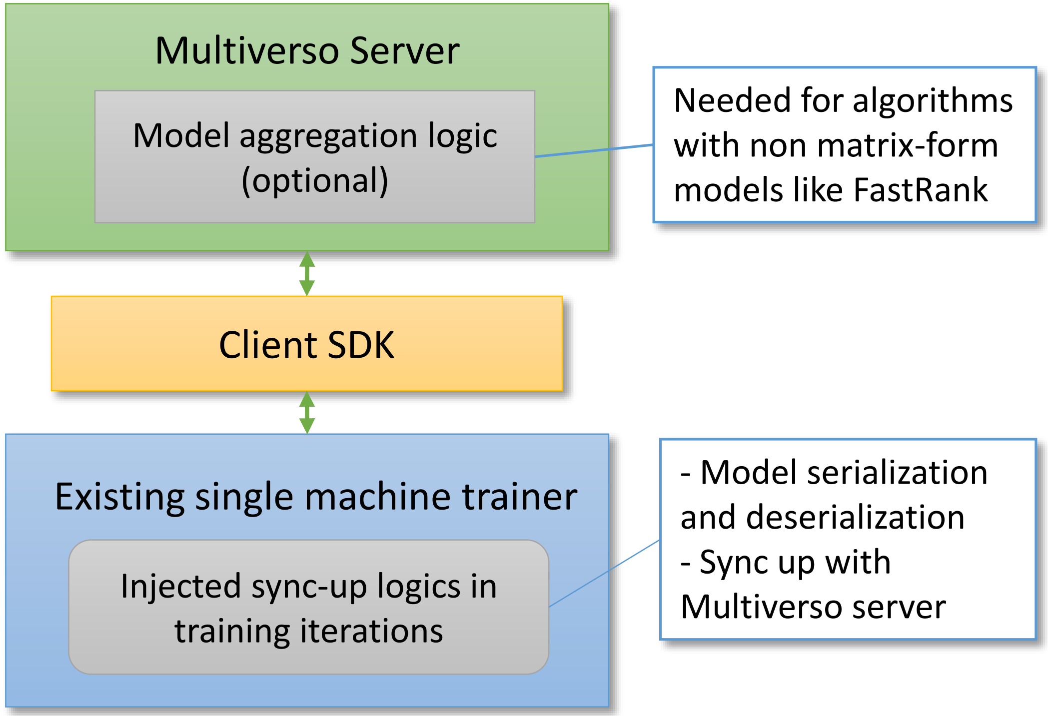
For FastRank/Decision trees:
- `models = trees(with split point information) + histogram`
- `agg_type = max info gain/histogram merge`

For Ensemble Models:
- `models = trees + (sparse) matrix + …`
- `agg_type = voting/max/min/weighted sum`

For other algorithms, one can implement their own model data structures and aggregation operators.

Flexibility: Plug-in Mode

- Scenario: existing codebase; model is dense and can fit into local machine memory.
- Examples: CNTK, CNN for image classification.

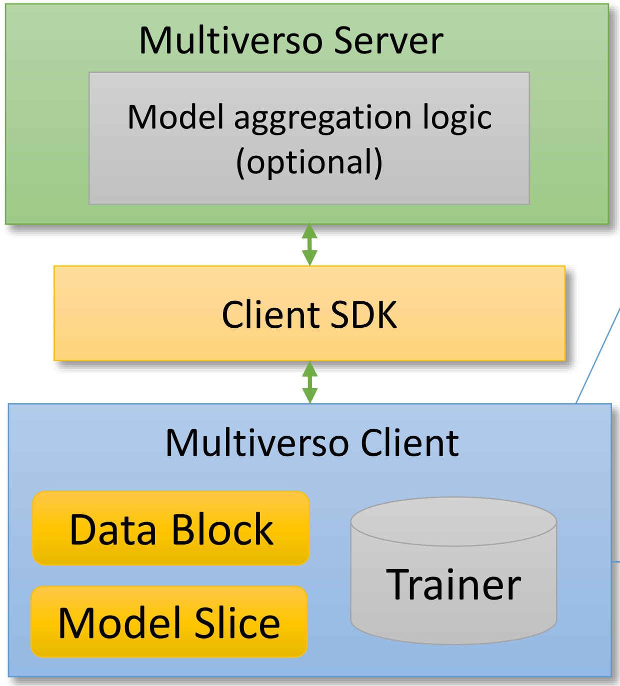


Tiny Code Changes

```
//initial parameter server
printf("@@@ Intializing parameter server ...\n");
_multiversoWrapper = new Multiverso::Wrapper();

_multiversoWrapper->Init(_adapterID, _strConfig);
printf("@@@ connected to the parameter server. \n");
Mode _test;

//Get model from parameter server finished;
_convNet->GetModelFromMultiverso();

for (int i = 0; i < GetNumMiniBatchs(); i++)
{
    _convNet->fprop(i, _test ? PASS_TEST : PASS_TRAIN);
    _convNet->getCost(batchCost);

    if (!_test)
    {
        _convNet->bprop(PASS_TRAIN);
        _convNet->updateWeights();
    }

    if (i % _sendInterv == 0)
        _convNet->SubmitModelToMultiverso(); //submit the update parameter finished

    if (i % _fetchInterv == 0)
         convNet->GetModelFromMultiverso(); //Get model from parameter server
}

//Training batch finished
if (!_test)
{
    //force to sync up in the last stage
    _convNet->SubmitModelToMultiverso();
    _convNet->GetModelFromMultiverso();
}
```

Flexibility: Embedded Mode

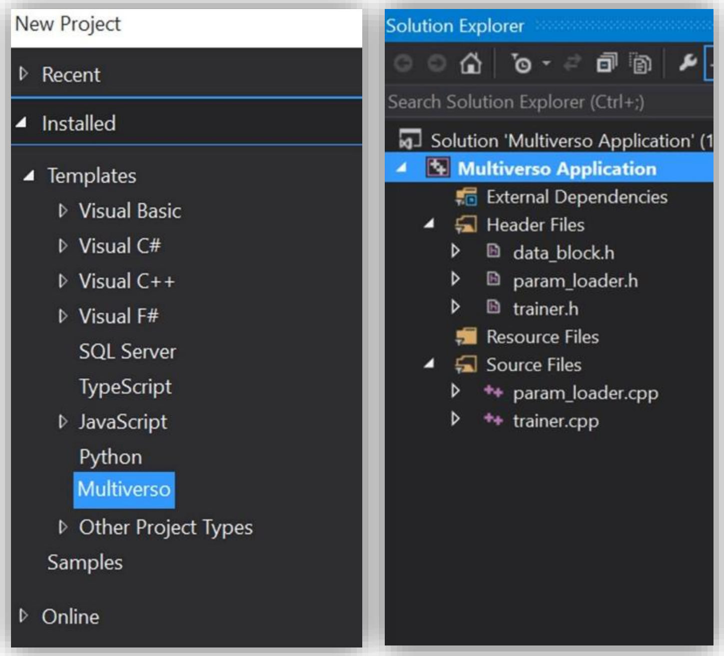
- Scenario: model exceeds single machine memory; sparse model training (only a small subset of model parameters are used when training a data block)
- Examples: LightLDA, Word Embeding, Logistic Regression.



User needs to define
- Data block & model slices
- Train logic for one data block
- Model parsing and update logics

Multiverso client manages the pipelining of the following procedures

- Training threads to obtain model updates
- Parameter loading thread to fetch model slices
- Local aggregation thread to aggregate and send out updates



Project template integrated with visual studio to assist algorithm developer

Record Breaking: Model Size & Training Speed

- Topic Models:

	Data Scale	Model Scale	#Core	Training time
Distributed LightLDA	$\mathbf{10^{11}}$	$\mathbf{10^{13}}$	**384**	**60 hrs**
Peacock LDA (Tencent)	10^{9}	10^{10}	3,000	50 hrs
Alias LDA (Google, Baidu, CMU)	10^{10}	10^{10}	10,000	70 hrs

- Word2vec:

	Data Scale	Model Scale	#Core	Training time
Distributed Word Embedding	$\mathbf{10^{11}}$	$\mathbf{10^{10}}$	**96**	**40 hrs**
Word2Vec (Google)	10^{11}	10^{8}	N/A	N/A

Rich Learning Algorithms on Multiverso

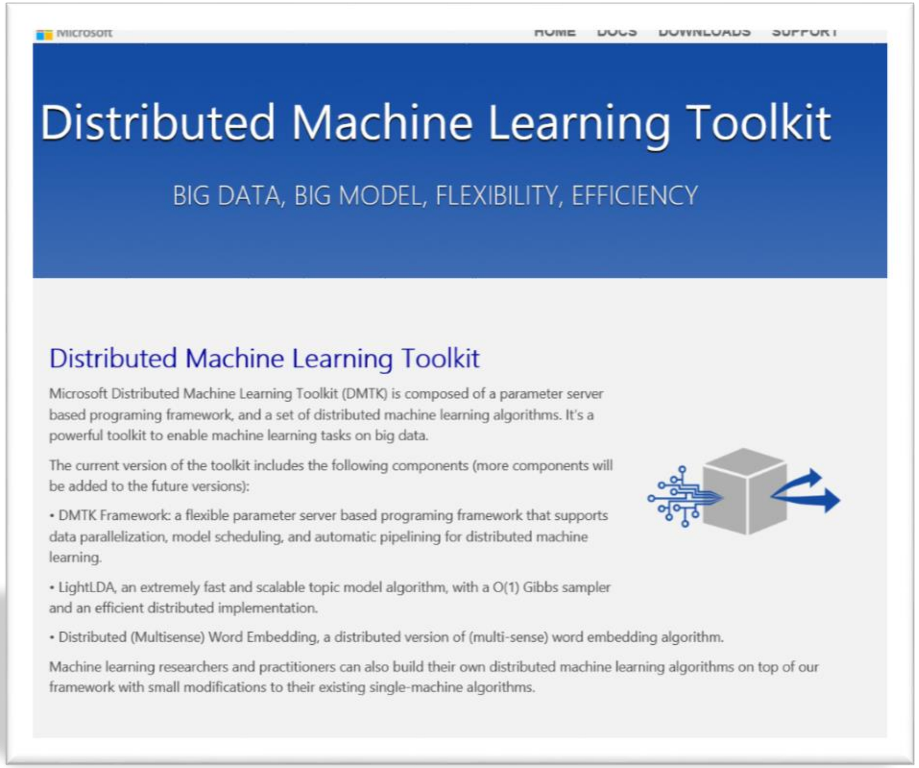
	LightLDA	Word2Vec	GBDT	LSTM	CNN	Online FTRL
Model	**20M** vocab, **1M** topics (largest topic model)	**10M** vocab, **1000** dim (largest word embedding)	**3000** trees (**120**-node) (GBDT)	**20M** parameters (4 hidden layer, LSTM)	**60M** parameters (AlexNet)	**800M** parameters (Logistic Regression)
Data	**200B** tokens (Bing web chunk)	**200B** samples (Bing web chunk)	**7M** records (Bing HRS data)	**375** hrs speech data (Win phone data)	**2M** images (ImageNet 1K dataset)	**6.4B** impressions (Bing Ads click log)
Training time	**60** hrs on **24** machines (nearly **linear** speed-up)	**40** hrs on **8** machines (nearly **linear** speed-up)	**3** hrs on **8** machines (**4**x of speed-up)	**11180** on **4** GPU (**3.8**x speed-up)	**2** hrs on **16** GPU cards (**12**x speed-up)	**2400**s on **24** machines (**12**x speed-up)

Our New Platform

Open Source

- Releasing to Github
  - https://github.com/Microsoft/multiverso
  - Containing a parameter server based framework, LightLDA and distributed word embedding
- Next steps:
  - Release more distributed machine learning algorithms, and new features of Multiverso.

http://dmtk.io



Future Research

- Data exchange vs. model exchange
- Data server vs. parameter server
- Adaptive communication filters
- Automatic hyper-parameter tuning
- Machine learning for distributed machine learning

Thanks!

tyliu@microsoft.com

http://research.microsoft.com/users/tyliu/