NOMAD: A Distributed Framework for Latent Variable Models

Inderjit S. Dhillon

Department of Computer Science
University of Texas at Austin

Joint work with H.-F. Yu, C.-J. Hsieh, H. Yun,
and S.V.N. Vishwanathan

NIPS 2014 Workshop:
Distributed Machine Learning and Matrix Computations

Outline

- Challenges
- Matrix Completion
  - Stochastic Gradient Method
  - Existing Distributed Approaches
  - Our Solution: NOMAD-MF
- Latent Dirichlet Allocation (LDA)
  - Gibbs Sampling
  - Existing Distributed Solutions: AdLDA, Yahoo LDA
  - Our Solution: F+NOMAD-LDA

Large-scale Latent Variable Modeling

- Latent Variable Models: very useful in many applications
  - Latent models for recommender systems (e.g., MF)
  - Topic models for document corpus (e.g., LDA)
- Fast growth of data
  - Almost 2.5×10^{18} bytes of data added each day
  - 90% of the world's data today was generated in the past two year

Challenges

- *Algorithmic* as well as *hardware* level
  - Many effective algorithms involve fine-grain iterative computation
    $\Rightarrow$ hard to parallelize
  - Many current parallel approaches
    - bulk synchronization
      $\Rightarrow$ wasted CPU power when communicating
    - complicated locking mechanism
      $\Rightarrow$ hard to scale to many machines
    - asynchronous computation using parameter server
      $\Rightarrow$ not serializable, danger of stale parameters
- Proposed NOMAD Framework
  - access graph analysis to exploit parallelism
  - asynchronous computation, non-blocking communication, and lock-free
  - serializable (or almost serializable)
  - successful applications: MF and LDA

Matrix Factorization: Recommender Systems

Recommender Systems

Rating Matrix

Items

Users

Users	Movie 1	Movie 2								Movie 10	Movie 11
Hsiang-Fu	1			5			3		5		2
Cho-Jui		2		3			5		2	5	
Si Si					3	?	5		3		
Inderjit	2		5			3		4		2	
Kai-Yang				5			5				1
Donghyuk		5			1				5		
Naga	1			1				2			4

Matrix Factorization Approach $A \approx WH^T$

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

1			5			3		5		2
	2		3			5		2	5	
				3		5		3		
2		5			3		4		2	
			5			5				1
	5			1				5		
1			1				2			4

Matrix Factorization Approach $A \approx WH^T$

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

1			5			3		5		2
	2		3			5		2	5	
				3	?	5		3		
2		5			3		4		2	
			5			5				1
	5			1				5		
1			1				2			4

Matrix Factorization Approach

$$\min_{\substack{W \in \mathcal{R}^{m \times k} \\ H \in \mathcal{R}^{n \times k}}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2 + \lambda \left(\|W\|_F^2 + \|H\|_F^2 \right),$$

- $\Omega = \{(i,j) \mid A_{ij} \text{ is observed}\}$
- Regularized terms to avoid over-fitting

A transform maps users/items to latent feature space $\mathbb{R}^k$

- the i^{th} user $\Rightarrow$ i^{th} row of W, w_i^T,
- the j^{th} item $\Rightarrow$ j^{th} column of H^T, h_j.
- $w_i^T h_j$: measures the interaction.

SGM: Stochastic Gradient Method

SGM update: pick $(i, j) \in \Omega$

- $R_{ij} \leftarrow A_{ij} - w_i^T h_j$,
- $w_i \leftarrow w_i - \eta(\frac{\lambda}{|\Omega_i|} w_i - R_{ij} h_j)$,
- $h_j \leftarrow h_j - \eta(\frac{\lambda}{|\bar{\Omega}_j|} h_j - R_{ij} w_i)$,

  Ω_i: observed ratings of i-th row.

  $\bar{\Omega}_j$: observed ratings of j-th column.

$$\begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix}$$

$$\begin{pmatrix} w_1^T \\ w_2^T \\ w_3^T \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

An iteration : $|\Omega|$ updates

- Time per update: $O(k)$
- Time per iteration: $O(|\Omega| k)$,

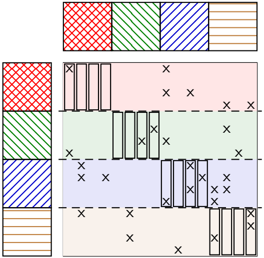
  better than $O(|\Omega| k^2)$ for ALS

Parallel Stochastic Gradient Descent for MF

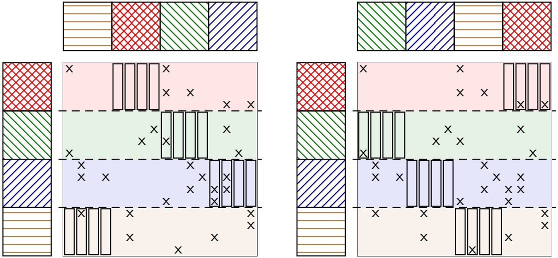
Challenge: direct parallel updates $\Rightarrow$ memory conflicts.

- Multi-core parallelization
  - Hogwild [Niu 2011]
  - Jellyfish [Recht et al, 2011]
  - FPSGD** [Zhuang et al, 2013]
- Multi-machine parallelization:
  - DSGD [Gemulla et al, 2011]
  - DSGD ++ [Teflioudi et al, 2013]

DSGD/JellyFish [Gemulla et al, 2011; Recht et al, 2011]



Synchronize and communicate



Synchronize and communicate

Proposed Asynchronous Approach: NOMAD-MF [Yun et al, 2014]

Motivation

Most existing parallel approaches require

- **Synchronization** and/or
  - E.g., ALS, DSGD/JellyFish, DSGD++, CCD++
  - Computing power is wasted:
    - Interleaved computation and communication
    - Curse of the last reducer
- **Locking** and/or
  - E.g., parallel SGD, FPSGD**
  - A standard way to avoid conflict and guarantee *serializability*
  - Complicated remote locking slows down the computation
  - Hard to implement efficient locking on a distributed system
- **Computation using stale values**
  - E.g., Hogwild, Asynchronous SGD using parameter server
  - Lack of serializability

Q: Can we avoid both *synchronization* and *locking* but keep CPU from being *idle* and guarantee *serializability*?

Our answer: NOMAD

A: Yes, NOMAD keeps CPU and network busy simultaneously

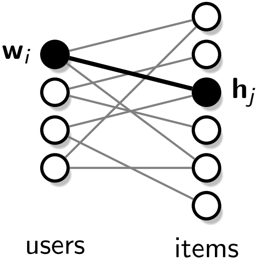
- Stochastic gradient update rule
  - only a small set of variables involved
- *Nomadic token passing*
  - widely used in telecommunication area
  - avoids conflict without explicit remote locking
  - Idea: “owner computes”
  - NOMAD: multiple “active tokens” and nomadic passing

Features:

- fully asynchronous computation
- lock-free implementation
- non-blocking communication
- serializable update sequence

Access Graph for Stochastic Gradient

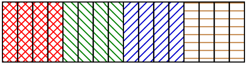
- Access graph $G = (V, E)$:
  - $V = \{\mathbf{w}_i\} \cup \{\mathbf{h}_j\}$
  - $E = \{e_{ij} : (i, j) \in \Omega\}$
- Connection to SG:
  - each e_{ij} corresponds to a SG update
  - only access to $\mathbf{w}_i$ and $\mathbf{h}_j$
- Parallelism:
  - edges without common node can be updated in parallel
  - identify "matching" in the graph
- Nomadic Token Passing:
  - mechanism s.t. active edges always form a "matching"
  - serializability guaranteed



More Details

Nomadic Tokens for $\{\mathbf{h}_j\}$:

- n tokens
- (j, h_j): $O(k)$ space



Worker:

- p workers
- a computing unit + a concurrent token queue
- a block of W: $O(mk/p)$
- a block row of A: $O(|\Omega|/p)$

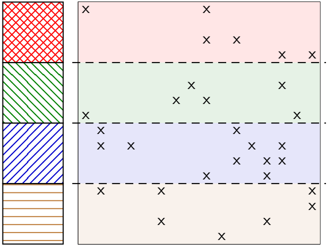


Illustration of NOMAD communication

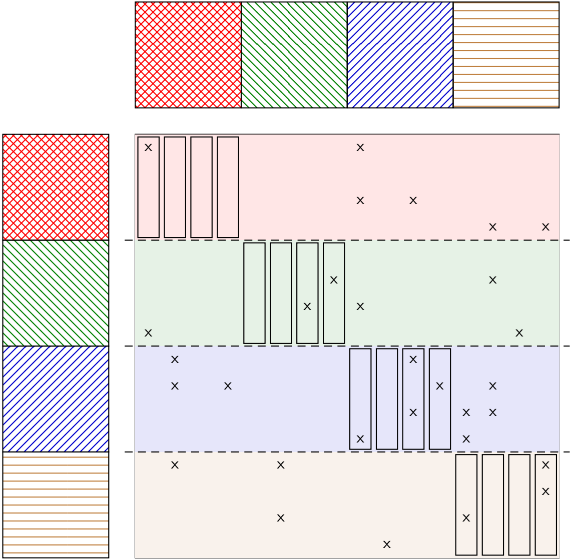


Illustration of NOMAD communication

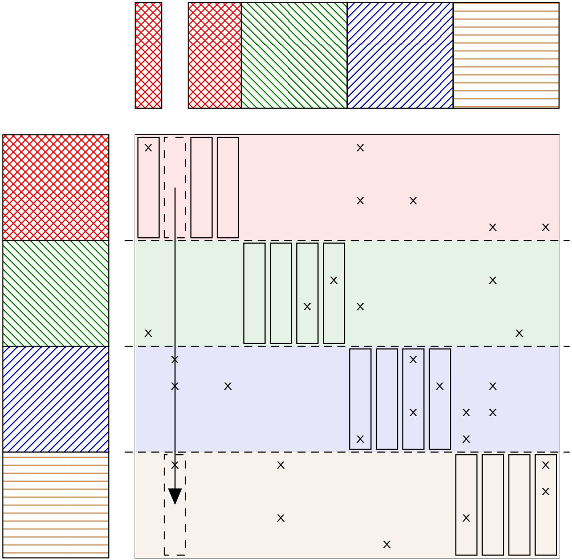


Illustration of NOMAD communication

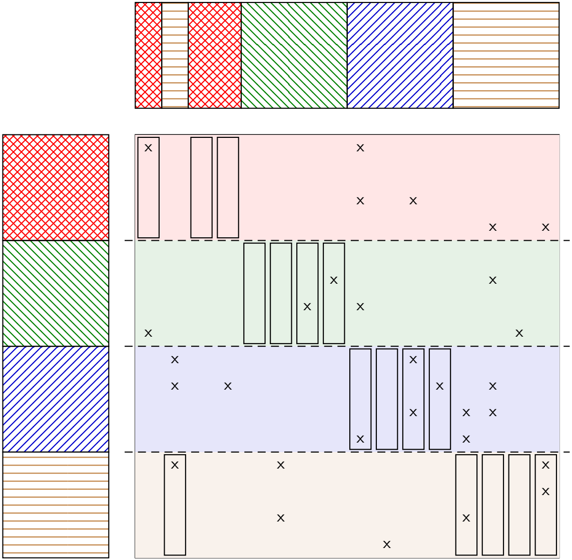


Illustration of NOMAD communication

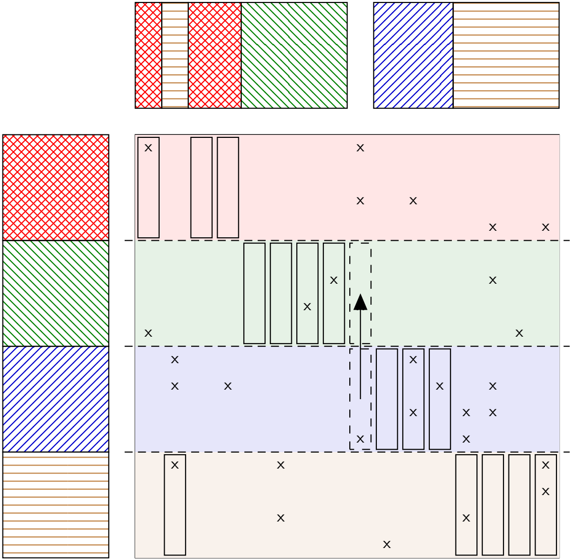


Illustration of NOMAD communication

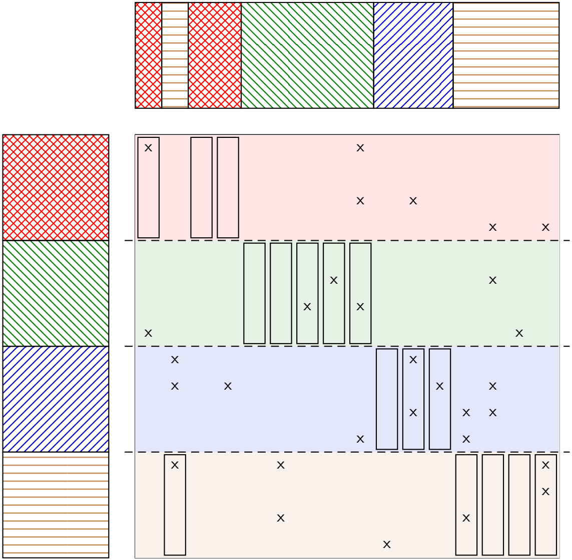


Illustration of NOMAD communication

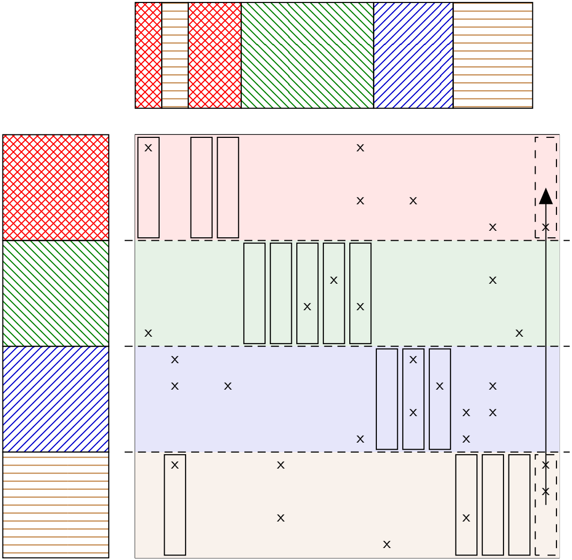


Illustration of NOMAD communication

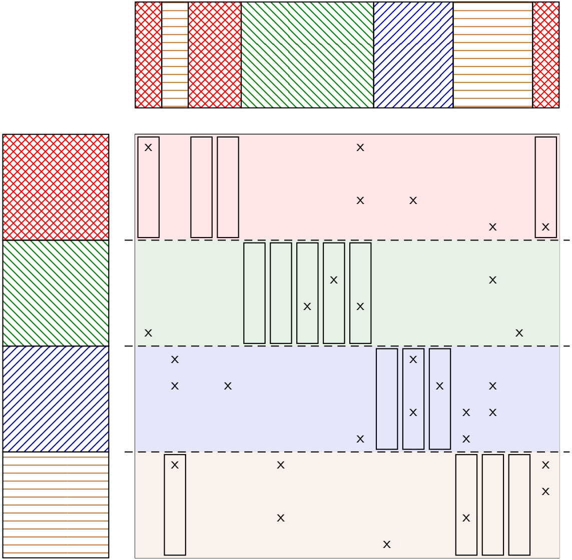


Illustration of NOMAD communication

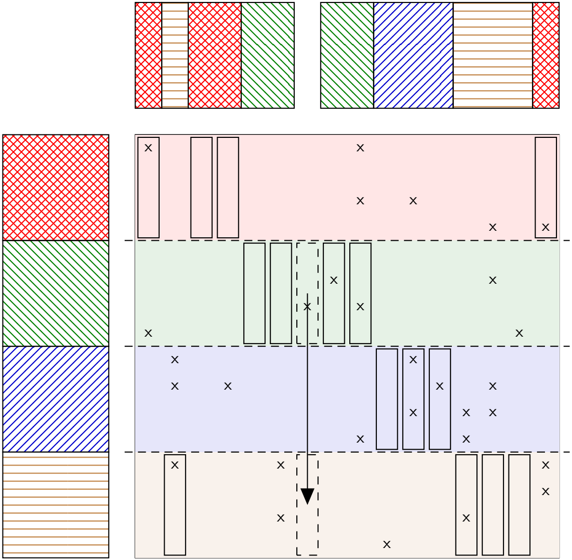


Illustration of NOMAD communication

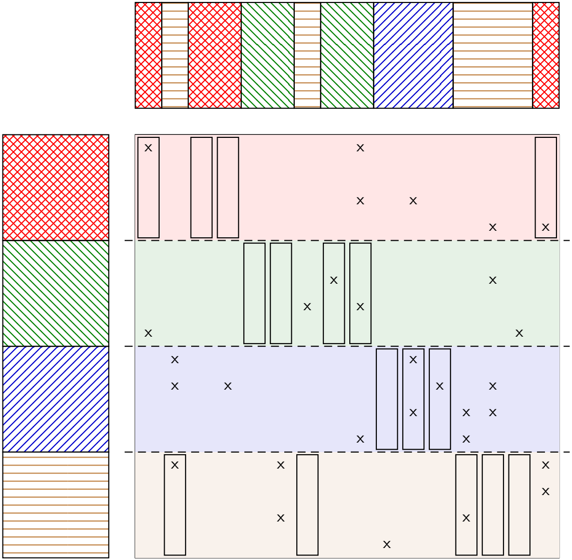


Illustration of NOMAD communication

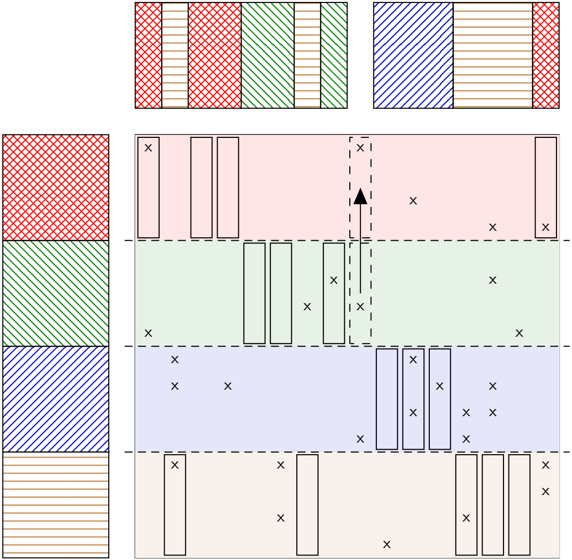
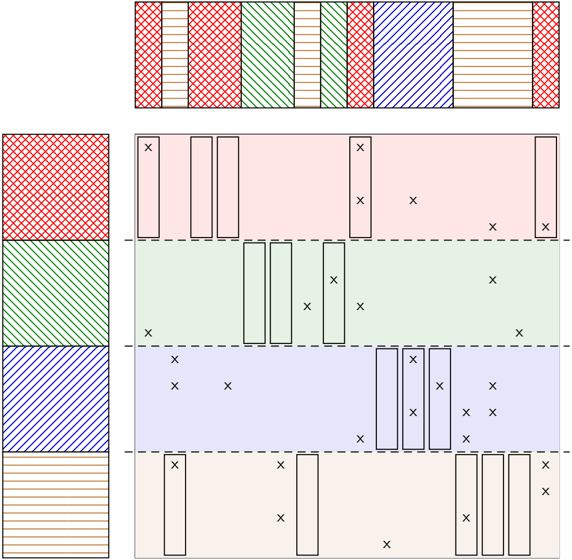


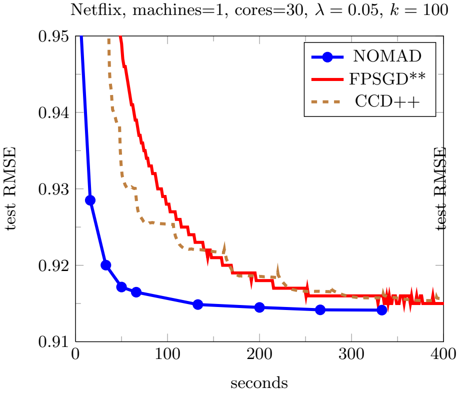
Illustration of NOMAD communication



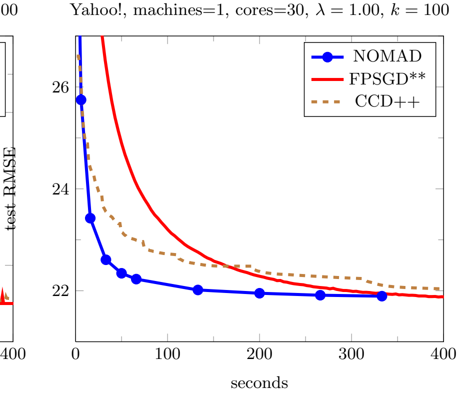
Comparison on a Multi-core System

- On a 32-core processor with enough RAM.
- Comparison: NOMAD, FPSGD**, and CCD++.

(100M ratings)



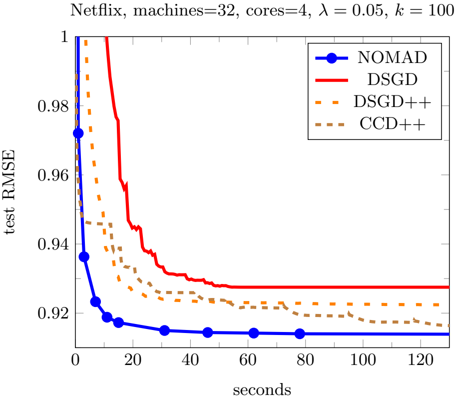
(250M ratings)



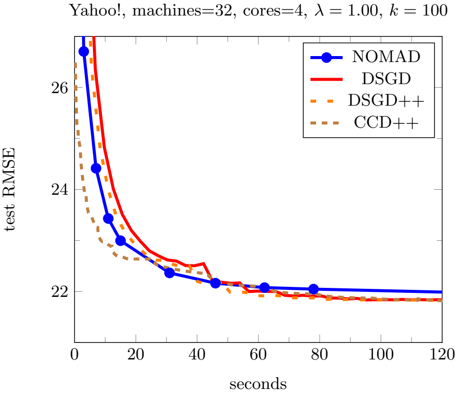
Comparison on a Distributed System

- On a distributed system with 32 machines.
- Comparison: NOMAD, DSGD, DSGD++, and CCD++.

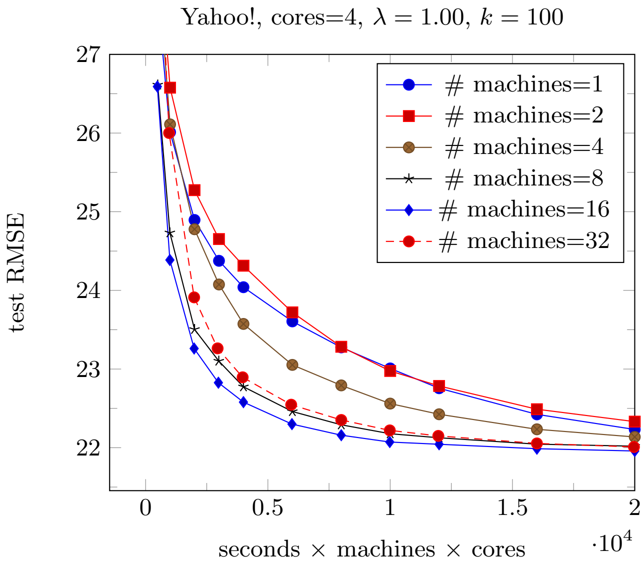
(100M ratings)



(250M ratings)

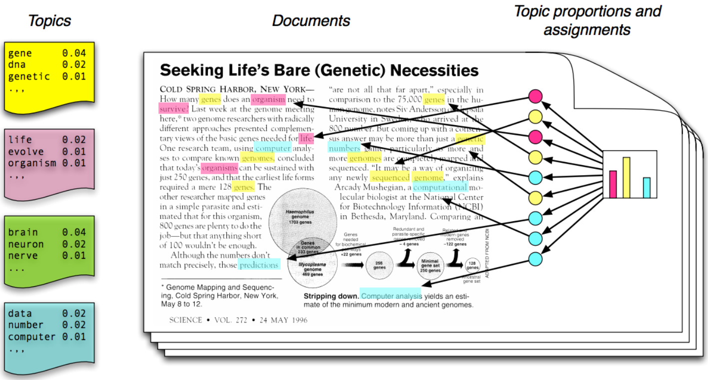


Super Linear Scaling of NOMAD-MF



Topic Modeling: Latent Dirichlet Allocation

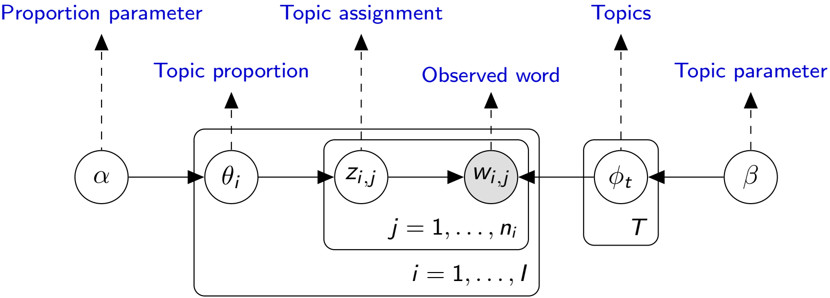
Latent Dirichlet Allocation (LDA)



- Each **topic** is a multinomial distribution over words
- Each **document** is a multinomial distribution over topics
- Each **word** is drawn from one of these topics

[1]source: http://www.cs.columbia.edu/~blei/papers/icml-2012-tutorial.pdf

Graphical Model for LDA

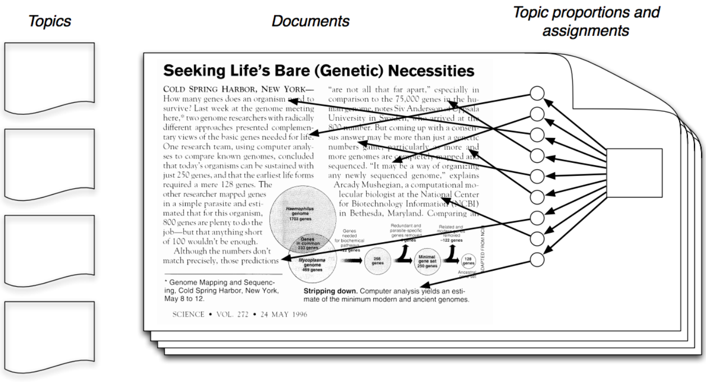


- Joint distribution

$$Pr(\cdot) = \prod_{t=1}^{T} Pr(\phi_t \mid \beta) \prod_{i=1}^{I} Pr(\theta_i \mid \alpha) \left(\prod_{j=1}^{n_i} Pr(z_{i,j} \mid \theta_i) Pr(w_{i,j} \mid \phi_{z_{i,j}}) \right)$$

- $Pr(\phi_t \mid \beta)$, $Pr(\theta_i \mid \alpha)$: Dirichlet distributions
- $Pr(w \mid \phi_t)$, $Pr(z \mid \theta_i)$: multinomial distributions

Inference for LDA



- Only documents are observed
- $\theta_t, \phi_t, z_{i,j}$ are latent
- Goal: infer these latent structures

[1] source: http://www.cs.columbia.edu/~blei/papers/icml-2012-tutorial.pdf

Posterior Inference for LDA

Task: $Pr(\theta_i, \phi_t, z_{i,j} \mid \{d_i\}, \alpha, \beta)$

- Given
  - a corpus of documents $\{d_i : i = 1, \ldots, N\}$, α, β
  - each document $d_i = \{w_{i,j} : j = 1, \ldots, n_i\}$
- Exact inference for $z_{i,j}$, θ_i, ϕ_t
  - Intractable
  - Latent variables are dependent when conditioned on data

Approximate Inference approaches:

- Variational Methods
  - See [Blei et al, 2003]
  - an optimization approach
  - runs faster
  - but generates biased results
- Gibbs Samplings
  - See [Griffiths & Steyvers, 2004]
  - an MCMC approach
  - more accurate
  - but slower with a vanilla implementation

Goal: Design a scalable Gibbs sampler for LDA

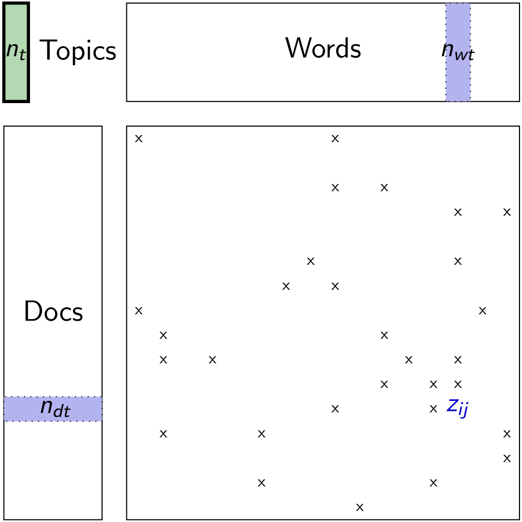
Gibbs Sampling for LDA [Griffiths & Steyvers, 2004]

- Count matrices for topic assignment $\{z_{i,j}\}$:
  - n_{dt}: # words of document d assigned to topic t
  - n_{wt}: # of times word w assigned to topic t
  - $n_t := \sum_w n_{wt} = \sum_d n_{dt}$
- Gibbs Sampling Step
  1. choose $w := w_{i,j}$ with old assignment $t_o := z_{i,j}$ of document $d := d_i$
  2. Decrease n_{dt_o}, n_{wt_o}, n_{t_o} by 1
  3. Resample a new assignment $t_n := z_{i,j}$ according to

$$Pr(z_{i,j} = t) \propto \frac{(n_{dt} + \alpha)(n_{wt} + \beta)}{n_t + \bar{\beta}}, \quad \forall t = 1, \ldots, T.$$

  4. Increase n_{dt_n}, n_{wt_n}, n_{t_n} by 1
- Constants
  - J: vocabulary size
  - $\bar{\beta} = \beta \times J$

Access Pattern for Gibbs Sampling



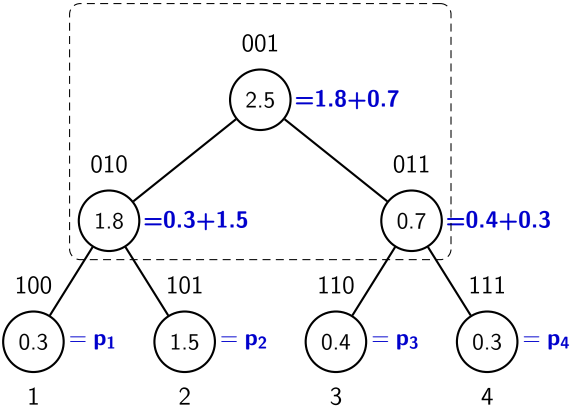
Multinomial Sampling Techniques for $\mathbf{p} \in R_+^T$

	Initialization Time	Initialization Space	Generation Time	Parameter Update Time
LSearch	$\Theta(T)$	$\Theta(1)$	$\Theta(T)$	$\Theta(1)$
BSearch	$\Theta(T)$	$\Theta(1)$	$\Theta(\log T)$	$\Theta(T)$
Alias Method	$\Theta(T)$	$\Theta(T)$	$\Theta(1)$	$\Theta(T)$
F+tree Sampling	$\Theta(T)$	$\Theta(1)$	$\Theta(\log T)$	$\Theta(\log T)$

- LSearch
  - maintain $c_T = \mathbf{p}^\top \mathbf{1}$
  - linear search
  - $\Theta(1)$ update
- BSearch
  - maintain $\mathbf{c} = \texttt{cumsum}(\mathbf{p})$
  - binary search
  - no support for update
- Alias Method
  - Alias table
  - construction has some overhead
  - no support for updates
- F+tree
  - a variant of Fenwick tree
  - construction has low overhead
  - logarithmic time for sampling and update

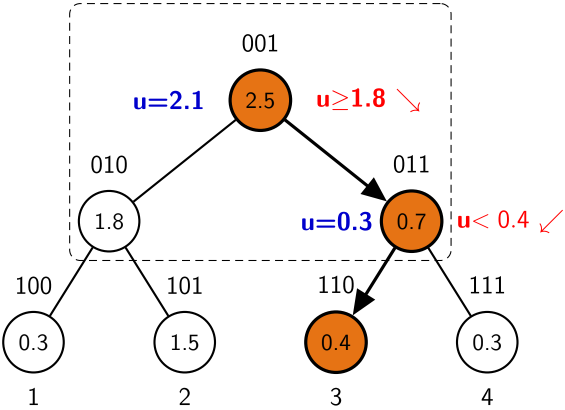
F+Tree: Construction

- Construction in $\Theta(T)$ time
- $\mathbf{p} = [0.3, 1.5, 0.4, 0.3]^\top$



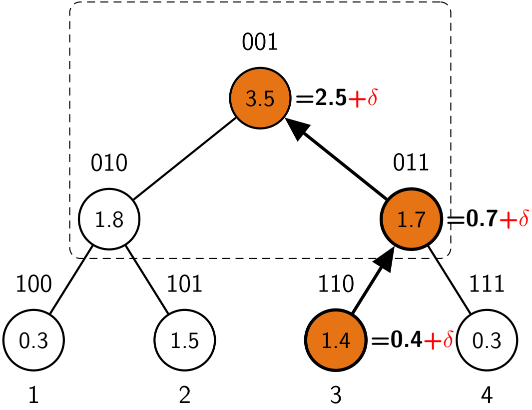
F+Tree: Sampling

- Multinomial sampling in $\Theta(\log T)$ time
- Initial u: a uniformly number drawn from $[0, F[1])$



F+Tree: Update

- Update in $\Theta(\log T)$ time
- $p_3 \leftarrow p_3 + \delta$



F+LDA = LDA with F+tree Sampling

- Decomposition of $\mathbf{p}$

$$p_t = \frac{(n_{dt}+\alpha)(n_{wt}+\beta)}{n_t+\bar{\beta}}, \quad \forall t = 1,\ldots,T.$$
$$= \beta \underbrace{\left(\frac{n_{dt}+\alpha}{n_t+\bar{\beta}}\right)}_{q_t} + \underbrace{n_{wt}\left(\frac{n_{dt}+\alpha}{n_t+\bar{\beta}}\right)}_{r_t}. \tag{1}$$

- $\mathbf{p} = \beta\mathbf{q} + \mathbf{r}$
  - two-level sampling for $\mathbf{p}$
- $\mathbf{q}$ is dense
  - only 2 entries (q_{t_o}, q_{t_n}) change for each Gibbs step in the same document
  - use F+Tree for $\mathbf{q}$
- $\mathbf{r}$ is sparse
  - nonzero entries: $T_w := \{t : n_{tw} \neq 0\}$
  - entire $\mathbf{r}$ changes for each Gibbs step
  - use BSearch for $\mathbf{r}$
- Can also work on word-by-word update sequence

F+LDA: Alternative Decomposition

- Word-by-word Gibbs sampling sequence
- Decomposition of $\mathbf{p}$

$$p_t = \frac{(n_{dt} + \alpha)(n_{wt} + \beta)}{n_t + \bar{\beta}}, \quad \forall t = 1, \ldots, T.$$
$$= \alpha \underbrace{\left(\frac{n_{wt} + \beta}{n_t + \bar{\beta}} \right)}_{q_t} + \underbrace{n_{dt} \left(\frac{n_{wt} + \beta}{n_t + \bar{\beta}} \right)}_{r_t}. \tag{2}$$

- $\mathbf{p} = \alpha \mathbf{q} + \mathbf{r}$
- $\mathbf{q}$: slight changes for this sequence $\Rightarrow$ use F+Tree
- $\mathbf{r}$: $|T_d := \{t : n_{dt} \neq 0\}|$ nonzeros $\Rightarrow$ use BSearch

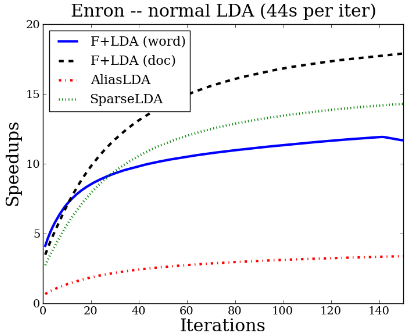
Comparison to Other LDA Sampling

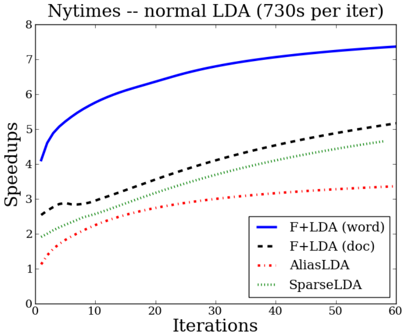
	F+LDA		F+LDA		Sparse-LDA			Alias-LDA	
Sequence	Word-by-Word		Doc-by-Doc		Doc-by-Doc			Doc-by-Doc	
Exact?	Yes		Yes		Yes			No	
Decomposition	$\alpha\left(\frac{n_{wt}+\beta}{n_t+\beta}\right)$	$+n_{dt}\left(\frac{n_{wt}+\beta}{n_t+\beta}\right)$	$\beta\left(\frac{n_{dt}+\alpha}{n_t+\beta}\right)$	$+n_{wt}\left(\frac{n_{dt}+\alpha}{n_t+\beta}\right)$	$\frac{\alpha\beta}{n_t+\beta}$	$+\beta\left(\frac{n_{dt}}{n_t+\beta}\right)$	$+n_{wt}\left(\frac{n_{dt}+\alpha}{n_t+\beta}\right)$	$\alpha\left(\frac{n_{wt}+\beta}{n_t+\beta}\right)$	$+n_{dt}\left(\frac{n_{wt}+\beta}{n_t+\beta}\right)$
Structure	F+tree	BSearch	F+tree	BSearch	LSearch	LSearch	LSearch	Alias	Alias
Fresh samples	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Initialization	$\Theta(\log T)$	$\Theta(\lvert T_d\rvert)$	$\Theta(\log T)$	$\Theta(\lvert T_w\rvert)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\lvert T_w\rvert)$	$\Theta(1)$	$\Theta(\lvert T_d\rvert)$
Sampling	$\Theta(\log T)$	$\Theta(\log\lvert T_d\rvert)$	$\Theta(\log T)$	$\Theta(\log\lvert T_w\rvert)$	$\Theta(T)$	$\Theta(\lvert T_d\rvert)$	$\Theta(\lvert T_w\rvert)$	$\Theta(\#MH)$	$\Theta(\#MH)$

- F+LDA: word-by-word faster than doc-by-doc for large I
  - $|T_d|$ bounded by n_i, but $|T_w|$ approaches to T
  - per Gibbs step cost: $\rho_F \log T + \rho_B |T_d|$
- SparseLDA:
  - per Gibbs step cost: $\Theta(T + |T_d| + |T_w|)$
  - the first $\Theta(T)$ rarely happens but $|T_w| \to T$ for large I
- AliasLDA:
  - per Gibbs step cost: $\rho_A |T_d| + \#MH$
  - $\rho_A \approx 3 \times \rho_B$: construction overhead of Alias table
  - If $(\rho_A - \rho_B)|T_d| > \rho_F \log T \Rightarrow$ AliasLDA slower than F+LDA
  - say $|T_d| \approx 100$, F+LDA still faster for $T < 2^{50}$

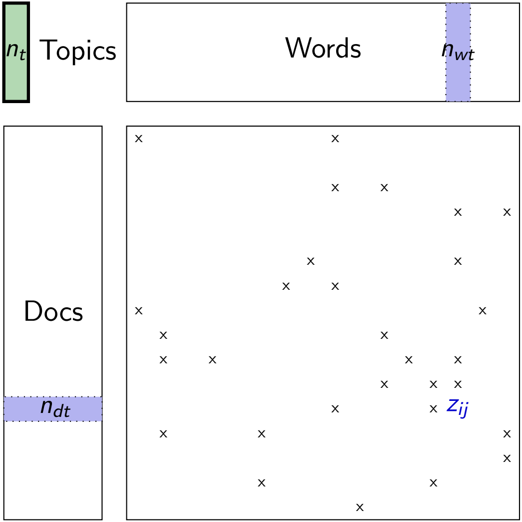
Comparison of various sampling methods

- Single machine, single thread
- y-axis: speedup over normal $O(T)$ multinomial sampling
- Enron: 38K docs with 6M tokens
- NyTimes: 0.3M docs with 100M tokens





Access Pattern for Gibbs Sampling

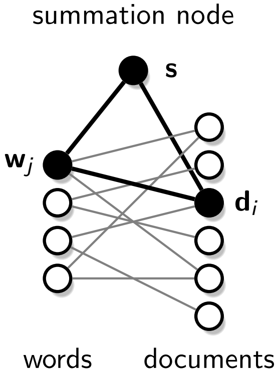


Access Graph for Gibbs Sampling

- $G = (V, E)$: a hyper graph

$$V = \{\mathbf{d}_i\} \cup \{\mathbf{w}_j\} \cup \{\mathbf{s}\}$$
$$E = \{e_{ij} = (\mathbf{d}_i, \mathbf{w}_j, \mathbf{s})\}$$

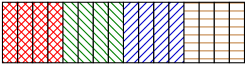
- Connection to Gibbs sampling
  - $(\mathbf{d}_i)_t := n_{d_i t}$, $(\mathbf{w}_j)_t := n_{w_j t}$, $(\mathbf{s})_t := n_t$
  - each e_{ij}: a Gibbs step for word w_j in d_i access to $(\mathbf{d}_i, \mathbf{w}_j, \mathbf{s})$
- Parallelism: more challenging
  - all edges incident to $\mathbf{s}$
  - all $(\mathbf{s})_t$ are large in general
    $\Rightarrow$ slightly stale $\mathbf{s}$ is fine for accuracy
  - duplicate $\mathbf{s}$ for parallelism



Nomadic Tokens for $\mathbf{w}_j$

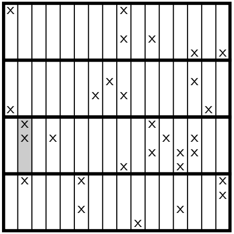
Nomadic Tokens for $\{\mathbf{w}_j : j = 1, \ldots, J\}$:

- J tokens
- $(j, \mathbf{w}_j)$: $O(T)$ space



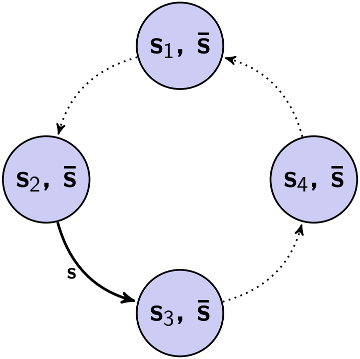
Worker:

- p workers
- a computing unit + a concurrent token queue
- a subset of $\{\mathbf{d}_i\}$: $O(IT/p)$
- "x": an occurrence of a word
- bigger rectangle: a subset of corpus
- smaller rectangle: a unit subtask



Nomadic Token for $\mathbf{s}$: Circular Delta Update

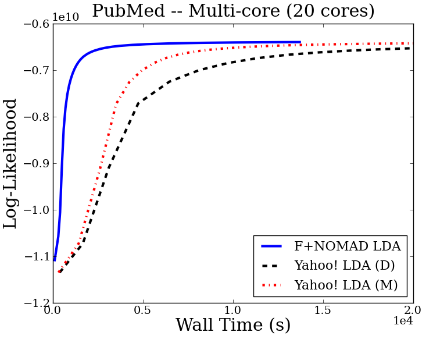
- Single global $\mathbf{s}$
  - travels among machines as a messenger
  - broadcasts local delta updates
- Every machine p: $(\mathbf{s}_p, \bar{\mathbf{s}})$
  - $\mathbf{s}_p$: local working copy
  - $\bar{\mathbf{s}}$: snapshot version of global $\mathbf{s}$

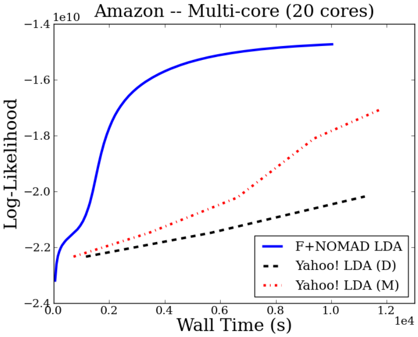


$$\mathbf{s} \leftarrow \mathbf{s} + (\mathbf{s}_3 - \bar{\mathbf{s}})$$
$$\bar{\mathbf{s}} \leftarrow \mathbf{s}$$
$$\mathbf{s}_3 \leftarrow \mathbf{s}$$

Comparison on a single multi-core machine

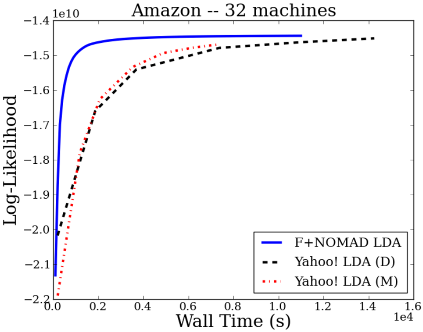
- On a machine with a 20-core processor
- Comparison: F+NOMAD LDA, Yahoo! LDA
- PubMed: 9M docs with 700M tokens
- Amazon: 30M docs with 1.5B tokens

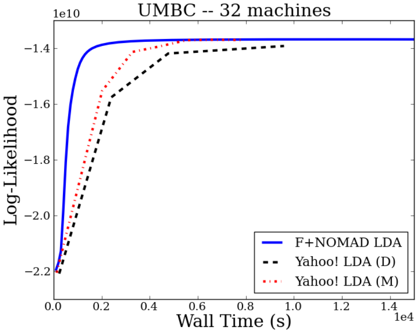




Comparison on a Multi-machine System

- 32 machines, each with a 20-core processor.
- Comparison: F+NOMAD LDA, Yahoo! LDA
- Amazon: 30M docs with 1.5B tokens
- UMBC: 40M docs with 1.5B tokens





Conclusions

- NOMAD framework uses nomadic tokens to provide
  - Asynchronous computation
  - Non-blocking communication
  - Lock-free implementation
  - Serializable or near Serializable
- Recommender System: Matrix factorization
  - scalable parallel stochastic gradient
  - Serializability guarantee
- Topic Modeling: Latent Dirichlet Allocation
  - Logarithmic F+tree sampling
  - Efficient Gibbs Sampling
  - Duplicated nomadic tokens for the common node
  - Outperforms Yahoo! LDA