Nomadic Computing for Big Data Analytics

Original authors: Hsiang-Fu Yu, Cho-Jui Hsieh, Hyokun Yun, S.V.N. Vishwanathan, Inderjit Dhillon

November 15, 2017

Introduction

Analysis of big data

Two approaches of big data analysis

- Stochastic optimization & inference (sequential);
- Distributed computing based on MapReduce.

Introduction

Analysis of big data

Two approaches of big data analysis

- Stochastic optimization & inference (sequential);
- Distributed computing based on MapReduce.

Stochastic optimization and inference algorithms are inherently sequential, making them hard to be run on distributed system.

Introduction

Analysis of big data

Two approaches of big data analysis

- Stochastic optimization & inference (sequential);
- Distributed computing based on MapReduce.

Stochastic optimization and inference algorithms are inherently sequential, making them hard to be run on distributed system.

NOMAD (Nonlocking, stOchastic Multimachine framework for Asynchronous and Decentralized computation), is presented in this paper, which combines stochastic optimization's and distributed computing's advantages without incurring their drawbacks.

Matrix Completion

Some notations

Notation	Meaning
$\langle \cdot, \cdot \rangle$	the Euclidean inner product of two vectors
Ω_i	$\{j : (i,j) \in \Omega\}$
$\overline{\Omega}_j$	$\{i : (i,j) \in \Omega\}$
$\lvert \cdot \rvert$	The cardinality of a set
$\lVert \cdot \rVert$	The L^2 norm of a vector
$[a..b]$	$\{a, a+1, \cdots, b\}$

Matrix Completion

The problem

Matrix completion problem

Given an incomplete matrix $A \in R^{m \times n}$, where only entries of indices $(i, j) \in \Omega \subset [1..m] \times [1..n]$ are known. The task is to find two matrices $W \in R^{m \times k}$ and $H \in R^{k \times n}$ with $k \ll \min\{m, n\}$, such that $A \approx WH^{\mathsf{T}}$.

Matrix Completion

The problem

Matrix completion problem

Given an incomplete matrix $A \in R^{m \times n}$, where only entries of indices $(i, j) \in \Omega \subset [1..m] \times [1..n]$ are known. The task is to find two matrices $W \in R^{m \times k}$ and $H \in R^{k \times n}$ with $k \ll \min\{m, n\}$, such that $A \approx WH^{\mathsf{T}}$.

The matrix W can be considered as m k-dimensional row vectors, and H as n k-dimensional column vectors. Then, for every entry A_{ij}, it is simply the inner product of the corresponding row and column vectors:

$$A_{ij} = \langle \mathbf{w}_i, \mathbf{h}_j \rangle$$

Matrix Completion

The problem

We use loss function to measure the goodness of the model, typically given by mean squared error:

$$\frac{1}{2|\Omega|}\sum_{(i,j)\in\Omega}(A_{ij}-\langle\mathbf{w}_i,\mathbf{h}_j\rangle)^2$$

also, we need a regularizer to avoid overfitting

$$\frac{\lambda}{2}(\sum_{i=1}^{m}|\Omega_i|\cdot\|\mathbf{w}_i\|^2+\sum_{i=1}^{n}|\overline{\Omega}_j|\cdot\|\mathbf{h}_j\|^2)$$

Combine these two items, the problem can be described as

$$\min_{W,H} J(W,H)=\frac{1}{2|\Omega|}\sum_{(i,j)\in\Omega}(A_{ij}-\langle\mathbf{w}_i,\mathbf{h}_j\rangle)^2+\text{regularizer}$$

Matrix Completion

Stochastic gradient descent

Take the gradients of the target function with respect to $\mathbf{w}_i$ and $\mathbf{h}_j$:

$$\nabla_{\mathbf{w}_i} J(W,H) = -(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{h}_j + \lambda \mathbf{w}_i$$
$$\nabla_{\mathbf{h}_j} J(W,H) = -(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{w}_i + \lambda \mathbf{h}_j$$

Matrix Completion

Stochastic gradient descent

Take the gradients of the target function with respect to $\mathbf{w}_i$ and $\mathbf{h}_j$:

$$\nabla_{\mathbf{w}_i} J(W,H) = -(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{h}_j + \lambda \mathbf{w}_i$$
$$\nabla_{\mathbf{h}_j} J(W,H) = -(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{w}_i + \lambda \mathbf{h}_j$$

The function decrease at the greatest rate in the opposite direction of the gradient of a function points, so we tend to move our approximation to somewhere in the opposite the direction of the gradient:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - s_t[-(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{h}_j + \lambda \mathbf{w}_i] \tag{1}$$
$$\mathbf{h}_j \leftarrow \mathbf{h}_j - s_t[-(A_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{w}_i + \lambda \mathbf{h}_j] \tag{2}$$

where s_t is called the learning rate.

Matrix Completion

Stochastic gradient descent

Repeatedly performing such updates, we will finally obtain an optimal result within admissible error. However, computing all the gradients and updating all the vectors take too much time.

The main idea of stochastic gradient descent, is that we randomly choose a pair of vectors $\mathbf{w}_i, \mathbf{h}_j$, calculate the gradients with respect to the two vectors, and perform updates (1) and (2).

Matrix Completion

NOMAD in matrix completion

Note that, to update $\mathbf{w}_i$ and $\mathbf{h}_j$, one only need to know $\mathbf{w}_i$, $\mathbf{h}_j$ and A_{ij}. To parallelize the process, we partition the indices $[1..m]$ to p disjoint sets $I_1, I_2, \cdots, I_p$ of approximately equal size, each worker processes one set. The data is partitioned and distributed in the following ways:

Matrix Completion

NOMAD in matrix completion

Note that, to update $\mathbf{w}_i$ and $\mathbf{h}_j$, one only need to know $\mathbf{w}_i$, $\mathbf{h}_j$ and A_{ij}. To parallelize the process, we partition the indices $[1..m]$ to p disjoint sets $I_1, I_2, \cdots, I_p$ of approximately equal size, each worker processes one set. The data is partitioned and distributed in the following ways:

- The q-th worker stores $\overline{\Omega}_j^{(q)} := \{(i,j) \in \overline{\Omega}_j; i \in I_q\}$;
- The q-th worker stores $\mathbf{w}_i, i \in I_q$ and $A_{ij}, i \in I_q$;
- The vectors $\mathbf{h}_j$ are randomly distributed to the workers at the beginning, and moving between the workers during processing.

Matrix Completion

NOMAD in matrix completion

Note that, to update $\mathbf{w}_i$ and $\mathbf{h}_j$, one only need to know $\mathbf{w}_i$, $\mathbf{h}_j$ and A_{ij}. To parallelize the process, we partition the indices $[1..m]$ to p disjoint sets $I_1, I_2, \cdots, I_p$ of approximately equal size, each worker processes one set. The data is partitioned and distributed in the following ways:

Matrix Completion

NOMAD in matrix completion

Note that, to update $\mathbf{w}_i$ and $\mathbf{h}_j$, one only need to know $\mathbf{w}_i$, $\mathbf{h}_j$ and A_{ij}. To parallelize the process, we partition the indices $[1..m]$ to p disjoint sets $I_1, I_2, \cdots, I_p$ of approximately equal size, each worker processes one set. The data is partitioned and distributed in the following ways:

- The q-th worker stores $\overline{\Omega}_j^{(q)} := \{(i,j) \in \overline{\Omega}_j; i \in I_q\}$;
- The q-th worker stores $\mathbf{w}_i, i \in I_q$ and $A_{ij}, i \in I_q$;
- The vectors $\mathbf{h}_j$ are randomly distributed to the workers at the beginning, and moving between the workers during processing.

Matrix Completion

NOMAD in matrix completion

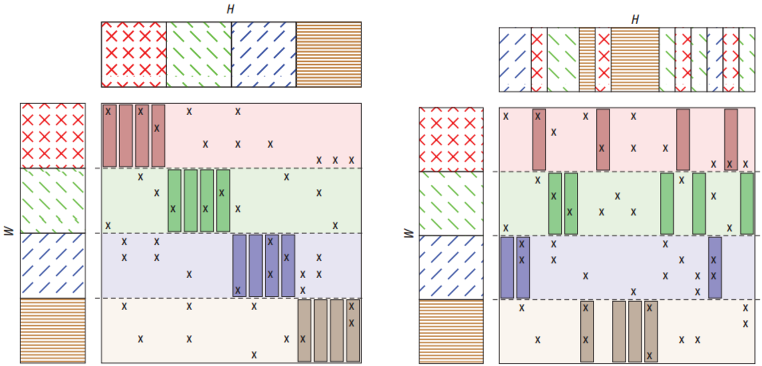

Figure: Ownership of the data

Matrix Completion

NOMAD in matrix completion

Every worker randomly chooses an index $i \in I_q$ and vector $\mathbf{h}_j$ it owns, and update vectors $\mathbf{w}_i$ and $\mathbf{h}_j$. The worker transfers the ownership of vector $\mathbf{h}_j$ to another worker immediately after the update.

Matrix Completion

NOMAD in matrix completion

Every worker randomly chooses an index $i \in I_q$ and vector $\mathbf{h}_j$ it owns, and update vectors $\mathbf{w}_i$ and $\mathbf{h}_j$. The worker transfers the ownership of vector $\mathbf{h}_j$ to another worker immediately after the update.

The advantages of this scheme are

- Decentralized;
- Asynchronous computation and communication;
- Serializability.

Matrix Completion

NOMAD in matrix completion

Every worker randomly chooses an index $i \in I_q$ and vector $\mathbf{h}_j$ it owns, and update vectors $\mathbf{w}_i$ and $\mathbf{h}_j$. The worker transfers the ownership of vector $\mathbf{h}_j$ to another worker immediately after the update.

The advantages of this scheme are

- Decentralized;
- Asynchronous computation and communication;
- Serializability.

Disadvantages?

Matrix Completion

NOMAD in matrix completion

TABLE 1. Dataset details.			
Dataset	**Rows**	**Columns**	**Nonzeros**
Netflix[3]	2,649,429	17,770	99,072,112
Yahoo! Music[10]	1,999,990	624,961	252,800,275
Hugewiki	50,082,603	39,780	2,736,496,604

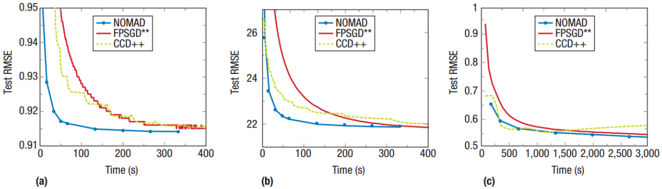

Figure: Performance of NOMAD in matrix completion

Latent Dirichlet Allocation

Some notations

Suppose we are given I documents.

Notation	Meaning
d_i	the ith document
n_i	number of words in the ith document
$w_{i,j}$	the jth word in the ith document
$z_{i,j}$	the latent topic from which $w_{i,j}$ was drawn
$n(z,i,w)$	$\sum_{j=1}^{n_i} I(z_{i,j} = z \wedge w_{i,j} = w)$
$n(z,i,*)$	$\sum_w n(z,i,w)$
$n(z,*,w)$	$\sum_i n(z,i,w)$
$n(z,*,*)$	$\sum_{i,w} n(z,i,w)$

Latent Dirichlet Allocation

Some notations

Notation	Meaning
$\mathbf{n}_t$	the vector $n(z, *, *)$ over z
$\mathbf{n}_w$	the vector $n(z, *, w)$ over z
$\mathbf{n}_d$	the vector $n(z, d, *)$ over z

Latent Dirichlet Allocation

Collapsed Gibbs sampling

The inference task for LDA requires Collapsed Gibbs sampling (CGS).

Latent Dirichlet Allocation

Collapsed Gibbs sampling

The inference task for LDA requires Collapsed Gibbs sampling (CGS).

The update rule for CGS of indices (i, j) can be written as

1. Decrease $n(z_{i,j}, i, *)$, $n(z_{i,j}, *, w_{i,j})$ and $n(z_{i,j}, *, w_{i,j})$ by 1;
2. Resample $z_{i,j}$ according to:

$$\Pr(z_{i,j} | w_{i,j}, \alpha, \beta) \propto \frac{(n(z_{i,j}, i, *) + \alpha)(n(z_{i,j}, *, w_{i,j}) + \beta)}{n(z_{i,j}, *, *) + J \cdot \beta}$$

3. Increase $n(z_{i,j}, i, *)$, $n(z_{i,j}, *, w_{i,j})$ and $n(z_{i,j}, *, w_{i,j})$ by 1.

Latent Dirichlet Allocation

Nomadic approach for LDA

To perform an update, we need to access $z_{i,j}, \mathbf{n}_w, \mathbf{n}_d$ and $\mathbf{n}_t$. If there is no $\mathbf{n}_t$, the access pattern is identical to that of matrix completion problem.

Latent Dirichlet Allocation

Nomadic approach for LDA

To perform an update, we need to access $z_{i,j}, \mathbf{n}_w, \mathbf{n}_d$ and $\mathbf{n}_t$. If there is no $\mathbf{n}_t$, the access pattern is identical to that of matrix completion problem.

Note that, the elements in $\mathbf{n}_t$ are very large, and any small change to $\mathbf{n}_t$ is negligible. This enables us to design a special nomadic scheme for $\mathbf{n}_t$:

- There is only one worker keeps $\mathbf{n}_t$, while each worker has its own local copy $\mathbf{n}_t^{(i)}$;
- Whenever a worker receives $\mathbf{n}_t$, it updates $\mathbf{n}_t$ with the change in its local copy, keeps a snapshot $\bar{\mathbf{n}}_t$, and passes $\mathbf{n}_t$ to the next worker.

Latent Dirichlet Allocation

Nomadic approach for LDA

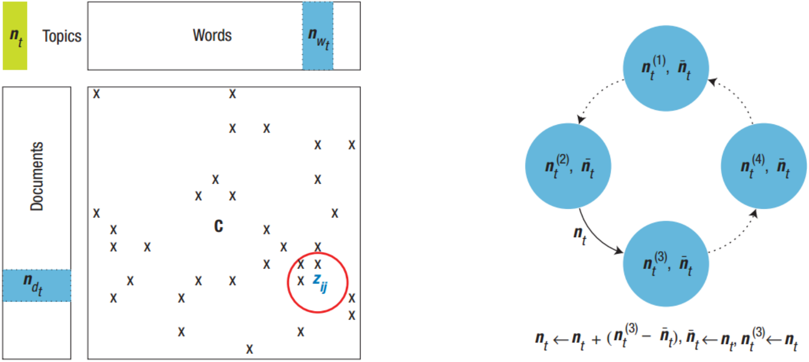

Figure: Data access graph and the nomadic scheme for $\mathbf{n}_t$

Latent Dirichlet Allocation

Nomadic approach for LDA

TABLE 2. Data statistics.

Dataset	No. of documents (*I*)	No. of vocabulary in the corpus (*J*)	No. of word tokens
PubMed	8,200,000	141,043	737,869,083
Amazon	29,907,995	1,682,527	1,499,602,431
University of Maryland, Baltimore County	40,559,164	2,881,476	1,483,145,192

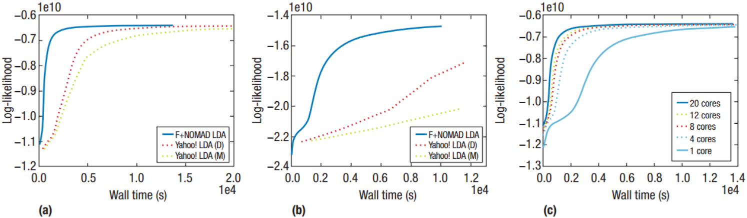

Figure: Performance of NOMAD for LDA

Summary

The main idea of nomadic computing is that, we partition and distribute the data to the workers. Some data is fixed to the workers, while the other is nomadic. Every worker only performs the operations involving the data it owns, and transfer the nomadic data to other workers. The transfer of the nomadic data makes the worker able to do all operations about the fixed data it owns. If some data is involved in the operations, we have to design some special scheme to make the data synchronized.

Q & A