GPU Accelerated Latent Dirichlet Allocation and Differential Privacy

A dissertation proposal presented
by
Shilong Wang
to
The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Electrical Engineering

University of Massachusetts Lowell
Lowell, Massachusetts
April 2020

Thesis advisor: Professor Hengyong Yu Shilong Wang

GPU Accelerated Latent Dirichlet Allocation and Differential Privacy

Abstract

Latent Dirichlet Allocation (LDA) is a statistical approach for topic modeling with a wide range of applications. Attracted by the exceptional computing and memory throughput capabilities, we introduce two new GPU-based LDA methods, sparse-aware based ezLDA and Metropolis-Hasting based mhLDA, to achieve efficient and scalable LDA training on GPUs. For ezLDA, we have following three contributions: First, ezLDA introduces three-branch sampling method which takes advantage of the convergence heterogeneity of various tokens to reduce the redundant sampling task. Second, to enable sparsity-aware format for both D and W on GPUs with fast sampling and updating, we introduce hybrid format for W along with corresponding token partition to T and inverted index designs. Third, we design a hierarchical workload balancing solution to address the extremely skewed workload imbalance problem on GPU and scale ezLDA across multiple GPUs. Taken together, ezLDA achieves superior performance over the state-of-the-art attempts with lower memory consumption. For mhLDA, we have following two contributions: First, instead of three-branch sampling, we preserve token topics in previous iteration to utilize the convergence heterogeneity of various tokens to reduce redundant sampling. Second, documents of different lengths are dealt with differently for balanced workload.

Differential privacy is proposed to keep data private even when maliciously created differential queries occur. Formally, differential privacy requires that the computation results to be indistinguishable when run with or without any specific single record, analogous to the fact that person A has opted out of the dataset. In practice, differential privacy introduces a privacy budget (i.e., noise) to a specific database so that all the queries come to this database will be answered with some noise.We need a budget allocation mechanism which can dispatch privacy budgets to various sub-queries so that the privacy is preserved yet the accuracy of the answer is optimal. We explore a solution that can arrive at optimal budget allocation within an acceptable time envelop. Particularly, the algorithm part of research introduces a novel gradient decent based design to rapidly optimize the budget allocation towards optimal.

Contents

1

ezLDA: Sparse-aware based Efficient and Scalable LDA on GPUs

1.1 Introduction

Topic modeling is a type of statistical approach that reveals the latent (i.e., unobserved) topics for a collection of documents (also referred to as corpus).

LDA[6], which carefully chooses the Dirichlet distribution as the statistical model to formulate topic distributions, is one of the most popular topic modeling approaches that find applications in not only text analysis[9,77], but also computer vision[10], recommendation system[14,32] and network analysis[11] among many others. Thanks to the practical implications, contemporary search engines rely upon LDA to handle billions of news with 10K of topics and 1000K of words[73].

Recently, we also observe interesting interactions between LDA and popular deep learning models. First, Functional and Contextual attention-based Long Short-Term Memory (FC-LSTM)[54] utilizes LDA to preprocess the data and feed corresponding results into LSTM model to improve the accuracy in a recommendation system. LDA can also cooperate with Convolutional Neural Network (CNN) model as a preprocessing method to deal with automobile insurance fraud problems[64]. Second, logistic LDA[76], which is a modified supervised LDA model, can achieve comparable accuracy with Syntax Aware LSTM (SA-LSTM)[49] on document classification with much shorter training time than SA-LSTM. BPLDA[12] can achieve comparable accuracy on classification and regression as deep learning with much shorter training time. Compared with recent deep learning based natural language processing (NLP) tools, e.g., Embeddings from Language Models (ELMo)[44] and Bidirectional Encoder Representations from Transformers (BERT)[17,53], LDA also presents a solid theoretical foundation which is absent for deep learning models.

As the size of NLP problems continues to rise, it becomes imperative for us to scale the training of LDA towards more computing resources, as well as accommodating larger corpus with more topics. Graphics Processing Units (GPUs) exhibits remarkable performance over traditional CPU system and are hence widely applied on compute-intensive problems such as deep learning[59,56,26,3,18] and graph[62]. Towards expediting LDA training, GPUs are a tempting platform for two, if not more, reasons. First, modern GPUs yield extraordinary computing and data delivering capabilities, both of which are crucial for LDA training. Second, GPUs possess a thriving community with steady updates in both hardware and software support, which helps extend the impacts of LDA.

Generally speaking, LDA encompasses three tensors and two tasks. First, the three tensors are: the token list T - an array of <wordId, docId, topicId> triplets, a document-topic matrix (i.e., D) and a word-topic matrix (i.e., W). Second, the two tasks are sampling and updating. During sampling, LDA takes as input each specific token, i.e., t, and relies on D and W to generate a new topic for this token. The intuition of sampling is that the probability of assigning new topic t is positively correlated to the number of tokens for each topic of the document and word this t belongs to. During updating stage, T, D and W are updated to reflect the new topic generated for each token t.

1.1.1 Related Work

As one of the most popular topics in machine learning, LDA[6] has received enormous attentions. This section restricts our discussions to the projects that are closely related to ezLDA, that is, efficiency, scalability and GPU-based LDA.

There mainly exist four directions to make LDA more efficient than the original attempt[6], that is, sparsity-aware, Metropolis-Hasting (MH) and Expectation Maximization (EM) approaches, as well as the hybrid of them. i). Sparsity-aware method utilizes the sparsity of word-topic and document-topic matrix to make the sampling time sublinear to number of topics K (detailed in Section 1.2.2). SparseLDA[70] pioneers this attempt. ii). MH[58] method generates a complex distribution by constructing a Markov Chain (MC) with a simple easy-to-sample distribution and update the topic with some acceptance rates at each step. Since MH requires frequent random memory address to word-topic and doc-topic matrices, thus is not friendly for sparse matrix. iii). LDA*[73] uses sparsity-aware and MH samplers to deal with short and long documents separately. The follow-up variations are[34,72,13,35,66,79,71]. iiii). EM[69] divides the LDA training into E-step and M-step while the former is responsible for sampling and the latter for updating. Comparing with standard LDA sampling, EM can enjoy better parallelism because frequent random memory access to update word-topic and document-topic matrices during sampling can be avoided.

Large-scale training is another important field for LDA considering real-world corpus often contains billions of tokens. LightLDA[72] leads this effort. Particularly, it trains LDA model with 1 million topics and 1 million words on eight machines via data parallelism (corpus partition) and model parallelism (word-topic matrix splitting). While LightLDA allows both D and W to be sparse, it relies upon hash table for fast sampling, which is a hardership for GPU because collision handling in hash table remains elusive on GPUs[5]. This concern is evident by SaberLDA[35] which only stores D in sparse format for fast sampling.

cuLDA[66] further attempts this challenge on multi-GPU but ends up with identical strategy as SaberLDA except scaling the sampling towards multiple GPUs. As we will evaluate in Table 1.1, only allowing D to be sparse will greatly hinder the scalability of LDA.

Last, for GPU-based LDA, which is the interest of this work, we find very few efforts. Yan et al[67] implements collapsed Gibbs sampling[46] and collapsed variational Bayesian[57] on GPU. BIDMach[79] toolkit implements Monte Carlo Expectation Maximization (MCEM)[65] method on GPU without much GPU specific optimizations thus ends up with moderate performance. SaberLDA[35] proposes the PDOW (partition by document, order by word) strategy to reduce random memory access. Warp-based sampling is also adopted, which means using a warp to process a token and a block to process a word, to avoid thread-divergence and uncoalesced memory access. Further, cuLDA[66] scales LDA to multiple GPUs based on collapsed Gibbs sampling with similar optimizations as SaberLDA. In summary, the curse of GPU-based LDA is the limited number of topics because they have to store W in dense format - larger topics will exhaust the limited memory space of GPUs.

1.1.2 Contributions

This paper introduces ezLDA[*], an efficient and scalable LDA project that trains LDA across multiple GPUs. Notably, ezLDA can train LDA on UMBC dataset within 700 seconds while supporting the unprecedented 32,768 topics on merely

[*]ezLDA, pronounced as "easy LDA", implies that this project achieves efficiency and scalability without the involvement of users.

one V100 GPU[Nvidia]. This achievement is not possible without the following contributions:

First, ezLDA introduces one more direction to make LDA efficient, i.e., the three-branch sampling method which takes advantage of the convergence heterogeneity of various tokens to reduce the redundant sampling task. While the convergence heterogeneity is promising, the caveat is that one cannot simply avoid sampling a token because its topic remains unchanged for a number of iterations, as detailed in Figure 1.6. Inspired by our key observation that the majority of the tokens often fall in the top popular topics, we single out these popular topics as the third sampling branch in addition to the traditional two branches (detailed in Section 1.2.2). During sampling, we introduce an algorithm to accurately estimate whether this token will remain in the top popular topics thus be skipped or not. Meanwhile, in order to minimize the overhead of three-branch sampling, we introduce processing by both word and document strategy along with inverted index, and top topics pair-storage. Our evaluation shows three-branch sampling can avoid sampling over 70% of the tokens with negligible overhead after 100 iterations on large datasets, PubMed and UMBC.

Second, to enable sparsity-aware format for both D and W on GPUs with fast sampling and updating upon this format, we introduce hybrid format for W along with corresponding token partition to T and inverted index designs. Particularly, we store W in sparse and dense hybrid format and D in sparse format. During sampling, we will keep a canonical copy of the dense format of W, which accounts for the majority of the tokens but with very few number of words, in

GPU memory to cache the updates. After sampling, we will update both the sparse part of W and D. Since sparse part of W holds very few tokens, the update is trivial. Pair-storing the row index and value of D is also adopted for fast sampling. For rapid update of D, we, again, leverage the inverted index of D to navigate through the token list T for tokens of interest. We also notice that SaberLDA[35] has attempted sparsity aware LDA but ends up with only sparse D which cannot solve memory exhaustion problem caused by dense W when vocabulary and topic size become too large. Consequently, ezLDA, as shown in Table 1.1, doubles the space saving over SaberLDA thus supports models that SaberLDA cannot.

Third, we improve the scalability of ezLDA across GPU threads and GPUs. For single GPU, we introduce hierarchical workload management to ultimately balance the workload which consists of two optimizations. Specifically, we first use atomic operation to dynamically decide which word should be assigned to which GPU warp thanks to light-weighted GPU atomic operation[23]. Further, we propose to split the extremely large words for better workload balancing. To efficiently combine dynamic inter-word scheduling and large word splitting design, we introduce efficient indexing to achieve light-weighted workload management. Towards multi-GPU support, we propose to cache W, partitioned D and partitioned T in GPU memory to further boost the performance of ezLDA.

The novelty of this paper is that we introduce the efficient and scalable techniques to achieve fast LDA training. Particularly, to the best of our knowledge, our three-branch sampling is the first successful design to exploit the con-

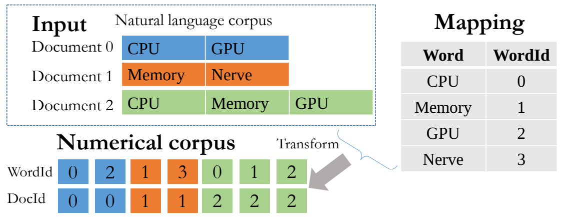


Figure 1.1: The running example used in this paper.

vergence heterogeneity of various tokens for fast LDA sampling. We also shed lights on the possibility of using inverted index to achieve sparsity-aware LDA training where both W and D are sparse. It is also important to note that this paper strives to make sense of the complicated mathematical designs of LDA with an intuitive example which will also benefit the community.

1.1.3 Organization

The rest of the paper is organized as follows: Section 1.2 explains the background. Section 1.3 presents the novel three-branch sampling design. Section 1.4 discusses sparsity-aware LDA and scalable LDA is introduced in Section 1.5. Section 1.6 evaluates this work and we conclude in Section 1.7.

1.2 Background and Challenges

1.2.1 General Purpose GPUs

Without loss of generality, this section uses Volta GPUs as an example to illustrate the essential backgrounds about modern GPUs, mainly from three aspects, that is, processor, memory and programming primitives. For more details about GPUs, we refer the readers to[43].

The streaming multiprocessor (SMX), which consists of several CUDA cores,

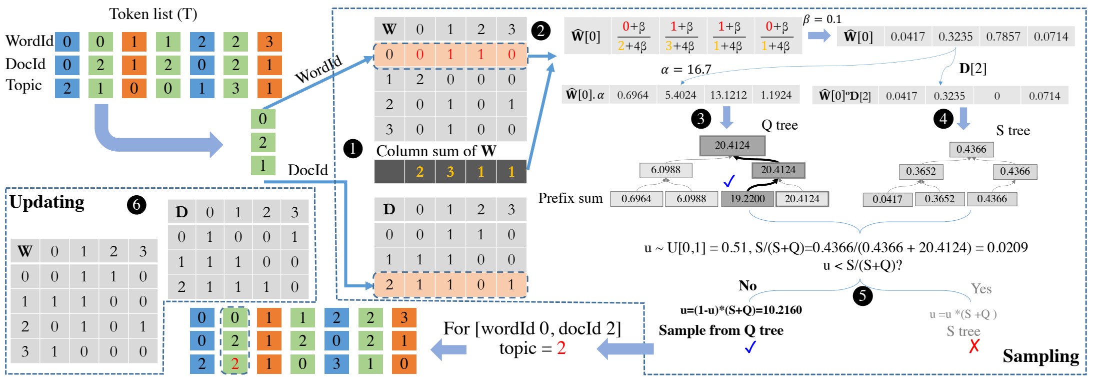


Figure 1.2: Two-branch sampling of one token for the corpus in Figure 1.1. Better viewed in color.

is a basic processing chip for GPUs. For instance, Nvidia Tesla V100 GPUs[Nvidia] contain 80 SMXs, each of which has 64 single-precision CUDA cores and 32 double-precision units and a 96 KB shared memory/L1 cache and 65,536 registers. V100 also features 6MB L2 cache and 16 GB global memory, which is shared by all SMXs. Similar to CPU, the memory access latency increases from register to shared memory, further to L2 cache and global memory.

With massive CUDA cores, GPUs can run a large number of threads. Contemporary GPUs thus manage threads by warps, which is a group of 32 consecutive threads. It is important to mention that a warp of threads is executed in Single Instruction Multiple Thread (SIMD) fashion, which is also one of the most representative features of modern GPUs. In terms of programming primitive, recent GPUs provide several warp-level primitives such as ___shfl() and ___ballot() for fast intra-warp communication.

1.2.2 LDA Algorithm and Theory

Before explaining LDA designs, Figure 1.1 describes how to transform a real world natural language corpus into numerical corpus which can be used by LDA. Particularly, for a natural language corpus which consists of three documents with 2, 2 and 3 tokens, respectively, the preprocessing step will extract the unique words and assign each of them a specific wordId in mapping. This step is necessary because identical word might appear repeatedly, where each occurrence is called a token, e.g., memory appears in both documents 1 and 2. After transformation, we arrive at the numerical corpus.

Algorithm. LDA is a three-layer Bayesian model, that is, each document is viewed as a distribution of topics and each topic is further deemed as a distribution of vocabulary. For a given token, a new topic can be generated based on these two distributions. So, during training, two matrices are involved, i.e., D and W. While detailed theory behind why LDA would work can be found in[6], this paper focuses on the algorithm.

For each token, ESCA[74] - one of the popular LDA version - assigns this token to topic k, i.e., $p(k)$ through the following equation:

$$p(k) \propto \underbrace{(\mathrm{D}[d][k] + \alpha)}_{part1} \cdot \underbrace{\frac{\mathrm{W}[v][k] + \beta}{(\sum_{v=1}^{V} \mathrm{W}[v][k] + V\beta)}}_{part2,\ \widehat{\mathrm{W}}[v][k]}, \tag{1.1}$$

where α and β are two constant hyper parameters. Similarly to[35,66], we adopt

$\alpha = 50/K$ and $\beta = 0.1$ for ezLDA, where K is the total number of topics. $\mathrm{D}[d][k]$ is the number of tokens in document d that belongs to topic k in D. Similarly, $\mathrm{W}[v][k]$ is the number of tokens of word v that belongs to topic k in W.

The intuition of Equation (1.1) is that, for token t that belongs to word v and document d, if more tokens from document d and word v fall in topic k, LDA will be more likely to assign topic k to this token t, that is, $\mathrm{D}[d][k] + \alpha$ and $\mathrm{W}[d][k]+\beta$ will be larger. Further, the total number of tokens in v - $\sum_{v=1}^{V} \mathrm{W}[v][k]+ V\beta$ - is negatively correlated.

Defining part 2 of Equation (1.1) as $\widehat{\mathrm{W}}[v][k]$, which can be regarded as the normalized version of W matrix, we get:

$$p(k) \propto (\mathrm{D}[d][k] + \alpha) \cdot \widehat{\mathrm{W}}[v][k], \tag{1.2}$$

It is important to note that we choose to extend ESCA[74] because ESCA is sparsity-aware, which means the time complexity is sub-linear with respect to the number of topics. ESCA achieves this sparsity-aware goal by decomposing the part 1 of Equation (1.1) into two separate terms. So Equation (1.1) can be rewritten in the following format:

$$p(k) \propto= \underbrace{\mathrm{D}[d][k] \cdot \widehat{\mathrm{W}}[v][k]}_{p_s(k)} + \underbrace{\alpha \cdot \widehat{\mathrm{W}}[v][k]}_{p_q(k)}, \tag{1.3}$$

Equation (1.3) can be further written into vector format:

$$\mathrm{p} \propto (\mathrm{D}[d] + \boldsymbol{\alpha}) \circ \widehat{\mathrm{W}}[v] = \underbrace{\mathrm{D}[d] \circ \widehat{\mathrm{W}}[v]}_{\mathrm{p}_s,\ \text{or S tree}} + \underbrace{\boldsymbol{\alpha} \circ \widehat{\mathrm{W}}[v]}_{\mathrm{p}_q,\ \text{or Q tree}}, \tag{1.4}$$

where $\circ$ is the Hadamard Product (HP) operator. $\widehat{\mathrm{W}}[v]$ is the normalized v-th row of W. D$[d]$ is d-th row of D. $\boldsymbol{\alpha}$ is a vector with all elements to be α.

Finally, ESCA defines S and Q as the sum of p_s and p_q, respectively, sampling process of LDA becomes as follow. Note, we term this sampling method as two-branch because it has S and Q two branches.

- Generating a random number $u \sim U[0,1]$.
- Generating the new topic by $\begin{cases} \mathrm{p}_s, & \text{if } u \leq \frac{S}{S+Q}; \\ \mathrm{p}_q, & \text{otherwise.} \end{cases}$

Example. Figure 1.2 presents one iteration of LDA on the same corpus as shown Figure 1.1 with randomly assigned topics. During initialization (❶), one will generate the W and D matrices from the token list T, where W and D are document-topic and word-topic matrices, respectively. Particularly, the dotted box in W means the document 0 has 0, 1, 1 and 0 tokens for topics 0, 1, 2 and 3, respectively. Similarly, the dotted box in D means that word 2 has 1, 1, 0 and 1 tokens for topics 0, 1, 2 and 3, respectively. Note, the column sum of W is also computed, as shown below W, which means, in total, we have 2, 3, 1 and 1 tokens for topics 0, 1, 2 and 3, respectively.

LDA training encompasses two steps, i.e., sampling and update. Further for sampling, LDA uses either Q or S tree to conduct sampling thus is called two-branch sampling. For the second token of the first word from the token list T - {0, 2, 1}, we follow Equation (1.1) to compute the $\widehat{\mathrm{W}}[0]$ as $\{\frac{0+\beta}{2+4\beta}, \frac{1+\beta}{3+4\beta}, \frac{1+\beta}{1+4\beta}, \frac{0+\beta}{1+4\beta}\}$ (❷). Since $\alpha = 50/3 = 16.7$ and $\beta = 0.1$, we obtain $\widehat{\mathrm{W}}[0] \circ \alpha$ as {0.6964, 5.4024, 13.1212, 1.1924} and $\widehat{\mathrm{W}}[0] \circ \mathrm{D}[2]$ as {0.0417, 0.3235, 0, 0.0714}. Conducting prefix-sum[52] of $\widehat{\mathrm{W}}[0] \circ \alpha$ and $\widehat{\mathrm{W}}[0] \circ \mathrm{D}[2]$, we arrive at the ranges of {0.6964, 6.0988, 19.2200, 20.4124} and {0.0417, 0.3652, 0.4366, 0.4366} for the Q and S tree, respectively.

The rule of tree construction is that the parent node should be the larger one of the two child nodes. Using the first two pairs of Q tree as an example, 6.0988 is the parent node of {0.6964, 6.0988} (❸). Similarly for the rest of Q tree and S tree construction (❹). The ❺ step draws a random number from uniform distribution $U[0, 1]$, u = 0.51 in this case, and compares it against $\frac{S}{S+Q}$ to decide which tree to sample in order to derive a new topic for this token. Since 0.51 is not smaller than $\frac{S}{S+Q}$, we use Q tree to conduct the sampling by adjusting $u = (1 - u) \cdot (S + Q) = 10.2160$ and descending the Q tree to arrive at new topic = 2. Following this way, LDA will update the topics for all the tokens T, subsequently the D and W matrices (❻).

Since T is sorted by wordId, we only need to construct Q tree once for all the tokens of the same word. S tree construction, in contrast, needs to be done more frequently because adjacent tokens often come from different documents, as shown in Figure 1.2.

Evaluation metric. We use log-likelihood per token (LLPT), also known as negative logarithm of perplexity, as the parameter to evaluate the converge of

LDA.

$$LLPT = \frac{1}{N}\sum_{n=1}^{N}(log_2 \sum_{k=1}^{K}(\frac{\mathrm{D}[d][k]+\alpha}{\sum_{k=1}^{K}\mathrm{D}[d][k]+K\alpha}\cdot \frac{\mathrm{W}[v][k]+\beta}{\sum_{v=1}^{V}\mathrm{W}[v][k]+V\beta})), \tag{1.5}$$

where N is the total number of tokens in this corpus. The rule is that LLPT should increase and gradually become stable when computation proceeds iteratively.

1.2.3 LDA Challenges

Challenge #1. LDA sampling presents high time complexity. For each token, one needs to conduct topic number of floating point multiplication and addition operations in order to get a new topic. Assuming we have N tokens and K topics, the time complexity of sampling in one iteration is as high as $\mathcal{O}(N \cdot K)$. As a result, our three-branch sampling (discussed in Section 1.3) will largely reduce the computation complexity of sampling.

Challenge #2. LDA presents high space complexity. As shown in Figure 1.2, LDA allocates three data structures: T, D and W, which consume $\mathcal{O}(N)$, $\mathcal{O}(M \cdot K)$ and $\mathcal{O}(V \cdot K)$ space, respectively, where M, V and K are number of documents, words and topics, respectively. As we will discuss in Section 1.4, ezLDA introduces sparsity-aware compression and computation to save space for both D and W.

1.3 Three-Branch Fast Sampling

This section discusses two important observations, our three-branch sampling and implementation optimizations that lower the overhead of three-branch sampling.

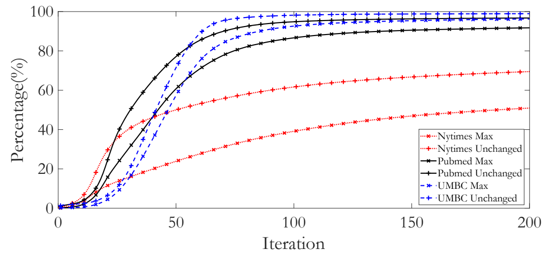


Figure 1.3: Percentage of tokens with unchanged topic and tokens corresponding to the max topic.

1.3.1 Observations

This section makes the following two important observations that inspire our three-branch sampling.

First, different tokens converge at dissimilar speeds. As shown in Figure 1.3, when iteration proceeds, more and more tokens experience unchanged topics. In other words, some tokens converge earlier and some later. For instance, at 50-th iteration, over 70% of the tokens keep their topic unchanged in PubMed dataset.

Second, the majority of the tokens tend to converge to the most popular topic. This observation is self-explanatory - because a topic contains more tokens, and becomes the most popular topic. In fact, Figure 1.3 quantitatively showcases this observation. In particular, more than 60% of the tokens converge to the

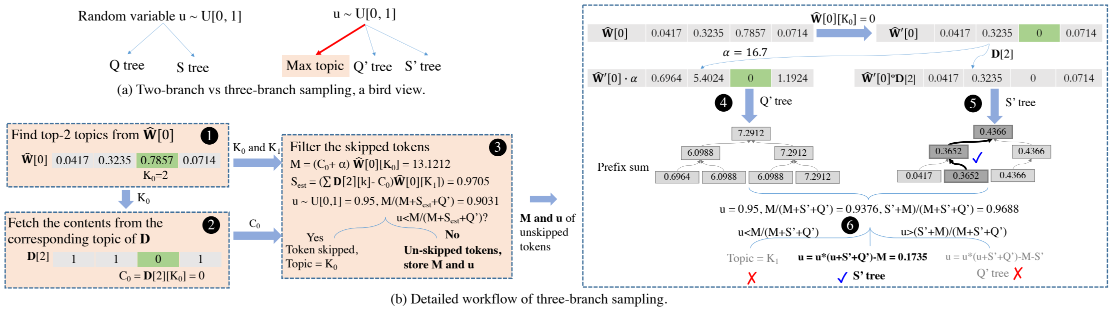


Figure 1.4: ezLDA three-branch sampling: (a) Two-branch vs ezLDA three-branch sampling, a bird view and (b) Detailed workflow of three-branch sampling.

most popular topic in PubMed dataset at 50-th iteration.

The first observation implies that we can reduce the sampling workload for early converged tokens. However, since reducing the sampling task needs extra checking operations, this might incur significant overhead. Fortunately, our second observation further suggests that only focusing on the most popular topic would be adequate, which lowers the overhead.

1.3.2 Three-Branch Sampling

Since traditional two-branch sampling cannot leverage our observations in Section 1.3.1 for workload reduction, we introduce three-branch sampling which singles out the most popular topic as one more branch. Below we discuss the theoretical soundness and implementation details of this design. Note, we cannot simply avoid sampling all the tokens from the most popular topic because, as discussed in Section 1.3.4, very few tokens from the most popular topic might change their topic, though more tokens will converge to the most popular topic.

Theoretical soundness. Our three-branch sampling rewrites Equation (1.3) into the following format:

$$\mathrm{p} \propto \underbrace{\mathrm{D}[d] \circ \widehat{\mathrm{W}}'[v]}_{\mathrm{p}_s} + \underbrace{\alpha \circ \widehat{\mathrm{W}}'[v]}_{\mathrm{p}_q} + \underbrace{(\mathrm{D}[d] + \alpha) \circ \widehat{\mathrm{W}}[v]^m}_{\mathrm{p}_m}, \tag{1.6}$$

where $\widehat{\mathrm{W}}'[v]$ is derived by setting the maximum entry of $\widehat{\mathrm{W}}[v]$ as 0. On the contrary, $\widehat{\mathrm{W}}[v]^m$ is achieved by setting all except the maximum entry of $\widehat{\mathrm{W}}[v]$ as 0.

Consequently, p_m has only one non-zero entry which corresponds to the most popular topic. As shown in the left of Figure 1.4(a), traditional two-branch sampling approach conducts sampling from two branches – either S or Q tree. Particularly, S and Q trees are constructed from p_s and p_q in Equation (1.4), respectively. The proposed three-branch sampling, as shown in the right of Figure 1.4(a), consists of three branches. That is, S' and Q' trees which are constructed from p_s and p_q in Equation (1.6), respectively, and the max topic branch which is the p_m in Equation (1.6).

Three-branch sampling is exemplified by Figure 1.4(b). For each unique word, ezLDA first finds the top-2 topics in $\widehat{\mathrm{W}}[v]$, which are $K_1 = 3$ and $K_2 = 1$ (❶). Here "top-2 topics" means these topics correspond to top-2 largest values, 4.5 and 0.3, in $\widehat{\mathrm{W}}[v]$. Then given the third topic is the most popular one, we will extract the number of tokens from the same index in $\mathrm{D}[d]$, that is, 2 (❷). Consequently, $K_1 = 3, K_2 = 1$ and $C_1 = 2$. Afterwards, as shown in ❸ of Figure 1.4(b), we can calculate $M = 9.45$ and $S_{est} = 1.8$ and generate $u \sim U[0, 1]$. Compare u against $\frac{M}{M+S_{est}+Q'}$ to determine whether this token remains in the most popular topic. If yes, this token will not involve in the following steps and corresponding topic will be updated to be K_1. Otherwise, store M and u for this un-skipped token. Finally, we will execute the remaining steps (❹, ❺ and ❻), which are similar to two-branch sampling except following two differences: First, these steps only need to be done for the remaining un-skipped tokens. As training goes, more and more tokens are skipped and linear time decrease will be introduced. Second, max topic is singled out and considered separately. So $\widehat{\mathrm{W}}[K_1]$ should be set to be zero and final sampling will include an additional M branch, as shown in the bottom right of Figure 1.4(b), even after construction of S' tree, we can still avoid the sampling if $u < \frac{M}{M+S'+Q'}$. Besides, the Q' tree and S' tree (❹ and❺) constructions are the same as two-branch sampling method.

S_{est} computation. In order to skip as many tokens as possible, ezLDA needs to make S_{est} as close to S' as possible. Meanwhile, to ensure theoretical soundness, we must also make sure tokens that go to 'Yes' branch in step ❸ must belong to the left branch in step ❻, which requires S' not greater than S_{est}. We use the following inequality to extract the S_{est} and calculate M. Assuming $\widehat{\mathrm{W}}[v]$ is sorted in descending order:

$$\begin{aligned} \widehat{\mathrm{W}}[v] &= [a_1, a_2, ..., a_n], \\ \mathrm{D}[d] &= [b_1, b_2, ..., b_n]. \end{aligned} \tag{1.7}$$

This means $a_i > a_j$ if $i > j$. Thus, maximum topic branch is:

$$M = a_1 \cdot (b_1 + \alpha). \tag{1.8}$$

Given

$$\begin{aligned} S' &= \widehat{\mathrm{W}}[v] \cdot \mathrm{D}[d] - M + a_1 \cdot \alpha \\ &= (a_2 \cdot b_2 + a_3 \cdot b_3 + \cdots + a_n \cdot b_n) \\ &< \sum_{2 \le i \le g} a_i \cdot b_i + a_{g+1} \cdot \sum_{g < i \le n} b_i, \end{aligned} \tag{1.9}$$

we hence propose:

$$S_{est} = \sum_{2 \le i \le g} a_i \cdot b_i + a_{g+1} \sum_{g < i \le n} b_i. \tag{1.10}$$

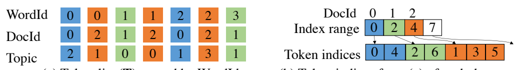
where $g \geq 1$ controls the accuracy and cost of the estimation. That is, the larger the value of g, the higher the cost, as well as the accuracy between S' and S_{est}.

Parameter tuning. First, the choice of g is a trade-off between benefit and overhead. ezLDA uses $g = 2$ because we can achieve significant better performance than g=1 with similar overhead after our optimization in Section 1.3.3.

1.3.3 Optimizations for Three-Branch Sampling

While three-branch sampling can avoid expensive S' tree constructions and sampling for all the skipped tokens, it also introduces three more steps, i.e., ❶, ❷ and ❸ as shown in Figure 1.4(b).

Across all the steps, the cost for steps ❶ and ❸ is negligible. For step ❸, the workload is simple. For step ❶, the reason lies in that the token list (i.e., T) is sorted by wordId, as shown in Figure 1.5(a), three-branch sampling only needs to find the top topics, that is, K_1 and K_2 pair in Figure 1.4(b) - once for all the tokens falling to the same word v. But also because T is sorted by wordId, step ❷ would take significant amount of time if we want to find the corresponding C_1 and C_2 pair across all documents for each v right after we find K_1 and K_2 pair.



(a) Token list (**T**), sorted by WordId (b) Token indices from (a) of each document

Figure 1.5: Inverted index for document.

In order to combat this overhead, ezLDA designs processing by word and document for K_1 and K_2 pair, and C_1 and C_2 pair, respectively. While processing by word is straightforward because T is sorted by word, processing by document turns out to be challenging. In this context, we introduce an inverted index for each document which stores the indices of tokens belonging to each document. This inverted index idea adopts Compressed Sparse Row (CSR) format[47,15] to store the indices of the tokens for each document. As shown in Figure 1.5(b), indices {0, 4}, {2, 6} and {1, 3, 5} are from documents 0, 1 and 2, respectively.

Scanning through the inverted index, we can fetch the corresponding row of D, as well as all the tokens of the same document easily. Note, in this processing by document design for C_1 and C_2 pair, we need to first write all the K_1 and K_2 pairs for all tokens into global memory and load them back for computation. However, this cost is way lower than we conduct processing by word for both K_1 and K_2 pair and C_1 and C_2 pair. Particularly, processing by word for C_1 and C_2 pair needs to repeatedly scan through the token list and search for C_1 and C_2 for each token because tokens of the same document are not stored together. It is also worthy to note that we combine K_1 and K_2, C_1 and C_2 pairs into a single value K_{12} and C_{12}, where the higher half bits of them store K_1 and C_1, and the lower half bits store K_2 and C_2 respectively, in order to further reduce the overhead.

1.3.4 Discussion

This section shares a failed trial. Inspired by the traditional iterative graph computing algorithms, such as, delta-step Pagerank[78] and Single Source Shortest Path[40,55], we falsely assume the tokens that already converged will no longer change their topics. Therefore, our naive design introduces a tracker array to indicate whether the topic of a token remains unchanged for several iterations. If so, this naive dropping method will not sample this token in the following iterations.

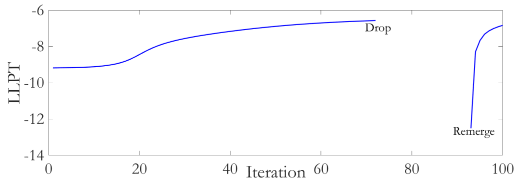


Figure 1.6: Perplexity of naive dropping strategy on PubMed.

However, the naive dropping strategy fails to work mainly because it betrays the nature of LDA. Particularly, the core of LDA, i.e., Bayes model, is that the sampling process of LDA is a non-deterministic process. That is, even if the topic of a token remains unchanged for several iterations, which means the probability of assigning this specific token to the same topic is very close to 1, this token still has a chance to change topics because the random number generated from $U[0,1]$ might fall in other topics whose probabilities are low.

Figure 1.6 shows the failure of the naive dropping strategy. In this test, the dropping starts at iteration 72. At iteration 90, all dropped tokens are re-included in the training to check whether this strategy works. Clearly, the results are not good. At the point of re-including, perplexity becomes even smaller than the value before dropping and severely deviates from the correct convergence point.

1.4 Sparsity-Aware Optimization

Reducing the sampling time is important for LDA, so does the space consumption. This section introduces the sparsity-aware storage format for both D and W, as well as our new mechanisms to facilitate rapid sampling and updating dwelling on these sparse formats.

1.4.1 Observations

The space problem faced by ezLDA appears for two types of data, that is, corpus data and algorithmic data. Corpus data is concerned with the gigantic corpus size while algorithmic data is related to both corpus size and number of topics. Below, we discuss the details surrounding these two challenges.

The space consumption incurred by the large corpus can be tackled by simply partitioning the corpus into multiple chunks. This way, each GPU will need much less memory for corpus. Doing so, however, comes with one obvious drawback - one needs to repeatedly load each chunk in and out the GPU, which could introduce overhead. ezLDA uses asynchronous kernel launching and memory transfer to hide this cost, similar to existing work[25,28].

In fact, curbing the space consumption of algorithmic data (e.g. D and W) is

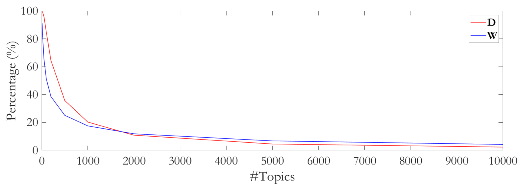


Figure 1.7: Density of W and D on NYTimes dataset.

even more imperative. Below, we unveil the reason from column and row perspective of a matrix. First, with the climbing of the corpus size, the diversity of the tokens will also increase, indicating the need of a larger number of topics (i.e., number of columns in D and W). Second, for a corpus with abundant documents or unique words, the number of rows in both D and W will also soar.

The good news is that both D and W are often very sparse because very few, if any, of the words or documents will occupy all the topics. As shown in Figure 1.7, the density of D and W decreases rapidly along with the increase of number of topics for the NYTimes dataset.

1.4.2 Sparsity-Aware Representation

We introduce compressed CSR format to store the sparse W and D matrices. While the traditional CSR contains three major components: row offset, column indices and values, we further compress column indices along with values in order to save space and improve performance. Knowing that the maximum number of topics are seldom larger than 65,536, 16 bits are enough for storing a single column index or value. Inspired by the pair-storage in Section 1.3.3, we compress the column indices and values of CSR into and a single integer, where the higher and lower 16 bits are for column index and corresponding value, respectively.

Storing the entire W in sparse format might not always save space. Particularly, despite sparse format will save memory space for sparse rows in W, words with large number of topics (i.e. dense rows) will, unfortunately, suffer from extra space consumption because CSR requires to store the column indices. In contrast, dense format only needs to store the values since the position of the value can automatically indicate its column location.

We thus advocate to store W in hybrid format. That is, the rows with large volume of nonzero columns (i.e., topics) will be stored in dense format while the remaining rows in sparse format. ezLDA comes up with a light-weighted heuristic to estimates the upper bound of W in order to decide whether we store a corresponding row in sparse or dense format. That is, the maximum number of topics one word can possess will not go beyond the number of tokens this word has in the entire corpus. With this rule, one can assign the words with tokens that is larger than the assumed number of topics (i.e. k) as dense row and the remaining rows to be the same as the number of tokens.

To enjoy the space saving from the hybrid format, we propose to group dense words together in token list T. Toward that end, the word identities (IDs) are relabeled based upon their token counts. Basically, words with larger number of tokens hold smaller IDs. Further, words with token count more than the topic number are stored in dense format in W. Subsequently, in each chunk from the token list, we remap the word IDs from the token list into T_{dense} and T_{sparse}, respectively, which represent the dense and sparse parts of T, respectively.

In summary, this hybrid approach comes with the following two advantages. First, comparing to the dense or sparse alone approach, the proposed method will yield the most space saving. Second, storing dense rows into dense format explicitly will reduce the overhead of $\widehat{\mathrm{W}}[v] \circ \mathrm{D}[d]$.

1.4.3 Sparsity-Aware Computation

Once storing W and D in sparse format is resolved, conducting updating and sampling atop the sparse W and D become a ground challenge for two reasons. First, during sampling, we need to do element-wise product of $\widehat{\mathrm{W}}[v] \circ \mathrm{D}[d]$. Given the elements from the same storage index of sparse $\widehat{\mathrm{W}}[v]$ and D[d] are most likely not from the same column, we will need to match their columns. Second, to update an element in sparse matrix, we must first find the correct row and column to write the update. So unlike dense matrix, it is impossible to update W once after a new topic is known in sampling kernel. The update of W can only be done by reconstructing from T after sampling, which will consume much more time.

A naive design could easily combine sampling with updating via keeping a canonical copy of W. During sampling, this design will compute the S' and Q', thus the ratio $t_1 = \frac{S'}{S'+Q'+M}$ and $t_2 = \frac{Q'}{S'+Q'+M}$. Based upon the generated random number u, this design could determine to sample from either S' or Q' tree. After arriving at the updated topic for a token, we can immediately update the canonical copy.

However, this naive design also faces two challenges. First, keeping extra

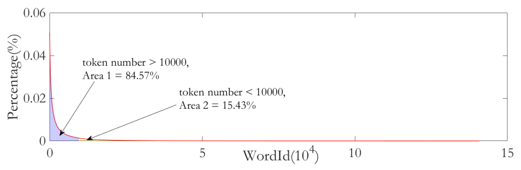


Figure 1.8: Token distribution of PubMed dataset.

canonical copies for W will consume more space. Second, it is hard to predict, for a random token, where to update the canonical copies of W providing they are in the sparse format. Third, given LDA is memory intensive, reading W twice (one for sampling, the other for updating) will hamper the performance.

We only keep a canonical copy for W_{dense} and reconstruct W_{sparse} as well as the entire D from T after sampling. Below, we discuss the details. ezLDA keeps a canonical copy of W_{dense} for update because the words in the dense rows often contain exceeding number of tokens which span across multiple chunks. In that context, we would need to transfer a large number of chunks if we choose to reconstruct W_{dense} from T. In contrast, with a canonical copy of W_{dense}, the update to W_{dense} can be done quickly because W_{dense} is in dense format, as well as in memory during sampling.

For W_{sparse}, we only need to read in the T_{sparse} part of the token list. Since more than 80% tokens contribute to W_{dense}, T_{sparse} will be relatively small. Thus reconstructing W_{sparse} will be very fast.

Since a majority of the tokens belong to W_{dense} which is updated during sampling, the update of the entire W usually consumes very short time.

The update of textbfD is aided by the inverted index which is discussed in Figure 1.5(b). In order to discuss the update to D, we need to understand how ezLDA partitions and preprocesses the corpus. Particularly, each document is solely assigned to one chunk, and all the tokens in each chunk are sorted by wordId. Here, assuming this chunk contains three documents of a corpus. This way, we can reconstruct three rows of D with this chunk. Inside of each chunk, the tokens are sorted by wordId for ease of update of W_{sparse}. Towards updating D, we resort to the inverted index in Figure 1.5(b). Particularly, one can scan through the CSR to decide which tokens are needed to update rows 0, 1 and 2 of D.

With the updating process being taken care of, sampling becomes the immediate bottleneck. To facilitate a fast S' tree construction, which involves the HP between two CSR rows of W and D, we reconstruct the entire row of W into dense format in shared memory. Afterwards, HP is done by scanning through the specific row of D and use the column index to access that of the dense W. Note, this shared memory will be repeatedly used for all rows of W.

1.5 Scalable ezLDA on GPUs

This section discusses novel techniques we exploit to better scale ezLDA across GPU threads, as well as GPUs.

1.5.1 Intra-GPU Workload Balancing

For a corpus, the number of tokens per word often follows power law, that is, a few high frequent words occupy majority of tokens, as shown in Figure 1.8. The workloads associated with various words are hence largely unbalanced. However, the contemporary LDA projects[35] typically assign a block to a word, regardless

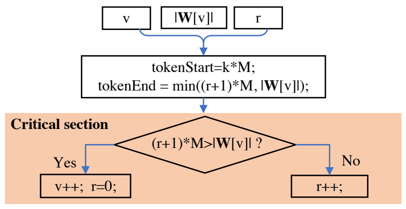


Figure 1.9: Hierarchical workload balancing. Note, v, $|W[v]|$, r and M are the word index, number of tokens for word v, region index, and maximum number of token processed by each index increment, respectively.

of the associated workload, leading to severe workload imbalance issue in LDA training. This section thus introduces two methods to overcome the workload imbalance problem, that is, dynamic workload balancing for small words and workload splitting for large words.

Small word. Given various words come with different number of tokens, we adopt the dynamic workload balance strategy from a recent work[23] to address the inter-word workload imbalance issue. Note, instead of each block processing the words in a pre-determined manner, this approach will use atomicAdd() to, on-the-fly, determine which word will be processed by the available thread block.

Large word. While applying the dynamic workload balance strategy can largely address the inter-word workload imbalance problem faced by small words, it will not work for large words which govern too many tokens. In this context, the block that processes the extremely large words will become the straggler. For instance, assuming one corpus has 128 tokens and the most frequent word holds 50 tokens, we use 8 blocks for training. Then, each block should process 16 tokens in workload balanced setting. However, with the dynamic workload balance strategy, the block that processes the word with 50 tokens will be responsible for this entire word alone, leading to workload imbalance.

In order to solve this problem, we introduce large word dissection, i.e, very high frequency words are partitioned and processed by multiple thread blocks. Particularly, we can quickly derive the maximum number of tokens a block can process through dividing the total amount of workloads by the number of thread blocks. If the token number of a specific word is larger than this maximum value, we will partition this word into several parts and assign them to multiple blocks. In this work, we use 10,000 as the threshold for ezLDA.

It is important to note that applying dynamic small word workload balancing and large word dissection together will pose challenges for word assignment. For instance, we need to decide which word and what portion of that word are the next workload. ezLDA introduces a two-level index strategy to deal with this challenge.

Figure 1.9 shows the design of our two-level index strategy. Word index v determines the word to be processed and region index r determines which region of tokens in that word should be processed. Apparently, the increment of v and r are correlated and must be executed atomically. Considering an atomic function can only be used for a single operation, we propose to use critical section to fulfill that goal. To remedy the absent of critical section support on GPU, ezLDA relies on atomic operations to build a critical section[cri].

1.5.2 Multi-GPU ezLDA

As the size of corpus and number of topics continue to grow, the training time of LDA also prolongs, which leads to our support of multi-GPU ezLDA. When extending to multiple GPUs, ezLDA is concerned with two essential data structures, that is, data (i.e., T) and algorithmic data (i.e., D and W), and the correlated workload partition, and communication.

The good news is that T and D are well partitioned in the single GPU-based design, as discussed in Section 1.4. Particularly, each chunk is responsible for similar number of documents. And surprisingly, each chunk actually contains a similar number of tokens. Using UMBC dataset on four GPUs as an example, the maximum and minimum workload chunks only have a difference of 5% in terms of the number of tokens. This partition of T also leads to evenly partitioned D across GPUs.

For word topic matrix, i.e., W, unlike the single GPU version, we keep an in-memory canonical copy for both W_{dense} and W_{sparse} given we have adequate space for all the data structures. After all chunks are processed, we can update both W_{dense} and W_{sparse} by summing up the canonical copies across all GPUs and broadcasting the result back to all of them.

1.6 Experiments

We implement ezLDA with ∼4,000 lines of C++/CUDA code and compile the source code with Nvidia CUDA 9.2 toolkit and -O3 optimization compilation flag. We use two platforms to evaluate ezLDA. For comparison with state-of-the-art SaberLDA, we use an Nvidia GTX 1080 GPU - identical platform used in SaberLDA - on an Alienware with 24 GB memory and Intel(R) Core(TM) i7-8700 (3.20Hz) CPU. For ezLDA internal study, we use a customized server which installs a dual-socket Xeon processor with 24 cores, and four Nvidia V100 GPUs.

Dataset. We evaluate ezLDA with two popular datasets that are also studied by cuLDA[66] and SaberLDA[35]:

- PubMed[19]: 8,200,000 documents, 141,043 unique words and 738M tokens.
- NYTimes[19]: 299,752 documents, 101,636 unique words and 100M tokens.

To better study the scalability and real-world impacts, we further prepare the following dataset by text splitting, stop words removing and non-frequent words stemming:

- UMBC: 40,000,000 documents, 200,000 unique words and 1.5 billion tokens. This dataset is obtained from UMBC webbase corpus[38].

1.6.1 ezLDA vs. State-of-the-art

Table 1.1 and Figure 1.10 compare ezLDA against the state-of-the-art SaberLDA for both space consumption and turnaround time performance, respectively.

Space. As shown in Table 1.1, ezLDA consumes 33% more space but saves 47% and 78% space compared with SaberLDA for #topic = 1,000, 10,000 and 32768 respectively. Note, ezLDA would require more space than SaberLDA for

Method	#Topics	W	D	T	Total
SaberLDA	1,000	1.08 GB	1.45 GB	2.16 GB	4.69 GB
	10,000	10.8 GB	1.45 GB	2.16 GB	14.41 GB
	32,768	35.4 GB	1.45 GB	2.16 GB	39.01 GB
ezLDA	1,000	0.31 GB	0.98 GB	4.97 GB	6.26 GB
	10,000	1.63 GB	0.98 GB	4.97 GB	7.58 GB
	32,768	2.5 GB	0.98 GB	4.97 GB	8.44 GB

Table 1.1: ezLDA vs SaberLDA memory consumption on PubMed dataset. The corpus is partitioned into 8 chunks and 2 asynchronous streams are applied to hide data transfer time.

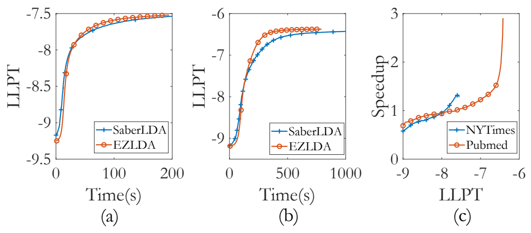


Figure 1.10: (a) The convergence of ezLDA vs SaberLDA with 1,000 topics on NYTimes Dataset. (b) The convergence of ezLDA vs SaberLDA with 1,000 topics on Pubmed Dataset. (c) Speedup for EZLDA vs SaberLDA.

T, which can be solved by partitioning corpus into smaller chunks, but saves significantly much more memory on W thanks to sparsity aware representation. Consequently, ezLDA can train LDA with up to 32,768 topics which is beyond the capacity of SaberLDA.

Turnaround time performance. Figure 1.10 further shows that ezLDA climbs to higher perplexity with less training time. At initial iterations, ezLDA falls behind because the three-branch sampling introduces slightly more overhead than benefits. ezLDA achieves an overall 1.3× and 2.9× speedup on NYTimes and PubMed respectively. The speedup on PubMed is significantly higher than NYTimes because more tokens are skipped.

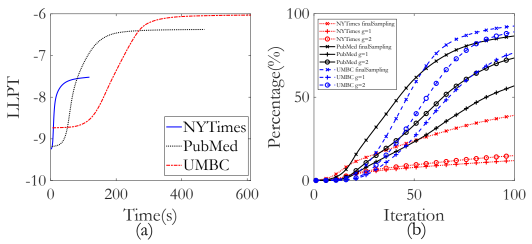


Figure 1.11: (a) ezLDA for large number of topics, i.e., #topics = 32,768. (b) The percentage of tokens skipped by three-branch sampling for #topics = 1,000, where g is the parameter from Equation 1.10.

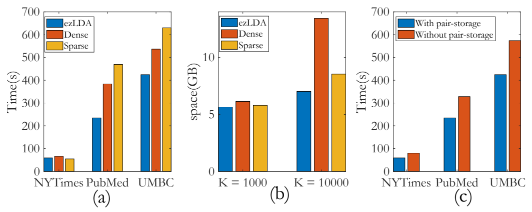


Figure 1.12: (a) Computational time compared with Dense-only and Sparse-only version for topic = 1,000. (b) Space consumption compared with Dense-only and Sparse-only version for UMBC dataset. (c) The performance impacts of pairStorage.

1.6.2 ezLDA Performance Study

Large number of topics. As shown in Figure 1.11(a), ezLDA can handle all the datasets with 32,768 topics on a single GPU. This is an important capability for real-world scale corpus[73]. Particularly, ezLDA reaches convergence around 180, 400 and 700 seconds for NYTimes, PubMed and UMBC dataset, respectively.

Three-branch sampling. Figure 1.11(b) profiles the impact of three-branch sampling. In general, we find this method is more effective when dealing with larger dataset. Particularly, NYTimes enjoys skipping 39% of the tokens during the final sampling and nearly 15% tokens skip the S construction at iteration 100. For PubMed, 87% tokens skip the final sampling and nearly 74% tokens skip the S construction at iteration 100. For UMBC, 93% tokens skip final sampling and nearly 89% tokens skip the S' construction at iteration 100. We also study different g in Equation 1.10. As expected, more tokens are skipping S' construction for larger g, because larger g makes S_{est} closer to S'.

Profiling ezLDA techniques. Figure 1.12 profiles various optimizations introduced by ezLDA. Particularly, dense and sparse hybrid representation as shown in Figure 1.12(a) and 1.12(b) both improve performance and save space. On average, hybrid format is 1.34× and 1.47× faster than dense and sparse only formats. Compared with ezLDA, sparse format needs to update W after all chunks are processed, which means all chunks need to be transferred back to GPU a second time to finish the update. For dense format, much time will be wasted on updating rows of W corresponding to small words. Further, the hybrid format consumes 17.8% and 47.8% less space than sparse format and dense format for K = 10,000, respectively.

Pair K1/K2, C1/C2 and D storage, as shown in Figure 1.12(c), yields 1.35×, 1.40× and 1.35× speedup on NYTimes, PubMed and UMBC datasets, respectively. The great speedup is achieved because LDA training is memory-bound[61] and pair-storage significantly reduces the global memory traffic in three-branch sampling.

1.6.3 Scalable ezLDA

Figure 1.13 studies the scalability optimizations in ezLDA.

Hierarchical workload balancing. Choose three-branch sampling without workload balance as baseline. As shown in Figure 1.12(a), on average, our hierarchical workload balancing technique yields 1.08×, 1.77× and 1.18× speedup on

NYTimes, PubMed and UMBC, respectively. The speedup on PubMed dataset is more obvious because performance is severely lagged without large word dissection. This result reflects the need of hierarchical workload balancing across GPU threads.

Multi-GPU scalability. Figure 1.12(b) that ezLDA can scale to four V100 GPUs with 3.11×, 3.18× and 3.16× speedup on NYTimes, PubMed and UMBC dataset, respectively. Clearly, this relatively close to linear scalability showcases a balanced workload distribution and light-weighted communication in ezLDA framework.

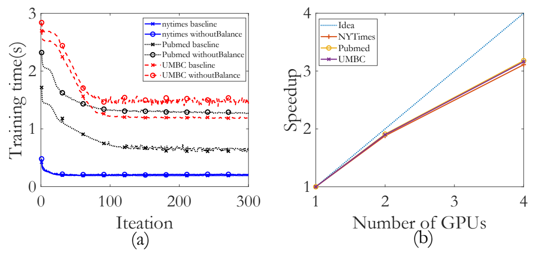


Figure 1.13: The performance impacts of (a) hierarchical workload balancing and (b) scalable ezLDA.

1.7 Conclusion

In this paper, we present ezLDA that achieves superior performance over the state-of-the-art attempts with lower memory consumption. Particularly, ezLDA introduces a novel three-branch sampling method which takes advantage of the convergence heterogeneity of various tokens to reduce the redundant sampling task. Further, to enable sparsity-aware format for both D and W on GPUs with fast sampling and updating, we introduce hybrid format for W along with corresponding token partition to T and inverted index designs. Last but not the least, we design strategies to balance workload across GPU threads and scale ezLDA across multiple GPUs.

2

mhLDA: Metropolis-Hasting based Efficient and Scalable LDA on GPUs

2.1 Introduction

As discussed in Chapter 1, both sparsity-aware and Metropolis-Hasting (MH) can make LDA more efficient. However, the optimization mechanisms behind these two methods are fundamentally different. Sparsity-aware method utilizes the sparsity of word-topic and document-topic matrix to make the sampling time sublinear to number of topics K. MH[58] method generates a complex distribution by constructing a Markov Chain (MC) with a simple easy-to-sample distribution and update the topic with some acceptance rates at each step. Compared with sparsity-aware based LDA method, MH based LDA method have easier sampling but more frequent random memory address to word-topic and doc-topic matrices. In Chapter 1, great speedup is witnessed by applying three-branch sampling method, which utilizes convergence heterogeneity of various tokens to reduce the redundant sampling task. Inspired by that, we propose mhLDA, a more efficient Metropolis-Hasting (MH) based LDA method compared with ezLDA on GPU. However, instead of singling out the most popular topic, we simply cache the topic of each token in previous iteration and skip the tokens whose topics keep unchanged. As a result, more tokens can be skipped compared with ezLDA and greater speedup can be achieved. Besides, we also optimize the memory access to word-topic and doc-topic matrices and deal with long-length and short-length documents separately for further speedup.

2.2 mhLDA

2.2.1 Observation

From Figure 1.3 in Chapter 1, the topic of a token will become unchanged and most of them converge to the most popular topic during the training. The percentage of topic-unchanged tokens are more than that converges to the most popular topic. For sparsity-aware based LDA method such as ezLDA, due to the complexity of sampling function (Equation 1.1), ezLDA can only skip tokens belonging to the most popular topic. So ezLDA shows no significant speedup on NYTimes dataset, tokens of which show no significant convergence to the most popular topic. However, our mhLDA can perfectly utilize the topic-unchanged property of converged tokens and skip more tokens during training compared with ezLDA.

2.2.2 Traditional MH based LDA

For each token, MH based LDA selects a new topic k for each token through following equation:

$$\begin{aligned} p^{word}(k) &\propto (\mathrm{W}[w][k] + \beta) \\ p^{doc}(k) &\propto (\mathrm{D}[d][k] + \alpha), \end{aligned} \tag{2.1}$$

and accept the new topic with following acceptance rate:

$$\begin{aligned} r^{word}(k) &= min(1, \frac{\mathrm{D}[d][k] + \alpha}{\mathrm{D}[d][k'] + \alpha} \frac{\mathrm{W}[k'] + V\beta}{\mathrm{W}[k] + V\beta}) \\ r^{doc}(k) &= min(1, \frac{\mathrm{W}[w][k] + \beta}{\mathrm{W}[w][k'] + \beta} \frac{\mathrm{W}[k'] + V\beta}{\mathrm{W}[k] + V\beta}). \end{aligned} \tag{2.2}$$

Here, k' is the topic in current state. The topic of a token is alternately sampled from word-topic distribution and document-topic distribution.

Example. Figure 2.1 presents one iteration of MH based LDA on the same corpus as shown Figure 1.1 with randomly assigned topics. During initialization

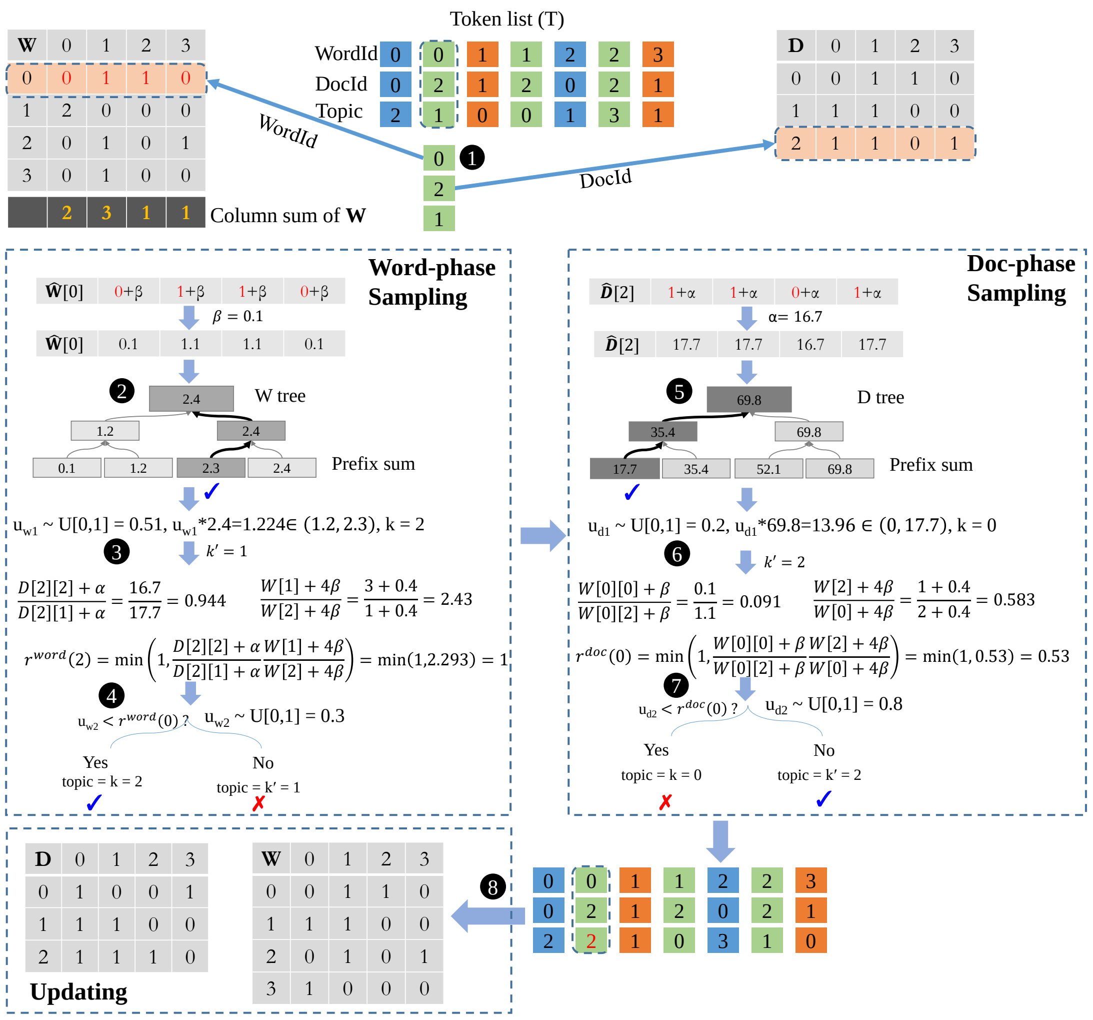

Token list (T)
WordId
DocId
Topic
WordId
DocId
Column sum of W
Word-phase Sampling
β = 0.1
W tree
Prefix sum
u_w1 ~ U[0,1] = 0.51, u_w1*2.4=1.224∈ (1.2, 2.3), k = 2
k′ = 1
(D[2][2] + α)/(D[2][1] + α) = 16.7/17.7 = 0.944
(W[1] + 4β)/(W[2] + 4β) = (3 + 0.4)/(1 + 0.4) = 2.43
r^word(2) = min(1, (D[2][2] + α)/(D[2][1] + α) · (W[1] + 4β)/(W[2] + 4β)) = min(1,2.293) = 1
u_w2 < r^word(0) ?
u_w2 ~ U[0,1] = 0.3
Yes
topic = k = 2
No
topic = k′ = 1
Doc-phase Sampling
α= 16.7
D tree
Prefix sum
u_d1 ~ U[0,1] = 0.2, u_d1*69.8=13.96 ∈ (0, 17.7), k = 0
k′ = 2
(W[0][0] + β)/(W[0][2] + β) = 0.1/1.1 = 0.091
(W[2] + 4β)/(W[0] + 4β) = (1 + 0.4)/(2 + 0.4) = 0.583
r^doc(0) = min(1, (W[0][0] + β)/(W[0][2] + β) · (W[2] + 4β)/(W[0] + 4β)) = min(1, 0.53) = 0.53
u_d2 < r^doc(0) ?
u_d2 ~ U[0,1] = 0.8
Yes
topic = k = 0
No
topic = k′ = 2
Updating

Figure 2.1: Traditional MH based LDA sampling of one token for the corpus in Figure 1.1. Better viewed in color.

(❶), one will generate the W and D matrices from the token list T, where W and D are document-topic and word-topic matrices, respectively. Particularly, the dotted box in W means the document 0 has 0, 1, 1 and 0 tokens for topics 0, 1, 2 and 3, respectively. Similarly, the dotted box in D means that word 2 has 1, 1, 0 and 1 tokens for topics 0, 1, 2 and 3, respectively. Note, the column sum of W is also computed, as shown below W, which means, in total, we have 2, 3, 1 and 1 tokens for topics 0, 1, 2 and 3, respectively.

MH based LDA training encompasses three steps, i.e., word-phase sampling, doc-phase sampling and update.

For word-phase sampling, a W tree is conducted for sampling. For the second token of the first word from the token list T - {0, 2, 1}, we follow the first equation in Equation 2.1 to compute the $\widehat{\mathrm{W}}[0]$ as $\{0+\beta, 1+\beta, 1+\beta, 0+\beta\}$. Conducting prefix-sum[52] of $\widehat{\mathrm{W}}[0]$, we arrive at the ranges of {0.1, 1.2, 2.3, 2.4} for W tree. For this token, a random number u_{w1} is drawn from uniform distribution $U[0,1]$ and a temporary new topic k = 2 will be derived by sampling from W tree (❷). Knowing that current topic is $k' = 1$, then we will calculate the acceptance rate of this temporary topic k by the first equation in Equation 2.2 (❸). We arrive at the acceptance rate of the new topic $r^{word}(2) = 1$. The last step (❹) will be determining whether accepting the new topic or not by random generating a number u_{w2} from uniform distribution $U[0,1]$ and comparing it with acceptance rate $r^{word}(2)$. Obviously, $u_{w2} < r^{word}(2)$, we accept this new topic k = 2.

For doc-phase sampling, it's much similar to word-phase sampling. a D tree is conducted for sampling. For the second token of the first word from the token list T - {0, 2, 1}, we follow the second equation in Equation 2.1 to compute the $\widehat{\mathrm{D}}[2]$ as $\{1+\alpha,\ 1+\alpha,\ 0+\alpha,\ 1+\alpha\}$. Conducting prefix-sum[52] of $\widehat{\mathrm{D}}[2]$, we arrive at the ranges of {17.7, 35.4, 52.1, 69.8} for D tree. For this token, a random number u_{d1} is drawn from uniform distribution $U[0,1]$ and a temporary new topic k = 0 will be derived by sampling from Q tree (❺). Knowing that current topic is $k' = 2$, then we will calculate the acceptance rate of this temporary topic k by the second equation in Equation 2.2 (❻). We arrive at the acceptance rate of the new topic $r^{doc}(0) = 0.53$. The last step (❼) will be determining whether accepting the new topic or not by random generating a number u_{d2} from uniform distribution $U[0,1]$ and comparing it with acceptance rate $r^{doc}(0)$. Obviously, $u_{d2} > r^{doc}(0)$, we reject this new topic k and topic will remain to be $k' = 2$.

For update stage, it's very similar to sparsity-aware based LDA method. Just reconstruct word-topic matrix and doc-topic matrix from the updated token list (❽).

2.2.3 mhLDA

Figure 2.2 presents one iteration of our proposed mhLDA on the same corpus as shown Figure 1.1 with randomly assigned topics. During initialization (❶), one will generate the W and D matrices from the token list T, where W and D are document-topic and word-topic matrices, respectively. Particularly, the dotted box in W means the document 0 has 0, 1, 1 and 0 tokens for topics 0, 1, 2 and 3, respectively. Similarly, the dotted box in D means that word 2 has 1, 1, 0 and

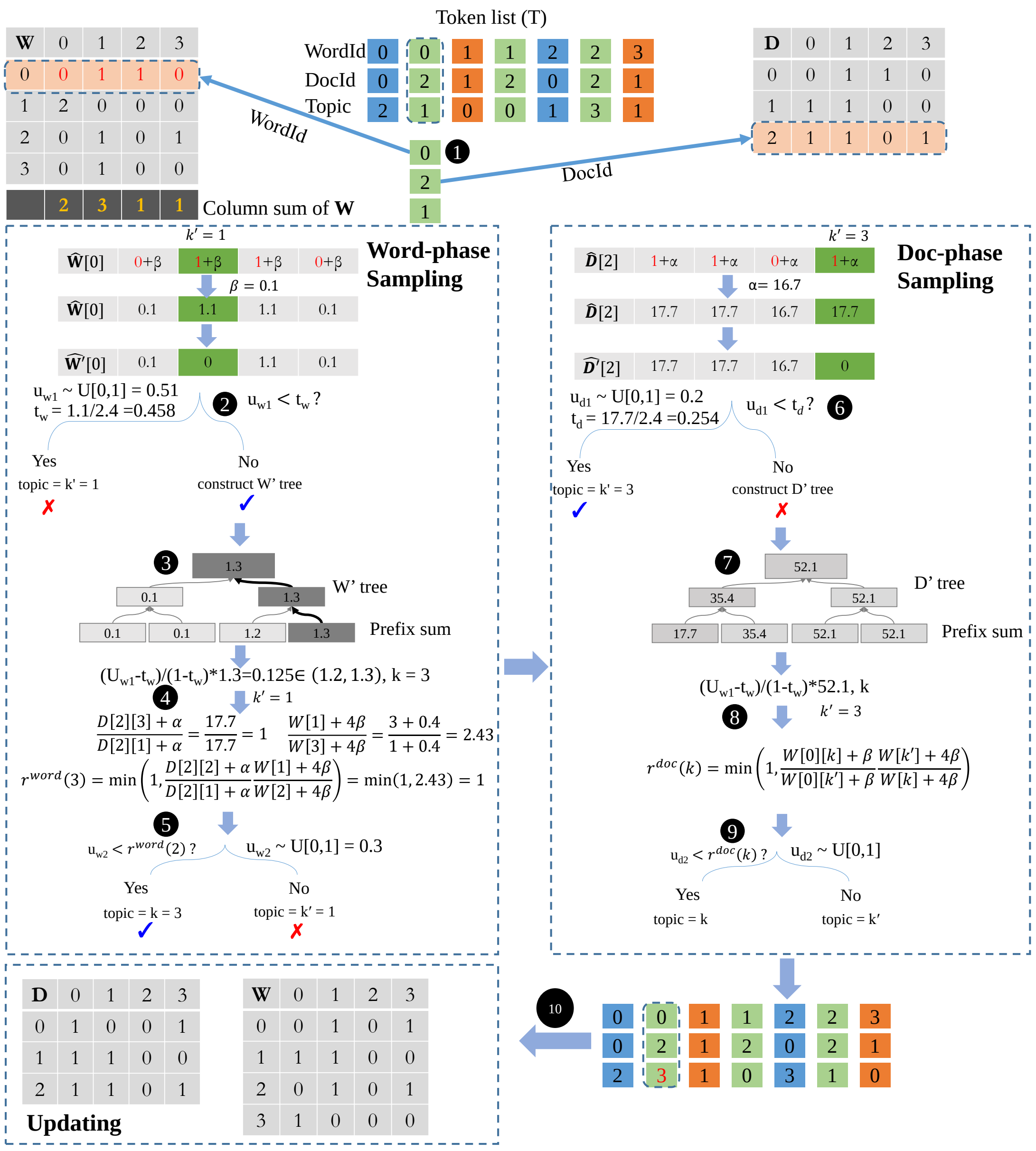


Figure 2.2: mhLDA sampling of one token for the corpus in Figure 1.1. Better viewed in color.

1 tokens for topics 0, 1, 2 and 3, respectively.

For word-phase sampling, we single out the current topic $k' = 1$ and deal with it separately (❷). First, a random number u_{w1} is drawn from uniform distribution $U[0, 1]$ and probability of choosing k', i.e. t_w, is calculated. Then we can determine whether the topic keeps unchanged by comparing t_w and u_{w1}. In the example, $u_{w1} > t_w$, which means the generated temporary new topic is not current topic. So we still need to follow similar process as Figure 2.1 to get the new topic. Note, $\widehat{\mathrm{W}}[k']$ should be set to zero in case current topic is re-included. After tree construction (❸), acceptance rate calculation (❹) and new topic generation (❺), we can update current topic to be 3.

For doc-phase sampling, we follow similar strategy to word-phase sampling. We single out the current topic $k' = 3$ and deal with it separately (❻). First, a random number u_{d1} is drawn from uniform distribution $U[0, 1]$ and probability of choosing k', i.e. t_d, is calculated. Then we can determine whether the topic keeps unchanged by comparing t_d and u_{d1}. In the example, $u_{d1} > t_d$, which means the generated temporary new topic is current topic. So we no longer need to continue remaining steps and directly assign the new topic to be $k' = 3$. The time-consuming part in sampling including D' tree construction (❼), acceptance rate calculation (❽) and new topic generation (❾) can all be skipped.

From Figure 1.3 in Chapter 1, with the training goes, the topic of majority of tokens will keep unchanged. That means majority of tokens will skip the most time-consuming part in sampling and great speedup will be achieved.

For update stage, just reconstruct word-topic matrix and doc-topic matrix from the updated token list (❽).

3 GPU Accelerated Differential Privacy

3.1 Introduction

As “big data” analysis become more pervasive, so too does the concern about the privacy surrounding such gigantic information which is often sensitive in nature. Given the fact that innovation is moving away from human ingenuity toward data-driven machine learning[60], contemporary service providers usually offer free services to users, subsequently analyze the user data to create new marketing opportunities for profits. Even service providers put exceeding efforts in protecting the privacy of their customers by releasing anonymized or aggregated data, this data often reveals much more information than intended, which leads to the advent of differential privacy.

Formally, differential privacy requires that the computation results to be indistinguishable when run with or without any one record, analogous to the fact that one participant who concerns about information leakage has opted out of the dataset. Consequently, the database is differentially private. Note, albeit one can exploit covert channel to extract certain private information,[24] is evident that these channels can be closed completely and easily, leading differential privacy to be a promising avenue for data privacy.

Attracted by the great potentials, an increasing number of researches has surged in this area, mainly from three aspects. (1), differential privacy has drawn significant theoretical studies. Given differential privacy is enacted from cryptography[20], we observe noise foundation research[21], information flow control[50] and data mining design[37] from cryptography field. (2). Along with the theory development, we observe algorithm boom in differential privacy[22]. For instance, using differential privacy for machine learning[29,51], deep learning[2] and graph computing[31,63]. (3). Towards practical usefulness, a collection of simplified systems are invented to support differential privacy, such as, PINQ[39], eKTELO[75] and Flex[30].

In spite of such a prosperity, a practical problem - budget allocation for composite queries (detailed in Section 3.3.3 - is rarely touched, preventing differential privacy from delivering a widespread practical impacts. Briefly, this unveils an urgent need of designing a fast, accurate and privacy preserving budget allocation mechanism. Particularly, without a proper budget allocation, we might retain desirable privacy goal but suffer from unacceptable accuracy loss, which yields useless query results for the follow-up data-driven analysis. Consequently, companies will lose profits. On the other hand, if we exhaust all possibilities to derive the optimal budget allocation, the processing time will be extremely long, ending up breaking the service level agreement (SLA). In short, neither options is satisfoctory.

To this end, we propose an algorithm-system codesigned solution that can arrive at optimal budget allocation within an acceptable time envelop. Particularly, the algorithm part of research introduces a novel gradient decent based design to rapidly optimize the budget allocation towards optimal. On the system track, we plan to use Graphics Processing Units (GPUs) to accelerate the gradient decent computation so that we can achieve optimal budget allocation within an acceptable time envelop. Particularly, this proposal is comprised of the following twin research thrusts:

- Gradient Decent-based Budget Allocation (Section 3.3.2) starts with a random initial budget allocation. At the meantime, we will obtain the accurate answer that is computed without differential privacy mechanism. Subsequently, the difference between these two answers will help calculate the gradient that can optimize the privacy budget allocation for all sub-

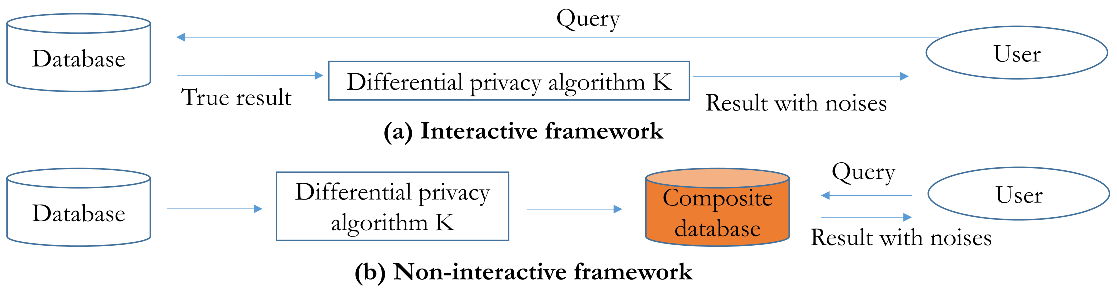

Query
Database
User
Differential privacy algorithm K
True result
Result with noises
(a) Interactive framework
Database
Differential privacy algorithm K
Composite database
Query
User
Result with noises
(b) Non-interactive framework

Figure 3.1: Two different differential privacy frameworks (a) interactive differential privacy framework (b) non-interactive differential privacy framework

queries. Note, this budget allocation is a constraint optimization problem which calls for special treatment in order to be solvable by the gradient decent method[45]. We have verified the feasibility of our design with MATLAB.

- GPU Accelerated Budget Allocation Optimization (Section 3.3.4) plans to leverage the massive parallelism of GPUs to compute the optimal privacy budget allocation in two ways. First, we will use GPUs to quickly compute the optimal sensitivity. Second, we will use GPUs to adjust the privacy budget toward the optimal allocation for all sub-queries.

3.2 Literature Review

This section presents the essential background of differential privacy, studies a collection of related projects and papers, and summarize them with respect to focus areas, as shown in Figure 3.2.

3.2.1 Background

Definition 3.2.1. Differential Privacy. A randomized algorithm $G : D^n \rightarrow R^d$ satisfies (ε, δ) differential privacy if for any pair of databases $A, B \in D^n$ where d(A, B) = 1 (A and B are called neighboring databases), and for all sets S of possible outputs:

$$Pr[G(A) \in S] \leq e^{\varepsilon} Pr[G(B) \in S] + \delta \tag{3.1}$$

Definition 3.2.1 suggests that database B which is derived by either modifying, adding or deleting an element from the original database A will not result in much difference in results because the outputs $G(A)$ and $G(B)$ follow similar distribution. Therefore, the privacy of any specific element cannot be deduced in a differential way. ε and δ are used to describe the similarity of the two distribution. Obviously, the smaller ε is, the more similar the output distributions are and better privacy we can achieve. Usually δ can be neglected and we only care about $(\varepsilon, 0)$ differential privacy.

Briefly, there exist two differential privacy frameworks, that is, interactive and non-interactive frameworks, where the former one is more popular. As shown in Figure 3.1(a), for interactive differential privacy framework, the major concern is the exhaustion of privacy budget. Corresponding solution is to limit number of queries, which is a challenge of this framework. For non-interactive differential privacy framework, as shown in Figure 3.1(b), real database is not exposed to users directly. Users only query from composite database. The major difficulty of this framework lies in that the composite database is very customized and may suffer from incompatibility issues. For example, composite database for histogram distribution privacy may not work for the Average query.

In pursuit of equation 3.1, mainstream differential privacy solutions often exploit three mechanisms to add noise, that is, Laplace, Gaussian and exponential mechanisms. Note, Laplace and Gaussian mechanisms are more common options for the following reason. For Laplace and Gaussian mechanisms, one can simply add Laplace and Gaussian noises directly on the true results, respectively. Exponential mechanism on the contrary will need to sample the output from a distribution, like Latent Dirichlet Allocation (LDA)[7].

It is important to note that we evaluate a differential privacy system with two metrics. That is, the privacy protection intensity ε, which is the inverse of error that can include relative error, absolute error and variance of error, and the turn around time (i.e., performance) of the design. From the aspect of differential privacy algorithm, virtually all existing efforts are pursuing low error and better performance on the premise of privacy budget.

3.2.2 Related Work

While differential privacy theory has been proposed for decades, the corresponding applications are still limited to fields such as histogram analysis[8] and range queries[27]. The grand challenge of building a real-word system is the need of accommodating many complex settings, such as, storage of data (distributed or centralized), type of data (SQL database or graph data), query function (equijoin or non-equijoin or sum and average), noise mechanism (Laplace or exponential) and so on. Below, we summarize the recent progresses in differential privacy.

Machine learning & statistics: In this field, researchers focus on avoiding privacy leakage in machine learning, that is privacy-preserving machine learning and statistics. Paper[29] discusses some common privacy-preserving machine learning and statistics methods such as Naive Bayes Model, Support Vector Machine (SVM), Logistic Regression, Decision Tree and so on. Paper[80] introduces privacy-preserving algorithm of LDA. Paper[51] gives an overview in this field.[36] applies privacy-preserving Naive Bayes scheme on dealing with multiple-source data. Paper[33] applies adaptive privacy budget allocation for privacy-preserving gradient descent method. Paper[2] explores how to avoid privacy leakage from trained model of deep learning.

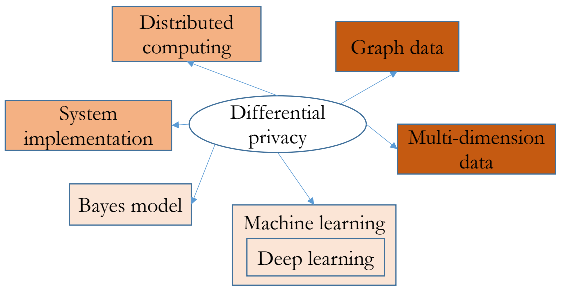


Figure 3.2: Differential privacy research area.

System implementation: In this field, researchers focus on system implementation of differential privacy algorithms. PrivacyBench[27] introduces differential privacyBench framework, which is a platform supporting 1- or 2- dimensional range queries. PINQ[39] provides analysts with a programming interface to unscrubbed data through a SQL-like language.[48] extended PINQ to weighted PINQ, which supports multiple query and multiple set. Paper[30] presents the first practical approach for differential privacy of SQL queries. eKTELO[75] presents eKTELO framework, which supports a serial of statistical tasks such as multidimensional histograms, answering OLAP and range queries.[16] explores differentially private protocols in a distributed setting, i.e, each user holds a private datum. The paper includes two models: the "central" model, in which a trusted server collects user data, which permits a greater accuracy; and the "local" model, in which users randomize their data per not trusting the server, accuracy hence is limited.

Data type. Differential privacy can be applied to graph data. Paper[31] explores differential privacy algorithms for node privacy. Paper[63] is about spectral graph. For high dimensional data,[4] explores the differential privacy analysis for two dimension data (position in a map). Location privacy is included in this area. Further,[68] applies Bayes theory in dealing with correlated data, such as relations in social network.

3.3 Proposed Research

This section discusses the proposed research, before which we summarize the mathematical foundations.

3.3.1 Mathematical Foundation

Before introducing our differential privacy design for SQL queries, we need to introduce some basic definition and theorem about differential privacy. Note they are not the proposed research.

Definition 3.3.1. Sensitivity. For $F : D^n \to R^d$ and all $A, B \in D^n$ where d(A, B) = 1, the sensitivity S of F is defined as:

$$S(F) = \max_{A,B:d(A,B)=1} ||F(A) - F(B)|| \tag{3.2}$$

Sensitivity describes the maximum variation in result introduced by modifying, adding or deleting an element in any database A. For example, we use function COUNT to get number of elements, then the sensitivity of COUNT will be one because modifying, adding or deleting an element will at most get a single variation in results.

Definition 3.3.2. Laplace Distribution. The Laplace Distribution (centered at 0) with scale b is defined as:

$$L(x|b) = \frac{1}{2b} exp(\frac{-|x|}{b}) \tag{3.3}$$

The variance of this distribution is $\sigma^2 = 2b^2$. We use L(b) to denote this distribution.

Definition 3.3.3. Laplace Noise Mechanism. Given any $F : D^n \to R^d$, the Laplace mechanism is defined as:

$$G_L(A, F, \varepsilon) = F(A) + (X_1, \ldots, X_d) \tag{3.4}$$

where X_i is a random variable drawn from L($\frac{S(F)}{\varepsilon}$), where $b = \frac{S(F)}{\varepsilon}$ in Equation 3.3.

Theorem 1. The Laplace mechanism satisfies (ε, 0) differential privacy.

The proof of Theorem 1 can be found in[22]. And it warrants that simply drawing noise from Laplace distribution as shown in Equation 3.4 will meet the (ε, 0) differential privacy.

On the other hand, in order to minimize the output error, which is reflected by the variance of Laplace distribution (σ^2 in Equation 3.3), we need to minimize the sensitivity S under privacy budget ε. Local sensitivity[41], which is data-based sensitivity definition, can be adopted for shrinking sensitivity. Compared with global sensitivity defined in Equation 3.2, whose sensitivity is only bounded by query, local sensitivity calculating from data is an additional cost. So local sensitivity will introduce more cost for large dataset.

Definition 3.3.4. Stable Transformation. We regard a transformation T as c-stable if for any two input databases A and B:

$$|T(A) \oplus T(B)| \leq c \times |A \oplus B| \tag{3.5}$$

Here $\oplus$ is symbol for describing difference between two databases.

Theorem 2. If G satisfies ε differential privacy in Equation 3.4, T is a c-stable transformation. The composite G$\circ$T will provides ($c \times \varepsilon$) differential privacy.

The proof of Theorem 2 can be referred in[39]. In fact, from the aspect of mathematics, T is a linear function, which means the derivative of T at A is a constant c, which is not relevant with the size of A. This theorem serves as an important theoretical insurance for composite SQL queries because some queries itself is actually a stable transformation. For example, Where and Select are all 1-stable transformation, GroupBy is 2-stable transformation, which means c = 1 and 2, respectively, in Equation 3.5. In detail, if databases A and B differ by 1 entry, assuming T is the transformation COUNT, T(A) and T(B) will also be c. With Theorem 2, we introduce the following important theorems for composite queries.

Theorem 3. Assume $G_i : D^n \to R^d$ provides ε_i differential privacy, the sequence of G_i , i.e., $G(A) = (G_1(A), \ldots, G_K(A))$ will provide $\sum_{i=0}^{K} \varepsilon_i$ differential privacy.

Theorem 4. Assume $G_i : D^n \to R^d$ provide ε_i differential privacy, $A_1, \ldots, A_K$ are disjointed sets and $A = A_1 \bigcup \cdots \bigcup A_K$. The sequence of G_i , i.e., $G(A) = (G_1(A_1), \ldots, G_K(A_K))$ will provide $\max \varepsilon_i$ differential privacy.

The proof of Theorem 3 and 4 can be found in[39]. For Theorem 3 , it is crucial for privacy platform that expects to deal with multiple queries because privacy loss ε will increase linearly with the number of queries. If we want total privacy budget ε for N queries, then each query will get about ε/N, which will be very small if N is large. Then we will need to add much more Laplace noise into the results, which will greatly decrease the accuracy. As a special case of Theorem 3, Theorem 4 provides very good performance - the total privacy loss ε will not increase linearly with respect to the number of queries if queries are applied on disjointed subsets of A. So more privacy loss ε_i will be distributed for each query given that total privacy budget ε is fixed. So simplifying the general multi-query case in Theorem 3 into the Theorem 4 case will often greatly help improve the accuracy of results under given privacy budget.

3.3.2 Gradient Decent-based Budget Allocation

For a composite query with multiple sub-queries, privacy budget allocation, which means how to distribute privacy budget across a collection of queries, is an important practical problem. The naive way is to equally assign ε/N budget for each sub-query. But this budget distribution plan might not be optimal. That is, assuming $(S_1, \ldots, S_N)$ represents the sensitivity for queries. If we apply equal budget $(\varepsilon/N, \ldots, \varepsilon/N)$, the total variance will become $Var = \frac{2N^2}{\varepsilon^2}\sum_{i=1}^{N}(S_i^2)$ according to Definition 3.3.2 and 3.3.3. In this context, the sub-queries with larger sensitivity will amplify the variance (a.k.a, error). Thus, it is ideal if we assign smaller budget to sub-queries with larger sensitivity. Towards achieving optimal budget allocation, we can formulate the following optimization objective:

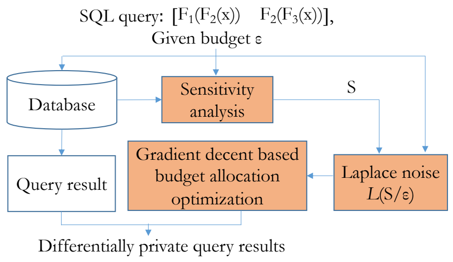

SQL query: [F₁(F₂(x)) F₂(F₃(x))],
Given budget ε
Database
Sensitivity analysis
S
Query result
Gradient decent based budget allocation optimization
Laplace noise L(S/ε)
Differentially private query results

Figure 3.3: Proposed differential privacy framework.

$$\varepsilon_i = arg\min\sum_{i=1}^{N}\left(\frac{S_i^2}{\varepsilon_i^2}\right), \qquad subject\ to \qquad \sum_{i=1}^{N}(\varepsilon_i) = \varepsilon. \tag{3.6}$$

Obviously, this is a constraint optimization problem, we can solve this problem by Lagrange multiplier method. In fact, the objective function is not fixed and varies based on the demand. For example, the objective function can be sum of L1 norm of difference between the output and true result or we set more constraints to optimization condition based on specific demand. So for this case, we will get a different optimized privacy allocation plan from the case in Equation 3.6. But for all objective functions, we can approach them with Gradient Descent method to getting the optimal privacy budget allocation plan.

Figure 3.3 plots the framework we plan to build in order to supports all SQL queries and their composite ones. Assuming we have three queries, that is, F_1, F_2 and F_3, we will

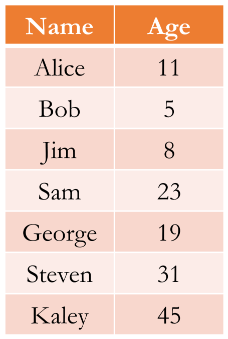
- Conduct sensitivity and stability analysis. For each basic query, such as F_1, F_2 and F_3, we will extract the sensitivity with respect to Equation 3.2. For each composite query, i.e., $F_1(F_2(A))$ and $F_2(F_3(B))$, we will run the transformation check based upon Equation 3.5. Particularly, the sensitivity analysis aims to extract the minimum upper bound of sensitivity S in order to minimize the variance.

- Define budget objective function. For multi-query problem, we will further check whether these sub-queries are disjoint - stated in Theorem 4 thus can help find a better budget. Otherwise, we have to adopt Theorem 3 in order to find the eventual budget $\sum_{i=0}^{K} \varepsilon_i$.

- Allocating privacy budget with gradient decent method. Better privacy budget management can be achieved by solving the optimization problem similar as the problem in Equation 3.6. Knowing sensitivity S and privacy loss ε, the output can be derived by adding true result with Laplace noise $L(\frac{S}{\varepsilon})$.

3.3.3 Composite Query Example

Figure 3.4 shows the database that we will execute our SQL queries upon. For simplicity, we use three independent sub-queries as examples. However, it should be noted that the proposed solutions can also accommodate composite queries with related sub-queries.



Name	Age
Alice	11
Bob	5
Jim	8
Sam	23
George	19
Steven	31
Kaley	45

Figure 3.4: Example database.

SELECT COUNT(*) FROM A WHERE Age BETWEEN 0 AND 18

SELECT COUNT(*) FROM A WHERE Age BETWEEN 19 AND 30

SELECT COUNT(*) FROM A WHERE Age BETWEEN 31 AND 50

According to these queries, the user wants to retrieve the count histogram over three age intervals : [0, 18], [19, 30], [31, 50]. In fact, each query can be summarized as SELECT(COUNT(X)), where X is the input database. We know that COUNT and SELECT function are both 1-stable transformation. Based on Theorem 2, the privacy loss will not change after this composite transformation. Besides, we also find that [0, 18], [19, 30] and [31, 50] are three disjointed intervals. So if the total privacy budget is ε, based on Theorem 4, the privacy budget allocation problem will be summarized as:

$$(\varepsilon_1, \ldots, \varepsilon_N) = arg\min \sum_{i=1}^{N} (\frac{S_i^2}{\varepsilon_i^2}) \qquad subject\,to \qquad \max(\varepsilon_i) = \varepsilon;\ S_i = 1\,for\,i = 1, 2, 3 \tag{3.7}$$

Obviously, the best privacy budget plan for the query sequence will be $(\varepsilon, \varepsilon, \varepsilon)$, which lead to a total variance $Var = 3/\varepsilon^2$.

Assuming we modify the three sub-queries to the following sequential queries:

SELECT COUNT(*) FROM A WHERE Age BETWEEN 0 AND 18

SELECT COUNT(*) FROM A WHERE Age BETWEEN 10 AND 30

SELECT COUNT(*) FROM A WHERE Age BETWEEN 31 AND 50

This time, these queries will count the histogram over these three Age intervals: [0, 18], [10, 30], [31, 50]. Obviously, the first two intervals are overlapped. Theorem 2 thus will not work, which forces us to use Theorem 3. In this context, the budget allocation problem will be summarized as:

$$(\varepsilon_1, \ldots, \varepsilon_N) = \mathit{arg}\min \sum_{i=1}^{N} (\frac{S_i^2}{\varepsilon_i^2}) \qquad \mathit{subject\ to} \qquad \sum_{i=1}^{N} (\varepsilon_i) = \varepsilon;\ S_i = 1\,\mathit{for}\,i = 1, 2, 3 \tag{3.8}$$

The the best privacy budget plan for the query sequence will be $(\varepsilon/3, \varepsilon/3, \varepsilon/3)$ by solve the optimization problem in Equation 3.8, which lead to a total variance $Var = 27/\varepsilon^2$.

Now we consider another situation for the same problem in Figure 3.4(c): the privacy of juveniles, whose age are under 18, needs to be strictly protected per legal requirements. We can reinforce the importance of juveniles' privacy by limiting corresponding privacy budget to a small value. For the example in Figure 3.4(c), if we want to reinforce the privacy protecting of juveniles, then [0, 18] and [10, 30] range in Figure 3.4(c) will be influenced. We force the privacy budget within these two intervals to be not larger than 1/2 times of any other intervals. Budget allocation problem will change to:

$$(\varepsilon_1, \ldots, \varepsilon_N) = arg\min \sum_{i=1}^{N} (\frac{S_i^2}{\varepsilon_i^2}) \qquad subject\,to \qquad \sum_{i=1}^{N} (\varepsilon_i) = \varepsilon;\ \varepsilon_1, \varepsilon_2 \leq 0.5{*}\varepsilon_3;\ S_i = 1 for\, i = 1, 2, 3 \tag{3.9}$$

The the best privacy budget plan for the query sequence will be $(0.25\varepsilon, 0.25\varepsilon, 0.5\varepsilon)$ by solve the optimization problem in Equation 3.9, which lead to a total variance $Var = 36/\varepsilon^2$.

3.3.4 GPU Accelerated Budget Allocation Optimization

The computation complexity of the proposed differential privacy framework for SQL lies in two steps. First, extracting the minimal sensitivity for a large database. Second, using gradient decent method to effectively derive the optimal privacy budget allocation for a collection of sub-queries. This proposal plans to exploit GPUs to accelerate these twin steps.

GPU accelerated sensitivity extraction. For large sequential queries, we can apply parallel sensitivity analysis by GPU. For global sensitivity analysis, because sensitivity is only related to the query itself, sensitivity of single query can be processed by single thread. For local sensitivity analysis, which is a data-based sensitivity analysis, sensitivity of single query can be processed by a GPU thread block and each element in the input database will be processed by a thread to get the query's sensitivity. In fact, considering that the hierarchical relations for queries (sequential queries vs single query) and the data (high-dimensional data vs data in one dimension), we can explore many flexible combinations by fully utilizing the thread and memory hierarchy of GPUs.

GPU accelerated gradient decent computation for budget allocation optimization. For sequential queries, after sensitivity analysis, we need to solve the optimization problem to get the best privacy budget allocation plan for this sequential queries. The objective function is known in advance, then the gradient of the objective function over privacy of single query can be deduced. Sensitivity for each single query is calculated by sensitivity analysis. So optimal budget plan for this sequential queries can be done by a block on GPU, privacy budget for each query will be done by single thread. However, for the optimization problem in Equation 3.6, gradient decent method can't be applied directly because it's a constraint optimization problem. We need to convert the constraint optimization problem to unconstraint optimization problem. Paper[45] provided a method called basic differential multiplier method (BDMM), which can convert the constraint optimization problem to unconstrained extreme problem by Lagrange multiplier method. Some special modifications in gradient need to be done before it can be solved by gradient decent method. Problem in Equation 3.6 can be converted into following unconstrained minimization problem:

$$(\varepsilon_1, \ldots, \varepsilon_N, \lambda) = arg\min(\sum_{i=1}^{N}(\frac{S_i^2}{\varepsilon_i^2}) + \lambda(\sum_{i=1}^{N}(\varepsilon_i) - \varepsilon)) \tag{3.10}$$

We know gradient descent method finds minimum by sliding variables downhill in the opposite direction of gradient. The negative gradient of the objective function will be:

$$\begin{aligned}\dot{\varepsilon}_i &= -(\tfrac{-2S_i^2}{\varepsilon_i^3} + \lambda) \\ \dot{\lambda} &= -(\textstyle\sum_{i=1}^{N}(\varepsilon_i) - \varepsilon)\end{aligned} \tag{3.11}$$

However, if Equation 3.11 is applied directly in gradient decent method, we will not find the local minimum because the optimal solution for the constrained problem tend to be a saddle point in this unconstrained problem. Based on BDMM, we can modify the negative gradient into following format:

$$\begin{aligned}\dot{\varepsilon}_i &= -(\tfrac{-2S_i^2}{\varepsilon_i^3} + \lambda) \\ \dot{\lambda} &= +(\textstyle\sum_{i=1}^{N}(\varepsilon_i) - \varepsilon)\end{aligned} \tag{3.12}$$

Compared with Equation 3.11, we just invert the sign of partial derivative of λ , it can be proved that the local minimum of this unconstrained problem can be found and the minimum is the optimal solution of constrained problem in Equation 3.6. Corresponding proof can be referred in[45]. The steps of gradient decent can be summarized as follows:

1. Give an initial guess of $(\varepsilon_1, \ldots, \varepsilon_N, \lambda)$ for sequential queries which consists of N queries and calculate the result of objective function.

2. Calculate the negative gradient at point $(\varepsilon_1, \ldots, \varepsilon_N, \lambda)$.

3. Move a small step in direction of negative gradient to get new $(\varepsilon_1, \ldots, \varepsilon_N, \lambda)$.

4. Calculate the result of objective function and compare it with result of previous step. If the difference is within a small value, then terminate, else

continue to step 2.

4
publication

Shilong Wang, Da Li, Hengyong Yu, and Hang Liu. 2020. ELDA: LDA made efficient via algorithm-system codesign submission. In Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '20). Association for Computing Machinery, New York, NY, USA, 407–408. DOI:https://doi.org/10.1145/3332466.3374517.

Shilong Wang, Hang Liu, Anil Gaihre, and Hengyong Yu. 2020. EZLDA: Efficient and Scalable LDA on GPUs. Paper submitted to SC20.

Manyun Yang, Xiaobo Liu, Yaguang Luo, Arne J. Pearlstein, Shilong Wang, Hayden Dillow, Kevin Reed, Zhen Jia, Arnav Sharma, Bin Zhou, Dan Pearlstein, Hengyong Yu, Boce Zhang. 2020. Paper Chromogenic Array Enabled by Machine Learning - a Tool for Nondestructive Surveillance and Monitoring of Viable Pathogens in Food. Paper submitted to Nature Food.

References

[cri] https://stackoverflow.com/questions/18963293/cuda-atomics-change-flag.

[2] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 308–318).: ACM.

[3] Akiba, T., Fukuda, K., & Suzuki, S. (2017). Chainermn: scalable distributed deep learning framework. arXiv preprint arXiv:1710.11351.

[4] Andrés, M. E., Bordenabe, N. E., Chatzikokolakis, K., & Palamidessi, C. (2012). Geo-indistinguishability: Differential privacy for location-based systems. arXiv preprint arXiv:1212.1984.

[5] Ashkiani, S., Farach-Colton, M., & Owens, J. D. (2018). A dynamic hash table for the gpu. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 419–429).: IEEE.

[6] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003a). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993–1022.

[7] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003b). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993–1022.

[8] Blocki, J., Datta, A., & Bonneau, J. (2016). Differentially private password frequency lists. In NDSS, volume 16 (pp. 153).

[9] Boyd-Graber, J. L., Blei, D. M., & Zhu, X. (2007). A topic model for word sense disambiguation. In EMNLP-CoNLL.

[10] Cao, L. & Fei-Fei, L. (2007). Spatially coherent latent topic model for concurrent segmentation and classification of objects and scenes. ICCV, (pp. 1–8).

[11] Chang, J. & Blei, D. M. (2009). Relational topic models for document networks. Artificial Intelligence and Statistics, (pp. 81–88).

[12] Chen, J., He, J., Shen, Y., Xiao, L., He, X., Gao, J., Song, X., & Deng, L. (2015a). End-to-end learning of lda by mirror-descent back propagation over a deep architecture. In Advances in Neural Information Processing Systems (pp. 1765–1773).

[13] Chen, J., Li, K., Zhu, J., & Chen, W. (2015b). Warplda: a cache efficient o(1) algorithm for latent dirichlet allocation. PVLDB, 9, 744–755.

[14] Chen, W.-Y., Chu, J.-C., Luan, J., Bai, H., Wang, Y., & Chang, E. Y. (2009). Collaborative filtering for orkut communities: Discovery of user latent behavior. WWW, (pp. 681–690).

[15] Chen, Y., Hayes, A. B., Zhang, C., Salmon, T., & Zhang, E. Z. (2018). Locality-aware software throttling for sparse matrix operation on gpus. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (pp. 413–425).: USENIX Association.

[16] Cheu, A., Smith, A., Ullman, J., Zeber, D., & Zhilyaev, M. (2019). Distributed differential privacy via shuffling. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 375–403).: Springer.

[17] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[18] Dryden, N., Maruyama, N., Moon, T., Benson, T., Yoo, A., Snir, M., & Van Essen, B. (2018). Aluminum: An asynchronous, GPU-aware communication library optimized for large-scale training of deep neural networks on HPC systems. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[19] Dua, D. & Graff, C. (2017). UCI machine learning repository.

[20] Dwork, C. (2011). Differential privacy. Encyclopedia of Cryptography and Security, (pp. 338–340).

[21] Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., & Naor, M. (2006). Our data, ourselves: Privacy via distributed noise generation. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 486–503).: Springer.

[22] Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science, 9(3–4), 211–407.

[23] Gaihre, A., Wu, Z., Yao, F., & Liu, H. (2019). Xbfs: exploring runtime optimizations for breadth-first search on gpus. In Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (pp. 121–131).: ACM.

[24] Haeberlen, A., Pierce, B. C., & Narayan, A. (2011). Differential privacy under fire. In USENIX Security Symposium.

[25] Han, W., Mawhirter, D., Wu, B., & Buland, M. (2017). Graphie: Large-scale asynchronous graph traversals on just a gpu. In 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT) (pp. 233–245).: IEEE.

[26] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.

[27] Hay, M., Machanavajjhala, A., Miklau, G., Chen, Y., & Zhang, D. (2016). Principled evaluation of differentially private algorithms using dpbench. In Proceedings of the 2016 International Conference on Management of Data (pp. 139–154).: ACM.

[28] Holmes, C., Mawhirter, D., He, Y., Yan, F., & Wu, B. (2019). Grnn: Low-latency and scalable rnn inference on gpus. In Proceedings of the Fourteenth EuroSys Conference 2019 (pp.41).: ACM.

[29] Ji, Z., Lipton, Z. C., & Elkan, C. (2014). Differential privacy and machine learning: a survey and review. arXiv preprint arXiv:1412.7584.

[30] Johnson, N., Near, J. P., & Song, D. (2018). Towards practical differential privacy for sql queries. Proceedings of the VLDB Endowment, 11(5), 526–539.

[31] Kasiviswanathan, S. P., Nissim, K., Raskhodnikova, S., & Smith, A. (2013). Analyzing graphs with node differential privacy. In Theory of Cryptography Conference (pp. 457–476).: Springer.

[32] Krestel, R., Fankhauser, P., & Nejdl, W. (2009). Latent dirichlet allocation for tag recommendation. In Proceedings of the third ACM conference on Recommender systems (pp. 61–68).: ACM.

[33] Lee, J. & Kifer, D. (2018). Concentrated differentially private gradient descent with adaptive per-iteration privacy budget. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1656–1665).: ACM.

[34] Li, A. Q., Ahmed, A., Ravi, S., & Smola, A. J. (2014). Reducing the sampling complexity of topic models. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 891–900).

[35] Li, K., Chen, J., Chen, W., & Zhu, J. (2017). Saberlda: Sparsity-aware learning of topic models on gpus. ACM SIGOPS Operating Systems Review, 51(2), 497–509.

[36] Li, T., Li, J., Liu, Z., Li, P., & Jia, C. (2018). Differentially private naive bayes learning over multiple data sources. Information Sciences, 444, 89–104.

[37] Lindell, Y. & Pinkas, B. (2000). Privacy preserving data mining. In Annual International Cryptology Conference (pp. 36–54).: Springer.

[38] Lushan Han, Abhay L. Kashyap, T. F. J. M. & Weese, J. (2013). UMBC-EBIQUITY-CORE: Semantic Textual Similarity Systems. In Proceedings of the Second Joint Conference on Lexical and Computational Semantics: Association for Computational Linguistics.

[39] McSherry, F. D. (2009). Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In Proceedings of SIGMOD International Conference on Management of data: ACM.

[40] Meyer, U. & Sanders, P. (1998). δ-stepping: A parallel single source shortest path algorithm. In European symposium on algorithms (pp. 393–404).: Springer.

[41] Nissim, K., Raskhodnikova, S., & Smith, A. (2007). Smooth sensitivity and sampling in private data analysis. In Proceedings of the thirty-ninth annual ACM symposium on Theory of computing (pp. 75–84).: ACM.

[Nvidia] Nvidia. V100 GPU. Retrived from https://www.nvidia.com/en-us/data-center/v100/. Accessed: 2019, August 5.

[43] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). Gpu computing. Proceedings of the IEEE, 96(5), 879–899.

[44] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365.

[45] Platt, J. C. & Barr, A. H. (1988). Constrained differential optimization. In Neural Information Processing Systems (pp. 612–621).

[46] Porteous, I., Newman, D., Ihler, A., Asuncion, A., Smyth, P., & Welling, M. (2008). Fast collapsed gibbs sampling for latent dirichlet allocation. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 569–577).: ACM.

[47] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press.

[48] Proserpio, D., Goldberg, S., & McSherry, F. (2014). Calibrating data to sensitivity in private data analysis: a platform for differentially-private analysis of weighted datasets. Proceedings of the VLDB Endowment, 7(8), 637–648.

[49] Qian, F., Sha, L., Chang, B., Liu, L.-c., & Zhang, M. (2017). Syntax aware lstm model for chinese semantic role labeling. arXiv preprint arXiv:1704.00405.

[50] Robling Denning, D. E. (1982). Cryptography and data security. Addison-Wesley Longman Publishing Co., Inc.

[51] Sarwate, A. D. & Chaudhuri, K. (2013). Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data. IEEE signal processing magazine, 30(5), 86–94.

[52] Sengupta, S., Harris, M., Zhang, Y., & Owens, J. D. (2007). Scan primitives for gpu computing. In Graphics hardware, volume 2007 (pp. 97–106).

[53] Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., & Keutzer, K. (2019). Q-bert: Hessian based ultra low precision quantization of bert. arXiv preprint arXiv:1909.05840.

[54] Shi, M., Liu, J., et al. (2018). Functional and contextual attention-based lstm for service recommendation in mashup creation. IEEE Transactions on Parallel and Distributed Systems, 30(5), 1077–1090.

[55] Song, S., Liu, X., Wu, Q., Gerstlauer, A., Li, T., & John, L. K. (2018). Start late or finish early: a distributed graph processing system with redundancy reduction. Proceedings of the VLDB Endowment, 12(2), 154–168.

[56] Svyatkovskiy, A., Kates-Harbeck, J., & Tang, W. (2017). Training distributed deep recurrent neural networks with mixed precision on gpu clusters. In Proceedings of the Machine Learning on HPC Environments (pp. 1–8). Association for Computing Machinery.

[57] Teh, Y. W., Newman, D., & Welling, M. (2007). A collapsed variational bayesian inference algorithm for latent dirichlet allocation. In Advances in neural information processing systems (pp. 1353–1360).

[58] Tierney, L. (1994). Markov chains for exploring posterior distributions. the Annals of Statistics, (pp. 1701–1728).

[59] Ueno, Y. & Yokota, R. (2019). Exhaustive study of hierarchical allreduce patterns for large messages between gpus. In Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.

[60] Viktor Mayer-Schönberger and Thomas Ramge (2018). Are the Most Innovative Companies Just the Ones With the Most Data? https://hbr.org/2018/02/are-the-most-innovative-companies-just-the-ones-with-the-most-data.

[61] Wahib, M. & Maruyama, N. (2014). Scalable kernel fusion for memory-bound gpu applications. In SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 191–202).: IEEE.

[62] Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., & Owens, J. D. (2016). Gunrock: A high-performance graph processing library on the gpu. In Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (pp. 1–12).

[63] Wang, Y., Wu, X., & Wu, L. (2013). Differential privacy preserving spectral graph analysis. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 329–340).: Springer.

[64] Wang, Y. & Xu, W. (2018). Leveraging deep learning with lda-based text analytics to detect automobile insurance fraud. Decision Support Systems, 105, 87–95.

[65] Wei, G. C. & Tanner, M. A. (1990). A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. Journal of the American statistical Association, 85(411), 699–704.

[66] Xie, X., Liang, Y., Li, X., & Tan, W. (2018). Culda_cgs: Solving large-scale lda problems on gpus. arXiv preprint arXiv:1803.04631.

[67] Yan, F., Xu, N., & Qi, Y. (2009). Parallel inference for latent dirichlet allocation on graphics processing units. In Advances in neural information processing systems (pp. 2134–2142).

[68] Yang, B., Sato, I., & Nakagawa, H. (2015). Bayesian differential privacy on correlated data. In Proceedings of ACM SIGMOD international conference on Management of Data (pp. 747–762).: ACM.

[69] Yang, Y., Chen, J., & Zhu, J. (2016). Distributing the stochastic gradient sampler for large-scale lda. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1975–1984).

[70] Yao, L., Mimno, D., & McCallum, A. (2009). Efficient methods for topic model inference on streaming document collections. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 937–946).

[71] Yu, H.-F., Hsieh, C.-J., Yun, H., Vishwanathan, S., & Dhillon, I. S. (2015). A scalable asynchronous distributed algorithm for topic modeling. In Proceedings of the 24th International Conference on World Wide Web (pp. 1340–1350).

[72] Yuan, J., Gao, F., Ho, Q., Dai, W., Wei, J., Zheng, X., Xing, E. P., Liu, T.-Y., & Ma, W.-Y. (2015). Lightlda: Big topic models on modest computer clusters. In Proceedings of the 24th International Conference on World Wide Web (pp. 1351–1361).

[73] Yut, L., Zhang, C., Shao, Y., & Cui, B. (2017). Lda*: a robust and large-scale topic modeling system. Proceedings of the VLDB Endowment, 10(11), 1406–1417.

[74] Zaheer, M., Wick, M., Tristan, J.-B., Smola, A., & Steele Jr, G. L. (2016). Exponential stochastic cellular automata for massively parallel inference. Artificial Intelligence and Statistics, (pp. 966–975).

[75] Zhang, D., McKenna, R., Kotsogiannis, I., Hay, M., Machanavajjhala, A., & Miklau, G. (2018a). Ektelo: A framework for defining differentially-private computations. In Proceedings of the 2018 International Conference on Management of Data (pp. 115–130).: ACM.

[76] Zhang, F., Zhai, J., Shen, X., Mutlu, O., & Chen, W. (2018b). Efficient document analytics on compressed data: Method, challenges, algorithms, insights. Proceedings of the VLDB Endowment, 11(11), 1522–1535.

[77] Zhang, F., Zhai, J., Shen, X., Mutlu, O., & Chen, W. (2018c). Zwift: A programming framework for high performance text analytics on compressed data. In Proceedings of the 2018 International Conference on Supercomputing (pp. 195–206).: ACM.

[78] Zhang, Y., Gao, Q., Gao, L., & Wang, C. (2013). Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation. IEEE Transactions on Parallel and Distributed Systems, 25(8), 2091–2100.

[79] Zhao, H., Jiang, B., Canny, J. F., & Jaros, B. (2015). Same but different: Fast and high quality gibbs parameter estimation. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1495–1502).

[80] Zhu, T., Li, G., Zhou, W., Xiong, P., & Yuan, C. (2016). Privacy-preserving topic model for tagging recommender systems. Knowledge and information systems, 46(1), 33–58.