Машинное обучение на больших данных

Лекция

Параллельные, распределённые и онлайновые алгоритмы тематического моделирования

Мурат Апишев (mel-lain@yandex.ru)

МФТИ (ГУ)

5 декабря, 2019

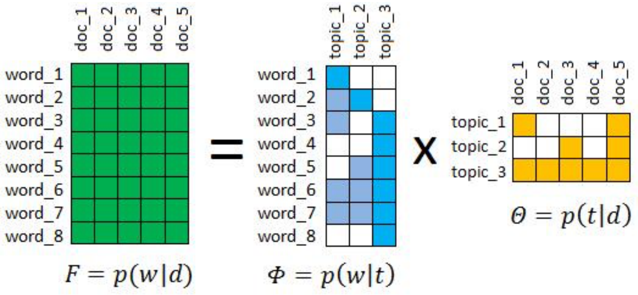
Содержание занятия

- Тематическое моделирование:
  - Модели PLSA и LDA, алгоритмы обучения
  - Оффлайн vs. онлайн обучение
- Алгоритмы на основе LDA
  - AD-LDA, Y!LDA, Mr.LDA
  - Light LDA, Zen-LDA
  - Saber LDA
  - Gensim LDA, Vowpal Wabbit LDA
- Библиотека BigARTM
  - Модель АРТМ как обобщение PLSA и LDA
  - Вариации алгоритма обучения
  - Сравнение с VW.LDA и Gensim

Тематическое моделирование

Тематическое моделирование – приложение машинного обучения к статистическому анализу текстов.

Тема – терминология предметной области, набор терминов (униграм или $n-$грам) часто встречающихся вместе в документах.



- *тема* t – это вероятностное распределение $p(w|t)$ над терминами w
- *документ* d – это вероятностное распределение $p(t|d)$ над темами t

Приложения

Приложения:

- информационный поиск по длинным текстовым запросам;
- анализ данных социальных медиа (соцсети, блоги);
- мягкая кластеризация, визуализация данных;
- классификация текстов;
- суммаризация текстов.

Проблемы при работе с большими данными:

- слишком медленная обработка данных в однопоточном режиме;
- невозможность потоковой обработки;
- необходимость хранения в памяти больших массивов информации.

Необходимо использовать параллельные, распределённые и онлайновые варианты алгоритмов.

Задача тематического моделирования

Дано: W – словарь терминов (униграм или $n-$биграм),
D – коллекция текстовых документов $d \subset W$,
n_{dw} – счётчик частоты появления слова w в документе d.

Найти: модель $p(w|d) = \sum\limits_{t \in T} \phi_{wt}\theta_{td}$ с параметрами $\underset{W\times T}{\Phi}$ и $\underset{T\times D}{\Theta}$:
$\phi_{wt} = p(w|t)$ – вероятности терминов w в каждой теме t,
$\theta_{td} = p(t|d)$ – вероятности тем t в каждом документе d.

Критерий максимизация логарифма правдоподобия:

$$\sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt}\theta_{td} \;\to\; \max_{\phi,\theta};$$

$$\phi_{wt} \geqslant 0; \quad \textstyle\sum_w \phi_{wt} = 1; \qquad \theta_{td} \geqslant 0; \quad \textstyle\sum_t \theta_{td} = 1.$$

PLSA и EM-алгоритм

Максимизация логарифма правдоподобия:

$$\sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_{t} \phi_{wt} \theta_{td} \; \to \; \max_{\Phi, \Theta}$$

EM-алгоритм: метод простых итерация для решения системы уравнений

$$\begin{array}{r} \text{E-шаг:} \\ \\ \text{M-шаг:} \\ \\ \end{array} \begin{cases} p_{tdw} = \operatorname{norm}_{t \in T} (\phi_{wt} \theta_{td}) \\ \phi_{wt} = \operatorname{norm}_{w \in W} (n_{wt}), \quad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw} \\ \theta_{td} = \operatorname{norm}_{t \in T} (n_{td}), \quad n_{td} = \sum_{w \in d} n_{dw} p_{tdw} \end{cases}$$

где $\operatorname{norm}_{i \in I} x_i = \frac{\max\{x_i, 0\}}{\sum_{j \in I} \max\{x_j, 0\}}$

Модель LDA и MAP-оценка

- Добавим для регуляризации априорные распределения на параметры модели:

$$\phi_t \sim \mathrm{Dir}(\phi_t \mid \beta), \forall t = 1, \dots, T; \qquad \theta_d \sim \mathrm{Dir}(\theta_d \mid \alpha), \forall d = 1, \dots, D$$

- Решать эту задачу можно путём поиска *максимума апостериорной вероятности*:

$$\phi_{wt} = \frac{n_{wt} + \beta_w - 1}{\sum_{v=1}^{W} (n_{vt} + \beta_v - 1)}; \qquad \theta_{td} = \frac{n_{td} + \alpha_t - 1}{\sum_{s=1}^{T} (n_{sd} + \alpha_s - 1)}$$

- Отличие от EM-алгоритма для PLSA только в сглаживающих слагаемых
- Но можно обучать LDA и по-другому

Сэмплирование Гиббса для LDA

- Приближённо оценивается распределение на темах $P(Z \mid X, \alpha, \beta)$
- Полученные переменные используются для оценки параметров Φ и Θ
- Генератор быстро начнёт сэмплировать из нужного распределения:

$$p(z_{di} = t \mid X, Z_{\setminus(d,i)}, \alpha, \beta) \propto \left(\sum_{j=1, j\neq i}^{N_d} [z_{dj} = t] + \alpha_t \right) \frac{\sum_{(c,j)\neq(d,i)} [x_{cj} = x_{di}][z_{cj} = t] + \beta_{x_{di}}}{\sum_{w=1}^{W} \left(\sum_{(c,j)\neq(d,i)} [x_{di} = w][z_{cj} = t] + \beta_w \right)}$$

- Дальше можно оценить Φ и Θ по формулам:

$$\phi_{wt} = \frac{n_{wt} + \beta_w}{\sum_{v=1}^{W}(n_{vt} + \beta_v)}, \quad n_{wt} = \sum_{d=1}^{D} \sum_{i=1}^{N_d} [x_{di} = w][z_{di} = t]$$

$$\theta_{td} = \frac{n_{td} + \alpha_t}{\sum_{s=1}^{T}(n_{sd} + \alpha_s)}, \quad n_{td} = \sum_{i=1}^{N_d} [z_{di} = t]$$

Общая схема параллельности

Базовая операция – обработка одного документа: `ProcessDocument`

- Принимает на вход текущие счётчики n_{wt} (может и n_{td}, если они хранятся, иначе можно использовать случайные θ_d) и документ d
- Возвращает инкременты $\tilde{n}_{wt}$ (может и итоговые векторы θ_d, если они вычисляются и нужны)

Внутри могут быть разные реализации: сэмплирование Гиббса, итерации EM-алгоритма

Любой процесс распараллеливания концептуально состоит из двух этапов:

1. Параллельный многократный запуск операций `ProcessDocument`
2. Агрегирование всех инкрементов $\tilde{n}_{wt}$ и их прибавление, возможно, с некоторым весом, к исходным n_{wt}

Процесс повторяется итеративно до сходимости

Метрика качества

- Методов оценивания качества тематического моделирования много, они разнообразны и специфичны для разных задач
- Сходимость модели с заданным словарём W оценивается правдоподобием или его функцией – *перплексия*:

$$P(D) = \exp\left(-\frac{1}{n}\sum_{d\in D}\sum_{w\in d} n_{dw}\ln\sum_{t\in T}\phi_{wt}\theta_{td}\right), \quad n = \sum_{d} n_d.$$

AD-LDA

- Алгоритм Approximate Distributed LDA (AD-LDA) был предложен в «D. Newman, A. Asuncion, P. Smyth, and M. Welling – Distributed algorithms for topic models»
- Основан на коллапсированной схеме Гиббса
- Распараллеливается по ядрам, т.е. в рамках машины используется не многопоточная, а многопроцессорная архитектура

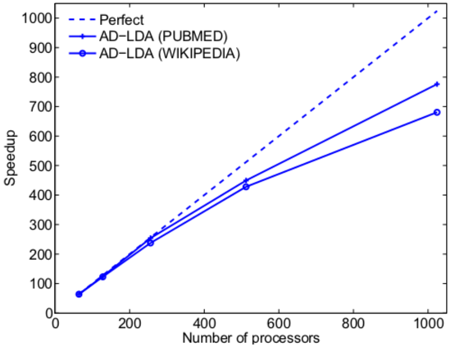
Описание:

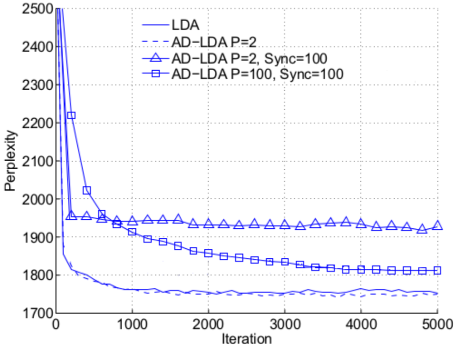
- Коллекция D распределяется по P. Документы распределяются случайным образом, без предварительной кластеризации
- Информация о словах каждого документа и счётчики n_{td} также распределены (обозначим последние n_{tdp})
- Каждый процессор имеет свою локальную *полную* копию n_{wtp} глобальных счётчиков n_{wt}

Алгоритм

1. На потоки (*сэмплеры*) загружается по частям коллекция и копируются счётчики n_{wt}
2. Каждый сэмплер производит обработку своих данных, вызывая `ProcessDocument` для каждого документа. Аккумулируются инкременты $\tilde{n}_{wtp}$
3. Обновляются локальные счётчики n_{tdp}
4. Общий шаг синхронизации для обновления глобальных n_{wt}
5. Новая n_{wt} копируется на процессоры и начинается следующая итерация обработки

Эксперименты





Выводы

AD-LDA имеет ряд серьёзных недостатков:

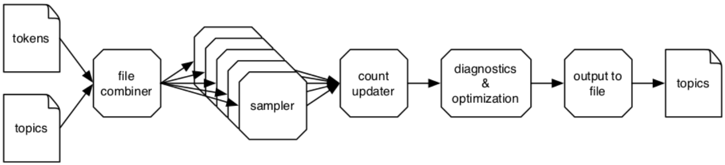
- Необходимость частых синхронизаций
- Из-за синхронизации скорость определяется самым медленным процессором
- Во время итераций сеть простаивает, а во время синхронизаций – перегружена
- Большое потребление памяти – копия глобальных счётчиков n_{wt} хранится на каждом ядре

Y!LDA

- Y!LDA был предложен в «A. Smola and S. Narayanamurthy – An architecture for parallel topic models,»
- Он так же основан на коллапсированной схеме Гиббса
- Производит двухуровневую обработку:
  1. многопоточную в рамках одного нода
  2. многопроцессорную в рамках кластера, в соответствии с т.н. *архитектурой классной доски*

Схема многопоточного параллелизма

- На каждом ноде создаётся несколько потоков-сэмплеров
- И один поток, задача которого – сливать полученные от сэмплеров обновления $\tilde{n}_{wt}$ в глобальную n_{wt}
- Глобальное (в рамках нода) состояние n_{wtp} является общим для всех ядер нода



Описание вычислений на кластере

Проблема: синхронизация глобальной (в рамках кластера) матрицы n_{wt} между всеми нодами-обработчиками

Архитектура классной доски:

- глобальная матрица счётчиков n_{wt} хранится в единственном экземпляре
- она является общей для всех нодов
- её обновление производится сэмплерами асинхронно по одному слову w за один раз
- Для хранения глобального состояния используется `memcached`

Схема обновлений n_{wt}

```
Инициализировать n_wt = n_wtp = n_wtp^old, для всех нодов p;
repeat
    Заблокировать глобально n_wt для некоторого слова;
    Заблокировать локально n_wtp для данного слова;
    Обновить глобальное состояние: n_wt := n_wt + (n_wtp - n_wtp^old);
    Обновить локальное состояние: n_wtp^old = n_wtp = n_wt;
    Разблокировать n_wtp;
    Разблокировать n_wt;
until производится сэмплирование;
```

- n_{wt} – глобальное состояние
- n_{wtp} – текущее локальное состояние
- n_{wtp}^{old} – копия локального состояния на момент последней синхронизации с n_{wt}

Достоинства Y!LDA

Таким образом, Y!LDA решает описанные выше проблемы AD-LDA:

- Отсутствие выделенного шага синхронизации позволяет более быстрым сэмплерам не ждать медленных
- Сеть равномерно загружена всё время работы
- Количество памяти, используемой для хранения копий глобальных счётчиков n_{wt} определяется не числом ядер в кластере, а числом нодов

Mr. LDA

- Реализация Mr. LDA описана в «K. Zhai, J. Boyd-Graber, N. Asadi, M. Alkhouja – Mr. LDA: A Flexible Large Scale Topic Modeling Package using Variational Inference in MapReduce»
- Алгоритм основан на вариационном выводе (вариационный EM-алгоритм)
- Обработка производится в рамках парадигмы MapReduce и реализована на Hadoop
- Для простоты опишем алгоритм в обычного EM-алгоритма – суть та же

Описание схемы MapReduce

- На каждый документ коллекции создаётся mapper. Он производит вызов `ProcessDocument`
- На каждую тему создается reducer. Он занимается слиянием полученных инкрементов n_{wt}
- Третий компонент – driver – присутствует в системе в единственном экземпляре, управляет всем процессом обучения и вычисляет перплексию
- Глобальные параметры n_{wt} хранятся в специальной, доступной только для чтения и общей для всех mapper-ов памяти, называемой *распределённым кэшем*

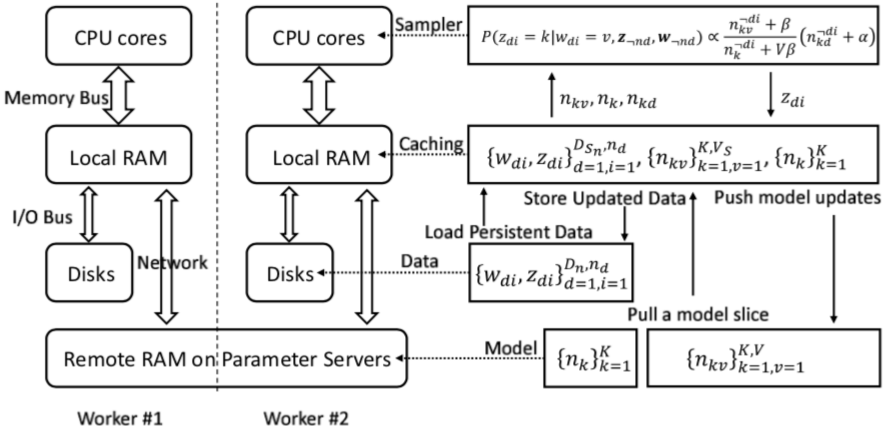
LightLDA

- Реализация LightLDA описана в «Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric P. Xing, Tie-Yan Liu, Wei-Ying Ma. LightLDA: Big Topic Models on Modest Computer Clusters»
- Алгоритм основан на алгоритме Метрополиса-Гастингса (обобщение сэмплирования Гиббса)
- Реализован на фреймворке Petuum
- Основная идея – распределённое хранение модели и данных

Схема обработки

- Модель (счётчики n_{wt}) хранится распределённо
- Коллекция делится на блоки данных (единица обработки узла)
- Все уникальные слова в блоке сохраняются в его метаданных
- При обработке блока за раз извлекается небольшое число слов из метаблока
- Счётчики для этих слов загружаются из хранилища в оперативную память
- Далее происходит обработка только этих слов во всех документах блока
- После завершение обработки одной части слов, начинается обработка следующей и так далее
- Пересылка параметров, подгрузка данных с диска и обработка происходят одновременно

Схема обработки



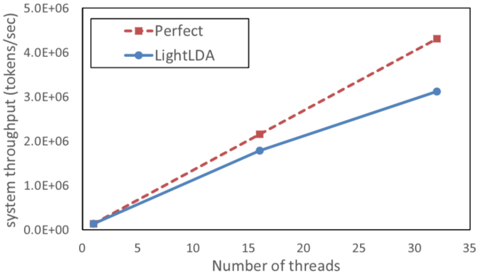
Хранение модели

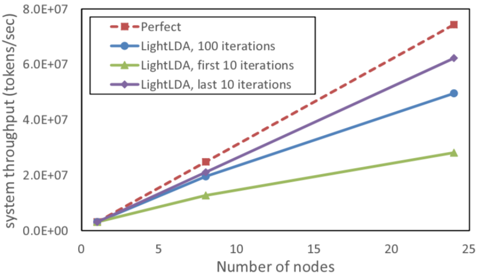
- Распределённое хранение модели требует оперативную память
- Объёмы требуемой памяти растут линейно с ростом числа тем в случае плотной матрицы n_{wt}
- Плотные модели перестают помещаться даже в ОЗУ нескольких узлов
- Выход – использовать разреженное хранение, например, в хэш-таблицах
- Но случайный доступ к элементу в хэш-таблице существенно более затратная операция, чем в плотном массиве
- В LightLDA предлагается компромиссное решение:
  - 10% самых частых слов, на которые приходится 90-95% словопозиций, хранятся в плотном виде
  - Остальные – в хэш-таблице

Технические детали

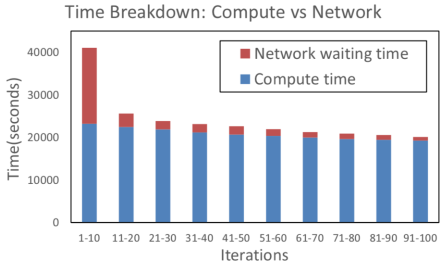
- Для повышения локальности данных слова x_{di} и присваивания тем z_{di} хранятся не по-отдельности, а в виде массива пар
- Это позволяет лучше использовать процессорные кэши, а дополнительные дисковые операции могут быть скрыты
- Отказоустойчивость достигается хранением на диске данных и присваиваний тем: при сбое достаточно загрузить их, инициализировать n_{wt} и продолжить обработку
- Случайное разбиение множества слов блока на фрагменты делает обработку более сглаженной
- В рамках каждого узла сэмплеры работают в многопоточном режиме

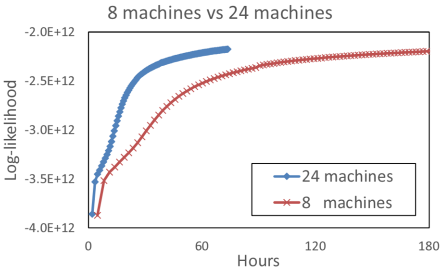
Масштабируемость LightLDA





Утилизация ресурсов и сходимость





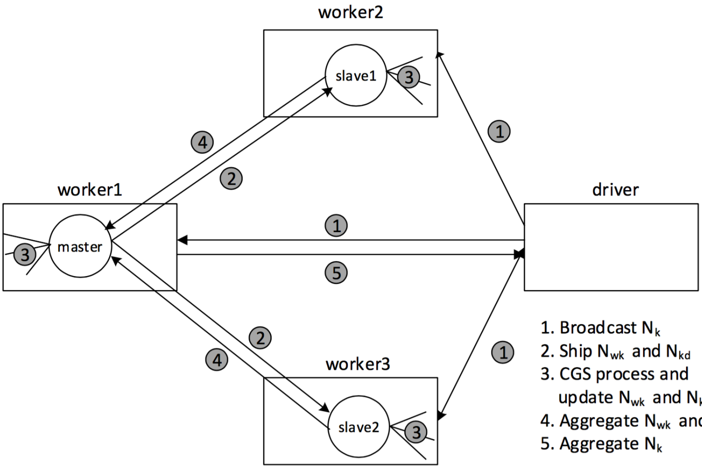
ZenLDA

- Реализация ZenLDA описана в «B. Zhao, H. Zhou, G. Li, Y. Huang – ZenLDA: An Efficient and Scalable Topic Model Training System on Distributed Data-Parallel Platform»
- Алгоритм основан на коллапсированной схеме Гиббса
- Реализован на фреймворке Spark
- Основная идея – хранение данных и модели в виде графа

ZenLDA: граф модели

- Три типа вершин: слова, документы и вершины n_t нормировочных констант
- Каждая вершина-слово соединена ребром со всеми вершинами документами, в которых она встречается хоть раз
- Каждому слову приписана строка матрицы n_{wt}, каждому документу – его n_{td}
- Текущее присваивание тем слов документа Z_{dw} приписано ребру $w - d$
- Граф некоторым образом разбивается на части
- Части передаются нодам-обработчикам
- Обработчики выполняют итерации сэмплирования
- Результаты отправляются на мастер
- Мастер производит обновление счётчиков n_{wt} и n_t

ZenLDA: схема обработки



1. Broadcast N_k
2. Ship N_{wk} and N_{kd}
3. CGS process and update N_{wk} and N_{kd} locally
4. Aggregate N_{wk} and N_{kd}
5. Aggregate N_k

ZenLDA: эксперименты

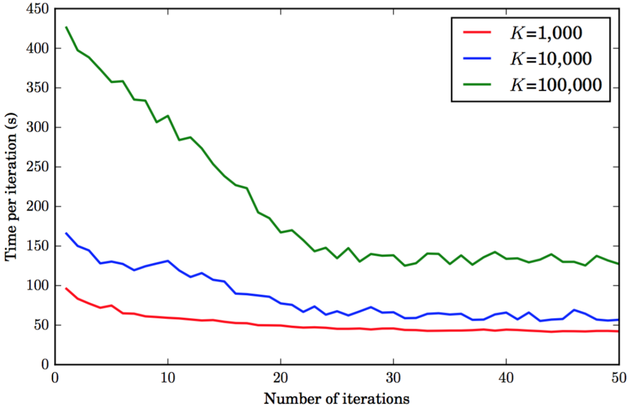
Данные:

Название	Словарь	Документы	Длина
BingWebC1Mon	302.098	16.422.424	3.150.765.984
VW.BingWebC320G	4.780.428	406.038.204	54.059.670.863

- **Количество тем в экспериментах:** 1k, 10k, 100k
- По сравнению с LightLDA реализация ZenLDA показала в 2-6 раз большую производительность

ZenLDA: эксперименты

Зависимость скорости обработки одной итерации от числа тем при фиксированном датасете (**разреженность!**):



SaberLDA

- SaberLDA предложен в работе «Kaiwei Li, Jianfei Chen, Wenguang Chen, Jun Zhu. SaberLDA: Sparsity-Aware Learning of Topic Models on GPUs»
- Как и многие другие, использует модифицированный алгоритм сэмплирования:

$$p(z_{di} = t) \propto p_1(t) \times p_2(t)$$

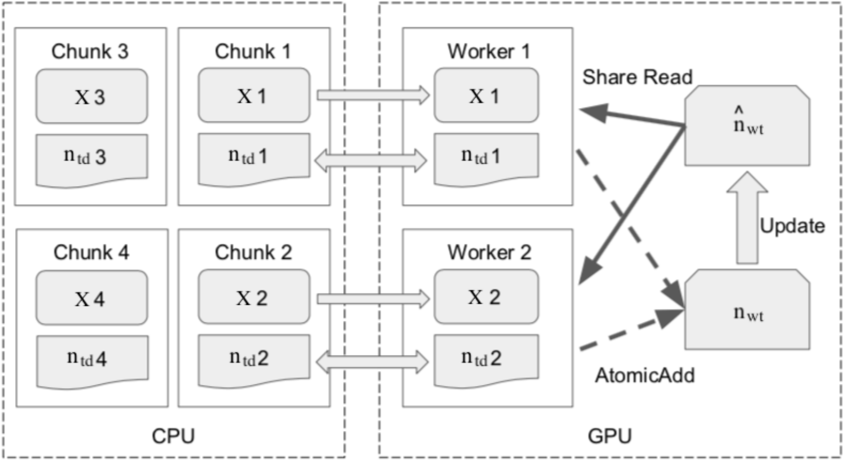
$$p_1(t) \propto n_{td}\, \hat{n}_{wt}, \quad p_2(t) \propto \alpha\, \hat{n}_{wt} \propto \hat{n}_{wt}, \quad \hat{n}_{wt} = \frac{n_{wt} + \beta}{\sum_{w=1}^{W} (n_{wt} + \beta)}$$

- Последний множитель не зависит от документа и может быть предпосчитан для каждого слова заранее
- Документные счётчики обрабатываются разреженно, без учёта нулевых значений
- Работает на GPU, для чего было придумано и реализовано много интересных технических решений

Размещение и обработка данных

- Разреженность нужна не только для сублинейности сложности по T, но и для локальности данных
- Счётчики n_{td} хранятся в виде CSR-матрицы размера $D \times T$
- Обращение к счётчикам n_{wt} и $\hat{n}_{wt}$ производятся случайно, поэтому они хранятся плотно
- **Важное ограничение:** данные модели хранятся на карте перманентно!
- Данные и связанные с ними счётчики хранятся в ОЗУ и подгружаются на GPU частями
- Пересылки данных скрываются за счёт большого числа обработчиков

Размещение и обработка данных



Размещение и обработка данных

- Кэш GPU меньше, чем у CPU, и у него более длинные блоки
- Если модель может поместиться в кэш, то лучше обрабатывать словопозиции по-документно, храня $\hat{n}_{wt}$ в кэше
- На GPU так не получится, зато можно хранить длинные векторы в блоках
- Поэтому все словопозиции в коллекции сортируются не по документам, а по словам
- Для этого в кэш грузится строка $\hat{n}_{wt}$ для данного слова и переиспользуется для сэмплирования всех его вхождений в документах пачки
- Строки n_{td} нужных документов подгружаются и используются целиком

Детали сэмплирования

- Минимальная вычислительная единица GPU – варп – состоит из 32 частей
- В CUDA каждой из частей соответствует один поток, в каждом потоке выполняется один и тот же код
- Можно попробовать обрабатывать каждую словопозицию одним потоком, но это плохо:
  - Разная длина соответствующего вектора n_{td} – производительность будет определяться самым медленным
  - Доступ к n_{td} не локален, разные потоки будут обращаться к различным непоследовательным адресам в общей памяти
- Вместо этого можно всеми потоками обрабатывать подсчёт присваиваний тем для одной словопозиции

Оффлайновые и онлайновые ЕМ-алгоритмы

Оффлайновый ЕМ-алгоритм:

1. Многократное итерирование по коллекции
2. Однократная обработка документа
3. $\Phi\ (n_{wt})$ обновляется в конце каждого прохода по коллекции
4. Применяется при обработке небольших коллекций

Онлайновый ЕМ-алгоритм:

1. Однократный проход по коллекции
2. Многократная обработка одного документа
3. $\Phi\ (n_{wt})$ обновляется через определённое число обработанных документов
4. Применяется при обработке больших коллекций в потоковом режиме

Online LDA: VW.LDA

- Online LDA был предложен в «M. D. Hoffman, D. M. Blei, F. Bach – Online Learning for Latent Dirichlet Allocation»
- Основан на вариационном EM-алгоритме

Vowpal Wabbit LDA:

- Онлайновый
- Не параллельный
- Реализован на C++ без STL и сторонних библиотек
- Относительно эффективный инструмент для моделирования больших коллекций в потоковом режиме

Online LDA: Gensim

- Онлайновый
- Параллельный (однопоточная реализация LdaModel, многопоточная – LdaMulticore)
- Архитектурно похож на Y!LDA. Многопоточный параллелизм реализован не очень эффективно, плохая масштабируемость
- Распределённый (неэффективная реализация)
- Реализован на Python
- Удобный инструмент для моделирования небольших коллекций

ARTM: ещё раз о задаче тематического моделирования

Дано: W – словарь терминов (униграм или $n-$биграм),
D – коллекция текстовых документов $d \subset W$,
n_{dw} – счётчик частоты появления слова w в документе d.

Найти: модель $p(w|d) = \sum\limits_{t \in T} \phi_{wt}\theta_{td}$ с параметрами $\underset{W\times T}{\Phi}$ и $\underset{T\times D}{\Theta}$:
$\phi_{wt} = p(w|t)$ – вероятности терминов w в каждой теме t,
$\theta_{td} = p(t|d)$ – вероятности тем t в каждом документе d.

Критерий максимизация логарифма правдоподобия:

$$\sum_{d\in D}\sum_{w\in d} n_{dw} \ln \sum_{t\in T} \phi_{wt}\theta_{td} \to \max_{\phi,\theta};$$

$$\phi_{wt} \geqslant 0; \quad \textstyle\sum_w \phi_{wt} = 1; \qquad \theta_{td} \geqslant 0; \quad \textstyle\sum_t \theta_{td} = 1.$$

Проблема: задача стохастического матричного разложения некорректно поставленная: $\Phi\Theta = (\Phi S)(S^{-1}\Theta) = \Phi'\Theta'$.

ARTM и регуляризованный EM-алгоритм

Максимизация логарифма правдоподобия с дополнительными аддитивными регуляризаторами R:

$$\sum_{d\in D}\sum_{w\in W} n_{dw} \ln \sum_{t} \phi_{wt}\theta_{td} + R(\Phi,\Theta) \;\to\; \max_{\Phi,\Theta}$$

EM-алгоритм: метод простых итераций для системы уравнений

$$\begin{array}{rl}
\text{E-шаг:} & \left\{\begin{array}{l}
p_{tdw} = \underset{t\in T}{\operatorname{norm}}\bigl(\phi_{wt}\theta_{td}\bigr) \\[2ex]
\phi_{wt} = \underset{w\in W}{\operatorname{norm}}\Bigl(n_{wt} + \phi_{wt}\frac{\partial R}{\partial \phi_{wt}}\Bigr), \quad n_{wt} = \sum_{d\in D} n_{dw}p_{tdw} \\[2ex]
\theta_{td} = \underset{t\in T}{\operatorname{norm}}\Bigl(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\Bigr), \quad n_{td} = \sum_{w\in d} n_{dw}p_{tdw}
\end{array}\right. \\
\text{M-шаг:} &
\end{array}$$

Регуляризаторы

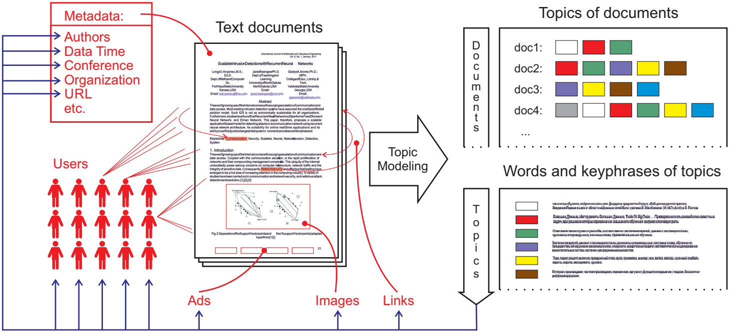
Многие байесовские модели могут быть интерпретированы в терминах ARTM

Примеры регуляризаторов:

1. Сглаживание Φ / Θ (приводит к известной модели LDA)
2. Разреживание Φ / Θ
3. Декорреляция тем в Φ
4. Частичное обучение
5. Максимизация когерентности тем
6. Отбор тем
7. ...

Мультимодальная тематическая модель

Мультимодальная тематическая модель распределения тем на терминах $p(w|t)$, авторах $p(a|t)$, метках времени $p(y|t)$, изображениях $p(o|t)$, связанных документвх $p(d'|t)$, рекламных баннерах $p(b|t)$, пользователях $p(u|t)$, и объдиняет все эти модальности в одно тематическую модель.



M-ARTM и мультимодальный регуляризованный EM-алгоритм

W^m – словарь терминов m-й модальности, $m \in M$,
$W = W^1 \sqcup W^m$ как объединение словарей всех модальностей.

Максимизация логарифма мультимодального правдоподобия с аддитивными регуляризаторами R:

$$\sum_{m \in M} \lambda_m \sum_{d \in D} \sum_{w \in W^m} n_{dw} \ln \sum_t \phi_{wt} \theta_{td} + R(\Phi, \Theta) \to \max_{\Phi,\Theta}$$

EM-алгоритм: метод простых итерация для системы уравнений

$$\begin{array}{rl} \text{E-шаг:} & \begin{cases} p_{tdw} = \operatorname{norm}_{t \in T}(\phi_{wt}\theta_{td}) \\ \phi_{wt} = \operatorname{norm}_{w \in W^m}\left(n_{wt} + \phi_{wt}\frac{\partial R}{\partial \phi_{wt}}\right), \quad n_{wt} = \sum_{d \in D} \lambda_{m(w)} n_{dw} p_{tdw} \\ \theta_{td} = \operatorname{norm}_{t \in T}\left(n_{td} + \theta_{td}\frac{\partial R}{\partial \theta_{td}}\right), \quad n_{td} = \sum_{w \in d} \lambda_{m(w)} n_{dw} p_{tdw} \end{cases} \\ \text{M-шаг:} & \end{array}$$

Проект BigARTM

Особенности BigARTM:

- Быстрая[1] параллельная и онлайновая обработка данных;
- Поддержка мультимодальных регуляризованных тематических моделей;
- Встроенная расширяемая библиотека регуляризаторов и метрик качества;

Сообщество BigARTM:

- Открытый репозиторий `https://github.com/bigartm`
- Описание и документация `http://bigartm.org`

Лицензия BigARTM и программные особенности:

- Бесплатное коммерческое использование (BSD 3-Clause license)
- Кроссплатформенная – Windows, Linux, Mac OS X (32 bit, 64 bit)
- Программные API: command line, C++, Python

---

[1] Vorontsov K., Frei O., Apishev M., Romov P., Dudarenko M. BigARTM: Open Source Library for Regularized Multimodal Topic Modeling of Large Collections Analysis of Images, Social Networks and Texts. 2015

Операция ProcessDocument

Input: документ $d \in D$, матрица $\Phi = (\phi_{wt})$;

Output: матрица $(\tilde{n}_{wt})$, вектор θ_{td};

инициализировать $\theta_{td} := \frac{1}{|T|}$ для всех $t \in T$;

repeat

  $p_{tdw} := \underset{t \in T}{\mathrm{norm}}\bigl(\phi_{wt}\theta_{td}\bigr)$ для всех $w \in d$ и $t \in T$;

  $\theta_{td} := \underset{t \in T}{\mathrm{norm}}\Bigl(\sum_{w \in d} n_{dw} p_{tdw} + \theta_{td} \frac{\partial R}{\partial \theta_{td}}\Bigr)$ для всех $t \in T$;

until до сходимости θ_d;

$\tilde{n}_{wt} := n_{dw} p_{tdw}$ для всех $w \in d$ и $t \in T$;

Оффлайновый алгоритм: описание

- Коллекция документов D разбивается на пакеты, называемые *батчами*
- Оффлайновый алгоритм производит сканирование коллекции, вызывая ProcessDocument для каждого документа $d \in D$ в коллекции
- Затем он агрегирует результирующие матрицы $(\tilde{n}_{wt})$ в финальную матрицу (n_{wt}) размера $|W| \times |T|$
- После каждого прохода по коллекции алгоритм пересчитывает матрицу Φ по формуле

$$\phi_{wt} = \operatorname*{norm}_{w \in W}\left(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}} \right), \quad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw}$$

Оффлайновый алгоритм: листинг

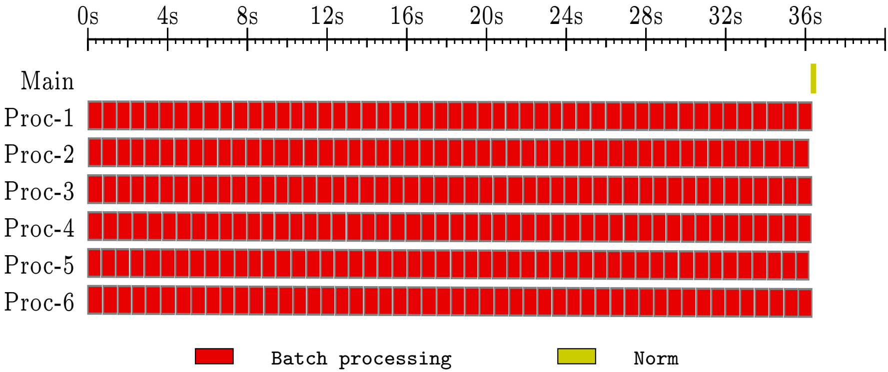
Input: коллекция D;
Output: матрица $\Phi = (\phi_{wt})$;

инициализировать (ϕ_{wt});
создать батчи $D := D_1 \sqcup D_2 \sqcup \cdots \sqcup D_B$;
repeat
$$(n_{wt}) := \sum_{b=1,\dots,B} \sum_{d \in D_b} \mathrm{ProcessDocument}(d, \Phi);$$
$$(\phi_{wt}) := \operatorname*{norm}_{w \in W}\Big(n_{wt} + \phi_{wt} \frac{\partial R}{\partial \phi_{wt}}\Big);$$
until до сходимости (ϕ_{wt});

Оффлайновый алгоритм: обсуждение

- Внешний цикл распараллеливается по потокам
- Внутри каждого пакета внутренний цикл по документам $d \in D_b$ производится в однопоточном режиме
- Значения θ_{td} появляются только в функции `ProcessDocument`
- Как следствие, реализация никогда не хранит в памяти целиком всю матрицу Θ
- Вместо этого значения θ_{td} пересчитываются при каждом проходе по коллекции
- Операция `ProcessDocument` может быть полезна как отдельная функция для поиска распределений θ_{td} новых документов, не участвовавших в обучении

Оффлайновый алгоритм: диаграмма Гантта



- Этот и последующие диаграммы Гантта создавались с помощью коллекции NYTimes: https://archive.ics.uci.edu/ml/datasets/Bag+of+Words
- Размер коллекции ≈ 300k документов, но алгоритмы запускались на подмножествах размером от 70% до 100 % для достижения единого времени работы (≈ 36 сек.)

Онлайновый алгоритм: описание

- Алгоритм является обобщением алгоритма Online variational Bayes для модели LDA (на нём основаны Gensim и VW.LDA)
- Онлайновый ARTM улучшает скорость сходимости оффлайнового алгоритма путём пересчёта матрицы Φ после каждых η батчей
- Введём элементарную операцию для упрощения нотации:

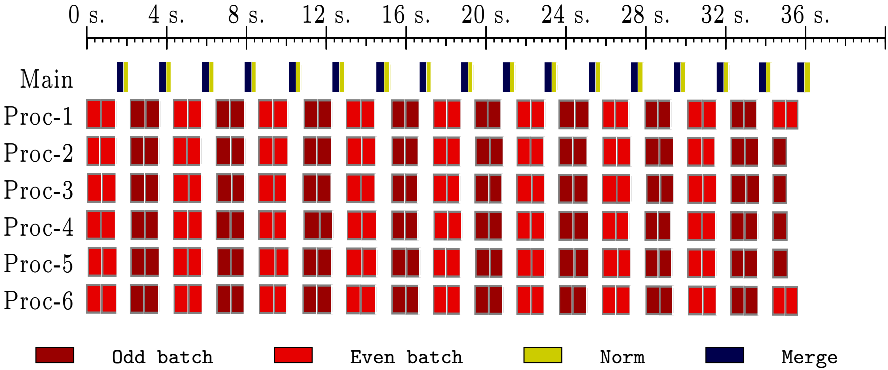
$$ProcessBatches(\{D_b\}, \Phi) = \sum_{D_b} \sum_{d \in D_b} ProcessDocument(d, \Phi)$$

Онлайновый алгоритм: листинг

Input: коллекция D, параметры η, τ_0, κ;
Output: матрица $\Phi = (\phi_{wt})$;

создать батчи $D := D_1 \sqcup D_2 \sqcup \dots \sqcup D_B$;
инициализация (ϕ^0_{wt});
for all обновить $i = 1, \dots, \lfloor B/\eta \rfloor$
  $(\tilde{n}^i_{wt}) := \mathrm{ProcessBatches}(\{D_{\eta(i-1)+1}, \dots, D_{\eta i}\}, \Phi^{i-1})$;
  $\rho_i := (\tau_0 + i)^{-\kappa}$;
  $(n^i_{wt}) := (1 - \rho_i) \cdot (n^{i-1}_{wt}) + \rho_i \cdot (\tilde{n}^i_{wt})$;
  $(\phi^i_{wt}) := \operatorname*{norm}_{w \in W}\big(n^i_{wt} + \phi^{i-1}_{wt} \frac{\partial R}{\partial \phi_{wt}}\big)$;

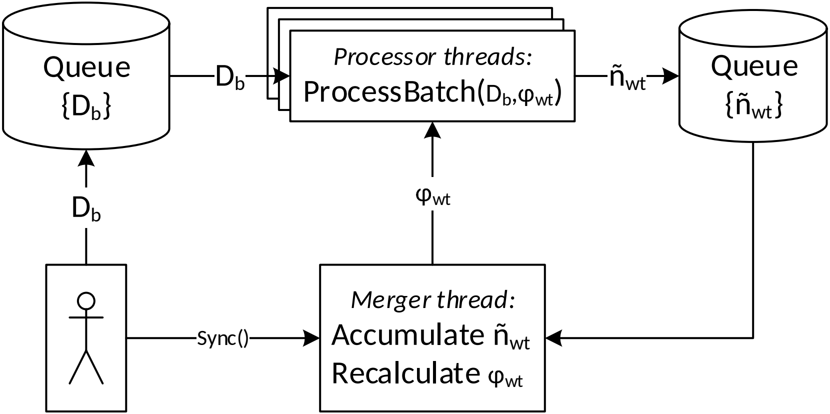
Онлайновый алгоритм: диаграмма Гантта



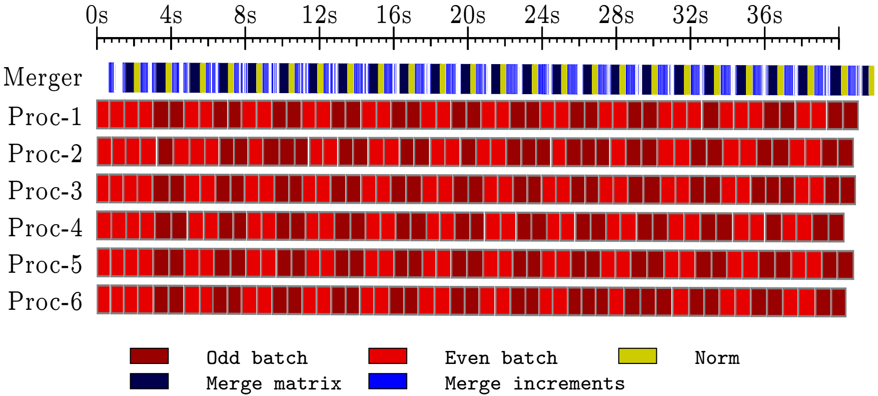
Асинхронный онлайновый алгоритм (Async): описание

- Есть поток `DataLoader`, который загружает батчи с диска в *очередь обработки* `Processor queue`
- Каждый поток обработчик `Processor` извлекает из очереди по одному батчу и производит на нём итерации EM-алгоритма.
- После того, как `Processor` окончил обработку батча, он помещает инкременты $\tilde{n}_{wt}$ в *очередь слияния* `Merger queue` (если в ней есть место) и начинает обработку следующего батча
- Когда число обновлений в очереди `Merger queue` становится равным параметру η, выделенный поток `Merger` производит обновление матрицы Φ

Алгоритм Async: схема



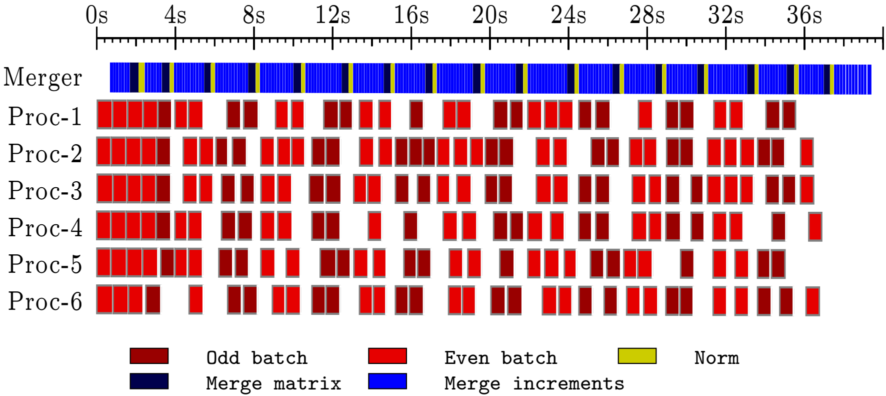
Алгоритм Async: диаграмма Гантта в нормальной ситуации



Алгоритм Async: недостатки

- Алгоритм Async не определяет порядок слияния $\tilde{n}_{wt}$ $\Rightarrow$ результирующая матрица Φ различна от запуска к запуску
- Помещение инкрементов $\tilde{n}_{wt}$ в очередь может серьёзно увеличить потребление памяти $\Rightarrow$ что приводит к перегрузке потока слияния *Merger*
- Особенно поток слияния могут перегрузить маленькие батчи или малое число внутренних итераций в `ProcessDocument`
- Это означает, что пользователь должен настраивать технические параметры, что недопустимо

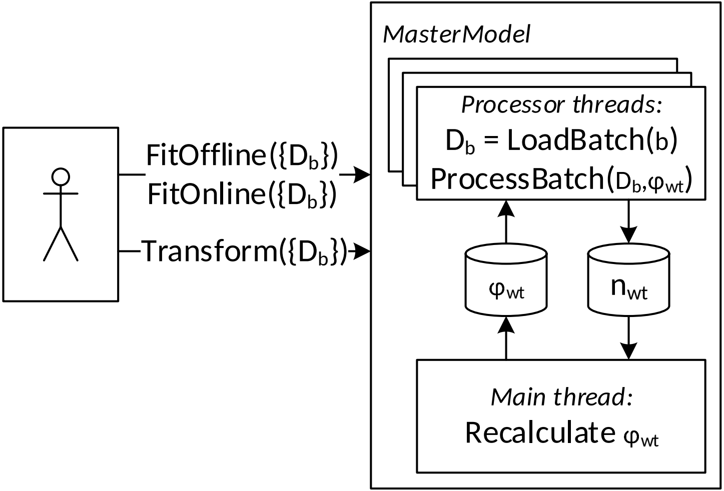
Алгоритм Async: диаграмма Гантта в плохой ситуации



Детерминированный асинхронный онлайновый алгоритм

- Чтобы избежать недетерминированного поведения, потребуем обновлений не после *первых* η батчей, а после *заданных* η батчей
- Введём две новые операции: `AsyncProcessBatches` и `Await`
- AsyncProcessBatches эквивалентна ProcessBatches, кроме того, что она берёт задачу на выполнение и сразу возвращает управление в вызвавший поток
- Она выдаёт объект типа future (прмером является `std::future` из стандарта C++11), который может быть затем подан в операцию `Await` для получения результатов, т.е. инкрементов $\tilde{n}_{wt}$
- Между вызовами `AsyncProcessBatches` и `Await` алгоритм может выполнять полезную работу по обновлению Φ, пока в фоновом режиме потоки `Processor` вычисляют матрицу $\tilde{n}_{wt}$

DetAsync: схема

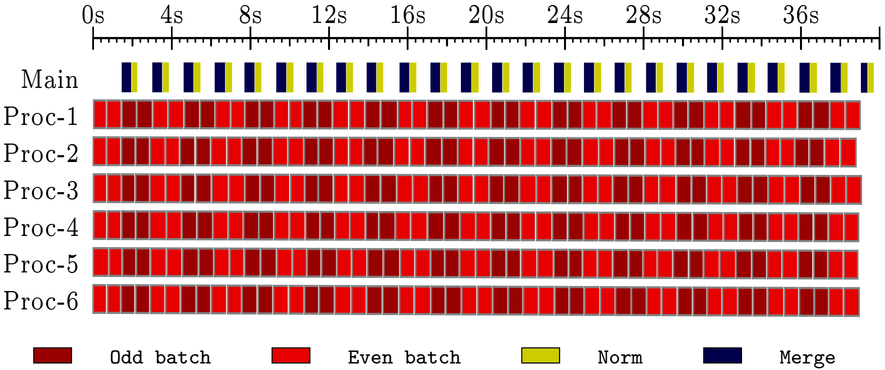


DetAsync: листинг

Input: коллекция D, параметры η, τ_0, κ;
Output: матрица $\Phi = (\phi_{wt})$;

создать батчи $D := D_1 \sqcup D_2 \sqcup \dots \sqcup D_B$;
инициализировать (ϕ^0_{wt});
$F^1 :=$ AsyncProcessBatches$(\{D_1, \dots, D_\eta\}, \Phi^0)$;
for all обновления $i = 1, \dots, \lfloor B/\eta \rfloor$
    if $i \neq \lfloor B/\eta \rfloor$ **then**
        $F^{i+1} :=$ AsyncProcessBatches$(\{D_{\eta i+1}, \dots, D_{\eta i+\eta}\}, \Phi^{i-1})$;
    $(\hat{n}^i_{wt}) :=$ Await(F^i);
    $\rho_i := (\tau_0 + i)^{-\kappa}$;
    $(n^i_{wt}) := (1 - \rho_i) \cdot (n^{i-1}_{wt}) + \rho_i \cdot (\hat{n}^i_{wt})$;
    $(\phi^i_{wt}) := \operatorname*{norm}_{w \in W}\left(n^i_{wt} + \phi^{i-1}_{wt} \frac{\partial R}{\partial \phi_{wt}}\right)$;

DetAsync: диаграмма Гантта



DetAsync: детали реализации

- В старой архитектуре матрицы $\tilde{n}_{wt}$, построенные по разным батчам, хранились в очереди и агрегировались потоком слияния `Merger`
- В новой архитектуре поток `Merger` был удалён, потоки-обработчики пишут обновления напрямую в n_{wt} (локальный аналог «архитектуры классной доски»)
- *Spin lock-и* предотвращают одновременное обновление одной строки несколькими потоками
- Тройка операций «lock-update-release» производится в цикле по всем словам документа $w \in d$ в конце операции `ProcessDocument`
- Выделенный поток загрузки данных `DataLoader` был так же ликвидирован, потоки-обработчики сами загружают себе батчи на обработку с диска

Все структуры данных, передаваемые внутри библиотеки, сгенерированы технологией Google protocol buffers

Сравнение реализаций и алгоритмов

	Gensim	VW-LDA	BigARTM (Online ARTM)	BigARTM (DetAsync)
P = 1, T= 50	142m (4945)	50m (5413)	42m (5117)	25m (5131)
P = 1, T= 100	287m (3969)	91m (4592)	52m (4093)	32m (4133)
P = 1, T= 200	637m (3241)	154m (3960)	83m (3347)	53m (3362)
P = 2, T= 50	89m (5056)		22m (5092)	13m (5160)
P = 2, T= 100	143m (4012)		29m (4107)	19m (4144)
P = 2, T= 200	325m (3297)		47m (3347)	28m (3380)
P = 4, T= 50	88m (5311)		12m (5216)	7m (5353)
P = 4, T= 100	104m (4338)		16m (4233)	10m (4357)
P = 4, T= 200	315m (3583)		26m (3520)	16m (3634)
P = 8, T= 50	88m (6344)		8m (5648)	5m (6220)
P = 8, T= 100	107m (5380)		10m (4660)	6m (5119)
P = 8, T= 200	288m (4263)		15m (3929)	10m (4309)

Таблица 5: Сравнение BigARTM с Vowpal Wabbit.LDA и Gensim (P – число потоков)

Фреймворк / T	2000	5000
BigARTM (Online ARTM)	166m (2377)	399m (1942)
BigARTM (DetAsync)	119m (2645)	281m (2216)

Таблица 6: Запуск BigARTM с большим числом тем.