Распределенные вычислительные системы для машинного обучения

И.Е. Трофимов

Аннотация Машинное обучение является активно развивающейся областью исследований. Во многих задачах машинного обучения и интеллектуального анализа данных возникает необходимость работать с большими массивами данных. Эти массивы зачастую не могут быть обработаны на одном компьютере, или обработка занимает слишком много времени. Если в этих задачах использовать для обучения только часть имеющихся данных, то точность модели, как правило, падает. Для решения этой проблемы используются распределенные вычислительные системы. Наиболее популярные подходы к разработке программного обеспечения таких систем: модели вычислений Map/Reduce, Spark, модели вычислений на графах, использование передачи сообщений по стандарту MPI, архитектура сервера параметров. В данной статье дан обзор таких систем, проведен анализ их достоинств и недостатков применительно к задачам машинного обучения. В отдельном разделе проведен анализ распределенных систем для обучения искусственных нейронных сетей.

Ключевые слова: машинное обучение, анализ данных, большие данные, распределенные системы.

Введение

В настоящее время во многих задачах машинного обучения и интеллектуального анализа данных возникают большие наборы данных. В качестве примера можно привести задачи поиска в интернете (ранжирование документов в выдаче поисковой системы), онлайн рекламы (прогнозирование вероятности клика по рекламе), обработки текстов (классификация текстов, определение векторных представлений слов), компьютерного зрения (классификация изображений), распознавания речи и т.д. Такие задачи характеризуются большим числом обучающих примеров, высокой размерностью пространства признаков, или и тем и другим одновременно. Обучающие выборки, как правило, разреженные. Если в этих задачах использовать для обучения только часть имеющихся данных, то точность модели, как правило, падает. Большие обучающие выборки часто не помещаются в оперативную память одного компьютера или обработка на одном компьютере занимает слишком долгое время. Объем параметров моделей также иногда превосходит размер оперативной памяти одного компьютера.

Поэтому важным направлением исследований является разработка методов машинного обучения, специально предназначенных для больших выборок, а также разработка распределенных вычислительных систем, позволяющих применять существующие методы на больших выборках. От таких систем требуется:

- **Масштабируемость** по количеству обучающих примеров и параметрам модели.
- **Отказоустойчивость**. При распределенном выполнении на кластере вероятность отказа хотя бы одного узла высока. Распределенная система должна быть устойчива к таким отказам.
- **Производительность**. Важной проблемой распределенных вычислений является "*проблема отстающего*", (slow node problem).

Суть ее состоит в том, что при необходимости синхронизации между узлами, один медленный узел будет тормозить вычисления в целом. Аналогично отказам - при работе на большом кластере хотя бы один узел с большой вероятностью будет медленным. Распределенная система должна эффективно решать "проблему отстающего". Передача данных по сети не должна быть узким местом вычислительной системы.

- **Универсальность**. Машинное обучение обычно является частью процесса анализа данных, включающего предобработку данных, вычисление признаков, подбор гиперпараметров обучения, анализ результатов использования модели в прикладной задаче и т.д. Поэтому возможность решения всех задач в одной системе также является весомым аргументом.

Методы машинного обучения, которые чаще всего применяются к большим выборкам из описанных ранее прикладных задач и требуют распределенного обучения: обобщенные линейные модели [1, 51, 32], LDA [47, 35, 32], коллаборативная фильтрация [35, 49], бустинг деревьев решений [10, 50, 52], word2vec [39], глубокие нейросети [27, 17, 19].

Следует отметить, что разработка распределенных вычислительных систем - сложная инженерная задача. Поэтому в статьях обычно проводится только фрагментарное сравнение систем между собой из-за сопутствующих технических сложностей. Данная обзорная статья призвана заполнить этот пробел.

В данной работе рассмотрены следующие подходы к разработке программ для распределенного машинного обучения: модель вычислений Map/Reduce, передача сообщений по стандарту MPI, архитектура "сервера параметров", система Spark, модели вычислений на графах. Каждый из данных подходов в чем-то ограничивает разработчика и предполагает определенный стиль разработки программ. В отдельной главе рассмотрены методы распределенного обучения нейросетей. Хотя эти методы частично пересекаются с остальным материалом статьи, данная тема выделена в отдельную главу в связи с важностью и перспективностью нейросетевых моделей в машинном обучении.

Каждый из подходов к распределенному машинному обучению имеет свои достоинства и недостатки. Их анализу и сравнению и посвящена данная статья.

1. Map/Reduce

Модель вычислений[1] Map/Reduce была предложена в статье [20]. В рамках этой модели каждая задача принимает на вход некоторый список пар *(key, value)*, применяет к ним операции *map* и *reduce* и возвращает список пар *(key, value)*. Данные обычно хранятся в файлах распределенной файловой системы.

Пользователю необходимо определить базовые операции *map* и *reduce*:

```
map:    (key1, value1) → list(key2, value2)
reduce: (key2, list(value2)) → list(key3, value3)
```

Операция *map* принимает на вход одну пару (*key1, value1)* и возвращает список пар *(key2, value2)*. Операция *reduce* принимает на вход все пары с одним ключом *key2* и возвращает список пар *(key3, value3)*. Между операциями *map* и *reduce* происходит группировка пар с одинаковым значением *key2*. На разных блоках данных вычисления проводятся параллельно. При запуске операций *map* и *reduce* программист не контролирует, на каком узле кластера и в какой последовательности на блоках данных будет выполняться программа. В тоже время, планировщик старается выполнять операции локально, т.е. минимизировать передачу данных по сети. Планировщик стремится равномерно распределить нагрузку на узлы: при выполнении операции *map* - по объему блоков, во время операции *reduce* - данные для обработки распределяются равномерно по хешу от ключа. При отказе одного узла кластера, выполняющиеся на нем вычисления перезапускаются на других узлах (файлы в распределенной файловой системе реплицируются), что обеспечивает отказоустойчивость.

Модель Map/Reduce является индустриальным стандартом для распределенных вычислений и широко используется в интернет-компаниях. Существует несколько проприетар-

[1] Модель вычислений – это стиль программирования, в котором выполняются вызовы инструкций из библиотек.

ных реализаций модели Map/Reduce: Google MapReduce, Yandex Tables (YT) [5], Disco [42] и т.д. На практике широко используется свободная реализация Apache Hadoop [54]. Кластеры Map/Reduce обычно включают от нескольких десятков до нескольких тысяч узлов. Для решения "проблемы отстающего", в реализациях модели Map/Reduce используется техника *спекулятивного выполнения* [20]. Она состоит в следующем: когда операции *map* или *reduce* завершились на большей части блоков данных, то копии задач на незавершившихся блоках запускаются на других узлах. Несмотря на дополнительные вычисления, такая техника часто ускоряет выполнение операций *map* и *reduce* на полном объеме данных. Техника спекулятивного выполнения подразумевает, что задачи *map* и *reduce* на разных блоках не обмениваются данными между собой в процессе исполнения.

С теоретической точки зрения, с помощью модели Map/Reduce можно реализовать алгоритмы, подходящие под "Statistical Query Model" [14]. Это означает, что в алгоритме используются только функции, вычисляемые в виде суммы по обучающей выборке. Например, если алгоритм требует только вычисления аддитивной по объектам функции потерь и ее градиента, то он удовлетворяет данному условию. Для большого числа популярных методов существуют алгоритмы обучения, подходящие под "Statistical Query Model": линейная и логистическая регрессии, наивный Байес, GDA, PCA, ICA, линейный SVM, обучение нейросети градиентным спуском, EM-алгоритм для смеси гауссиан и т.д. [14].

Несмотря на универсальность модели, отмечается [36, 1], что она хорошо приспособлена для вычисления признаков из больших выборок, но плохо подходит для итеративных алгоритмов машинного обучения. На это есть три основные причины:

1. Каждая итерация алгоритма предполагает запуск новой задачи Map/Reduce, что происходит с задержкой из-за планировщика задач, который ожидает выделения ресурсов.

2. На каждой итерации заново читаются входные данные с диска и пересылаются по сети.

3. Нет встроенных способов хранения состояния между итерациями.

Кроме того, в модели Map/Reduce вычисления на разных блоках данные выполняются параллельно, что не позволяет учитывать структуру данных. Поэтому при большом числе итераций такая реализация будет неэффективной.

Вариант решения проблемы 1 предложен в [45] - это "Forward Scheduling". Идея состоит в том, что мастер-программа, запускающая задачи Map/Reduce, запускает их заранее. После выделения ресурсов и начала фактической работы задач, мастер-программа сообщает им по RPC (в обход планировщика задач) какую конкретно операцию они должны выполнить. Естественно, что такой подход применим, если большое число задач выполняются на одних тех же входных данных. Для решения проблемы 2 были предложены расширения модели Map/Reduce [8, 22], поддерживающие кеширование данных между итерациями. Впрочем, они не получили широкого распространения из-за недостаточной отказоустойчивости. 3-я проблема решается обычно хранением состояния в HBase или кеш-файлах, которые передаются задачам Map/Reduce перед запуском.

Большое число реализаций алгоритмов с помощью модели Map/Reduce описаны в [31, 14, 6]. Несмотря на отмеченные недостатки, MapReduce используется в системах PLANET [45] и H2O [16] для обучения Gradient Boosted Decision Trees (GBDT). Впрочем, как отмечается в обзоре [7], реализации на MPI и RPC более вычислительно эффективны. В статье [11] предложена эффективная реализация метода оптимизации L-BFGS на базе Map/Reduce.

К **плюсам** реализации методов машинного обучения на Map/Reduce является чрезвычайная распространенность кластеров Map/Reduce, особенно Apache Hadoop. Также программы Map/Reduce обладают хорошей отказоустойчивостью даже при работе на перегруженном задачами (что является нормой) промышленном кластере. Универсальность модели позволяет реализовать широкий класс алгоритмов и проводить весь цикл анализа данных на Map/Reduce. Программы, написанные на Map/Reduce обладают хорошей масштабируемостью.

Минусом же является, как уже было сказано выше, плохая производительность при выполнении итеративных алгоритмов, в частном случае - алгоритмов машинного обучения.

2. MPI

Message Passing Interface (MPI) [38] - это стандарт передачи сообщений между процессами в параллельном программировании, широко использующийся при высокопроизводительных вычислениях. Существуют реализации MPI для многих популярных операционных систем и языков программирования. MPI поддерживает двухсторонние (point-to-point) и коллективные (collective) обмены данными. Существуют реализации стандарта как для систем с общей памятью - для обмена данными между процессами на одном компьютере, так и для распределенных систем [23, 46, 1]. MPI определяет только стандарты для передачи данных, не гарантируя отказоустойчивой работы программы в распределенной среде, в отличие от реализаций Map/Reduce.

Наиболее часто используемая функция из стандарта MPI в распределенном машинном обучении - это `AllReduce`, которая эквивалентна последовательному выполнению функций `Reduce` (из стандарта MPI) и `Broadcast`. Сигнатура функции `AllReduce` приведена ниже

```
int MPI_AllReduce(
void* input_data_p      /* in */,
void* output_data_p     /* out */,
int count             /* in */,
MPI_Datatype datatype   /* in */,
MPI_Op operator         /* in */,
MPI_Comm comm           /* in */)
```

Операция `AllReduce` применяет ассоциативную коммутативную операцию operator покомпонентно к массивам `input_data_p` одинаковой длины `count`, хранящимся на нескольких узлах. Примеры таких операций: сумма, произведение, максимум, минимум и т.д. Результат операции передается на все машины в массив `output_data_p`. Аналогично функции `AllReduce` для систем с общей памятью [43], распределенный вариант передает сообщения по структуре остовного бинарного дерева[2].

Плюсом программ, использующих MPI является более высокая производительность по сравнению с реализациями на Map/Reduce, так как отсутствуют накладные расходы по запуску задачи в итеративных алгоритмах. Программы, использующие MPI обычно пишутся по технике SPMD (Single Program Multiple Data) и получаются небольшим изменением кода последовательной программы, что также является плюсом.

Минусом же является более низкая отказоустойчивость - при отказе одного из узлов не могут продолжать вычисления остальные. Также программы, написанные с использованием MPI, страдают из-за "проблемы отстающего". Операция `AllReduce` является моментом синхронизации и не может быть выполнена, пока все машины не завершат предыдущие вычисления. Поэтому для эффективной работы нагрузка на узлы кластера должна быть *сбалансирована*, что является ответственностью программиста.

Примеры успешных проектов, использующих функции стандарта MPI в распределенных системах: Vowpal Wabbit [1], LibLinear [58], d-GLMNET [51], XGBoost [10], LambdaMART [50], pGBRT [52], COTS HPC [17].

3. Сервер параметров

Сервер параметров - это система для хранения и модификации параметров модели машинного обучения или оптимизации, предназначенная для использования в распределенной среде. При выполнении программ, использующих сервер параметров, все узлы кластера, не считая технических (например, планировщика задач), делятся на 2 группы: узлы сервера параметров и узлы обработки данных. Сервер параметров состоит одного или более узла кластера и поддерживает две базовые операции - чтения параметров (*pull*) и записи изменения параметров (*push*). Узлы обработки данных, в свою очередь, выполняют итерации алгоритма обучения, периодически читая синхронизированный массив параметров из сервера параметров и отправляя изменения на запись. Такого рода системы обычно обладают следующими свойствами:

- Параметры хранятся в *key-value* хранилище.
- Отказоустойчивость сервера параметров. Данные реплицируются между узлами.
- Отказоустойчивость обработки данных. При отказе узла обработки данных, вычисления по этому набору данных передаются другому узлу.

[2] Остовное дерево - минимальное подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам.

- Поддержка различных типов консистентности[3] обновлений, приходящих от узлов обработки:
  - **Bulk Synchronous Parallel (BSP)**. Все узлы обработки данных должны завершить итерацию, переслать обновления на сервер параметров. Все обновления агрегируются (чаще всего – суммируются) и только после этого начинается следующая итерация.
  - **Asynchronous**. При полностью асинхронной схеме, узлы обработки пересылают обновления в произвольном порядке. Эти обновления выполняются сервером параметров также в произвольном порядке. Соответственно, результат чтения параметров в разные моменты времени не детерминирован.
  - **Stale Synchronous Parallel (SSP).** Является компромиссом между полностью синхронным и асинхронным вариантами. Разрешается, чтобы рассинхронизация между самым быстрым и самым медленным узлами обработки данных составляла не более фиксированного числа s итераций. При достижении этого отставания, самый быстрый узел приостанавливается.

Асинхронные и частично асинхронные (SSP) типы консистентности призваны ускорить вычисления и решить “проблему отстающего”.

В статье [26] вариант консистентности Stale Synchronous Parallel (SSP) исследован теоретически. Рассматривается случай распределенной оптимизации суммы выпуклых функций

$$x^* = \operatorname{argmin}_x \frac{1}{T}\sum_{t=1}^{T} f_t(x)$$

удовлетворяющих критерию Липшица с константой L. Оптимизация делается с помощью шагов градиентного спуска $\mathrm{x}_{t+1} = x_t + u_t$, где $u_t = -\eta_t \nabla f_t(\tilde{x}_t)$, $\eta_t = \frac{\sigma}{\sqrt{t}}$, $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$. Тогда имеет место следующая оценка

$$\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathrm{x}}_t) \le f(\mathrm{x}^*) + O\left(\sqrt{\frac{2(s+1)P}{T}}\right)$$

Здесь $\tilde{x}_t$ - это неточный вектор u_t, в котором учтены не все шаги x_t за последние s итераций (узлы успели выполнить разное количество итераций), P - степень параллелизма. Это соответствует консистентности SSP с параметром s. Данные теоретические результаты гарантируют сходимость обучения в распределенной среде при использовании градиентных методов и консистентности SSP.

Наиболее развитым на текущий момент является архитектура сервера параметров (проект почему-то безымянный), описанная в работе [32]. Проект обладает большим числом возможностей и интересных технических решений. Поддерживаются варианты синхронизации BSP, Asynchronous, SSP, причем параметр s в варианте SSP может меняться динамически для увеличения производительности. На сервер параметров отсылаются только достаточно большие обновления, которые проходят через заданный пользователем *фильтр*. Обновления передаются в сжатом виде - только ненулевые компоненты векторов. Сервера параметров хранят пары *(key, value)* с использованием консистентного хеширования (consistent hashing) [48] и репликации по цепочке (chain replication) [53]. Система обладает высокой отказоустойчивостью, подключение новых узлов сервера параметров и узлов обработки данных может осуществляться "на лету". Для уменьшения передачи данных, необходимой для репликации[4], сервера параметров вначале выполняют агрегацию обновлений от нескольких источников, и только после этого - репликацию.

В численных экспериментах показаны примеры успешного распределенного машинного обучения с выборками размером больше 100 ТВ с использованием более 10^4 ядер и до 6000 серверов. Тестировались следующие методы: логистическая регрессия с L1 регуляризацией, LDA с использованием стохастического вариационного вывода и коллапсированного сэмплирования Гиббса, а также CountMin sketch. Время простоя серверов обработки было совсем небольшим: например только 1.7% при обучении логистиче-

[3] Консистентность - согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечивость.

4 Репликация - механизм синхронизации содержимого нескольких копий объекта (например, содержимого базы данных)/

ской регрессии с консистентностью SSP, $s = 16$. Отмечается, что при выборе слишком большого параметра s сходимость замедляется.

Из остальных популярных систем типа "сервер параметров" можно отметить Y!LDA [2, 47], Petuum [18], Distributed Machine Learning Toolkit (DMLTK) [39]. Y!LDA предназначен для распределенного обучение LDA и графических моделей. Особенностью системы Petuum [18] является планировщик STRADS, выполняющий балансировку нагрузки и выделяющий блоки для параллельного выполнения (например, шаги координатного спуска по слабо коррелированным переменным).

К **плюсам** систем машинного обучения, использующих сервер параметров можно отнести отличную производительность и масштабируемость. Как обработка данных, так и хранение параметров модели распределены между узлами кластера. Отказы узлов кластера не являются проблемой. При использования асинхронной и SSP консистентности вычислительные системы не подвержены "проблеме отстающего".

Недостатком такого рода систем является большая нагрузка на сеть при асинхронном и SSP типах консистентности. Данная проблема частично решается хранением локального кэша для часто обновляемых переменных. При асинхронном стохастическом градиентном спуске алгоритм перестает быть математически эквивалентным последовательному исполнению на одной машине. Ускорение является сублинейным, а сходимость гарантируется только при выполнении консистентности SSP. Кроме того, сервер параметров - это специализированная архитектура для машинного обучения и оптимизации, в которой нельзя выполнить полный цикл анализа данных.

4. Spark

Система Spark [56] ориентирована на выполнение итеративных алгоритмов и интерактивный анализ данных с помощью команд функционального программирования. Особенностью этих задач является многократное использование одного *рабочего множества* данных. Основной концепцией Spark является *resilient distributed dataset (RDD)*. RDD - это доступная только для чтения коллекция объектов, хранящаяся распределенно. В отличие от модели Map/Reduce, можно дать указание хранить ее в оперативной памяти. Один RDD можно получить из другого с помощью *преобразований* (команды функционального программирования), которые отложено выполняются только при совершении определенных *действий*. Действия - это операции, требующие возврата данных.

Для каждого RDD система Spark запоминает последовательность команд, с помощью которых данный RDD был получен, например, из файла HDFS. При отказе одного узла и потери части RDD, последовательность команд воспроизводится и эта часть вычисляется заново. Таким образом достигается отказоустойчивость при работе с RDD, кэшированными в оперативной памяти. Обмен данными между параллельными процессами, обрабатывающими RDD на разных узлах также может происходить с помощью *широковещательных переменных* (broadcast variables) и *аккумуляторов* (accumulators). Широковещательные переменные позволяют передать объект всем процессами на разных узлах. Аккумуляторы же позволяют агрегировать данные с помощью коммутативной ассоциативной операции в процессе перебора всех элементов RDD.

Численные эксперименты показывают [56] значительное ускорение итеративных алгоритмов машинного обучения по сравнению с реализацией Map/Reduce на Hadoop.

Spark MLLib [4] - это пакет для машинного обучения, написанный на Scala, в котором используются реализации операций линейной алгеброй на C++. Он поддерживает большое число распространенных методов машинного обучения. Также пакет Spark MLLib содержит функции для предобработки данных и конструирования признаков. Авторы отмечают [4] существенное ускорение решения задачи коллаборативной фильтрации с помощью метода ALS по сравнению с реализацией на Hadoop.

Система SparkNet [40] предназначена для обучения глубоких нейросетей с помощью комбинирования Spark и Caffe [28]. После преобразования обучающей выборки в RDD и ее распределения по кластеру, на каждом разбиении (partition) запускается несколько итераций обучения нейросети в Caffe. Далее каждые T

итераций параметры модели усредняются с помощью встроенной в Spark операции *reduce*. Усредненные параметры передаются на узлы с помощью техники широковещательных переменных. Авторы [40] считают, что основное преимущество SparkNet - это совместимость с существующими кластерами Spark, а также низкие требования к пропускной способности сети, так как усреднение происходит лишь после нескольких итераций обучения. SparkNet является частным случаем BSP модели вычислений. Для решения "проблемы отстающего", предлагается усреднять параметры не каждые T итераций, а после фиксированного интервала времени.

Плюсом реализации методов машинного обучения на Spark является совместимость с широко распространенными кластерами Apache Hadoop. Необходимая предобработка данных и конструирование признаков также могут быть выполнены в Spark, что упрощает работу. Программы, написанные на Spark имеют хорошую отказоустойчивость и масштабируемость. Производительность Spark приложений достаточно хорошая, т.к. система изначально была разработана для итеративных алгоритмов и позволяет хранить данные в оперативной памяти. В тоже время, производительность может страдать из-за "проблемы отстающего", так как Spark приложения не являются полностью асинхронными и реализуют модель вычислений BSP.

Минусом является более низкая по сравнению с реализациями на MPI производительность. Отмечается [34], что реализация обучения логистической регрессии методом TRON быстрее с помощью MPI, чем Spark. В тоже время, реализация на Spark более отказоустойчива. В сравнении [44] отмечается, что реализация ансамблей решающих деревьев в Spark MLLib медленнее и требует больше оперативной памяти, чем H2O и XGBoost, сравнение выполнялось на одной машине.

5. Системы вычислений на графах

Во многих задачах машинного обучения и аналитики данные имеют структуру разреженного графа, например:

- Тематическое моделирование: граф "документы-слова".
- Коллаборативная фильтрация: граф "пользователи-объекты".
- PageRank: граф веб-документов и ссылок между ними.
- Классификация и регрессия: матрицу "примеры-признаки", можно рассматривать как граф. Если элемент матрицы равен нулю, то соответствующего ребра между примером и признаком не будет.
- Нахождение сообществ в социальных сетях: граф связей людей в социальной сети.

Одной из первых моделей вычислений для графов, успешно работающей распределенно была разработанная Google система Pregel [37]. Существует ее свободная реализация Apache Giraph [13, 49]. Впоследствии была предложена система GraphLab [36], а в [35] была описана ее распределенная реализация. В отличие от Pregel, система GraphLab более гибкая и с самого начала направлена на решение задач машинного обучения и анализа данных.

Авторы отмечают три основных свойства GraphLab: *асинхронность*, *динамичность*, *параллельность на графе*. В отличие от Pregel, GraphLab не использует синхронизацию BSP. Более гибкие возможности синхронизации позволяют обойти "проблему отстающего", которая также актуальна при распределенной обработке графов. Объемы вычислений, связанные с каждой вершиной могут существенно отличаться, т.к. степени вершины обычно распределены по степенному закону. Также в GraphLab существует возможность выполнять вычисления параллельно и асинхронно, сохраняя математическую эквивалентность последовательному выполнению. Динамичность означает, что порядок обработки вершин может изменяться динамически в зависимости от заданного пользователем вычисляемого критерия. В модели GraphLab пользователю необходимо задать:

1. Ненаправленный граф $G = (V, D, E)$. Здесь D - это данные, связанные с каждой вершиной и ребром. Направление вершины можно опционально хранить в данных, связанных с ребром. Структура графа не изменяется в процессе работы.

2. Функцию *update*: $(v, \mathcal{S}_v) \to (\mathcal{S}_v, \mathcal{T})$ Функция получает на вход вершину $v \in V$ и данные

вершин и ребер в ее окрестности, $\mathcal{S}_v$ после чего изменяет эти данные. Также функция *update* возвращает набор вершин $\mathcal{T}$ для последующей обработки.

3. Функция *sync*: выполняет агрегацию, вычисляя произвольную коммутативную ассоциативную операцию по всем данным графа. Функция *sync* может использоваться для вычисления критерия останова.

Функция *update* выполняется на вершинах параллельно и асинхронно, поэтому необходимы блокировки данных. GraphLab поддерживает три типа консистентности. *Полная консистентность* (full consistency) выполняет блокировку на чтение и запись всего региона $\mathcal{S}_v$. В этом случае существует математически эквивалентная последовательная обработка вершин. *Консистентность по ребрам* (edge consistency) выполняет блокировку на чтение и запись обрабатываемой вершины и соседних ребер. *Консистентность по вершинам* (vertex consistency) накладывает только блокировку на чтение и запись обрабатываемой вершины.

При распределенном выполнении GraphLab стремится максимально равномерно распределить граф по серверам, так, чтобы число ребер между машинами было минимально. Также каждый сервер хранит информацию о *призраках* (ghosts) - соседних к части графа вершинах и ребрах. Таким образом кешируются данные соседних вершин и ребер с других серверов.

GraphLab поддерживает два алгоритма выполнения и синхронизации задач - *раскраска графа* (chromatic engine) и *распределенные блокировки* (distributed lock engine). В алгоритме выполнения через раскраску графа, вершины графа раскрашиваются в несколько цветов. За одну итерацию обрабатываются все вершины одного цвета. Алгоритм выполнения с использованием распределенных блокировок более гибок и поддерживает параллельное и распределенное выполнение со всеми описанными ранее типами консистентности. Для гарантирования отказоустойчивости используются снэпшоты[5] Ченди-Лэмпорта [9]. Из остальных систем вычислений на графах стоит отметить GraphX [55, 24], реализованную на базе Spark. GraphX сравним по производительности с GraphLab и Apache Giraph для задач обработки графов. В данной системе реализованы популярные методы коллаборативной фильтрации и тематического моделирования. Система GraphX достаточно проста, т.к. нет необходимости в специальных механизмах отказоустойчивости и обмена данными между машинами - это обеспечивает Spark.

Заметим, что системы вычислений на графах Pregel, GraphLab, GraphX достаточно сильно отличаются между собой. Поэтому анализ достоинств и недостатков приведен для наиболее популярной из них - GraphLab.

К **плюсам** систем вычисления на графах стоит отнести более высокую, по сравнению с реализациями на Hadoop/MapReduce производительность. В статье [35] показано, что GraphLab в 20-60 раз быстрее Hadoop/MapReduce для задач коллаборативной фильтрации (ALS), видео ко-сегментации и Named Entiny Recognition (NER). В тоже время, скорость выполнения примерно совпадает с реализациями на MPI.

К **недостаткам** систем вычислений на графах можно отнести их достаточно узкую направленность. Не всякую задачу машинного обучения можно представить в виде алгоритма обработки графа. Полный цикл анализа данных в такой системе не выполним. В работе [18] показано, что сервер параметров Petuum более масштабируем за счет меньшего потребления оперативной памяти, чем GraphLab для задач LASSO и LDA. В работе [25] отмечается, что асинхронный режим GraphLab неэффективен из-за чрезмерного использования сети. А работе [33] отмечается, что GraphLab недостаточно масштабируем по сравнению с сервером параметров из-за механизма снэпшотов.

Таким образом, нельзя сделать вывод о превосходстве GraphLab с точки зрения производительности и масштабируемости над системами, предназначенными для итеративных вычислений в оперативной памяти (MPI, сервера параметров, Spark).

[5] Снэпшот - моментальный снимок, копия состояния системы в определенный момент времени.

6. Искусственные нейронные сети

Искусственные нейронные сети являются стандартом в решении задач компьютерного зрения, распознавания речи, обработки естественных текстов. Используемые для этого архитектуры сети - глубокие, т.е. содержат много скрытых слоев. Увеличение объема обучающей выборки и количества параметров сети имеют существенное значение для качества классификации [12].

Нейросети обычно обучают с помощью градиентных методов. Что касается обучения на одном компьютере, то наиболее эффективным является использование GPU (Graphics Processing Unit). В тоже время, память GPU, как правило, не превышает 2-4 GB, что накладывает ограничение на размер нейросети. Также проблемой является долгое (дни и даже недели) обучение нейросети на одной машине, что замедляет прогресс в разработке новых нейросетевых архитектур. Наиболее простое решение - использование нескольких GPU, соединенных высокопроизводительной шиной на одном узле [15, 29, 3]. Однако пропускная способность шины не позволяет использовать более 4-8 GPU [17]. Поэтому для дальнейшего ускорения необходимо переходить к распределенным архитектурам.

Существует два принципиальных способа распараллеливания обучения нейросети: *по данным* (data parallel) и *по модели* (model parallel). При распараллеливании по данным, машины независимо вычисляют градиенты по частям выборки, после чего эти градиенты суммируются, синхронно или асинхронно. Объем передаваемых между узлами данных пропорционален количеству параметров модели. Вычисление градиентов на GPU происходит достаточно быстро, поэтому параллельные по данным вычисления непрактичны, если передача данных будет занимать больше времени, чем вычисление градиентов. Это имеет место для больших нейросетей при передаче данных по Ethernet. При распараллеливании по модели, необходимо передавать между машинами выходы нейронов для выполнения прямого и обратного прохода. В этом случае, обмен данными может быть существенно сокращен, если архитектура сети - не полносвязная, что типично для задач компьютерного зрения.

Пример системы, параллельной по данным - FireCaffe [27]. Используемая вычислительная архитектура - кластер из 128 узлов, по 1 GPU на каждом узле, соединенные сетью Infiniband. Вычислительные эксперименты показывают, что обучение небольших нейросетей GoogLeNet и NetwokInNetwork ($30*10^6$ и $54*10^6$ параметров) ускоряется в 47 и 39 раз соответственно. Авторы FireCaffe реализовали суммирование градиентов с помощью передачи сообщений по остовному дереву - аналогично функции `AllReduce` стандарта MPI.

Система COTS HPC [17] реализует параллелизм по модели, в качестве примера рассматривается обучение разреженного авто-кодировщика [30]. Максимальный размер модели в вычислительных экспериментах – $11*10^9$ параметров. Обучение проводилось на 16 узлах кластера, соединенных Infiniband, на каждом узле было установлено по 4 GPU с 4 GB памяти в каждом. В результате система COTS HPC показала ускорение обучения в 40 раз относительно реализации на одном GPU, однако эффективная реализация существенно использует информацию о структуре обучаемой нейросети.

Наконец, существуют подходы, использующие одновременно параллельность по данными и по модели: DistBelief [19] и Adam [12]. Данные системы предназначены для запуска на CPU кластере с сетевым интерфейсом Ethernet. Для хранения и синхронизации параметров модели в DistBelief и Adam используются асинхронные сервера параметров, аналогичные рассмотренным в гл. 4. Вычислительные эксперименты с системами DistBelief и Adam проводились с датасетом ImageNet 22k (16 млн. изображений) на кластерах из 120 и 5000 узлов соответственно. Вычислительные эксперименты показывают возможность обучения нейросетей, имеющих до $2*10^9$ параметров. Другие сервера параметров [26, 33] также могут быть использованы для распределенного обучения нейросетей.

Распределенное обучение нейросетей - сложная инженерная задача. Она требует правильной комбинации аппаратного обеспечения, сетевого интерфейса, а также оптимизированного кода, предназначенного, зачастую, для определенной архитектуры сети. Для обучения небольшой

нейросети уместно использовать параллелизм по данным, для большой - по модели.

Заключение

В данном обзоре были рассмотрены современные распределенные вычислительные системы для машинного обучения. Пожалуй, наиболее масштабируемой и производительной системой является архитектура "сервера параметров", для которой описаны успешные примеры обучения с 100 ТВ данных и кластера из 6000 узлов. Сервера параметров были разработаны специально для распределенного машинного обучения и совместимы с большим числом методов. Явным аутсайдером выглядит Map/Reduce, хорошо подходящий для вычисления признаков из "сырых", данных, но не предназначенный изначально для итеративных алгоритмов. Реализации на Map/Reduce используются во многих статьях как простой бейзлайн (эталон) для сравнения, который легко превзойти. С другой стороны, плюсом Map/Reduce является чрезвычайная распространенность поддерживающих его кластеров Hadoop. Остальные системы: MPI, Spark, вычисления на графах - занимают промежуточное положение и обладают своими достоинствами и недостатками. Немного отдельно находятся системы для распределенного обучения нейросетей, т.к. они, как правило, требуют одновременной комбинации программных и аппаратных средств, высокопроизводительных сетевых интерфейсов и кода, оптимизированного под конкретную структуру нейросети.

Практически все системы используют асинхронные вычисления для решения "проблемы отстающего". Однако такие асинхронные алгоритмы не эквивалентны математически последовательным алгоритмам и могут обладать плохой (и недоказанной) сходимостью. Поэтому некоторая синхронизация все-таки необходима: перспективным выглядит консистентность типа Stale Synchronous Parallel (SSP). Еще одним минусом асинхронных схем является недетермированность. Недетерминированность затрудняет исследования, т.к. точность модели становится более шумной, зависящей от распределения вычислений между узлами кластера.

Интересно, что на базе практически любой описанной системы можно реализовать большой спектр популярных методов машинного обучения: обобщенные линейные модели, LDA, коллаборативная фильтрация, бустинг деревьев решений, word2vec, глубокие нейросети. Поэтому при выборе системы зачастую решающим фактором оказывается ее доступность. При условии, что разработчику доступен только один кластер, весомым плюсом является возможность выполнять весь цикл анализа данных в одной системе. С этой точки зрения наиболее интересной является система Spark, сочетающая в себе производительность, масштабируемость и универсальность. Разработка систем для распределенного машинного обучения является активно развивающимся направлением.

Ускорение методов машинного обучения за счет вычислений на кластере позволяет на этапе исследования пробовать больше вариантов моделей и решать практические задачи, в которых возникают большие обучающие выборки. Все это способствует, в конечном счете, научно-техническому прогрессу.

Литература

1. Agarwal, A., Chapelle, O., Dudik, M., and Langford, J. (2011). A reliable effective terascale linear learning system. Technical report. http://arxiv.org/abs/1110.4198.
2. Ahmed, A., Aly, M., Gonzalez, J., Narayanamurthy, S., and Smola, A. (2012). Scalable inference in latent variable models. In International conference on Web search and data mining (WSDM), pages 123-132. https://github.com/shravanmn/Yahoo_LDA.
3. Amazon (2016). Amazon DSSTNE: Deep Scalable Sparse Tensor Network Engine. https://github.com/amznlabs/amazon-dsstne.
4. Apache Foundation (2016). Spark MLLib. http://spark.apache.org/mllib/.
5. Babenko, M. (2013). YT - a new framework for distributed computations. Technical report. https://events.yandex.ru/lib/talks/1091/.
6. Bekkerman, R., Bilenko, M., and Langford, J. (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.
7. Bilenko, M. et al. (2011). Scaling up decision tree ensembles. Technical report. http://hunch.net/~large_scale_survey/TreeEnsembles.pdf.
8. Bu, Y., Howe, B., and Ernst, M. D. (2010). HaLoop: Efficient Iterative Data Processing on Large Clusters. Proceedings of the VLDB Endowment, 3(1-2):285-296.
9. Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS), 3(1):63-75.
10. Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. arXiv preprint arXiv:1603.02754.

11. Chen, W., Wang, Z., and Zhou, J. (2014). Large-scale L-BFGS using MapReduce. In Advances in Neural Information Processing Systems, pages 1332-1340.
12. Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project Adam: Building an efficient and scalable deep learning training system. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 571-582.
13. Ching, A. and Kunz, C. (2011). Giraph: Large-scale graph processing infrastructure on Hadoop. Hadoop Summit, 6(29):2011. http://giraph.apache.org.
14. Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. Advances in neural information processing systems
15. Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classifcation. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence, volume 22, page 1237.
16. Click, C., Lanford, J., Malohlava, M., Parmar, V., and Roark, H. (2015). Gradient boosted machines with H2O. Technical report. http://h2o.gitbooks.io/gbm-with-h2o/.
17. Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y., and Catanzaro, B. (2013). Deep learning with COTS HPC systems.
18. Dai, W., Wei, J., Kim, J. K., Lee, S., Yin, J., Ho, Q., and Xing, E. P. (2013). Petuum: A framework for iterative-convergent distributed ML. http://www.petuum.com.
19. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223-1231.
20. Dean, J. and Ghemawat, S. (2004). MapReduce : Simplifed Data Processing on Large Clusters. In OSDI' 04, San Francisco.
21. Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121-2159.
22. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-h., Qiu, J., and Fox, G. (2010). Twister: a runtime for iterative MapReduce. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10 , pages 810-818.
23. Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation 38 implementation. In Proceedings, 11th European PVM/38 Users' Group Meeting, pages 97-104.
24. Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 599-613.
25. Han, M., Daudjee, K., Ammar, K., Ozsu, M. T., Wang, X., and Jin, T. (2014). An experimental comparison of pregel-like graph processing systems. Proceedings of the VLDB Endowment, 7(12):1047-1058.
26. Ho, Q., Cipar, J., Cui, H., Kim, J. K., Lee, S., Gibbons, P. B., Gibson, G. A., Ganger, G. R., and Xing, E. P. (2013). More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In NIPS' 13.
27. Iandola, F. N., Ashraf, K., Moskewicz, M. W., and Keutzer, K. (2015). Firecaffe: nearlinear acceleration of deep neural network training on compute clusters. arXiv preprint arXiv:1511.00175.
28. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia , pages 675-678. ACM.
29. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classiffcation with deep convolutional neural networks. In NIPS, pages 1097-1105.
30. Le, Q. V., Ranzato, M. A., Devin, M., Corrado, G. S., and Ng, A. Y. (2012). Building High-level Features Using Large Scale Unsupervised Learning.
31. Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). Mining of Massive Datasets. Cambridge University Press.
32. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 583-598. https://github.com/dmlc/ps-lite.
33. Li, M., Zhou, L., Yang, Z., Li, A., and Xia, F. (2013). Parameter Server for Distributed Machine Learning. Big Learning Workshop, pages 1-10.
34. Lin, C.-Y., Tsai, C.-H., Lee, C.-P., and Lin, C.-J. (2014). Large-scale logistic regression and linear support vector machines using spark. In Big Data (Big Data), 2014 IEEE International Conference on, pages 519-528. IEEE.
35. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. (2012). Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716-727. http://graphlab.org.
36. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. In UAI' 10, Cataline Island, California.
37. Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data , pages 135-146. ACM.
38. Message Passing Interface Forum (2012). 38: A message-passing interface standard, version 3.0. http://www.mpi-forum.org.
39. Microsoft (2016). Distributed machine learning toolkit. http://www.dmtlk.io.
40. Moritz, P., Nishihara, R., Stoica, I., and Jordan, M. I. (2015). SparkNet: Training Deep Networks in Spark. arXiv preprint arXiv:1511.06051. https://github.com/amplab/SparkNet.
41. Niu, F., Recht, B., Re, C., and Wright, S. J. (2011). HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. page 21.

42. Nokia Research Center (2008). Disco project. http://discoproject.org/.
43. Pacheco, P. (2011). An introduction to parallel programming. Elsevier.
44. Pafka, S. (2016). Simple/limited/incomplete benchmark for scalability, speed and accuracy of machine learning libraries for classiffcation. https://github.com/szilard/benchm-ml.
45. Panda, B., Herbach, J. S., Basu, S., and Bayardo, R. J. (2009). Planet: massively parallel learning of tree ensembles with mapreduce. Proceedings of the VLDB Endowment, 2(2):1426-1437.
46. Rabit developers (2016). Rabit: Reliable allreduce and broadcast interface. https://github.com/dmlc/rabit.
47. Smola, A. and Narayanamurthy, S. (2010). An architecture for parallel topic models. Proceedings of the VLDB Endowment, 3(1-2): pp. 703-710.
48. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 31(4):149-160.
49. Stylianou, M. (2014). Apache Giraph for applications in Machine Learning & Recommendation Systems. http://people.inf.ethz.ch/jaggim/meetup/5/slides/ML-meetup-5-Stylianou-Giraph.pdf.
50. Svore, K. M. and Burges, C. (2011). Large-scale learning to rank using boosted decision trees. Scaling Up Machine Learning: Parallel and Distributed Approaches.
51. Trofimov, I. and Genkin, A. (2014). Distributed coordinate descent for l1-regularized logistic regression. arXiv preprint:1411.6520
52. Tyree, S., Weinberger, K. Q., Agrawal, K., and Paykin, J. (2011). Parallel boosted regression trees for web search ranking. In Proceedings of the 20th international conference on World wide web, pages 387-396. ACM.
53. Van Renesse, R. and Schneider, F. B. (2004). Chain replication for supporting high throughput and availability. In OSDI, volume 4, pages 91-104.
54. White, T. (2012). Hadoop: The definitive guide. "O'Reilly Media, Inc.".
55. Xin, R. S., Gonzalez, J. E., Franklin, M. J., and Stoica, I. (2013). GraphX: A resilient distributed graph system on Spark. First Intern. Workshop on Graph Data Management Experiences and Systems.
56. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark : Cluster Computing with Working Sets. HotCloud'10 Proceedings of the 2nd USENIXconference on Hot topics in cloud computing, page 10.
57. Zhang, W., Gupta, S., Lian, X., and Liu, J. (2015). Staleness-aware async-sgd for distributed deep learning. arXiv preprint arXiv:1511.05950.
58. Zhuang, Y., Chin, W.-S., Juan, Y.-C., and Lin, C.-J. (2015). Distributed newton methods for regularized logistic regression. In Advances in KDD.

Трофимов Илья Егорович. Ведущий аналитик ООО “Яндекс Маркет Лаб”. Окончил Московский государственный университет им. М.В. Ломоносова в 2006 году. Количество печатных работ: 8. Область научных интересов: машинное обучение, распределенные системы, онлайн реклама, рекомендательные системы. E-mail: trofim@yandex-team.ru

Distributed systems for machine learning

Trofimov Ilya

Abstract. Machine learning is a rapidly developing area of research. Many machine learning and data science applications face very large datasets. These datasets are hard to process on a single computer or this processing will be very time consuming. Using only a subsample of a dataset typically leads to the worse quality of model’s predictions. Distributed computational systems are used to solve this problem. The most popular approaches for developing software of such systems include the following computational models: Map/Reduce, Spark, graph computational models and parameter server architecture. Current paper is a review of such systems with analysis of their advantages and disadvantages regarding to machine learning applications. Systems for training artificial neural networks are discussed separately.
Keywords: Machine learning, data science, big data, distributed systems.

References

1. Agarwal, A., Chapelle, O., Dudik, M., and Langford, J. (2011). A reliable effective terascale linear learning system. Technical report. http://arxiv.org/abs/1110.4198.
2. Ahmed, A., Aly, M., Gonzalez, J., Narayanamurthy, S., and Smola, A. (2012). Scalable inference in latent variable models. In International conference on Web search and data mining (WSDM), pages 123-132. https://github.com/shravanmn/Yahoo_LDA.
3. Amazon (2016). Amazon DSSTNE: Deep Scalable Sparse Tensor Network Engine. https://github.com/amznlabs/amazon-dsstne.

4. Apache Foundation (2016). Spark MLLib. http://spark.apache.org/mllib/.
5. Babenko, M. (2013). YT - a new framework for distributed computations. Technical report. https://events.yandex.ru/lib/talks/1091/.
6. Bekkerman, R., Bilenko, M., and Langford, J. (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.
7. Bilenko, M. et al. (2011). Scaling up decision tree ensembles. Technical report. http://hunch.net/~large_scale_survey/TreeEnsembles.pdf.
8. Bu, Y., Howe, B., and Ernst, M. D. (2010). HaLoop: E_cient Iterative Data Processing on Large Clusters. Proceedings of the VLDB Endowment, 3(1-2):285-296.
9. Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS), 3(1):63-75.
10. Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. arXiv preprint arXiv:1603.02754.
11. Chen, W., Wang, Z., and Zhou, J. (2014). Large-scale L-BFGS using MapReduce. In Advances in Neural Information Processing Systems, pages 1332-1340.
12. Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project Adam: Building an e_cient and scalable deep learning training system. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 571-582.
13. Ching, A. and Kunz, C. (2011). Giraph: Large-scale graph processing infrastructure on Hadoop. Hadoop Summit, 6(29):2011. http://giraph.apache.org.
14. Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. Advances in neural information processing systems.
15. Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classifcation. In IJCAI Proceedings-International Joint Conference on Arti_cial Intelligence, volume 22, page 1237.
16. Click, C., Lanford, J., Malohlava, M., Parmar, V., and Roark, H. (2015). Gradient boosted machines with H2O. Technical report. http://h2o.gitbooks.io/gbm-with-h2o/.
17. Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y., and Catanzaro, B. (2013). Deep learning with COTS HPC systems.
18. Dai, W., Wei, J., Kim, J. K., Lee, S., Yin, J., Ho, Q., and Xing, E. P. (2013). Petuum: A framework for iterative-convergent distributed ML. http://www.petuum.com.
19. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223-1231.
20. Dean, J. and Ghemawat, S. (2004). MapReduce : Simplifed Data Processing on Large Clusters. In OSDI' 04, San Francisco.
21. Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121-2159.
22. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-h., Qiu, J., and Fox, G. (2010). Twister: a runtime for iterative MapReduce. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10 , pages 810-818.
23. Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation 38 implementation. In Proceedings, 11th European PVM/38 Users' Group Meeting, pages 97-104.
24. Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 599-613.
25. Han, M., Daudjee, K., Ammar, K., Ozsu, M. T., Wang, X., and Jin, T. (2014). An experimental comparison of pregel-like graph processing systems. Proceedings of the VLDB Endowment, 7(12):1047-1058.
26. Ho, Q., Cipar, J., Cui, H., Kim, J. K., Lee, S., Gibbons, P. B., Gibson, G. A., Ganger, G. R., and Xing, E. P. (2013). More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In NIPS' 13.
27. Iandola, F. N., Ashraf, K., Moskewicz, M. W., and Keutzer, K. (2015). Fireca_e: nearlinear acceleration of deep neural network training on compute clusters. arXiv preprint arXiv:1511.00175.
28. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia , pages 675-678. ACM.
29. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classi_cation with deep convolutional neural networks. In NIPS, pages 1097-1105.
30. Le, Q. V., Ranzato, M. A., Devin, M., Corrado, G. S., and Ng, A. Y. (2012). Building High-level Features Using Large Scale Unsupervised Learning.
31. Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). Mining of Massive Datasets. Cambridge University Press.
32. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. (2014). Scaling distributed machine learning with the parameter
server. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 583_598. https://github.com/dmlc/ps-lite.

33. Li, M., Zhou, L., Yang, Z., Li, A., and Xia, F. (2013). Parameter Server for Distributed Machine Learning. Big Learning Workshop, pages 1-10.
34. Lin, C.-Y., Tsai, C.-H., Lee, C.-P., and Lin, C.-J. (2014). Large-scale logistic regression and linear support vector machines using spark. In Big Data (Big Data), 2014 IEEE International Conference on, pages 519-528. IEEE.
35. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. (2012). Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716_727. http://graphlab.org.
36. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. In UAI' 10, Cataline Island, California.
37. Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 135-146. ACM.
38. Message Passing Interface Forum (2012). 38: A message-passing interface standard, version 3.0. http://www.mpi-forum.org.
39. Microsoft (2016). Distributed machine learning toolkit. http://www.dmtlk.io.
40. Moritz, P., Nishihara, R., Stoica, I., and Jordan, M. I. (2015). SparkNet: Training Deep Networks in Spark. arXiv preprint arXiv:1511.06051. https://github.com/amplab/SparkNet.
41. Niu, F., Recht, B., Re, C., and Wright, S. J. (2011). HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. page 21.
42. Nokia Research Center (2008). Disco project. http://discoproject.org/.
43. Pacheco, P. (2011). An introduction to parallel programming. Elsevier.
44. Pafka, S. (2016). Simple/limited/incomplete benchmark for scalability, speed and accuracy of machine learning libraries for classi_cation. https://github.com/szilard/benchm-ml.
45. Panda, B., Herbach, J. S., Basu, S., and Bayardo, R. J. (2009). Planet: massively parallel learning of tree ensembles with mapreduce. Proceedings of the VLDB Endowment, 2(2):1426-1437.
46. Rabit developers (2016). Rabit: Reliable allreduce and broadcast interface. https://github.com/dmlc/rabit.
47. Smola, A. and Narayanamurthy, S. (2010). An architecture for parallel topic models. Proceedings of the VLDB Endowment, 3(1-2): pp. 703-710
48. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 31(4):149-160.
49. Stylianou, M. (2014). Apache Giraph for applications in Machine Learning & Recommendation Systems. http://people.inf.ethz.ch/jaggim/meetup/5/slides/ML-meetup-5-Stylianou-Giraph.pdf.
50. Svore, K. M. and Burges, C. (2011). Large-scale learning to rank using boosted decision trees. Scaling Up Machine Learning: Parallel and Distributed Approaches.
51. Trofimov, I. and Genkin, A. (2014). Distributed coordinate descent for l1-regularized logistic regression. arXiv preprint:1411.6520
52. Tyree, S., Weinberger, K. Q., Agrawal, K., and Paykin, J. (2011). Parallel boosted regression trees for web search ranking. In Proceedings of the 20th international conference on World wide web, pages 387-396. ACM.
53. Van Renesse, R. and Schneider, F. B. (2004). Chain replication for supporting high throughput and availability. In OSDI, volume 4, pages 91-104.
54. White, T. (2012). Hadoop: The definitive guide. "O'Reilly Media, Inc.".
55. Xin, R. S., Gonzalez, J. E., Franklin, M. J., and Stoica, I. (2013). GraphX: A resilient distributed graph system on Spark. First Intern. Workshop on Graph Data Management Experiences and Systems.
56. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark : Cluster Computing with Working Sets. HotCloud'10 Proceedings of the 2nd USENIXconference on Hot topics in cloud computing, page 10.
57. Zhang, W., Gupta, S., Lian, X., and Liu, J. (2015). Staleness-aware async-sgd for distributed deep learning. arXiv preprint arXiv:1511.05950.
58. Zhuang, Y., Chin, W.-S., Juan, Y.-C., and Lin, C.-J. (2015). Distributed newton methods for regularized logistic regression. In Advances in KDD.

Trofimov Ilya. Principal analyst in Yandex.Market. Received M.Sc. in theoretical physics from Moscow State University in 2006. Fields of research: Large-scale machine learning, Online advertising, Recommender systems. 8 publications.
E-mail: trofim@yandex-team.ru