Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Системного Программирования

Толстопятов Всеволод Андреевич

Развитие эволюционного программирования в Apache Spark

Курсовая работа

Научный руководитель:
Старший разработчик программного обеспечени я в Alpine Data Labs Пахомов Е. А.

Санкт-Петербург
2015

Оглавление

Введение

За последние несколько лет в высокопроизводительных вычислительных системах произошли значительные изменения. Сегодня даже небольшие бизнес-приложения генерируют огромное количество данных, не говоря уже о больших и сложных системах из таких сфер как медицина, анализ и моделирования физических процессов, криптография, и.т.д. Так как масштабы данных и сложность современных алгоритмов анализа данных и машинного обучения растут несоизмеримо быстрее, чем вычислительные мощности компьютеров, то неизбежно появляются библиотеки и экосистемы для распределённого анализа данных, такие как Apache Hadoop, Apache Spart, Google MapReduce и другие.

В данный момент требования к таким инструментам очень высокие: от них требуется возможность быстро обрабатывать потоковые данные, обучаться на данных, которые не помещаются в оперативную память современных компьютеров, в реальном времени реагировать на изменения данных и/или алгоритмов и выдавать результат. Одним из таких инструментов является Apache Spark, который благодаря своей стратегии обработки данных большую часть времени держит их в оперативной памяти и не обменивается данными с соседними узлами кластера, что позволяет ему выигрывать по скорости обработки данных даже у зарекомендовавшего себя в бизнес-приложениях фреймворка Apache Hadoop.

Apache Spark содержит в себе большое количество различных алгоритмов машинного обучения для всевозможных задач, но класс алгоритмов эволюционного программирования в нем не представлен. В то же время эволюционное программирование решает обширный класс задач и может конкурировать с более специализированными алгоритмами по точности результата и скорости. На сегодняшний день существует несколько библиотек для вычислений, основанных на эволюционном подходе, но ни одна из них не интегрирована с Apache Spark. Таким образом, целью моей работы является анализ спектра задач, которые решаются средствами эволюционного программирования и написание необходимого функционала и API для обобщённого распределённого эволюционного программирования с последующей отправкой кода на review в Apache Spark в качестве экспериментального API в под-библиотеку mllib.

0.1. Обзор Apache Spark

Apache Spark - библиотека с открытым исходным кодом для распределённых вычислений и анализа данных, разработанная в Калифорнийском университете в Беркли как более быстрая и удобная замена Apache Hadoop. На данный момент Apache

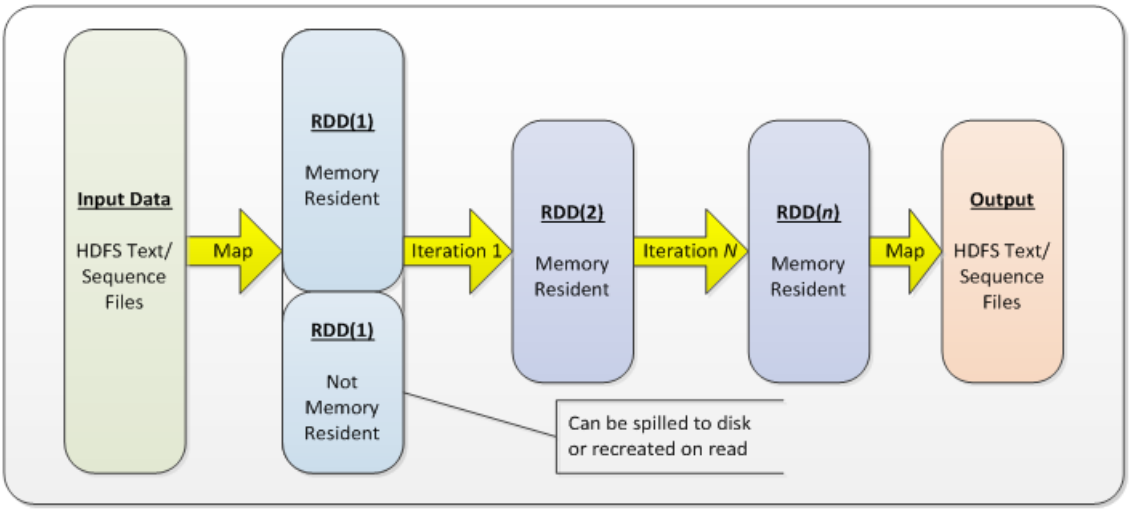

Рис. 1: Схема обработки данных

Spark является Apache Top Level Project и обладает самой быстрорастущей кодовой базой среди всех продуктов Apache. API Apache Spark поддерживается для таких языков как Java, Scala и Python. Так же для Scala и Python существует поддержка интерактивной консоли REPL.

Основным понятием в Apache Spark является RDD (Resilient Distributed Dataset), который представляет собой коллекцию, над которой можно делать преобразования двух типов (и, соответственно, вся работа с этими структурами заключается в последовательности этих двух действий): трансформации - распределённые аналоги широко распространённых преобразований коллекций в различных языках программирования, такие как .map(), .filter(), .distinct(), и действия, которые приводят к материализации коллекции: .save(), .reduce(), .zip(). Не смотря на то, что для программиста RDD является локальной коллекцией, Apache Spark обеспечивает то, что её содержимое будет равномерно распределено по всем машинам в вычислительном кластере и все операции будут производиться в памяти до тех пор, пока не будет вызвано одно из материализующих действий.

1. Постановка задачи

Моими задачами является:

- Изучения эволюционного программирования и классов задач, которые оно эффективно решает
- Изучить существующие инструменты, которые решают такие же или похожие задачи
- Реализация основных примитивов (строительных блоков) эволюционного программирования с использованием инфраструктуры Apache Spark
- Проработка публичного API для наиболее быстрой и выразительной работы с примитивами эволюционного программирования для быстрого построения распределённых эволюционных и генетических алгоритмов
- Сравнение моего решения с уже существующими библиотеками
- Код должен соответствовать стандартам Apache Spark: проходить автоматические проверки стиля программирования перед сборкой, покрытие исходного кода тестами должно стремиться к ста процентам.
- После предыдущих пунктов желательно отправить код на ревизию автору подбиблиотеки Apache Spark mllib с целью добавления созданного решения в mllib как экспериментальной функциональности

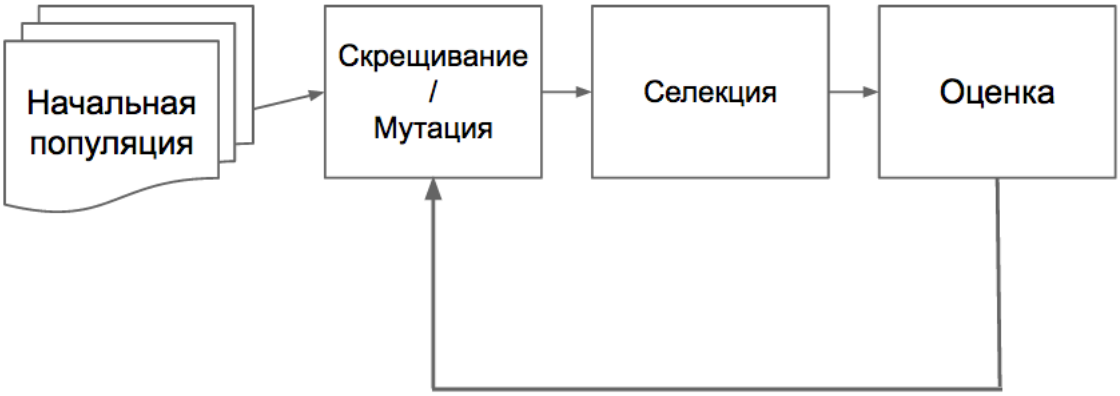

Рис. 2: Схема эволюции

2. Обзор эволюционного программирования

2.1. Эволюционное программирование

Эволюционные программирование – это метод решения задач, основанный на эволюции популяции решений, в процессе которой определяется оптимальное значение оценочной функции. Такой подход в основном применяется для оптимизации пло- хо определённых функций. Одна из основных идей эволюционного программирования - принцип естественного отбора, который заключается в том, что наиболее приспособленные особи дают потомство, которое формирует последующие поколения Утверждается, что каждое следующее поколение является более приспособленным для решения поставленной задачи, чем предыдущее. В настоящее время эволюционное программирование применяется в различных сферах: оптимизация функций, приближённое решение NP-полных и NP-сложных задач, построение конечных автоматов для АСУ и т.д.

2.2. Схема эволюционного программирования

Процесс эволюционного программирования, описанный на схеме выше, в общем случае работает так:

1. Случайным или псевдослучайным образом генерируется начальное поколение
2. Решения из предыдущего семейства скрещиваются с целью получения нового множества решений (особей)
3. Новое поколение оценивается с помощью оценочной функции (функции приспособленности)

4. Исходя из приспособленности особей одним из алгоритмов селекции выбирается их подмножество, которое перейдёт в следующее поколение

5. Проверяется один или несколько критериев остановки

   (a) Один или несколько индивидов достигли желаемого значения оценочной функции

   (b) Процесс эволюции остановился или зациклился

   (c) Прошёл заданный промежуток времени

3. Обзор существующих решений

3.1. The Watchmaker Framework

The Watchmaker Framework - расширяемая высокопроизводительная объектно-ориентированная библиотека для написания платформонезависимых эволюционных и генетических алгоритмов на Java, которая изначально была частью Apache Mahout, библиотеки алгоритмов машинного обучения и обработки данных для Apache Hadoop. Данная библиотека предоставляет типобезопасную эволюцию произвольных типов с помощью обобщенного интерфейса. The Watchmaker Framework является проектом с открытым исходным кодом и распространяется под лицензией Apache Software Licence, Version 2.

Достоинства данной библиотеки:

- Возможность использовать параллелизм для улучшения производительности на многоядерных машинах;
- Обобщённый интерфейс, который позволяет участвовать в эволюционном процессе экземплярам классов, которые не реализуют никакие специфичные интерфейсы;
- Большой выбор строительных блоков: в наличии имеется большое количество алгоритмов скрещивания, отбора и мутации
- Различные подходы к эволюции: модель островов, $\lambda+\mu$ стратегия, модель устойчивого состояния
- Гибкий интерфейс позволяет комбинировать существующие решения и дополнять их своими
- Дополнения к библиотеке для графических интерфейсов Swing позволяют легко визуализировать процесс эволюции

Недостатки:

- Слишком громоздкий интерфейс: попытка создать универсальный интерфейс с полной безопасностью типов на языке без вывода типов привёл к излишней сложности интерфейса и невозможности быстро прототипировать эволюционные алгоритмы;
- Из-за предыдущего пункта данная библиотека была вынесена за пределы Apache Mahout и распространяется отдельно
- Написана на устаревшей Java 6, которая больше официально не поддерживается компанией Oracle

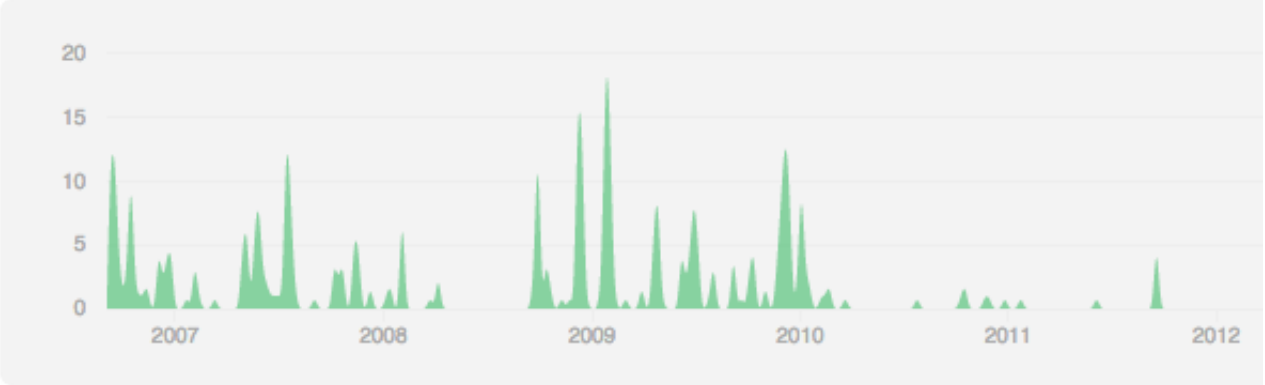

Рис. 3: Частота изменений в Watchmaker Framework

- Несмотря на заявленную поддержку многопроцессорного окружения, уровень параллелизма поддерживается только один, по применению оценочной функции, что затрудняет использование данной библиотеки в распределённом окружении;
- Последнее исправление данной библиотеке было произведено два года назад

3.2. ECJ

ECJ - библиотека для эволюционных вычислений, разработанная в Университете Джорджа Мейсона как академический исследовательский проект в области эволюционного программирования. Достоинства данной библиотеки:

- Написанный код покрывает очень большую область задач, начиная от классического эволюционного и генетического программирования и заканчивая алгоритмами мета-эвристики и многокритериальной оптимизацией
- Поддержка иерархических конфигурационных файлов
- Абстракции для использования различных форм эволюционного программирования
- Платформонезависимое логгирование
- Графический интерфейс с построением различных диаграмм

Недостатки:

- В процессе разработки использовались устаревшие на данный момент технологии (Java 1.4)
- Плохое качество кода;
- Невозможность интеграции с современными решениями для распределённой обработки данных

- Отсутствие системы контроля версий, исходный код выкладывался архивами на сайте университета
- Поддержка прекращена более восьми лет назад;

3.3. Вывод

В силу невозможности использования существующих решений в совокупности с Apache Spark было решено написать своё решение в рамках пакета mllib. Решение должно содержать преимущества вышеназванных решений, но избежать их недостатков. Также необходимо исследовать и реализовать различные подходы к эволюционному процессу с точки зрения распределённых вычислений.

4. Актуальность проблемы

4.1. Фундаментальная теорема о шаблонах

Понятие шаблона:
Шаблоном называется подмножество множества всех возможных генотипов (решений), возможных в данной популяции, заданное в виде хромосомы с фиксированными значениями некоторых битов. Остальные биты могут принимать любые значения, образуя примеры шаблона. Так, примерами схемы 00*1* являются хромосомы 00010, 00011, 00110, 00111, 00010. Оценочная функция шаблона — это среднее значение функции пригодности всех её индивидов.

Фундаментальная теорема показывает экспоненциальное распространение хорошо приспособленных шаблонов с оценочной функцией выше, чем в среднем по популяции.

Это также можно выразить неравенством:
$N(h, t+1) \geq N(h, t)\frac{f(h,t)}{f(t)}[1-p]$.
Где $N(h,t)$ количество шаблонов схемы h в поколении с номером t, $f(h,t)$ - оценочная функция шаблона, $f(t)$ - среднее значение оценочной функции по поколению t, а p - вероятность исчезновения шаблона из популяции.

4.2. Следствия из теоремы

- Линейное увеличение размера популяции экспоненциально ускоряет поиск решения задачи, но требует более осторожной настройки процесса эволюции
- Фактически единицей поколения становится не индивид, а шаблон
- Неправильно выбранные параметры эволюции уничтожают перспективные шаблоны решений

Из перечисленных следствий данной теоремы можно сделать вывод, что настройка процесса эволюции для оптимального результата - итеративный процесс, где на каждой итерации проверяется та или иная гипотеза относительно текущей задачи. При этом часто эти гипотезы могут быть не связаны, что приводит к мысли о том, что проверка таких гипотез - универсальная единица параллелизации и проверять такие гипотезы можно независимо, с линейным ускорением от количества вычислительных узлов. Именно для таких задач и был создан Apache Spark.

5. Реализация

5.1. Инструменты

Apache Spark поддерживает интерфейсы на таких языках как Scala, Java и Python. Язык Python был сразу отброшен, так как это не основной язык платформы и реализация большой инфраструктуры на нем не соответствует стандартам Apache Spark, который внутри использует в основном Scala и, в некоторых случаях, Java. Несмотря на близкое знакомство с языком программирования Java, в качестве основного инструмента была выбрана Scala, так как Java не обладает достаточной выразительностью и является императивным языком, когда как Apache Spark предоставляет функциональный подход к обработке данных.

Apache Spark использует Scala версии 2.10, но в последних версиях появилась экспериментальная поддержка версии 2.11, поэтому решено было использовать именно её, как последнюю.

Также, так как автор данной курсовой и его руководитель на момент начала работы являлись сотрудниками компании Яндекс, была использована вычислительная сеть Tasmania, состоящая из двухсот вычислительных узлов с возможностью быстрого обновления любых компонент на них, в которые в том числе входил Apache Spark. Однако в рамках данной работы из-за сложностей с развёртыванием полноценной вычислительной сети с обновлёнными зависимостями была использована лишь её часть.

5.2. Линейная регрессия и метод роя частиц

В рамках изучения задач эволюционного программирования было выбрано несколько типов задач, основываясь на которых строился функционал и реализация библиотеки для распределённых эволюционных вычислений. В моем случае такими задачами стали задача линейной регрессии и задача поиска минимума или максимума функции методом роя частиц.

Задача линейной регрессии состоит в в восстановлении линейной зависимости между N независимыми переменными, при наличии зависимой переменной Y.
В качестве индивида популяции для решения данной задачи было выбрано двоичное дерево разбора арифметических выражений, промежуточные вершины которого - операции, такие как умножение, сложение и вычитание, которые производились над детьми данных вершин, а лист может быть либо одной из независимых переменных, либо константой. В качестве оператора скрещивания был выбран самый простой подход: каждого из двух индивидов нужно разбить на два поддерева по какой-либо

случайной вершине, а потом скрестить эти поддеревья между друг другом.
Путём множественных экспериментов в рамках реализации библиотеки в качестве алгоритма селекции для больших размеров начальной популяции (более тысячи индивидов) был выбран алгоритм Roulette Wheel Selection (известный также как Fitness Proportionate Selection). Однако в случае небольших популяций данный алгоритм слишком часто выбирал наилучшее локальное решение и распространял его, что не всегда являлось оптимальным результатом. Поэтому для небольших размеров начальной популяции было решено использовать алгоритм Stochastic Universal Sampling (SUS), который благодаря разбиению популяции на интервалы позволяет даже потенциально неприспособленным индивидом быть выбранным для следующего поколения.

Оптимизация значения функции методом роя частиц. Метод роя частиц - метод численной оптимизации, для использования которого не требуется знать градиент оптимизируемой функции, единственное требование - непрерывность по всем переменным. Метод моделирует систему, где индивиды-частицы двигаются к оптимальным решениям, обмениваясь при этом информацией с соседями.

Каждая частица характеризуется координатами в пространстве решений, а также вектором скорости перемещения. Оба этих параметра выбираются случайным образом на этапе генерации начального поколения. Кроме того, каждая частица хранит координаты лучшего из найденных ей решений, а также лучшее из пройденных всеми частицами решений – этим имитируется мгновенный обмен информацией между роем. На каждой итерации алгоритма направление и длина вектора скорости каждой из частиц изменяются в соответствие со сведениями о найденных оптимумах и на основе некоторой случайной величины. В данном случае в алгоритм скрещивания комбинируется с алгоритмом мутации: в качестве скрещивания берётся среднее двух индивидов, а в качестве мутации - произвольное незначительное изменение скорости. В данном случае алгоритм селекции подходит двоичный алгоритм турнирного отбора, описанный в статье The LifeCycle Model. Для случая распределённых вычислений разумно разбить область определения функции на примерно одинаковые области, где количество областей должно совпадать с количеством вычислительных узлов и для каждой из таких областей генерировать начальную популяцию частиц. Тогда в каждой такой области будет найдено локальное оптимальное значение и лишь в конце выбрано самое оптимальное. Основное преимущество такого подхода - минимизация обмена информацией между вычислительными узлами.

5.3. Перебор с отсечениями

Также, в рамках изучения спектра задач, для которых подходит эволюционное программирование, было выяснено, что для многих NP-полных задач, таких как за-

дача о выполнимости булевых формул и задача коммивояжёра, а также других задач, требующих перебора всего множества решений с возможными отсечениями, алгоритмы эволюционного программирования быстро находят локальные оптимальные решения. Единственная проблема для таких задач - представление решения в виде индивида, которого можно скрестить с другим индивидом и представление оценочной функции для них. Благодаря этому было решено добавить в библиотеку возможно параллелизации по всем возможным параметрам.

5.4. Реализация API строительных блоков

В рамках данной работы были реализованы основные алгоритмы селекции, а также скрещивания геномов и условий остановки.
Для наибольшей производительности, а также для тестирования, было решено избавиться от чисто функционального подхода, поэтому строительные блоки инкапсулируют изменяемое состояние, которое, однако, незаметно коду, который работает с данными блоками. Интерфейс был спроектирован как типобезопасный, без жесткой привязки индивидов к библиотечным интерфейсам, поэтому в качестве индивидов-решений можно использовать любые объекты, даже если пользователи не имеют прямого контроля над ними (например, это объекты классов из сторонней библиотеки).

5.5. Реализация различных уровней параллелизма механизма эволюции

Несмотря на то, что Apache Spark в качестве единицы параллелизации (RDD) обычно использует данные, которые являются обучающей выборкой для выбранных алгоритмов, а процесс эволюции в большинстве своем не содержит никаких дополнительных данных, кроме самой популяции, сложностей с параллелизацией на различных уровнях не было, так как популяции можно интерпретировать и как данные, и как алгоритм, который обучается сам на себе с помощью оценочной функции.

В рамках данной работы были реализованы различные уровни параллелизма, а именно:

- Параллелизм на уровне обучающей выборки: стандартный параллелизм для алгоритмов в Apache Spark, наш случай не является исключением
- Параллелизм на уровне алгоритма селекции: каждый узел вычислительной сети производит вычисления со своим алгоритмом.
- Параллелизм на уровне алгоритма скрещивания: аналогично предыдущему пункту

- Параллелизм на уровне пары алгоритм селекции-алгоритм скрещивания. Так как в большинстве случаев параллелизация на уровне алгоритмов необходима для проверки гипотез, то разумно было предоставить возможность использовать различные пары алгоритмов, а в силу большого количества вычислительных узлов можно избежать экспоненциального взрыва времени эволюции со всеми парами таких алгоритмов.

5.6. Реализация островной модели

В рамках данной работы также был рассмотрен популярный подход островной модели, в которой популяция разбивается на множество подпопуляций, которые развиваются независимо и раз в небольшой промежуток времени обмениваются лучшими индивидами. Так как в распределённых вычислительных сетях обмен данными стоит очень дорого, то было проделано много технической работы для частичного преодоления данных ограничений. В результате появилось две модели островных вычислений: В первой модели раз в несколько десятков тысяч поколений "острова" перемешиваются, что позволяет избегать быстрой локальной сходимости каждого из независимых островов.

Во второй модели в одном узле агрегируются лучшие индивиды каждой из независимых популяций и на их основе вносятся изменения в каждый из островов.

Заключение

В результате данной работы было сделано следующее:

- Изучен спектр задач, которые решаются эволюционным программированием
- Произведён обзор и сравнение существующих библиотек для эволюционного программирования
- Был реализован основной интерфейс для работы с эволюционным программированием в экосистеме Apache Spark
- Были реализованы два алгоритма с оптимальным подбором параметров для решения классических задач машинного обучения
- Были исследованы и реализованы различные подходы к параллелизации процесса эволюции
- Были начаты переговоры по поводу попадания данных изменений в новые версии Apache Spark

Список литературы

- John R. Koza, "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems", Stanford University Computer Science, 1990
- John R. Koza, "Genetic Programming II: Automatic Discovery of Reusable Programs". Cambridge Massachusetts: MIT Press, may 1994.
- Manish Sarkar, B. Yegnanarayana, Deepak Khemani, "A clustering algorithm using an evolutionary programming-based approach", 1997
- Thiemo Krink, "The LifeCycle model: Combining Particle Swarm Optimisation, Genetic Algorithms and HillClimbers"
- Martin Odersky, Lex Spoon, Bill Venners, "Programming in Scala"
- Marko Bonaći, Petar Zečević, "Spark in Action", 2015, early access edition
- Stephanie Forrest, Westley Weimer, ThanhVu Nguyen, Claire Le Goues, "A Genetic Programming Approach to Automated Software Repair"
- Tuan Hao Hoang, Daryl Essam, Xuan Hoai Nguyen, "Developmental Evaluation in Genetic Programming: A Position Paper"