электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 05, май 2016

УДК 004.75+004.855.5

Применение Apache Spark в задачах интеллектуального обучения в гибридных интеллектуальных информационных системах

Леонтьев А.В., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

научный руководитель: Гапанюк Ю.Е., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»
gapiu@bmstu.ru

Введение

Цель работы: изучить возможности и оценить применимость Apache Spark для решения задач, связанных с гибридными интеллектуальными информационными системами.

Apache Spark (от англ. spark — искра, вспышка) — каркас приложений с открытым исходным кодом. Библиотека предназначена для создания распределённых приложений, однако в отличие от Hadoop, реализующего двухуровневую концепцию MapReduce с дисковым хранилищем, использует примитивы в оперативной памяти, благодаря чему позволяет получать большую производительность в определённых приложениях.

Spark базируется на таком представлении данных как RDD — Resilient Distributed Dataset (эластичный распределённый набор данных). Алгоритм обработки данных состоит из множества «ленивых» преобразований над RDD (например, операция map), завершающийся каким-то «неленивым» действием (например, reduce). [1]

Возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Проект предоставляет программные интерфейсы для языков **Java**, **Scala**, **R** и **Python**. В нижеследующем изложении используется программный интерфейс для языка Python - библиотека **PySpark**.

Всё исследование проводилось на персональном компьютере со следующими характеристиками:

- Intel Core i7 4.3 GHz (4 ядра, 8 потоков с учётом гипертрединга);
- 16 GB RAM DDR3;
- NVidia GeForce GTX 670 (1344 ядра, 980 MHz, 2 GB GDDR5).

Производительность. Число Pi

Эта проблема была выбрана как простой пример задачи, хорошо подходящей для распределённых вычислений.

Постановка задачи: найти значение числа Pi программно с произвольной фиксированной точностью.

Алгоритм решения: в цикле генерируются точки — пары случайных чисел (x, y). Эти точки попадают в квадрат со стороной R (этот квадрат отмечен линией из точек на рис. 1). Для каждой точки проверяется, попала ли она в окружность радиуса R или нет. Ясно, что отношение количества точек, попавших в окружность к общему количеству сгенерированных точек будет равно отношению площадей окружности с радиусом R к площади квадрата со стороной 2R, в которую она вписана.

Площади квадрата и круга равны соответственно:

$$S_{sq} = (2R)^2 = 4R^2$$

$$S_c = \pi R^2$$

Отсюда получаем, что число π равно:

$$\pi = 4\frac{S_c}{S_{sq}}$$

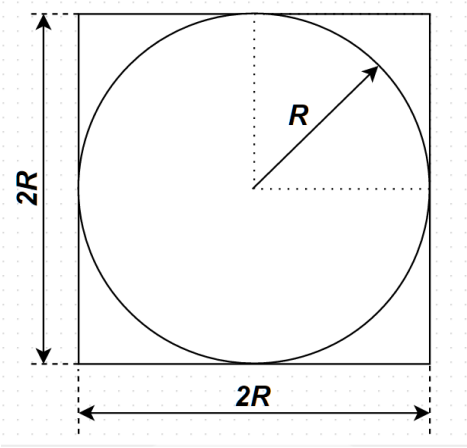

Рис. 1. Вычисление числа π

Сравниваемые решения:

Исходный код обоих решений представлен на github по ссылке [2].

1. Pure Python

   Реализация на обычном Python 2.7 с использованием обычных встроенных в Python функций map/reduce; используется один поток.

2. PySpark

   Реализация на Spark с использованием операций над RDD; используется восемь потоков.

Ожидаемый результат:

Приведённая ниже оценка является верхним теоретическим порогом прироста производительности. На практике же результат будет ниже из-за множества факторов: другие процессы, разделяющие доступ к CPU, синхронизация потоков, передача информации между ними и т.д.

Второй вариант в идеале должен дать прирост производительности в 6-7 раз (это эмпирическая оценка, основанная на том, что гипертрединг даёт в среднем не двухкратный прирост, как можно было бы ожидать, а прирост в ~1.7 раз).

Полученный результат:

1. Pure Python: 164 секунды
2. PySpark: 47 секунд

$$\frac{T_{pure}}{T_{spark}} = \frac{164}{47} \approx 3.5$$

Получили прирост в 3.5 раза, что почти в два раза хуже теоретического потолка.

Возможности ML и MLlib

Перейдём к рассмотрению возможностей библиотек машинного обучения, встроенных в PySpark. Таких библиотек две: ML и MLlib. [3]

MLlib предоставляет базовое API на базе RDD.

ML предоставляет API высокого уровня для pipelining'а разных методов ML, используя SQL-запросы над DataFrames.

MLlib включает в себя следующий набор инструментов для решения задач машинного обучения:

1. Классификация: LogisticRegression, DecisionTreeClassifier, GBTClassifier, RandomForestClassifier, NaiveBayes, MultilayerPerceptronClassifier
2. Кластеризация: KMeans

3. Рекомендации: ALS

4. Регрессия: AFTSurvivalRegression, DecisionTreeRegressor, GBTRegressor, IsotonicRegression, LinearRegression, RandomForestRegressor

Масштабируемость. MLP + Mnist

Цель данного эксперимента: оценить масштабируемость (scalability) Spark применительно к задаче обучения MLP на Mnist.

Mnist — это набор чёрно-белых рукописных изображений цифр от нуля до девяти. Всего в наборе 70000 разных картинок, каждая 28 x 28 пикселей.

В качестве модели классификатора используется многослойный персептрон (MLP – Multilayer Perceptron), логическая схема которого представлена на рис. 2.

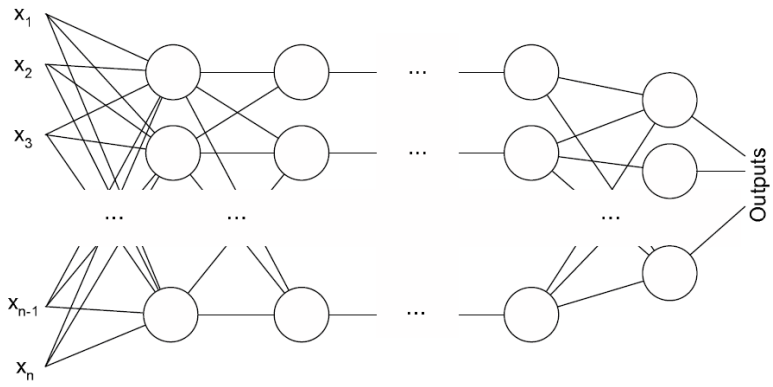

Рис. 2. Логическая структура многослойного персептрона

Конфигурация слоёв MLP: [28*28, 14*14, 5*5, 10], количество итераций: 100. Исходный код реализации этого эксперимента представлен по ссылке [4].

В этом эксперименте используется не весь Mnist, а только сотая его часть, так как цель эксперимента – оценить масштабируемость алгоритма.

Результаты представлены в табл. 1 и разделены на три группы:

1. зависимость времени от количества используемых ядер (степень параллелизма);

2. зависимость времени от объёма ОЗУ;

3. зависимость времени от обеих величин (ОЗУ и CPU).

Таблица 1

Масштабируемость Apache Spark

Ядер, шт.	ОЗУ, ГБ.	Время, мин
1	1	3.4
8	1	2.7
1	1	3.4
1	4	2.9
1	8	2.9
1	1	3.4
8	8	2.5

Как видно, даже при восьмикратном увеличении степени параллелизма и восьмикратном увеличении объёма ОЗУ производительность возросла всего в ~1.4 раза.

Производительность. MLP + Mnist

Цель данного эксперимента: оценить абсолютное время обучения MLP из Spark на всей выборке Mnist.

Конфигурация слоёв MLP: [28*28, 1024, 10], количество итераций: 100.

Исходный код реализации этого эксперимента представлен по ссылке [4].

Результат эксперимента представлен в (табл. 1).

Таблица 2

Производительность Apache Spark

Ядер, шт.	ОЗУ, ГБ.	Время, мин
8	8	51

Точность (процент верно распознанных изображений) при этом составила 0.961.

Производительность. Spark vs Theano + CUDA

Цель данного эксперимента: оценить абсолютное время обучения MLP с использованием библиотеки Theano и технологии CUDA на всей выборке Mnist (вычисления выполняются на видеокарте).

Theano — это библиотека для численных вычислений на Python. Вычисления в Theano описываются в стиле NumPy и компилируются для эффективного исполнения на CPU или GPU. В данном эксперименте используется именно GPU.

Конфигурация слоёв MLP: [28*28, 1024, 10], количество итераций: 100. (конфигурация такая же как в предыдущем эксперименте со Spark).

Исходный код реализации доступен по ссылке [5].

Результат: 2.65 мин.

Точность (процент верно распознанных изображений) при этом составила 0.947.

То есть прирост производительности решения на Theano/CUDA над решением на Apache Spark – более чем в 19 раз. Аналогичные результаты были получены и в статье [6] в задаче распознавания лиц с использованием MLP.

Выводы

Так как на данный момент Apache Spark не имеет встроенной поддержки вычислений на GPU, то для задач интеллектуального обучения с интенсивными вычислениями лучше отказаться от использования кластера с Apache Spark в пользу одной машины с мощной видеокартой.

Применение Apache Spark в большей степени оправдано для преимущественно параллельной обработки больших потоков данных пачками (batches). Кроме того, Apache Spark для полноценной работы имеет достаточно большие системные требования [7].

Следует также отметить, что при измерении производительности решений на базе CUDA стоит обращать внимание на процент загрузки CPU. Если тестируемое приложение загружает одно ядро на 100%, то такое решение стоит считать неэффективным, так как производительность CPU в нём является «бутылочным горлышком».

Список литературы

[1] Документация Apache Spark. Режим доступа: http://spark.apache.org (дата обращения: 14.02.2016).

[2] Реализация алгоритма поиска числа Пи. Режим доступа: https://gist.github.com/alekseyl1992/71f0af8daaeb0e5fea00 (дата обращения: 14.02.2016).

[3] Документация к библиотеке MLib. Режим доступа: http://spark.apache.org/mllib/ (дата обращения 14.02.2016).

[4] Реализация алгоритма обучения MLP на Mnist с использованием Apache Spark MLib. Режим доступа: https://gist.github.com/alekseyl1992/237cee3528d991b1a9d4 (дата обращения 14.02.2016).

[5] Реализация алгоритма обучения MLP на Mnist с использованием Theano и CUDA. Режим доступа: https://github.com/lisa-lab/DeepLearningTutorials/blob/c4e42a2d0a5d2c4e41a5dfb20aa7f725cff92dbc/code/mlp.py#L195 (дата обращения 14.02.2016).

[6] Терехов В.И. О реализации нейросетевого алгоритма распознавания лиц на графических процессорах. Режим доступа: http://technomag.bmstu.ru/doc/659387.html (дата обращения 14.02.2016).

[7] Системные требования Apache Spark. Режим доступа: http://spark.apache.org/docs/latest/hardware-provisioning.html (дата обращения: 14.02.2016).