Теория BigData

Поставщики: **Apache, Cloudera, Hortonworks, MapR**

Hadoop является проектом верхнего уровня организации Apache Software Foundation, поэтому основным дистрибутивом и центральным репозиторием для всех наработок считается именно Apache Hadoop. Однако этот же дистрибутив является основной причиной большинства сожжённых нервных клеток при знакомстве с данным инструментом: по умолчанию установка слонёнка на кластер требует предварительной настройки машин, ручной установки пакетов, правки множества файлов конфигурации и кучи других телодвижений. При этом документация чаще всего неполна или просто устарела. Поэтому на практике чаще всего используются дистрибутивы от одной из трёх компаний:

**Cloudera**. Ключевой продукт — CDH (Cloudera Distribution including Apache Hadoop) — связка наиболее популярных инструментов из инфраструктуры Hadoop под управлением Cloudera Manager. Менеджер берёт на себя ответсвенность за развёртывание кластера, установку всех компонентов и их дальнейший мониторинг. Кроме CDH компания развивает и другие свои продукты, например, Impala (об этом ниже). Отличительной чертой Cloudera также является стремление первыми предоставлять на рынке новые фичи, пусть даже и в ущерб стабильности. Ну и да, создатель Hadoop — Doug Cutting — работает в Cloudera.

**Hortonworks**. Так же, как и Cloudera, они предоставляют единое решение в виде HDP (Hortonworks Data Platform). Их отличительной чертой является то, что вместо разработки собственных продуктов они больше вкладывают в развитие продуктов Apache. Например, вместо Cloudera Manager они используют Apache Ambari, вместо Impala — дальше развивают Apache Hive. Мой личный опыт с этим дистрибутивом сводится к паре тестов на виртуальной машине, но по ощущениями HDP выглядит стабильней, чем CDH.

**MapR**. В отличие от двух предыдущих компаний, основным источником доходов для которых, судя по всему, является консалтинг и партнёрские программы, MapR занимается непосредственно продажей своих наработок. Из плюсов: много оптимизаций, партнёрская программа с Amazon. Из минусов: бесплатная версия (M3) имеет урезанный функционал. Кроме того, MapR является основным идеологом и главным разработчиком Apache Drill.

Фундамент: **HDFS**

Когда мы говорим про Hadoop, то в первую очередь имеем в виду его файловую систему — **HDFS** (Hadoop Distributed File System). Самый простой способ думать про HDFS — это представить обычную файловую систему, только больше. Обычная ФС, по большому счёту, состоит из таблицы файловых дескрипторов и области данных. В HDFS вместо таблицы используется специальный

**Hive** это СУБД на платформе **Hadoop**. В качестве языка запросов использует HiveQL — урезанный диалект SQL, который, тем не менее, позволяет выполнять довольно сложные запросы над данными, хранимыми в HDFS

Движки: **MapReduce, Spark, Tez**

При правильной архитектуре приложения, информация о том, на каких машинах расположены блоки данных, позволяет запустить на них же вычислительные процессы (которые мы будем нежно называть англицизмом «воркеры») и выполнить большую часть вычислений локально, т.е. без передачи данных по сети. Именно эта идея лежит в основе парадигмы **MapReduce** и её конкретной реализации в Hadoop.

Классическая конфигурация кластера Hadoop состоит из одного сервера имён, одного мастера MapReduce (т.н. JobTracker) и набора рабочих машин, на каждой из которых одновременно крутится сервер данных (DataNode) и воркер (TaskTracker). Каждая MapReduce работа состоит из двух фаз:

**map** — выполняется параллельно и (по возможности) локально над каждым блоком данных. Вместо того, чтобы доставлять терабайты данных к программе, небольшая, определённая пользователем программа копируется на сервера с данными и делает с ними всё, что не требует перемешивания и перемещения данных (shuffle).

**reduce** — дополняет map агрегирующими операциями

На самом деле между этими фазами есть ещё фаза combine, которая делает то же самое, что и reduce, но над локальными блоками данных. Например, представим, что у нас есть 5 терабайт логов почтового сервера, которые нужно разобрать и извлечь сообщения об ошибках. Строки независимы друг от друга, поэтому их разбор можно переложить на задачу map. Дальше с помощью combine можно отфильтровать строки с сообщением об ошибке на уровне одного сервера, а затем с помощью reduce сделать то же самое на уровне всех данных. Всё, что можно было распараллелить, мы распараллелили, и кроме того минимизировали передачу данных между серверами. И даже если какая-то задача по какой-то причине упадёт, Hadoop автоматически перезапустит её, подняв с диска промежуточные результаты. Круто!

Проблема в том, что большинство реальных задач гораздо сложней одной работы MapReduce. В большинстве случаев мы хотим делать параллельные операции, затем последовательные, затем снова параллельные, затем комбинировать несколько источников данных и снова делать параллельные и последовательные операции. Стандартный MapReduce спроектирован так, что все результаты — как конечные, так и промежуточные — записываются на диск. В итоге время считывания и записи на диск, помноженное на количество раз, которые оно делается при решении задачи, зачастую в несколько (да что там в несколько, до 100 раз!) превышает время самих вычислений.

И здесь появляется **Spark**. Спроектированный ребятами из университета Berkeley, Spark использует идею локальности данных, однако выносит большинство вычислений в память вместо диска. Ключевым понятием в Spark-е является RDD (resilient distributed dataset) — указатель на ленивую распределённую колекцию данных. Большинство операций над RDD не приводит к каким-либо вычислениям, а только создаёт очередную обёртку, обещая выполнить операции только тогда, когда они понадобятся. Впрочем, это проще показать, чем рассказать. Ниже приведён скрипт на Python (Spark из коробки поддерживает интерфейсы для Scala, Java и Python) для решения задачи про логи:

sc = ... # создаём контекст (SparkContext)

rdd = sc.textFile("/path/to/server\_logs") # создаём указатель на данные

rdd.map(parse\_line) \ # разбираем строки и переводим их в удобный формат

.filter(contains\_error) \ # фильтруем записи без ошибок

.saveAsTextFile("/path/to/result") # сохраняем результаты на диск

В этом примере реальные вычисления начинаются только на последней строчке: Spark видит, что нужно материализовать результаты, и для этого начинает применять операции к данным. При этом здесь нет никаких промежуточных стадий — каждая строчка поднимается в память, разбирается, проверяется на признак ошибки в сообщении и, если такой признак есть, тут же записывается на диск.

Такая модель оказалась настолько эффективной и удобной, что проекты из экосистемы Hadoop начали один за другим переводить свои вычисления на Spark, а над самим движком сейчас работает больше людей, чем над морально устаревшим MapReduce.

Но не Spark-ом единым. Компания Hortonworks решила сделать упор на альтернативный движок — **Tez**. Tez представляет задачу в виде направленного ациклического графа (DAG) компонентов-обработчиков. Планировщик запускает вычисление графа и при необходимости динамически переконфигурирует его, оптимизируя под данные. Это очень естественная модель для выполнения сложных запросов к данным, таких как SQL-подобные скрипты в Hive, куда Tez принёс ускорение до 100 раз. Впрочем, кроме Hive этот движок пока мало где используется, поэтому сказать, насколько он пригоден для более простых и распространённых задач, довольно сложно.

**Hive** — самая первая и до сих пор одна из самых популярных СУБД на этой платформе. В качестве языка запросов использует HiveQL — урезанный диалект SQL, который, тем не менее, позволяет выполнять довольно сложные запросы над данными, хранимыми в HDFS. Здесь надо провести чёткую линию между версиями Hive <= 0.12 и текущей версией 0.13: как я уже говорил, в последней версии Hive переключился с классического MapReduce на новый движок Tez, многократно ускорив его и сделав пригодным для интерактивной аналитики. Т.е. теперь вам не надо ждать 2 минуты, чтобы посчитать количество записей в одной небольшой партиции или 40 минут, чтобы сгруппировать данные по дням за неделю (прощайте длительные перекуры!). Кроме того, как Hortonworks, так и Cloudera предоставляют ODBC-драйвера, позволяя подключить к Hive такие инструменты как Tableau, Micro Strategy и даже (господи, упаси) Microsoft Excel.

**Impala** — продукт компании Cloudera и основной конкурент Hive. В отличие от последнего, Impala никогда не использовала классический MapReduce, а изначально исполняла запросы на своём собственном движке (написанном, кстати, на нестандартном для Hadoop-а C++). Кроме того, в последнее время Impala активно использует кеширование часто используемых блоков данных и колоночные форматы хранения, что очень хорошо сказывается на производительности аналитических запросов. Так же, как и для Hive, Cloudera предлагает к своему детищу вполне эффективный ODBC-драйвер.

Машинное обучение

Картинка выше прекрасно выражает состояние многих компаний: все знают, что большие данные — это хорошо, но мало кто реально понимает, что с ними делать. А делать с ними нужно в первую очередь две вещи — переводит в знания (читать как: использовать при принятии решений) и улучшать алгоритмы. С первым уже помогают инструменты аналитики, а второе сводится к машинному обучению. В Hadoop для этого есть два крупных проекта:

**Mahout** — первая большая библиотека, реализовавшая многие популярные алгоритмы средствами MapReduce. Включает в себя алгоритмы для кластеризации, коллаборативной фильтрации, случайных деревьев, а также несколько примитивов для факторизации матриц. В начале этого года организаторы приняли решение перевести всё на вычислительное ядро Apache Spark, которое гораздо лучше поддерживает итеративные алгоритмы (попробуйте прогнать 30 итераций градиентного спуска через диск при стандартном MapReduce!).

Форматы данных: **Parquet, ORC, Thrift, Avro**

Если вы решите использовать Hadoop по полной, то не помешает ознакомиться и с основными форматами хранения и передачи данных.

**Parquet** — колончатый формат, оптимизированный для хранения сложных структур и эффективного сжатия. Изначально был разработан в Twitter, а сейчас является одним из основных форматов в инфраструктуре Hadoop (в частности, его активно поддерживают Spark и Impala).

**ORC** — новый оптимизированный формат хранения данных для Hive. Здесь мы снова видим противостояние Cloudera c Impala и Parquet и Hortonworks с Hive и ORC. Интересней всего читать сравнение производительности решений: в блоге Cloudera всегда побеждает Impala, причём со значительным перевесом, а в блоге Hortonworks, как несложно догадаться, побеждает Hive, причём с не меньшим перевесом.

**Thrift** — эффективный, но не очень удобный бинарный формат передачи данных. Работа с этим форматом предполагает определение схемы данных и генерацию соответсвующего кода клинета на нужном языке, что не всегда возможно. В последнее время от него стали отказываться, но многие сервисы всё ещё используют его.

**Avro** — в основном позиционируется как замена Thrift: он не требует генерации кода, может передавать схему вместе с данными или вообще работать с динамически типизированными объектами.

Прочее: **ZooKeeper, Hue, Flume, Sqoop, Oozie, Azkaban**

Ну и напоследок коротко о других полезных и бесполезных проектах.

**ZooKeeper** — главный инструмент координации для всех элементов инфраструктуры Hadoop. Чаще всего используется как сервис конфигурации, хотя его возможности гораздо шире. Простой, удобный, надёжный.

**Hue** — веб-интерфейс к сервисам Hadoop, часть Cloudera Manager. Работает плохо, с ошибками и по настроению. Пригоден для показа нетехническим специалистам, но для серьёзной работы лучше использовать консольные аналоги.

**Flume** — сервис для организации потоков данных. Например, можно настроить его для получения сообщений из syslog, агрегации и автоматического сбрасывания в директорию на HDFS. К сожалению, требует очень много ручной конфигурации потоков и постоянного расширения собственными Java классами.

**Sqoop** — утилита для быстрого копирования данных между Hadoop и RDBMS. Быстрого в теории. На практике Sqoop 1 оказался, по сути, однопоточным и медленным, а Sqoop 2 на момент последнего теста просто не заработал.

**Oozie** — планировщик потоков задач. Изначально спроектирован для объединения отдельных MapReduce работ в единый конвеер и запуска их по расписанию. Дополнительно может выполнять Hive, Java и консольные действия, но в контексте Spark, Impala и др., этот список выглядит довольно бесполезным. Очень хрупкий, запутанный и практически не поддаётся отладке.

**Azkaban** — вполне годная замена Oozie. Является частью Hadoop-инфраструктуры компании LinkedIn. Поддерживает несколько типов действий, главное из которых — консольная команда (а что ещё надо), запуск по расписанию, логи приложений, оповещения об упавших работах и др. Из минусов — некоторая сыроватость и не всегда понятный интерфейс (попробуйте догадаться, что работу нужно не создавать через UI, а заливать в виде zip-архива с текстовыми файлами).