Распределенные вычисления. Hadoop

Hadoop не так уж сложен, ядро состоит из файловой системы HDFS и MapReduce фреймворка для обработки данных из этой файловой системы.

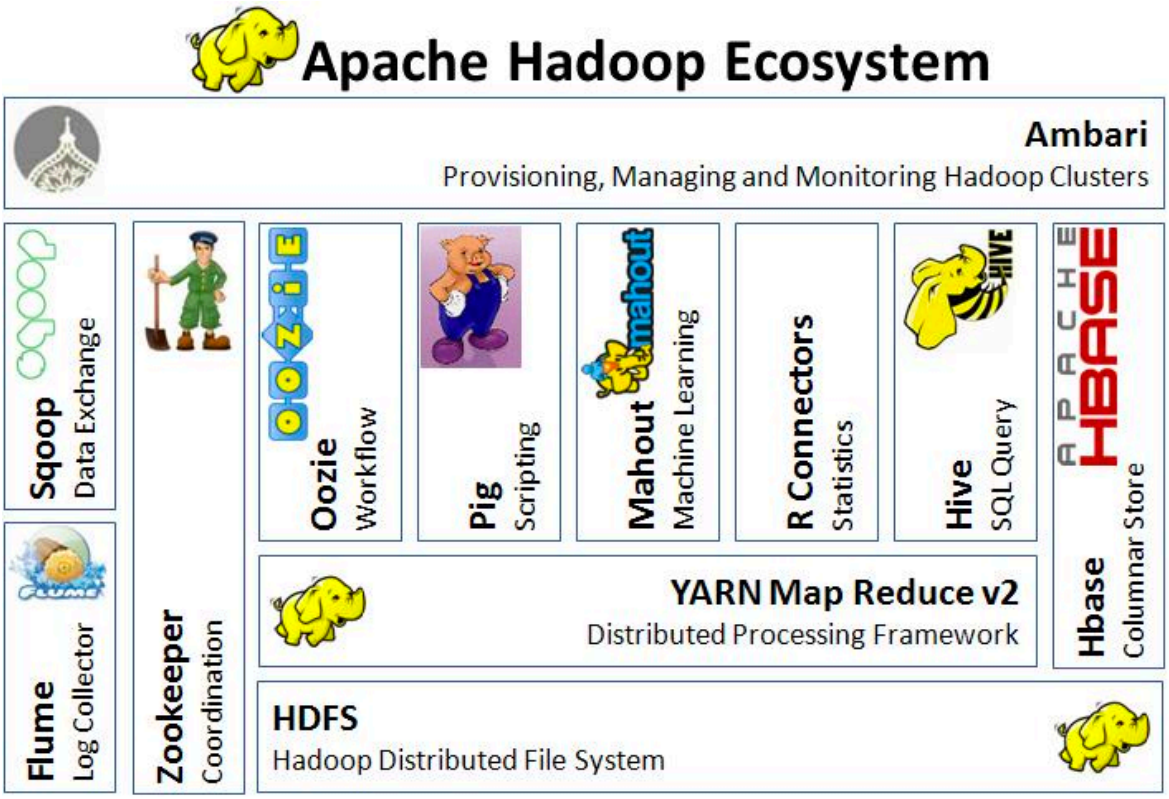
Hadoop следует использовать, если:

- Вычисления должны быть компонуемыми, другими словами, вы должны иметь возможность запустить вычисления на подмножестве данных, а затем слить результаты.
- Вы планируете обрабатывать большой объем неструктурированных данных — больше, чем можно уместить на одной машине (> нескольких терабайт данных).

Hadoop не следует использовать:

- Для некомпонуемых задач — например, для задач рекуррентных.
- Если весь объем данных умещается на одной машине. Существенно сэкономите время и ресурсы.
- Hadoop в целом — система для пакетной обработки и не подходит для анализа в режиме реального времени (здесь на помощь приходит система Storm).

Архитектура HDFS и типичный Hadoop кластер



HDFS подобна другим традиционным файловым системам: файлы хранятся в виде блоков, существует маппинг между блоками и именами файлов, поддерживается древовидная структура, поддерживается модель доступа к файлам основанная на правах и т. п.

Отличия HDFS:

- Предназначена для хранения большого количества огромных (>10GB) файлов. Одним Следствие — большой размер блока по сравнению с другими файловыми системами (>64MB)

- Оптимизирована для поддержки потокового доступа к данным (high-streaming read), соответственно производительность операций произвольного чтения данных начинает хромать.

- Ориентирована на использование большого количество недорогих серверов. В частности, серверы используют JBOB структуру (Just a bunch of disk) вместо RAID — зеркалирование и репликация осуществляются на уровне кластера, а не на уровне отдельной машины.

- Многие традиционные проблемы распределенных систем заложены в дизайн — уже по дефолту все выход отдельных нод из строя является совершенно нормальной и естественной операцией, а не чем-то из ряда вон.

Hadoop-кластер состоит из нод трех типов: NameNode, Secondary NameNode, Datanode.

Namenode — мозг системы. Как правило, одна нода на кластер (больше в случае Namenode Federation, но мы этот случай оставляем за бортом). Хранит в себе все метаданные системы — непосредственно маппинг между файлами и блоками. Если нода 1 то она же и является Single Point of Failure. Эта проблема решена во второй версии Hadoop с помощью Namenode Federation.

Secondary NameNode — 1 нода на кластер. Принято говорить, что «Secondary NameNode» — это одно из самых неудачных названий за всю историю программ. Действительно, Secondary NameNode не является репликой NameNode. Состояние файловой системы хранится непосредственно в файле fsimage и в лог файле edits, содержащим последние изменения файловой системы (похоже на лог транзакций в мире РСУБД). Работа Secondary NameNode заключается в периодическом мерже fsimage и edits — Secondary NameNode поддерживает размер edits в разумных пределах. Secondary NameNode необходима для быстрого ручного восстанавления NameNode в случае выхода NameNode из строя.

В реальном кластере NameNode и Secondary NameNode — отдельные сервера, требовательные к памяти и к жесткому диску. А заявленное “commodity hardware” — уже случай DataNode.

DataNode — Таких нод в кластере очень много. Они хранят непосредственно блоки файлов. Нода регулярно отправляет NameNode свой статус (показывает, что еще жива) и ежечасно — репорт, информацию обо всех хранимых на этой ноде блоках. Это необходимо для поддержания нужного уровня репликации.

Посмотрим, как происходит запись данных в HDFS:

1. Клиент разрезает файл на цепочки блокового размера.

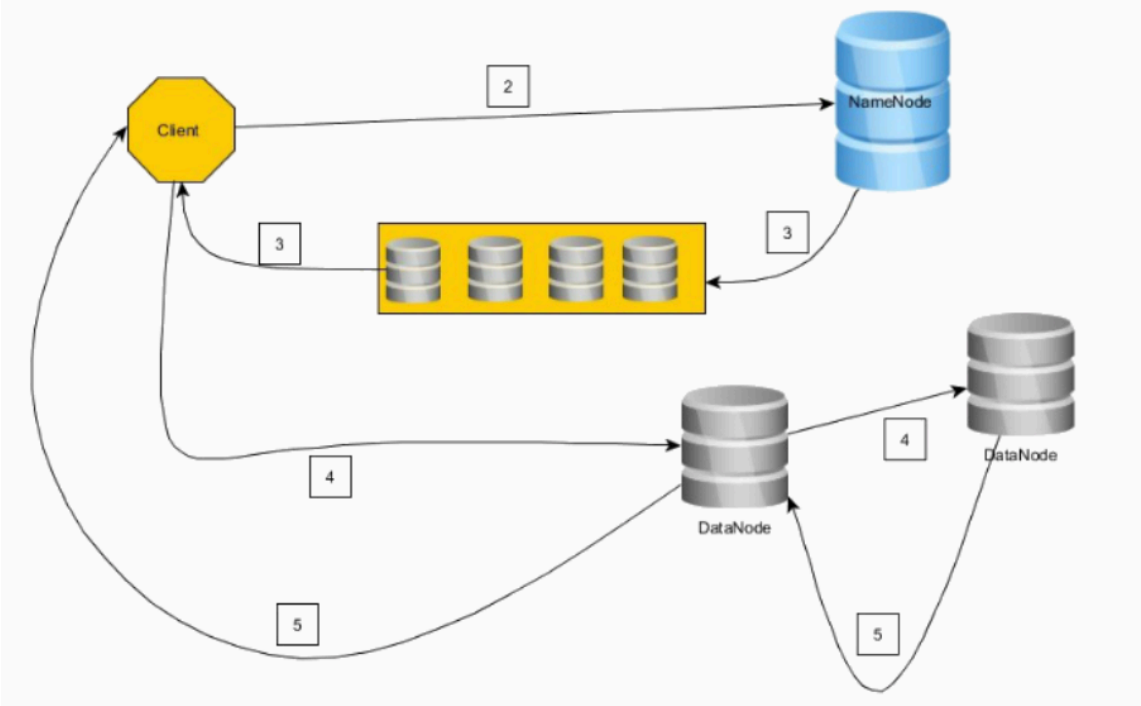
2. Клиент соединяется с NameNode и запрашивает операцию записи, присылая количество блоков и требуемый уроень репликации

3. NameNode отвечает цепочкой из DataNode.

4. Клиент соединяется с первой нодой из цепочки (если не получилось с первой, со второй и т. д. не получилось совсем — откат). Клиент делает запись первого блока на первую ноду, первая нода — на вторую и т. д.

5. По завершении записи в обратном порядке (4 -> 3, 3 -> 2, 2 -> 1, 1 -> клиенту) присылаются сообщения об успешной записи.

6. Как только клиент получит подтверждение успешной записи блока, он оповещает NameNode о записи блока, затем получает цепочку DataNode для записи второго блока и т.д.



Клиент продолжает записывать блоки, если сумеет записать успешно блок хотя бы на одну ноду, т. е. репликация будет работать по хорошо известному принципу «eventual», в дальнейшем NameNode обязуется компенсировать и таки достичь желаемого уровня репликации.

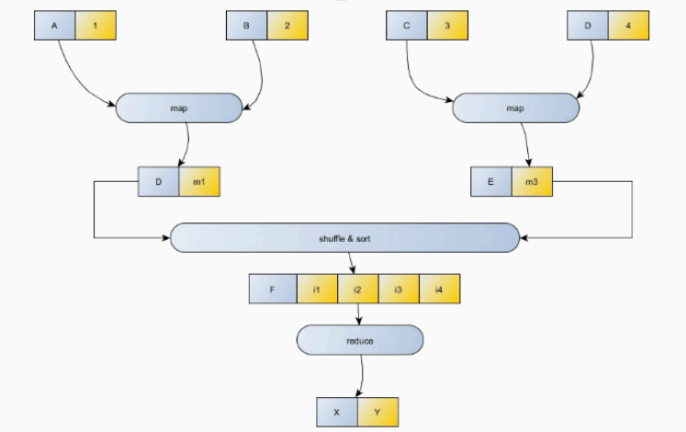
Завершая обзор HDFS и кластера, обратим внимание на еще одну замечательную особенность Hadoop'a — rack awareness. Кластер можно сконфигурировать так, чтобы NameNode имел представление, какие ноды на каких rack'ах находятся, тем самым обеспечив лучшую защиту от сбоев.

MapReduce

Единица работы job — набор map (параллельная обработка данных) и reduce (объединение выводов из map) задач. Map-задачи выполняют mapper'ы, reduce — reducer'ы. Job состоит минимум из одного mapper'a, reducer'ы опциональны. Здесь разобран вопрос разбиения задачи на map'ы и reduce'ы. Если слова «map» и «reduce» вам совсем непонятны, можно посмотреть классическую статью на эту тему.

Модель MapReduce

- Ввод/вывод данных происходит в виде пар (key, value)



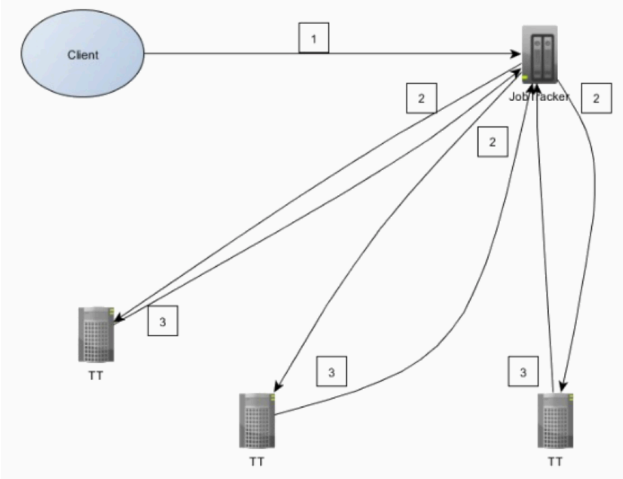
- Используются две функции map: (K1, V1) -> ((K2, V2), (K3,V3), ...) — отображающая пару ключ-значение на некое множество промежуточных пар ключей и значений, а также reduce: (K1, (V2,V3,V4,VN)) -> (K1, V1), отображающая некоторое множество значений, имеющий общий ключ на меньшее множество значений.

- Shuffle and sort нужна для сортировки ввода в reducer по ключу, другими словами, нет смысла отправлять значение (K1, V1) и (K1, V2) на два разных reducer'a. Они должны быть обработаны вместе.

Посмотрим на архитектуру MapReduce

1. Для начала расширим представление о hadoop-кластере, добавив в кластер два новых элемента — JobTracker и TaskTracker. JobTracker непосредственно запросы от клиентов и управляет map/reduce тасками на TaskTracker'ах. JobTracker и NameNode разносится на разные машины, тогда как DataNode и TaskTracker находятся на одной машине.

Взаимодействие клиента и кластера выглядит следующим образом:



1. Клиент отправляет job на JobTracker. Job представляет из себя jar-файл.

2. JobTracker ищет TaskTracker'ы с учетом локальности данных, т.е. предпочитая те, которые уже хранят данные из HDFS. JobTracker назначает map и reduce задачи TaskTracker'ам

3. TaskTracker'ы отправляют отчет о выполнении работы JobTracker'у.

Неудачное выполнение задачи — ожидаемое поведение, провалившиеся таски автоматически перезапускаются на других машинах.

В Map/Reduce 2 (Apache YARN) больше не используется терминология «JobTracker/ TaskTracker». JobTracker разделен на **ResourceManager** — управление ресурсами и **Application Master** — управление приложениями (одним из которых и является непосредственно MapReduce). MapReduce v2 использует новое API