Лабораторная работа 4. Развёртывание системы распределённых вычислений Hadoop

1.1 Цель работы

Цель лабораторной работы заключается в получении первоначальных навыков настройки и использования системы Hadoop.

1.2 Общие сведение

Hadoop – популярная программная платформа (software framework) построения распределенных приложений для массово-параллельной обработки (massive parallel processing, MPP) данных в рамках вычислительной парадигмы MapReduce.

Hadoop считается одним из основополагающих решений в области «больших данных» (big data). Вокруг Hadoop образовалась целая экосистема из связанных проектов и технологий.

1.2.1 Архитектура Hadoop

Hadoop состоит из четырёх модулей:

- связующее программное обеспечение Hadoop Common;
- распределённая файловая система HDFS;
- система для планирования заданий и управления кластером;
- платформа программирования и выполнения распределённых вычислений Hadoop MapReduce.

В Hadoop Common входят библиотеки управления файловыми системами, поддерживаемыми Hadoop, и сценарии создания необходимой инфраструктуры и управления распределённой обработкой.

HDFS (Hadoop Distributed File System) – файловая система, предназначенная для хранения файлов больших размеров, поблочно распределённых между узлами вычислительного кластера. Все блоки в HDFS (кроме последнего блока файла) имеют одинаковый размер, и каждый блок может быть размещён на нескольких узлах, размер блока и коэффициент репликации (количество узлов, на которых должен быть размещён каждый блок) определяются в настройках на уровне файла. Благодаря репликации обеспечивается устойчивость распределённой системы к отказам отдельных узлов. Файлы в HDFS могут быть записаны лишь однажды (модификация не поддерживается), а запись в файл в одно время может вести только один процесс. Организация файлов в пространстве имён — традиционная иерархическая: есть корневой каталог, поддерживается вложение каталогов, в одном каталоге могут располагаться и файлы, и другие каталоги.

Развёртывание экземпляра HDFS предусматривает наличие центрального узла имён (name node), хранящего метаданные файловой системы и метаинформацию о распределении блоков, и серии узлов данных (data node), непосредственно хранящих блоки файлов. Узел имён отвечает за обработку операций уровня файлов и каталогов — открытие и закрытие файлов, манипуляция с каталогами, узлы данных непосредственно отрабатываютоперации по записи и чтению данных. Узел имён и узлы данных снабжаются веб-серверами, отображающими текущий статус узлов и позволяющими просматривать содержимое файловой системы.

HDFS является неотъемлемой частью проекта, однако, Hadoop поддерживает работу и с другими распределёнными файловыми системами без использования HDFS, поддержка Amazon S3 и CloudStore реализована в основном дистрибутиве. С другой стороны, HDFS может использоваться не только для запуска MapReduce-заданий, но и как распределённая файловая система общего назначения, в частности, поверх неё реализована распределённая NoSQL-СУБД HBase, в её среде работает масштабируемая система машинного обучения Apache Mahout.

YARN (англ. Yet Another Resource Negotiator – «ещё один ресурсный посредник») – модуль, появившийся с версией 2.0 Hadoop, отвечающий за управление ресурсами кластеров и планирование заданий. Если в предыдущих выпусках эта функция была интегрирована в модуль MapReduce, где была реализована единым компонентом (JobTracker), то в YARN функционирует логически самостоятельный демон – планировщик ресурсов (ResourceManager), абстрагирующий все вычислительные ресурсы кластера и управляющий их предоставлением приложениям распределённой обработки. Работать под управлением YARN могут как MapReduce-программы, так и любые другие распределённые приложения, поддерживающие соответствующие программные интерфейсы. YARN обеспечивает возможность параллельного выполнения нескольких различных задач в рамках кластера и их изоляцию (по принципам мультиарендности). Разработчику распределённого приложения необходимо реализовать специальный класс управления приложением (ApplicationMaster), который отвечает за координацию заданий в рамках тех ресурсов, которые предоставит планировщик ресурсов; планировщик ресурсов же отвечает за создание экземпляров класса управления приложением и взаимодействия с ним через соответствующий сетевой протокол.

Hadoop MapReduce – программный каркас для программирования распределённых вычислений в рамках парадигмы MapReduce. Разработчику приложения для Hadoop MapReduce необходимо реализовать базовый обработчик, который на каждом вычислительном узле кластера обеспечит преобразование исходных пар «ключ – значение» в промежуточный набор пар

«ключ – значение» (класс, реализующий интерфейс Mapper, назван по функции высшего порядка Map), и обработчик, сводящий промежуточный набор пар в окончательный, сокращённый набор (свёртку, класс, реализующий интерфейс Reducer). Каркас передаёт на вход свёртки отсортированные выводы

от базовых обработчиков, сведе́ние состоит из трёх фаз – shuffle (тасовка, выделение нужной секции вывода), sort (сортировка, группировка по ключам выводов от распределителей – досортировка, требующаяся в случае, когда разные атомарные обработчики возвращают наборы с одинаковыми ключами, при этом правила сортировки на этой фазе могут быть заданы программно и использовать какие-либо особенности внутренней структуры ключей) и собственно reduce (свёртка списка) – получения результирующего набора. Для некоторых видов обработки свёртка не требуется, и каркас возвращает в этом случае набор отсортированных пар, полученных базовыми обработчиками.

Hadoop MapReduce позволяет создавать задания как с базовыми обработчиками, так и со свёртками, написанными без использования Java: утилиты Hadoop streaming позволяют использовать в качестве базовых обработчиков и свёрток любой исполняемый файл, работающий со стандартным вводом-выводом операционной системы (например, утилиты командной оболочки UNIX), есть также SWIG-совместимый прикладной интерфейс программирования Hadoop pipes на C++. Также, в состав дистрибутивов Hadoop входят реализации различных конкретных базовых обработчиков и свёрток, наиболее типично используемых в распределённой обработке.

1.2.2 Введение в MapReduce

Парадигма MapReduce была предложена Google в статье Джефри Дина и Санжая Чемавата «MapReduce: упрощенная обработка данных на больших кластерах» (Jeffrey Dean and Sanjay Ghemawat MapReduce: Simplied Data Processing on Large Clusters). Данная статья носит довольно поверхностный характер. Ее недосказанности компенсирует работа Ральфа Ламмеля из Исследовательского центра Microsoft в Редмонде «И снова модель программирования MapReduce компании Google» (Ralf Lammel Google's MapReduce Programming Model – Revisited). Во введении Ламмель заявляет, что он проанализировал статью Дина и Чемавата, которую он неоднократно и с почтением называет судьбоносной, используя метод, который называется «обратной инженерией» (reverse engineering). То есть, он постарался раскрыть смысл того, что же в ней на самом деле представлено. Ламмель увязывает MapReduce с функциональным программированием, языками Лисп и Haskell, что естественным образом вводит читателя в контекст.

MapReduce – это скорее инфраструктурное решение, способное эффективно использовать сегодня кластерные, а в будущем многоядерные архитектуры.

Большинство современных параллельных компьютеров принадлежит к категории «много потоков команд, много потоков данных» (Multiple Program Multiple Data, MPMD). Каждый узел выполняет фрагмент общего задания, работая со своими собственными данными, а в дополнение существует сложная система обмена сообщениями, обеспечивающая согласование совместной работы. Такой вид параллелизма раньше называют параллелизмом на уровне

задач (task level parallelism), но иногда его еще называют функциональным параллелизмом или параллелизмом по управлению. Его реализация сводится к распределению заданий по узлам и обеспечению синхронности происходящих процессов.

Альтернативный тип параллелизма называют параллелизмом данных (data parallelism), который попадает в категорию «один поток команд, много потоков данных» (Single Program Multiple Data, SPMD). Реализация SPMD предполагает, что сначала данные должны быть каким-то образом распределены между процессорами, обработаны, а затем собраны. Эту совокупность операций можно назвать map-reduce, как принято в функциональном программировании, хотя точнее было бы split-aggregate, то есть разбиение и сборка, но привилось первое. Возможны различные способы реализации SPMD.

Реализация SPMD требует выделения ведущей части кода, ее называют Master или Manager (далее менеджер), и подчиненных ей частей Worker (далее исполнитель). Менеджер «раздает» задания исполнителям и потом их собирает. До появления MapReduce эта модель рассматривалась как малоперспективная из-за наличия «бутылочного горла» на тракте обмена между множеством исполнителей и одним менеджером. Создание MapReduce разрешило эту проблему и открыло возможность для обработки огромных массивов данных с использованием архитектуры SPMD.

Допустим, что следует решить простейшую задачу: обработать массив, разбив его на подмассивы. В таком случае работа менеджера сводится к тому, что он делит этот массив на части, посылает каждому исполнителю положенный ему подмассив, а затем получает результаты и объединяет их (рисунок 1). В функцию исполнителя входит получение данных, обработка и возврат результатов менеджеру. Распределение нагрузок может быть статическим (static load balancing) или динамическим (dynamic load balancing). Парадигма создана по мотивам комбинации map и reduce в функциональном языке программирования Липс. В нем map использует в качестве входных параметров функцию и набор значений, применяя эту функцию по отношениюк каждому из значений, а reduce комбинирует полученные результаты.

Канонический пример приложения, написанного с помощью MapReduce это процесс, подсчитывающий, сколько раз различные слова встречаются в наборе документов:

```
// Функция, используемая исполнителями на Map-фазе
// для обработки пар ключ-значение из входного потока
void map(String name, String document):
    // Входные данные:
    //   name - название документа
    //   document - содержимое документа
    for each word w in document:
        EmitIntermediate(w, "1");
```

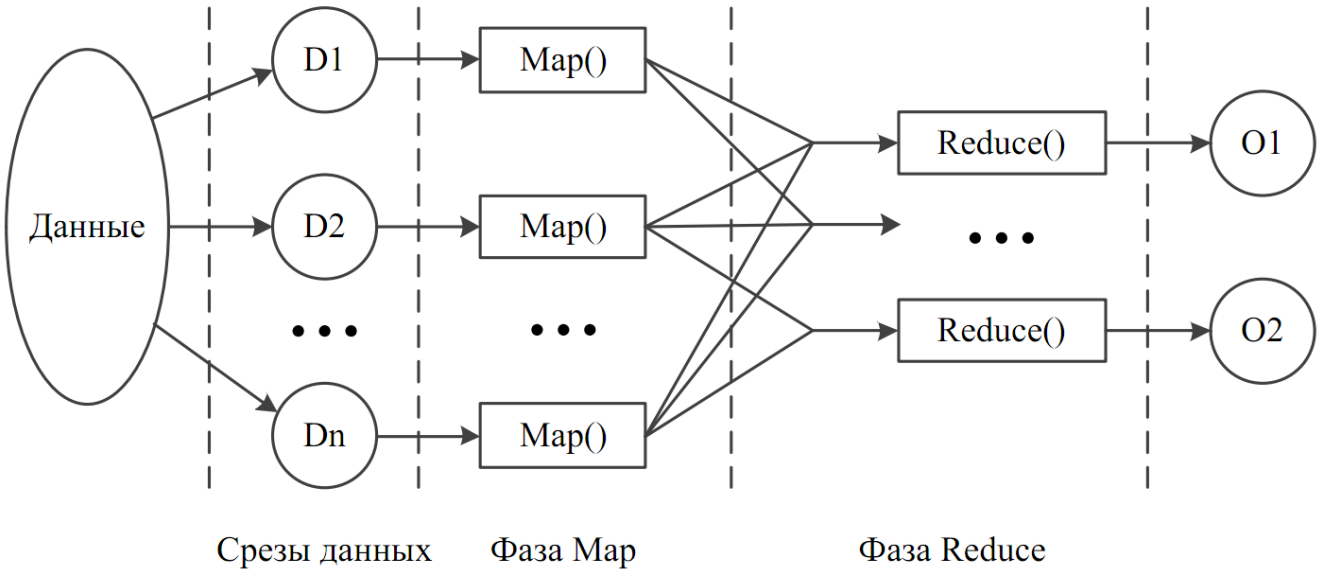

Рисунок 1 – Упрощенная схема потоков данных в парадигме MapReduce

```
// Функция, используемая исполнителями на Reduce-фазе
// для обрабоки пар ключ-значение, полученных на Map-шаге
void reduce(String word, Iterator partialCounts):
    // Входные данные:
    //   word - слово
    //   partialCounts - список группированных промежуточных результатов.
    //       Количество записей в partialCounts и есть требуемое значение
    int result = 0;
    for each v in partialCounts:
        result += parseInt(v);
    Emit(AsString(result));
```

В этом коде на map-фазе каждый документ разбивается на слова, и возвращаются пары, где ключом является само слово, а значением – «1». Если в документе одно и то же слово встречается несколько раз, то в результате предварительной обработки этого документа будет столько же этих пар, сколько раз встретилось это слово.

Далее выполняется объединение всех пар с одинаковым ключом, и они передаются на вход функции reduce, которой остается сложить их, чтобы получить общее количество вхождений данного слова во все документы.

Созданная в Google конструкция MapReduce делает примерно то же, нопо отношению к гигантским объемам данных. В этом случае map – это функция запроса от пользователя, помещенная в библиотеку MapReduce. Ее работа сводится к выбору входной пары, ее обработке и формированию результата в виде значения и некоторого промежуточного ключа, служащего указателем для reduce. Конструкция MapReduce собирает вместе все значения с одинаковыми промежуточными ключами и передает их в функцию reduce, также написанную пользователем. Эта функция воспринимает промежуточные значения, каким-то образом их собирает и воспроизводит результат, скорее всего выраженный меньшим количеством значений, чем входное множество.

Теперь свяжем функциональную идею MapReduce со схемой SPMD. Вызовы map распределяются между множеством машин путем деления входного потока данных на M срезов (splits или shard), каждый срез обрабатывается на

отдельной машине. Вызовы reduce распределены на R частей, количество которых определяется пользователем.

При вызове функции из библиотеки MapReduce выполняется примерно такая последовательность операций (рисунок 2):

1) входные файлы разбиваются на срезы размером от 16 до 64 Мбайт каждый, и на кластере запускаются копии программы. Одна из них менеджер, а остальные – исполнители. Всего создается M задач map и R задач reduce. Поиском свободных узлов и назначением на них задач занимается менеджер;

2) в процессе исполнения исполнители, назначенные на выполнение задачи map, считывают содержимое соответствующих срезов, осуществляют их грамматический разбор, выделяют отдельные пары «ключ и соответствующее ему значение», а потом передают эти пары в обрабатывающую их функцию map. Промежуточное значение в виде идентификатора и значения буферизуется в памяти;

3) периодически буферизованные пары сбрасываются на локальный диск, разделенный на R областей. Расположение этих пар передается менеджеру, который отвечает за дальнейшую передачу этих адресов исполнителям, выполняющим reduce;

4) исполнители, выполняющие задачу reduce, ждут сообщения от менеджера о местоположении промежуточных пар. По его получении они, используя процедуры удаленного вызова, считывают буферизованные данные с локальных дисков тех исполнителей, которые выполняют map. Загрузив все промежуточные данные, исполнитель сортирует их по промежуточным ключам и, если есть необходимость, группирует;

5) исполнитель обрабатывает данные по промежуточным ключам и передает их в соответствующую функцию reduce для вывода результатов;

6) когда все задачи map и reduce завершаются, конструкция MapReduce возобновляет работу вызывающей программы, и та продолжает выполнять пользовательский код.

Одно из важнейших преимуществ этой, замысловатой на первый взгляд, конструкции состоит в том, что она позволяет надежно работать на платформах с низкими показателями надежности. Для обнаружения потенциальных сбоев менеджер периодически опрашивает исполнителей, и, если какой-то из них задерживается с ответом сверх заданного норматива, менеджер считает его дефектным и передает исполнение на свободные узлы. Различие между задачами map и reduce в данном случае состоит в том, что map хранит промежуточные результаты на своем диске, и выход из строя такого узла приводит к их потере и требует повторного запуска, а reduce хранит свои данные в глобальном хранилище.

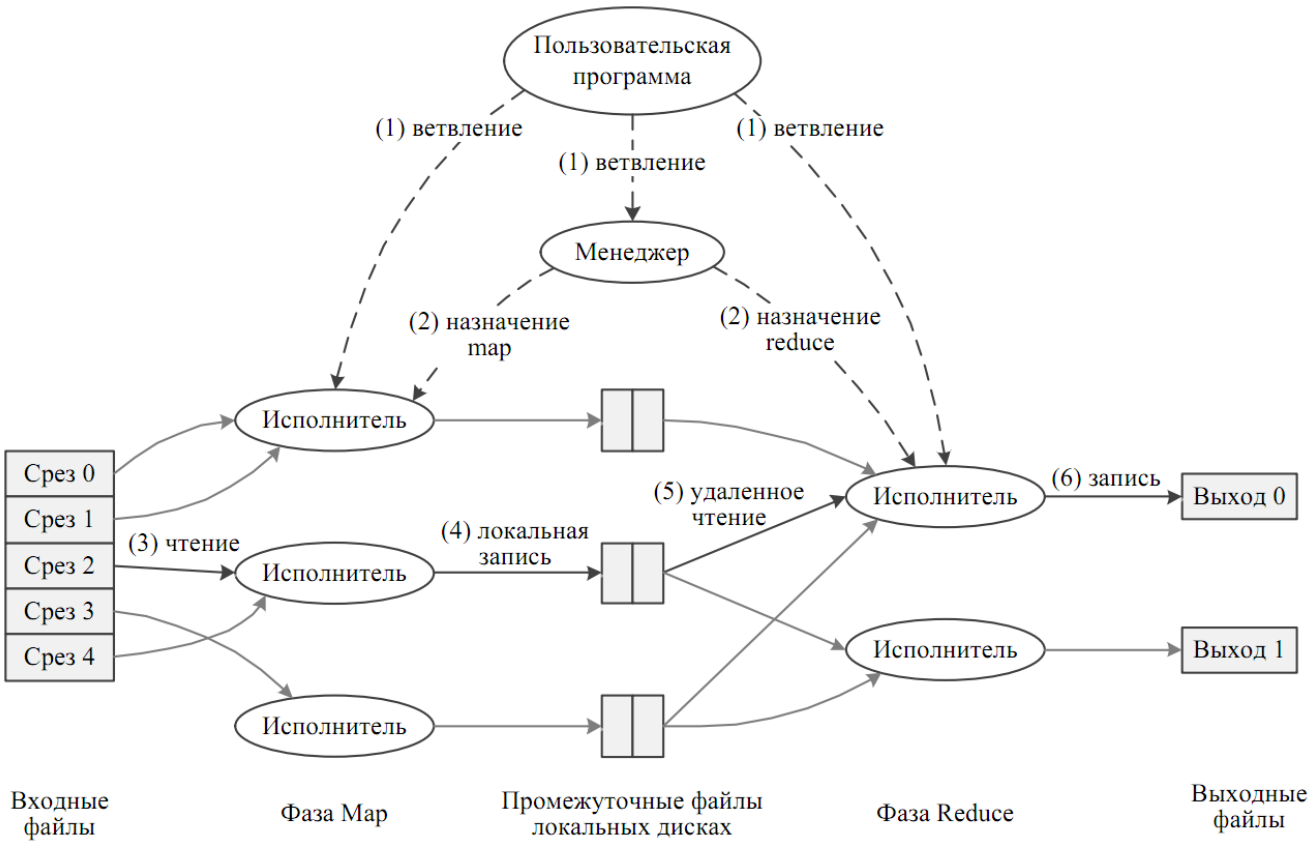

Рисунок 2 – Ход выполнения вызова MapReduce

1.3 Порядок выполнения работы

1.3.1 Установка операционной системы

В качестве операционной системы для нашего кластера будем использовать Ubuntu Server 16.04.3 LTS.

Все узлы будут работать на VirtualBox. Выставим следующие системные настройки для виртуальной машины: 10 GB пространства для жёсткого диска, два ядра и 1024 Мб памяти. Виртуальную машину можно оснастить двумя сетевыми адаптерами: один NAT, а другой для внутренней сети.

После того, как была скачена и установлена операционная система, необходимо обновиться и установить ssh и rsync:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install ssh
sudo apt-get install rsync
```

Для редактирования файлов с консоли будем использовать редактор nano.

Для его установки введем команду:

```
sudo apt-get install nano
```

Для запуска:

```
nano файл
```

или, если нужно редактировать системные файлы (с root правами), то

```
sudo nano файл
```

Для удобства также можно поставить оболочку Midnight Commander:

```
sudo apt-get install mc
```

для ее запуска

```
mc
```

или если хотите редактировать системные файлы (с root правами), то

```
sudo mc
```

Изменим имя узла на master в файле /etc/hostname.

1.3.2 Установка Java

Далее необходимо установить OpenJDK 8 версии:

```
sudo apt-get install openjdk-8-jdk
```

1.3.3 Создание отдельной учетной записи для запуска Hadoop

Мы будем использовать выделенную учетную запись для запуска Hadoop. Это не обязательно, но рекомендуется. Также предоставим новому пользователю права sudo, чтобы облегчить себе жизнь в будущем.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
sudo usermod -aG sudo hduser
```

Во время создания нового пользователя, необходимо будет ввести ему пароль.

1.3.4 Настройка статического IP-адреса

Для дальнейшей работы нам потребуются IP-адреса серверов. Для того чтобы узнать IP-адрес, можно воспользоваться командой

```
ifconfig
```

Вместо использования динамически выделенных адресов более удобным может оказаться использование статических адресов. Для настройки статического IP-адреса замените в файле /etc/network/interfaces для соответствующего интерфейса «dhcp» на «static» и укажите значения адреса, маски сети, шлюза и адрес DNS сервера для соответствия требованиям вашей сети:

```
auto enp0s3
iface enp0s3 inet static
  address 192.168.0.1
  netmask 255.255.255.0
  gateway 192.168.0.254
  dns-nameservers 192.168.0.254
```

В приведенных далее примерах для серверов используются адреса вида 192.168.0.X.

1.3.5 Настройка доменного имени узла

Нам необходимо, чтобы все узлы могли легко обращаться друг к другу. В большом кластере желательно использовать dns сервер, но для нашей маленькой конфигурации подойдет файл /etc/hosts. В нем мы будем описывать соответствие ip-адреса узла к его имени в сети. Для одного узла ваш файл должен выглядеть примерно так:

```
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.0.1 master
```

1.3.6 Настройка SSH

Для управления узлами кластера hadoop необходим доступ по ssh. Для созданного пользователя hduser предоставить доступ к master. Для начала необходимо сгенерировать новый ssh ключ:

```
ssh-keygen -t rsa -P ""
```

Следующим шагом необходимо добавить созданный ключ в список авторизованных:

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Проверяем работоспособность, подключившись к себе:

```
ssh master
```

Чтобы вернуться в локальную сессию, просто наберите:

```
exit
```

1.3.7 Отключение IPv6

Если не отключить IPv6, то в последствии можно получить много проблем. Для отключения IPv6 в Ubuntu 12.04 / 12.10 / 13.04 нужно отредактироватьфайл sysctl.conf:

```
sudo nano /etc/sysctl.conf
```

Добавляем следующие параметры:

```
# IPv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Сохраняем и перезагружаем операционную систему командой

```
sudo reboot
```

1.3.7 Установка Apache Hadoop

Скачаем необходимые файлы. Актуальные версии фреймворка располагаются по адресу: http://www.apache.org/dyn/closer.cgi/hadoop/common. Воспользуемся стабильной версией 2.7.4.

Создадим папку downloads в корневом каталоге и скачаем последнюю версию:

```
sudo mkdir /downloads
cd /downloads
sudo wget http://apache-mirror.rbc.ru/pub/apache/hadoop/common/stable/hadoop-
2.7.4.tar.gz
```

Распакуем содержимое пакета в /usr/local/, переименуем папку и выдадим пользователю hduser права создателя:

```
sudo mv /downloads/hadoop-2.7.4.tar.gz /usr/local/
cd /usr/local/
sudo tar xzf hadoop-2.7.4.tar.gz
sudo mv hadoop-2.7.4 hadoop
sudo chown -R hduser:hadoop hadoop
```

1.3.8 Обновление $HOME/.bashrc

Для удобства, добавим в .bashrc список переменных:

```
#Hadoop variables
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
```

На этом шаге заканчиваются предварительные подготовки.

1.3.9 Настройка Apache Hadoop

Вся последующая работа будет вестись из папки /usr/local/hadoop. Откроем etc/hadoop/hadoop-env.sh и зададим JAVA HOME:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

Опишем, какие у нас будут узлы в кластере в файле etc/hadoop/slaves:

```
master
```

Этот файл может располагаться только на главном узле. Все новые узлы необходимо описывать здесь.

Основные настройки системы располагаются в etc/hadoop/core-site.xml:

```
<configuration>
    <property>
       <name>fs.defaultFS</name>
       <value>hdfs://master:9000</value>
    </property>
```

```
</configuration>
```

Настройки HDFS лежат в etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/usr/local/hadoop/tmp/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/local/hadoop/tmp/hdfs/datanode</value>
    </property>
</configuration>
```

Здесь параметр dfs.replication задает количество реплик, которые будут хранится на файловой системе. По умолчанию его значение равно 3. Оно не может быть больше, чем количество узлов в кластере.

Параметры dfs.namenode.name.dir и dfs.datanode.data.dir задают пути, где будут физически располагаться данные и информация в HDFS. Необходимо заранее создать папку tmp.

Сообщим нашему кластеру, что мы желаем использовать YARN. Для этого изменим etc/hadoop/mapred-site.xml:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.job.reduces</name>
        <value>1</value>
    </property>
    <property>
        <name>mapreduce.task.io.sort.mb</name>
        <value>16</value>
    </property>
    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>mapreduce.map.cpu.vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>mapreduce.reduce.cpu.vcores</name>
        <value>1</value>
    </property>
```

```
    <property>
        <name>mapreduce.job.heap.memory-mb.ratio</name>
        <value>0.8</value>
    </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Djava.net.preferIPv4Stack=true -Xmx52428800</value>
    </property>
    <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Djava.net.preferIPv4Stack=true -Xmx52428800</value>
    </property>
</configuration>
```

Все настройки по работе YARN описываются в файле etc/hadoop/yarn-site.xml:

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>master:8030</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>master:8032</value>
    </property>
    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>master:8088</value>
    </property>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>master:8031</value>
    </property>
    <property>
        <name>yarn.resourcemanager.admin.address</name>
        <value>master:8033</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>yarn.scheduler.increment-allocation-vcores</name>
        <value>1</value>
```

```
    </property>
    <property>
       <name>yarn.scheduler.maximum-allocation-vcores</name>
       <value>2</value>
    </property>
    <property>
       <name>yarn.nodemanager.resource.memory-mb</name>
       <value>2060</value>
    </property>
    <property>
       <name>yarn.scheduler.minimum-allocation-mb</name>
       <value>1</value>
    </property>
    <property>
       <name>yarn.scheduler.increment-allocation-mb</name>
       <value>512</value>
    </property>
    <property>
       <name>yarn.scheduler.maximum-allocation-mb</name>
       <value>2316</value>
    </property>
    <property>
       <name>yarn.nodemanager.vmem-check-enabled</name>
       <value>false</value>
    </property>
</configuration>
```

Настройки resourcemanager нужны для того, чтобы все узлы кластераможно было видеть в панели управления.

Сменим пользователя на hduser:

```
su hduser
```

Отформатируем HDFS:

```
/usr/local/hadoop/bin/hdfs namenode –format
```

Запустим hadoop службы:

```
/usr/local/hadoop/sbin/start-dfs.sh
/usr/local/hadoop/sbin/start-yarn.sh
```

Необходимо убедиться, что запущены следующие java-процессы:

```
hduser@master:/usr/local/hadoop$ jps
4868 SecondaryNameNode
5243 NodeManager
5035 ResourceManager
4409 NameNode
4622 DataNode
5517 Jps
```

Теперь у нас есть готовый образ, который послужит основой для создания кластера. Далее можно создать требуемое количество копий нашего образа.

На копиях необходимо настроить сеть, сгенерировать новые MAC-адреса для сетевых интерфейсов, выдать им необходимые IP-адреса и поправить файл /etc/hosts на всех узлах кластера так, чтобы в нем были прописаны все соответствия. Например:

```
127.0.0.1       localhost
192.168.0.1     master
192.168.0.2     slave1
```

Заменим имя нового узла на slave1, для этого внесем изменения в файл /etc/hostname.

Сгенерируем на узле новые SSH-ключи и добавим их все в список авторизованных на узле master.

На каждом узле кластера изменим значения параметра dfs.replication в /usr/local/hadoop/etc/hadoop/hdfs-site.xml. Например, выставим везде значение 2:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
</configuration>
```

Добавим на узле master новый узел в файл /usr/local/hadoop/etc/hadoop/slaves:

```
master
slave1
```

1.3.10 Создание на главном узле файл подкачки

Создадим на главном узле папку, в которую попозже мы подмонтируем файл подкачки:

```
sudo mkdir /media/swap
```

Создаем файл подкачки:

```
sudo dd if=/dev/zero of=/media/swap/swapfile.img bs=2048 count=1M
```

Выставляем нужные права на файл:

```
sudo chmod 600 /media/swap/swapfile.img
```

Создаем swap:

```
sudo mkswap /media/swap/swapfile.img
```

Добавляем swap в fstab. Это нужно сделать чтобы каждый раз при старте ОС, автоматически монтировался файл подкачки, который мы создали, для этого открываем файл /etc/fstab в редакторе:

```
sudo nano /etc/fstab
```

и добавляем в файл:

```
# mount swap image
/media/swap/swapfile.img swap swap sw 0 0
```

Активируем (включаем) наш swap:

```
sudo swapon /media/swap/swapfile.img
```

Убедимся, что swap нормально работает. Для этого выполним:

```
cat /proc/swaps
```

1.3.11 Запуск Hadoop

Когда все настройки прописаны, то на главном узле можно запустить наш кластер:

```
/usr/local/hadoop/sbin/start-dfs.sh
/usr/local/hadoop/sbin/start-yarn.sh
```

На slave-узле должны запуститься следующие процессы:

```
hduser@slave1:/usr/local/hadoop$ jps
1748 Jps
1664 NodeManager
1448 DataNode
```

Теперь у нас есть свой мини-кластер. Посмотреть состояние нод кластера можно по адресу http://*master*:8088/cluster/nodes.

Давайте запустим задачу Word Count. Для этого нам потребуетсязагрузить в HDFS несколько текстовых файлов. Для примера, возьмём книги в формате txt с сайта Free ebooks – Project Gutenberg.

```
cd /home/hduser
mkdir books
cd books
wget http://www.gutenberg.org/files/20417/20417.txt
wget http://www.gutenberg.org/files/5000/5000-8.txt
wget http://www.gutenberg.org/files/4300/4300-0.txt
wget http://www.gutenberg.org/files/972/972.txt
```

Перенесем наши файлы в HDFS:

```
cd /usr/local/hadoop
bin/hdfs dfs -mkdir /in
bin/hdfs dfs -copyFromLocal /home/hduser/books/* /in
bin/hdfs dfs -ls /in
```

Запустим Word Count:

```
/usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoo
p-mapreduce-examples-2.7.4.jar wordcount /in /out
```

Отслеживать работу можно через консоль, а можно через веб-интерфейс ResourceManager'a по адресу http://*master*:8088/cluster/apps/.

По завершению работы, результат будет располагаться в папке /out в HDFS.

Для того, чтобы скачать его на локальную файловую систему выполним: `/usr/local/hadoop/bin/hdfs dfs -copyToLocal /out /home/hduser/`

Теперь в директории /home/hduser/out можно увидеть результаты выполнения задачи.

1.3.12 Дополнительные команды

Удаление директории из HDFS:

```
/usr/local/hadoop/bin/hdfs dfs –rm -r /out
```

Отмена режима HDFS только для чтения, возникшего из-за сбоя:

```
/usr/local/hadoop/bin/hdfs dfsadmin -safemode leave
```

Остановка Yarn:

```
/usr/local/hadoop/sbin/stop-yarn.sh
```

Остановка DFS:

```
/usr/local/hadoop/sbin/stop-dfs.sh
```

1.4 Контрольные вопросы

1 Для чего предназначен Hadoop?
2 Из каких компонентов состоит Hadoop?
3 В чем заключается парадигма MapReduce?
4 В чем заключается фаза map?
5 В чем заключается фаза reduce?
6 Какие преимущества дает использование парадигмы MapReduce при обработке «больших данных» (big data)?