МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

Кафедра «Безопасность информационных и автоматизированных систем»

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

Методические указания
к выполнению лабораторных работ для студентов
направлений 10.05.03 и 10.03.01

Курган 2018

Кафедра: «Безопасность информационных и автоматизированных систем».
Дисциплина: «Технология построения защищенных распределенных приложений».
Составил: канд. техн. наук, доцент Д.И. Дик.

Утверждены на заседании кафедры « 24 » ноября 2017 г.

Рекомендованы методическим советом университета « 12 » декабря 2016 г.

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1.
РАЗВЕРТЫВАНИЕ СИСТЕМЫ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ HADOOP

1.1 Цель работы

Цель лабораторной работы заключается в закреплении теоретических основ курса «Технологии построения распределенных защищенных приложений» и 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 MapReduce 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 отрабатывают операции 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 отношению к 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");
```

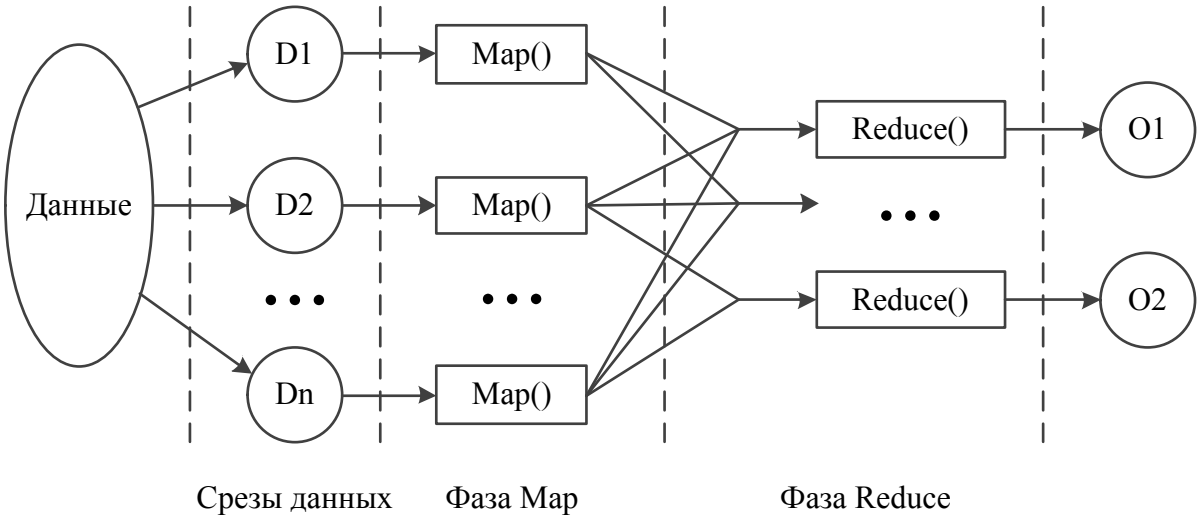

Рисунок 1 — Упрощенная схема потоков данных в парадигме MapReduce

```
// 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, но по 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;

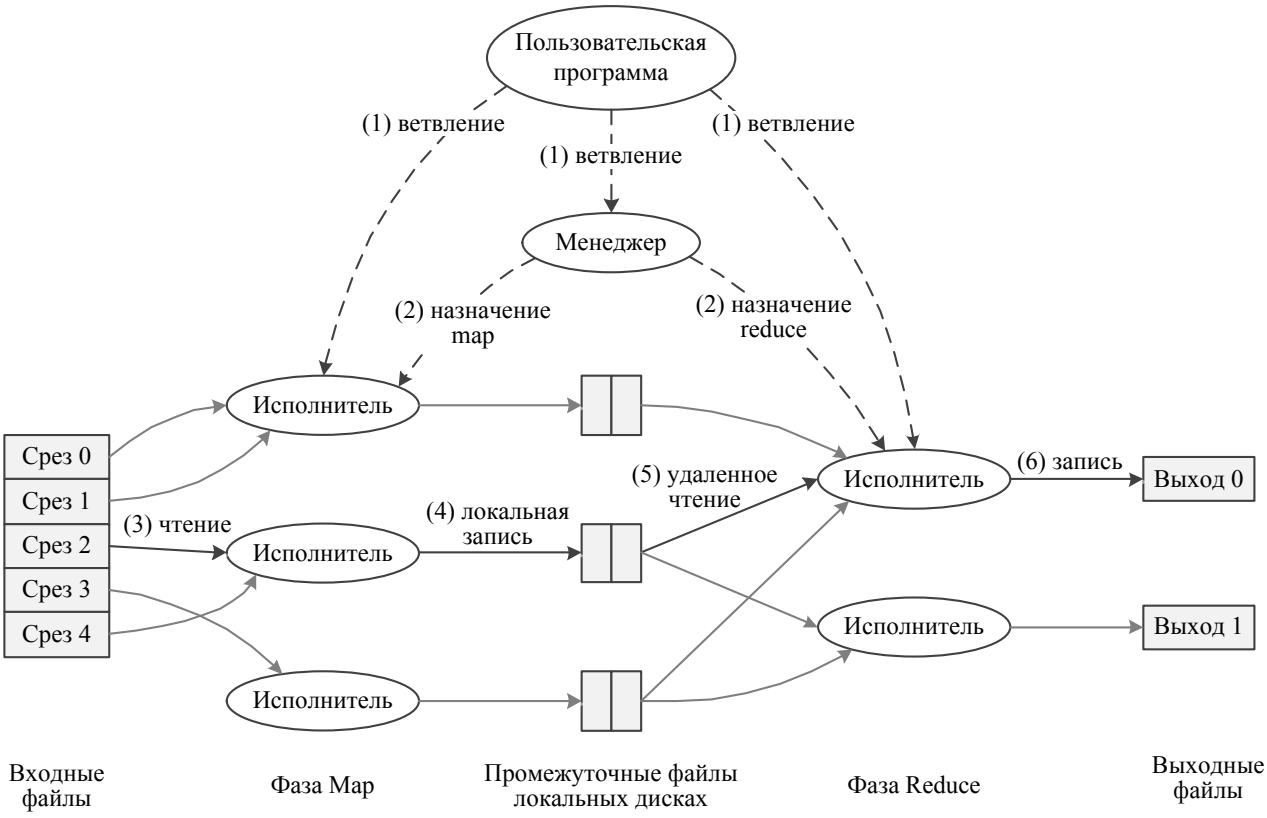

Рисунок 2 — Ход выполнения вызова MapReduce

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.

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 поставит 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 отредактировать файл 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
```

1.3.8 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
```

1.3.9 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.

1.3.10 Настройка Apache Hadoop

Все 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 кластера можно 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
```

1.3.11 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
```

1.3.12 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 потребуется загрузить 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)?

ЛАБОРАТОРНАЯ РАБОТА № 2.
НАСТРОЙКА РЕПЛИКАЦИИ НА СУБД MYSQL

2.1 Цель работы

Цель лабораторной работы заключается в закреплении теоретических основ курса «Технологии построения распределенных защищенных приложений» и получении первоначальных навыков настройки репликации на СУБД MySQL.

2.2 Общие сведение

2.2.1 Репликация данных

Репликация (англ. replication) — механизм синхронизации содержимого нескольких копий объекта (например, содержимого базы данных). Под репликацией также понимают процесс копирования данных из одного источника на другой (или на множество других).

Репликация является одной из техник масштабирования данных в распределенных системах. Репликация делятся на синхронные и асинхронные.

В случае синхронной репликации, если одна реплика (копия) обновляется, все другие реплики того же фрагмента данных также должны быть обновлены в одной и той же транзакции. Это означает, что все реплики остаются непротиворечивыми. Недостатком данного метода являются высокие накладные расходы на синхронизацию реплик.

В случае асинхронной репликации обновление одной реплики распространяется на другие спустя некоторое время, а не в той же транзакции. Таким образом, при асинхронной репликации вводится задержка, или время ожидания, в течение которого отдельные реплики могут быть фактически неидентичными. В тоже время накладные расходы на репликацию существенно уменьшаются.

Существует два основных подхода при работе с репликацией данных в MySQL:

− репликация master-slave (ведущий – ведомый, главный – подчиненный, мастер – слейв);

− репликация master-master (мастер – мастер) или в более общем виде multimaster (мультимастер).

2.2.2 Master-slave репликация

В этом подходе выделяется один основной сервер базы данных, который называется ведущим. На нем происходят все изменения в данных (любые запросы MySQL INSERT/UPDATE/DELETE). Ведомый сервер постоянно копирует все изменения с Мастера. С приложения на ведомый сервер отправляются запросы чтения данных (запросы SELECT). Таким образом,

ведущий сервер отвечает за изменения данных, а ведомый за чтение (рисунок 3).

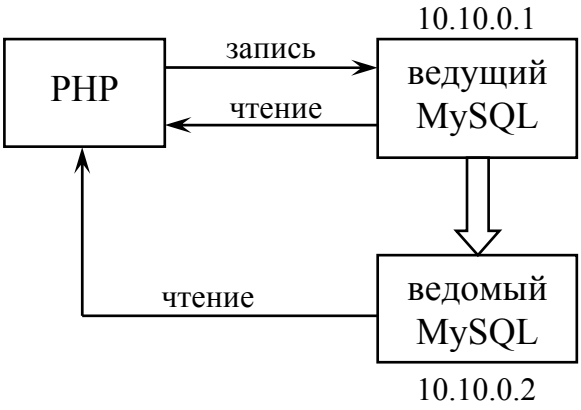

Рисунок 3 — Master-slave репликация

В приложении нужно использовать два соединения — одно для ведущего сервера, второе для ведомого.

```
<?
$master = mysql_connect('10.10.0.1', 'root', 'pwd');
$slave = mysql_connect('10.10.0.2', 'root', 'pwd');

# ...
mysql_query('INSERT INTO users ...', $master);

# ...
$q = mysql_query('SELECT * FROM photos ...', $slave);
# Используем два соединения для записи и чтения соответственно
```

2.2.3 Master-slave репликация на несколько slave серверов

Преимущество этого типа репликации в том, что можно использовать более одного ведомого сервера. Обычно следует использовать не более 20 ведомых серверов при работе с одним ведущим (рисунок 4).

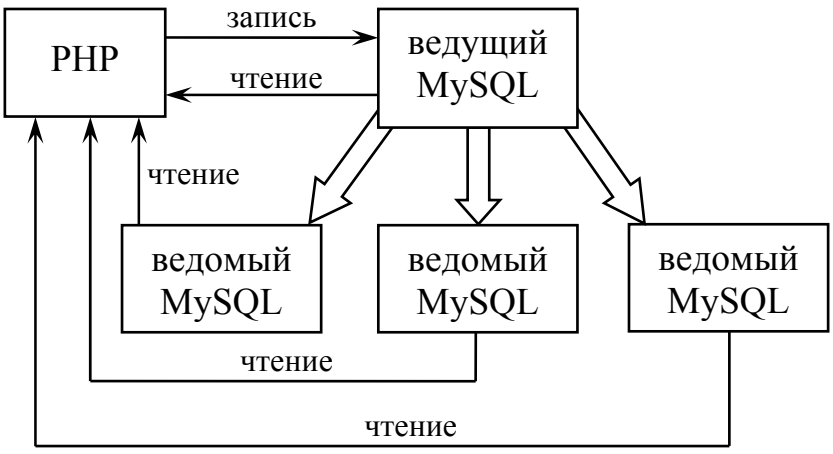

Рисунок 4 — Master-slave репликация на несколько slave серверов

Тогда в приложении один из ведомых серверов выбирается для обработки запросов случайным образом:

```
<?
$master = mysql_connect('10.10.0.1', 'root', 'pwd');
$slaves = [
    '10.10.0.2',
    '10.10.0.3',
    '10.10.0.4',
];
$slave = mysql_connect($slaves[array_rand($slaves)], 'root', 'pwd');

# ...
mysql_query('INSERT INTO users ...', $master);

# ...
$q = mysql_query('SELECT * FROM photos ...', $slave);
```

2.2.4 Задержка репликации

MySQL реализует асинхронную репликацию. Это означает, что данные на ведомом сервере могут появиться с небольшой задержкой. Поэтому, в последовательных операциях необходимо использовать чтение с ведущего сервера, чтобы получить актуальные данные:

```
<?
$master = mysql_connect('10.10.0.1', 'root', 'pwd');
$slave = mysql_connect('10.10.0.2', 'root', 'pwd');

# ...
mysql_query('UPDATE users SET age = 25 WHERE id = 7', $master);
$q = mysql_query ('SELECT * FROM users WHERE id = 7', $master);
# При обращении к изменяемым данным, необходимо использовать
# соединение с ведущим сервером

# ...
$q = mysql_query('SELECT * FROM photos ...', $slave);
```

2.2.5 Выход из строя сервера при master-slave репликации

При выходе из строя ведомого сервера, достаточно просто переключить все приложения на работу с ведущим. После этого восстановить репликацию на ведомом сервере и снова его запустить.

Если выходит из строя ведущий сервер, нужно переключить все операции (и чтения и записи) на ведомый. Таким образом, он станет новым ведущим. После восстановления старого ведущего, настроить на нем реплику, и он станет новым ведомым.

2.2.6 Резервирование

Намного чаще репликацию master-slave используют не для масштабирования, а для резервирования. В этом случае, ведущий сервер обрабатывает все запросы от приложения. Ведомый сервер работает в

пассивном режиме. Но в случае выхода из строя ведущего, все операции переключаются на ведомый.

2.2.7 Master-master репликация

В этой схеме, любой из серверов может использоваться как для чтения, так и для записи (рисунок 5).

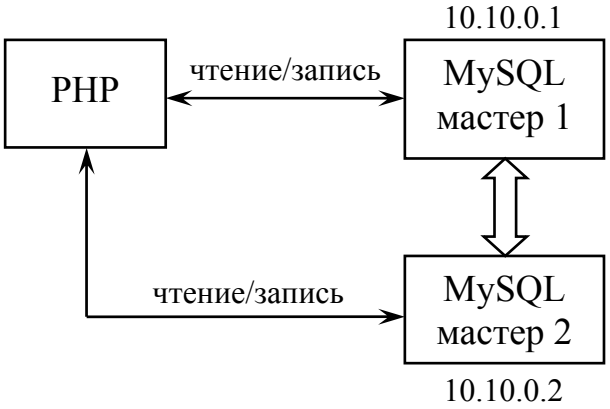

Рисунок 5 — Master-master репликация

При использовании такого типа репликации достаточно выбирать случайное соединение из доступных мастеров:

```
<?
$masters = [
    '10.10.0.1',
    '10.10.0.2',
    '10.10.0.3',
];
$master = mysql_connect($masters[array_rand($masters)], 'root', 'pwd');

# ...
mysql_query('INSERT INTO users ...', $master);
# Выбор случайного Мастера для обработки соединений
```

2.2.8 Выход из строя сервера при master-master репликации

Вероятные поломки делают master-master репликацию непривлекательной. Выход из строя одного из серверов практически всегда приводит к потере каких-то данных. Последующее восстановление также сильно затрудняется необходимостью ручного анализа данных, которые либо успели, либо не успели скопироваться.

2.2.9 Проблемы репликации в MySQL

Изначально в MySQL использовалась асинхронная репликация. Принцип работы очень прост, клиент производит изменение (commit), которое выполняется на ведущем сервере, результат заносится в бинарный журнал.

Клиент, не дожидаясь ответа подчиненных серверов, сразу же получает сообщение об успешном завершении процесса. Такой подход обеспечивает максимальную скорость и позволяет производить репликацию даже по коммутируемым соединениям, но имеет один существенный минус, особенно актуальный для кластеров. Если после подтверждения транзакции ведущий сервер выходит из строя, а подчиненные сервера еще не успевают получить реплику, данные будут потеряны, но клиент будет уверен, что все в порядке. В случае мультимастер репликации проблема только усугубляется.

В MySQL 5.5 появилась поддержка полу-синхронного (semi-synchronous) механизма репликации, основанного на патчах к InnoDB от компании Google. В этом случае ведущий сервер ожидает подтверждения от одного из ведомых, который сообщает, что получил и записал на диск реплику. При этом не ожидается, когда ведомый выполнит сам запрос, и нет никаких гарантий, что он вообще будет выполнен (например, ошибка).

Чтобы не быть зависимым от доступности подчиненного сервера, предусмотрен таймаут, позволяющий автоматически перейти асинхронный режим, если ни один из ведомых серверов не ответил за запрос. При восстановлении связи будет опять активирован semi-synchronous.

Также поддерживается отложенная репликация, когда подчиненный сервер начинает выполнять запрос после истечения указанного промежутка времени (устанавливается при помощи MASTER_DELAY, по умолчанию 0). Это может быть полезно, если на ведомом сервере понадобится откатить транзакцию.

По умолчанию используется асинхронная репликация, чтобы включить полу-синхронную, необходимо установить специальный плагин и активировать параметр в my.cnf.

Посмотрим, чем нам грозит асинхронность репликации. Данные между базами данных передаются с произвольной задержкой (от миллисекунд до дней). Для архитектуры master-slave с ведомого сервера можно просто не читать данные, отставшие, допустим, на 30 секунд. А вот для master-master все хуже — у нас нет и не будет (даже в случае полу-синхронной репликации) никаких гарантий, что копии БД — синхронны. Т.е. один и тот же запрос может выполняться по-разному на каждой из БД. А одновременное выполнение команд на разных серверах:

```
UPDATE mytable SET mycol=mycol+1;
UPDATE mytable SET mycol=mycol*3;
```

также приведет к рассинхронизации данных в обеих БД.

Одновременная вставка в обе БД одинакового уникального значения столбца — приведет к остановке репликации по ошибке (при авто инкременте такой проблемы не возникает). Таких «жутких» примеров можно привести множество. И хотя иногда советуют решения типа «ON DUPLICATE KEY UPDATE», игнорирование ошибок и пр. и заодно «перелопатить» приложение — здравый смысл подсказывает, что подобные подходы — скользкие и ненадежные.

Очевидно, к какому коллапсу и несогласованности это может привести ваше приложение.

Как результат, использовать асинхронный master-master для одновременной записи в обе БД без знания подводных камней — опасно и ненадежно и применяется в редких случаях.

2.2.10 Польза от асинхронной репликации

Однако, все не так печально. Как бы не ругали классическую MySQL master-slave репликацию за:

– асинхронность (рассинхронизация данных на узлах, отставания и т.п.);

– недостаточную надежность (flush_log_at_trx_commit=1, sync_binlog=1, sync_relay_log=1, sync_relay_log_info=1, sync_master_info=1, — иногда не достаточно, и репликация при рестарте сервера отваливается);

– недостаточную поддержка транзакционности,

master-slave репликация используется очень широко и приносит массу пользы системным администраторам:

– для создания горячей почти актуальной резервной копии;

– для кластеризации чтений;

– резервирования при вертикальном шардинге (разделение таблиц по разным серверам);

– для создания резервной копии без чрезмерной нагрузки рабочего сервера БД;

– и так далее.

Пользу можно извлечь и из асинхронной master-master репликации. При этом используется так называемая active-passive архитектура.

Идея проста: пишем в одну БД, вторая используется как горячая резервная копия, в которую при необходимости можно быстро начать писать данные, получая, таким образом, высокую степень доступности.

2.3 Порядок выполнения работы

2.3.1 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 512 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:

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

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 поставит 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.

2.3.2 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.

2.3.3 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
```

2.3.4 Настройка SSH

Для копирования базы данных с ведущего сервера на ведомый нам потребуется доступ по ssh. 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
```

2.3.5 Копирование образа виртуальной машины

Теперь у нас есть готовый образ, который послужит основой для создания репликации. Далее создадим копию образа. При клонировании образа необходимо сгенерировать новые MAC-адреса для сетевых интерфейсов и выдать им необходимые IP-адреса, выдать им необходимые IP-адреса и поправить файл /etc/hosts на всех узлах так, чтобы в нем были прописаны все соответствия:

```
127.0.0.1       localhost

192.168.0.1     master
192.168.0.2     slave
```

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

2.3.6 Установка MySQL сервера

На каждый из созданных узлов установим MySQL сервер:

```
sudo apt-get install mysql-server
```

Конфигурация MySQL сервера содержится в файле /etc/mysql/my.cnf, который подключает конфигурационные файлы из директорий /etc/mysql/conf.d и /etc/mysql/mysql.conf.d.

По умолчанию MySQL сервер принимает соединения только с локальной машины. Для того, чтобы разрешить подключаться к нему с других машин замените строку в файле /etc/mysql/mysql.conf.d/mysqld.cnf

```
bind-address            = 127.0.0.1
```

на

```
#разрешить подключатся с любого хоста
bind-address            = 0.0.0.0
```

на каждом из узлов.

2.3.7 Настройка master-slave репликация

2.3.7.1 Настройка ведущего сервера

На сервере, который будет выступать мастером, необходимо внести правки в /etc/mysql/mysql.conf.d/mysqld.cnf:

```
# выбираем ID сервера, произвольное число, лучше начинать с 1
server-id = 1

# путь к бинарному логу
log_bin = /var/log/mysql/mysql-bin.log

# название Вашей базы данных, которая будет реплицироваться
binlog_do_db = clusterdb
```

Перезапускаем Mysql:

```
sudo /etc/init.d/mysql restart
```

либо

```
sudo service mysql restart
```

Для управления демоном MySql можно использовать следующие команды:

– остановка сервиса MySql

```
sudo service mysql stop
```

– запуск сервиса

```
sudo service mysql start
```

– узнать состояние сервиса

```
sudo service mysql status
```

2.3.7.2 Создание базы данных на ведущем сервере

Для создания базы данных запустим консоль MySql:

```
mysql -u root -p --prompt='master> '
```

Создаем базу данных, создаем таблицу и вставляем в нее данные

```
create database clusterdb;
create table clusterdb.simples (id int not null primary key);
```

```
insert into clusterdb.simples values (999),(1),(2),(3);
```

Посмотрим содержимое таблицы

```
select * from clusterdb.simples;
```

2.3.7.3 Настройка прав на репликацию

Далее необходимо создать профиль пользователя, из-под которого будет происходить репликация. Для этого запускаем консоль:

```
mysql -u root -p --prompt='master> '
```

Далее создаем и назначаем права пользователю для реплики (даем права пользователю slave_user с паролем slavepass):

```
grant replication slave on *.* to 'slave_user'@'%' identified by 'slavepass';
```

Альтернативно можно вначале создать пользователя и лишь, затем дать ему права

```
create user 'repl'@'%.mydomain.com' identified by 'slavepass';
grant replication slave on *.* to 'slave_user'@'%';
```

Обновим права доступа:

```
flush privileges;
```

2.3.7.4 Получение статуса ведущего сервера

Далее блокируем все таблицы в нашей базе данных:

```
use clusterdb;
flush tables with read lock;
```

Проверяем статус ведущего сервера:

```
show master status;
```

Мы увидим что-то похожее на:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000004 |     1243 | clusterdb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Выделенные значения мы будем использовать для запуска ведомого сервера.

Выходим из консоли mysql:

```
quit
```

2.3.7.5 Создание дампа базы данных

Теперь необходимо сделать дамп базы данных:

```
mysqldump -u root -p clusterdb > /home/user/clusterdb.sql
```

Снова заходим в консоль

```
mysql -u root -p --prompt='master> '
```

и разблокируем таблицы в консоли mysql:

```
use clusterdb;
unlock tables;
```

2.3.7.6 Создание базы на ведомом сервере

Копируем файл clusterdb.sql с ведущего сервера на ведомый:

```
scp user@master:/home/user/clusterdb.sql /home/user/clusterdb.sql
```

Заходим в консоль

```
mysql -u root -p --prompt='slave> '
```

В консоли mysql на ведомом сервере создаем базу с таким же именем, как и на ведущем:

```
create database clusterdb;
```

Выходим из консоли и загружаем дамп (через bash):

```
mysql -u root -p clusterdb < /home/user/clusterdb.sql
```

2.3.7.7 Настройка ведомого сервера

В настройках my.cnf на ведомом сервере необходимо указать такие параметры:

```
# ID ведомого сервера, удобно выбирать следующим числом после Мастера
server-id = 2

# Путь к relay логу
relay-log = /var/log/mysql/mysql-relay-bin.log

# Путь к bin логу на Мастере
log_bin = /var/log/mysql/mysql-bin.log

# База данных для репликации
binlog_do_db = clusterdb
```

Перезапускаем MySql на ведомом сервере:

```
sudo /etc/init.d/mysql restart
```

2.3.7.8 Активации репликации на ведомом сервере

Нам осталось включить репликацию, для этого необходимо указать параметры подключения к мастеру. В консоли mysql на ведомом сервере необходимо выполнить запрос:

```
change master to master_host='master', master_user='slave_user',
master_password='slavepass',
master_log_file = 'mysql-bin.000004', master_log_pos = 1243;
```

Указанные значения мы берем из настроек ведущего сервера.

После этого запускаем репликацию на ведомом сервере:

```
start slave;
```

Проверить работу репликации можно запросом на ведомом сервере:

```
slave> show slave status\G
               Slave_IO_State: Waiting for master to send event
                  Master_Host: master
                  Master_User: slave_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 1243
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 548
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 1243
              Relay_Log_Space: 527
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_name:
1 row in set (0.00 sec)
```

2.3.7.9 Проверка работы репликации

Через mysql консоль на ведущем сервере добавим новую запись в таблицу simples нашей базы данных:

```
master> insert into clusterdb.simples values (5);
```

Через mysql консоль на ведомом сервере посмотрим содержимое таблицы и удостоверимся, что ведомый сревер получил изменения

```
slave> select * from clusterdb.simples;
```

2.3.8 Настройка master-master репликация

Перенастроим Master-Slave репликацию в Master-Master репликацию (перенастроим ведомый сервер во второй мастер).

2.3.8.1 Настройка прав на репликацию

На будущем втором мастер-сервере (бывшем ведомом) создадим пользователя и назначаем ему права для реплики (даем права пользователю replicator с паролем password):

```
grant replication slave on *.* to 'replicator'@'%' identified by 'password';
```

Проверяем статус второго мастер-сервера (бывшего ведомого):

```
SHOW MASTER STATUS;
```

Мы увидим что-то похожее на:

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      443 | clusterdb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Выделенные значения мы будем использовать для запуска репликации на первом мастере.

2.3.8.2 Активации репликации на первом мастер-сервере

Нам осталось включить репликацию на первом мастере, для этого необходимо указать параметры подключения ко второму мастеру. В консоли mysql на первом мастере необходимо выполнить запросы:

```
stop slave;
change master to master_host='slave', master_user='replicator',
master_password='password',
master_log_file = 'mysql-bin.000001', master_log_pos = 443;
start slave;
```

2.3.8.3 Проверка работы репликации

Через MySql консоль на втором мастере (бывшем ведомом) добавим новую запись в таблицу simples нашей базы данных

```
insert into clusterdb.simples values (10);
```

Через MySql консоль на первом мастере посмотрим содержимое таблицы и удостоверимся, что первый мастер получил изменения

```
select * from clusterdb.simples;
```

2.4 Контрольные вопросы

1 Что такое репликация данных?
2 В чем отличие синхронной и асинхронной репликации?
3 В чем заключается master-slave репликация?
4 В чем заключается master-master репликация?
5 Недостатки асинхронной репликации?
6 Области применения асинхронной репликации?

ЛАБОРАТОРНАЯ РАБОТА № 3.
РАЗВЕРТЫВАНИЕ PERCONA XTRADB CLUSTER

3.1 Цель работы

Цель лабораторной работы заключается в закреплении теоретических основ курса «Технологии построения распределенных защищенных приложений» и получении первоначальных навыков настройки кластера MySQL серверов, на основе Percona XtraDB Cluster.

3.2 Общие сведение

3.2.1 Общая информация о Percona XtraDB Cluster

Для гарантии целостности данных и возможности писать на все узлы кластера одновременно необходимо осуществление синхронной master-master репликации.

Для MySQL настоящее время наиболее известны следующие решения:

- MySQL NDB Cluster;
- Percona XtraDB Cluster;
- MariaDB Galera Cluster.

MySQL NDB Cluster (от производителя MySQL) в настоящее время еще слишком ограничен по своим возможностям. Percona XtraDB Cluster и MariaDB Galera Cluster построены на одном базовом решении «Galera synchronous multi-master replication library» и очень близки по своих возможностям.

Percona XtraDB Cluster – кластерная СУБД, предоставляющая решение для создания кластеров с синхронной репликаций между узлами, работающими в режиме multi-master. Percona XtraDB Cluster обеспечивает высокую производительность, быстрое восстановление узла кластера после падения и полный контроль состояния кластера. Исходные тексты проекта распространяются под лицензией GPLv2.

Основные преимущества:

– синхронная репликация – транзакция проходит либо на всех узлах, либо ни на одном;

- режим multi-master – писать данные можно в любой узел;
- параллельная репликация;
- высокая консистентность данных;
- полная совместимость с базами данных MySQL;
- полная совместимость с приложениями, работающими с MySQL;
- балансировщик нагрузки ProxySQL;
- легко настраиваемое зашифрованное общение между узлами;
- высокая масштабируемость.

Кластер состоит из множества узлов. Рекомендуется использовать хотя бы три узла, хотя возможно и использование и двух узлов.

В каждом узле находится обычный сервер MySQL или Precona Server. Таким образом, уже существующий сервер MySQL или Precona Server можно добавить в кластер, и наоборот: любой узел можно отсоединить от кластера и использовать как обычный сервер. В каждом узле находится полная копия данных.

Преимущества такой схемы:

– при исполнении запроса, он исполняется локально на узле. Все данные можно найти на локальной машине, без необходимости удаленного доступа;

– отсутствует централизация. Любой узел можно отсоединить в любой момент, и кластер продолжит работать;

– отличное решения для большого количества запросов на чтение. Их можно направлять к любому узлу.

Недостатки такой схемы:

– присоединение нового узла требует значительных ресурсов. Новый узел должен получить полную копию данных базы от одного из существующих узлов. Если база составляет 100 GB, то придется копировать все 100 GB;

– неоптимальное масштабирование для большого количества запросов записи. Есть возможность увеличить производительность за счет направления трафика на несколько узлов, но суть проблемы от этого не меняется - все записи должны попадать на все узлы.

Ограничения:

– в настоящее время репликация работает только с движком InnoDB. Записи в таблицы, работающие на другом движке, не будут реплицированы. Впрочем, DLL-выражения реплицируются на уровне запросов, и изменения таблиц mysql.* все-таки будут реплицированы. Так что можно спокойно писать «CREATE USER...», а вот «INSERT INTO mysql.user...» реплицировано не будет;

– неподдерживаемые запросы: LOCK/UNLOCK (принципиально невозможно в multi-master схеме) и сходные им функции;

– лог запросов нельзя класть в базу. Если вы хотите логировать запросы к базе, лог необходимо направлять в файл;

– максимальный размер транзакции определен значениями wsrep_max_ws_rows и wsrep_max_ws_size. LOAD DATA INFILE будет совершать коммит каждые 10 000 строк. Так что большие транзакции будут разделены на серию маленьких;

– из-за контроля многопоточности на уровне кластера, транзакции, совершающие COMMIT, все еще могут быть прерваны на этом этапе. Могут быть две транзакции, пишущие в одну и туже запись разных узлах, и только одна из них будет успешно завершена. Другая будет прервана на уровне кластера (Error: 1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK)). Это еще раз подтверждает, что масштабируемость поддерживается, в основном, для большого объема чтения, но не записи;

– XA транзакции не поддерживаются из-за возможного rollback-а на этапе коммита;

– способности кластера по записи ограничиваются возможностями самого слабого узла. Если один узел будет замедлен, с ним замедлится и весь кластер. Если вам необходима стабильная высокая производительность, она должна быть поддержана вашим аппаратным обеспечением.

3.2.2 Проблема split-brain и использование кворума

Минимальный рекомендованный размер кластера – 3 узла, что связано с проблемой split-brain.

Рассмотрим подробнее данную проблему. Допустим, в нашей системе ровно два совершенно одинаковых узла — A и B. Каждый из них хранит копию данных второго и может независимо обрабатывать запросы извне. Каждый, обрабатывая запрос, уведомляет второго об изменениях, чтобы данные оставались согласованы.

В случае аварии возможно два варианта:

1 Узел A выходит из строя. Система продолжает работать как ни в чем не бывало — B продолжает обрабатывать запросы. Когда A приведут в чувство, он первым делом синхронизируется с B и они вдвоем продолжат работать дальше. Ни доступность, ни согласованность не страдают.

2 Потеряна связь между A и B. При этом каждый из них продолжает принимать запросы извне, но не может уведомить второго об изменениях. Для каждого узла все выглядит так, будто второй узел «умер» и он действует в одиночку. Эту ситуацию часто называют «split-brain» — мозг разделился на два полушария, каждое из которых считает себя единственным хозяином ситуации. Если в этот момент на A был обработан запрос на удаление некой записи R, а на B был обработан запрос на модификацию той же самой записи, то данные стали не согласованы. Когда связь между A и B восстановится, при синхронизации всплывет конфликт — удалить R или оставить модифицированную версию? Согласованность данных утеряна.

Для решения данной проблемы Percona XtraDB Cluster в конфигурации с двумя узлами работает до тех пор, пока не пропадет связь между узлами. В случае исчезновения связи кластер полностью блокируется.

Теперь посмотрим, как ведет себя кластер из трех узлов. Будем считать, что все узлы имеют одинаковый вес, что является значением по умолчанию.

Что произойдет, если узел 1 штатно остановлен (выполнена остановка сервиса MySQL)? При завершении работы узел 1 сообщит, что он покидает кластер. Теперь у нас есть кластера из двух узлов, на оставшихся членов кластера приходится 2/2 = 100% голосов. Кластер продолжает работать нормально.

Что произойдет, если и узел 2 штатно будет остановлен? Теперь узел 3 знает, что узел 2 больше не является частью кластера. Узел 3 теперь имеет 1/1 = 100% голосов и кластер с одним узлом может продолжать работать.

В этих сценариях нет необходимости в достижении кворума при голосовании, так как остальные узлы всегда знают, что произошло с узлами, покидающими кластер.

Теперь рассмотрим другую ситуацию. На том же 3-узловом кластере, где запущены все 3 узла, если узел 1 аварийно завершит работу.

На этот раз узлы 2 и 3 должны провести голосование, чтобы определить, имеют ли они кворум и могут ли безопасно продолжить работу. Они имеют 2/3 голосов, 2/3 > 50%, так что оставшиеся 2 узла имеют кворум и продолжают работать нормально.

Необходимо иметь в виду, что голосование происходит не сразу, когда узлы 2 и 3 не могут присоединиться к узлу 1. Голосование происходит только после достижения тайм-аута (равного по умолчанию 5 секундам). Это позволяет кластеру быть устойчивым к кратковременным сбоям сети и может быть весьма полезны при работе кластера через глобальную сеть. Следствием является то, что в случае сбоя узла операции записи останавливаются на время тайм-аута.

Что произойдет, если узел 2 также выйдет из строя? Опять же необходимо провести голосование. На этот раз узел 3 имеет только 1/2 голосов, что не превышает 50% голосов. Узел 3 не имеет кворума, поэтому он прекращает обработку чтения и записи.

Если посмотреть в этом случае на переменную состояния wsrep_cluster_status на оставшемся узле, она покажет значение NON_PRIMARY. Это означает, что узел не является частью основного компонента.

3.2.3 Особенности репликации в Percona XtraDB Cluster

В Percona XtraDB Cluster можно выполнять запись на любом узле и при этом кластер гарантирует консистентность записи. То есть, запись либо произойдет на всех узлах, либо ни на одном.

Все запросы исполняются локально на узле, особым образом обрабатывается только коммит (commit). Как только дана команда коммит, транзакция должна пройти верификацию (подтверждение коммита) на всех узлах. Транзакция подтверждается только, если верификация проходит успешно, в противном случае происходит откат транзакции.

Из этой архитектуры вытекают два важных следствия:

– во-первых, существует небольшой период времени, когда slave не синхронизирован с master. Это происходит, потому что master может зафиксировать коммит быстрее, чем slave. И если в этот момент происходит чтение из slave, есть вероятность прочитать еще не изменившиеся данные. Впрочем, это поведение можно изменить, если установив переменную настройки wsrep_causal_reads=ON. В этом случае чтение на slave будет ожидать, пока коммита не будет зафиксирован, что впрочем, увеличивает время отклика на операцию чтения.

– во-вторых, если ведется запись на два разных узла, кластер будет использовать оптимистичную модель блокировки. То есть, транзакция не будет проверять возможные конфликты блокировок между узлами во время отдельных запросов, а только лишь на стадии коммита. Таким образом, можно

получить ошибку при выполнении коммита, что в обычном InnoDB MySQL не возможно. В InnoDB MySQL ошибки вида DEADLOCK и LOCK TIMEOUT могут возникать только в ответ на отдельные запросы, но не на коммит. Поэтому приложения, использующее Percona XtraDB Cluster, должны в обязательном порядке проверять ошибку коммита.

3.3 Порядок выполнения работы

3.3.1 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 512 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 поставит оболочку Midnight Commander

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

для ее запуска

```
mc
```

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

```
sudo mc
```

3.3.2 Добавление репозитария Percona

Скачаем пакет репозитария:

```
wget https://repo.percona.com/apt/percona-release_0.1-4.$(lsb_release -
sc)_all.deb
```

Инсталлируем скачанный пакет командой

```
sudo dpkg -i percona-release_0.1-4.$(lsb_release -sc)_all.deb
```

Обновим локальный **apt** кэш:

```
sudo apt-get update
```

Удостоверимся, что пакеты Percona стали доступными:

```
sudo apt-cache search percona
```

Вы увидите что-то подобное нижеследующему:

```
percona-xtrabackup-dbg - Debug symbols for Percona XtraBackup
percona-xtrabackup-test - Test suite for Percona XtraBackup
percona-xtradb-cluster-client - Percona XtraDB Cluster database client
percona-xtradb-cluster-server - Percona XtraDB Cluster database server
percona-xtradb-cluster-testsuite - Percona XtraDB Cluster database regression
test suite
percona-xtradb-cluster-testsuite-5.5 - Percona Server database test suite
...
```

Теперь у нас есть готовый образ, который послужит основой для создания кластера.

Далее создадим четыре копии нашего образа. Три будут использоваться для серверов кластера и четвертая для приложения проксирования SQL-запросов к базам данных.

При клонировании образов необходимо сгенерировать новые MAC-адреса для сетевых интерфейсов и выдать им необходимые IP-адреса.

3.3.3 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 192.168.1.X (таблица 1).

Таблица 1 – IP-адреса серверов кластера

Узел	Имя узла	IP-адрес
1	pxc1	192.168.1.164
2	pxc2	192.168.1.165
3	pxc3	192.168.1.166
4	proxysql	192.168.1.167

Первые три узла будут использоваться для Percona XtraDB Cluster, а четвертый для ProxySQL.

3.3.4 Установка Percona XtraDB Cluster

Установим пакет Percona XtraDB Cluster

```
sudo apt-get install percona-xtradb-cluster-57
```

Остановим сервис mysql:

```
sudo service mysql stop
```

Добавим на первом узле в конфигурационный файл /etc/mysql/percona-xtradb-cluster.conf.d/wsrep.cnf следующие переменные:

```
wsrep_provider=/usr/lib/libgalera_smm.so

wsrep_cluster_name=pxc-cluster
wsrep_cluster_address=gcomm://192.168.1.164,192.168.1.165,192.168.1.166

wsrep_node_name=pxc1
wsrep_node_address=192.168.1.164

wsrep_sst_method=xtrabackup-v2
wsrep_sst_auth=sstuser:passw0rd

pxc_strict_mode=ENFORCING

binlog_format=ROW
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
```

На остальных узлах добавим те же конфигурационные значения за исключением переменных wsrep_node_name и wsrep_node_address.

На втором узле установим значения переменных wsrep_node_name и wsrep_node_address в:

```
wsrep_node_name=pxc2
wsrep_node_address=192.168.1.165
```

На третьем узле:

```
wsrep_node_name=pxc3
wsrep_node_address=192.168.1.166
```

После конфигурирования узлов, инициализируем кластер развернув первый узел. Первый узел должна быть единственной содержащей реплицируемые данные на момент инициализации кластера.

Развертывание подразумевает запуск узла с пустой конфигурационнной переменной wsrep_cluster_address (без указания адресов машин входящих в кластер).

Чтобы не менять конфигурационный файл, можно воспользоваться командой:

```
sudo /etc/init.d/mysql bootstrap-pxc
```

При выполнении данной команды узел запускается в режиме развертывания с wsrep_cluster_address=gcomm://. После добавления остальных узлов можно перезапустить узел в нормальном режиме с использованием стандартной конфигурации.

Удостоверимся что кластер был успешно инициализирован. Для этого войдем в консоль mysql:

```
mysql -u root –p
```

и выполним команду

```
mysql@pxc1> show status like 'wsrep%';
+----------------------------------+--------------------------------------+
| Variable_name                    | Value                                |
+----------------------------------+--------------------------------------+
| wsrep_local_state_uuid           | 7e4a3888-d451-11e7-8bf9-037f76f76cc8 |
| ...                              | ...                                  |
| wsrep_local_state                | 4                                    |
| wsrep_local_state_comment        | Synced                               |
| ...                              | ...                                  |
| wsrep_incoming_addresses         | 192.168.1.164:3306                   |
| ...                              | ...                                  |
| wsrep_cluster_size               | 1                                    |
| wsrep_cluster_state_uuid         | 7e4a3888-d451-11e7-8bf9-037f76f76cc8 |
| wsrep_cluster_status             | Primary                              |
| wsrep_connected                  | ON                                   |
| ...                              | ...                                  |
| wsrep_ready                      | ON                                   |
+----------------------------------+--------------------------------------+
67 rows in set (0.01 sec)
```

Результат выполнения команды показывает, что кластер содержит 1 узел, это основной компонент кластера, узел находится в синхронизированном состоянии, подключен к кластеру и готов к репликации данных.

Перед добавлением других узлов создадим пользователя для передачи мгновенных снимков состояния (SST) и предоставим ему необходимые привилегии:

```
mysql@pxc1> CREATE USER 'sstuser'@'localhost' IDENTIFIED BY 'passw0rd';
mysql@pxc1> GRANT RELOAD, LOCK TABLES, PROCESS, REPLICATION CLIENT ON *.* TO
  'sstuser'@'localhost';
mysql@pxc1> FLUSH PRIVILEGES;
```

3.3.5 Добавление узлов в кластер

Корректно сконфигурированные новые узлы добавляются к кластеру автоматически. При запуске узла с адресом, по крайней мере, одного запущенного узла в переменной wsrep_cluster_address он автоматически добавляется к кластеру и синхронизируется с ним.

Все данные и настройки будут перезаписаны в соответствии с конфигурацией и данными узла донора. Не рекомендуется одновременно присоединять несколько узлов из-за возможных высоких накладных расходов на репликацию данных на новый узел.

3.3.5.1 Запуск второго узла

Для запуска второго узла воспользуемся командой (на машине сервера второго узла):

```
sudo /etc/init.d/mysql start
```

После запуска сервера mysql он должен автоматически выполнить репликацию данных (принять мгновенный снимок состояния (SST)).

Для проверки статуса второго узла войдем в консоль mysql:

```
mysql -u root –p
```

и выполним команду

```
mysql@pxc2> show status like 'wsrep%';
+----------------------------------+-------------------------------------------------+
| Variable_name                    | Value                                           |
+----------------------------------+-------------------------------------------------+
| wsrep_local_state_uuid           | 7e4a3888-d451-11e7-8bf9-037f76f76cc8            |
| ...                              | ...                                             |
| wsrep_local_state                | 4                                               |
| wsrep_local_state_comment        | Synced                                          |
| ...                              | ...                                             |
| wsrep_incoming_addresses         | 192.168.1.164:3306,192.168.1.165:3306           |
| ...                              | ...                                             |
| wsrep_cluster_size               | 2                                               |
| wsrep_cluster_state_uuid         | 7e4a3888-d451-11e7-8bf9-037f76f76cc8            |
| wsrep_cluster_status             | Primary                                         |
| wsrep_connected                  | ON                                              |
| ...                              | ...                                             |
| wsrep_ready                      | ON                                              |
+----------------------------------+-------------------------------------------------+
67 rows in set (0.00 sec)
```

Мы видим, что узел успешно добавлен к кластеру. Кластер содержит 2 узла, это основной компонент кластера, узел подключен к кластеру и готов к репликации данных.

Если состояние узла как в примере Synced, то узел полностью принял SST, синхронизирован с кластером, и мы можем добавлять следующий узел. Если состояние Joiner это означает, что репликация снимка состояния еще не завершена. В этом случае не рекомендуется добавлять другие узлы.

3.3.5.2 Запуск третьего узла

Для добавления третьего узла воспользуемся командой (на машине сервера третьего узла):

```
sudo /etc/init.d/mysql start
```

Для проверки статуса второго узла войдем в консоль mysql:

```
mysql -u root –p
```

и выполним команду

```
mysql@pxc3> show status like 'wsrep%';
+------------------------------+-------------------------------------------------------+
| Variable_name                | Value                                                 |
+------------------------------+-------------------------------------------------------+
| wsrep_local_state_uuid       | 7e4a3888-d451-11e7-8bf9-037f76f76cc8                  |
| ...                          | ...                                                   |
| wsrep_local_state            | 4                                                     |
| wsrep_local_state_comment    | Synced                                                |
| ...                          | ...                                                   |
| wsrep_incoming_addresses     | 192.168.1.164:3306,192.168.1.166:3306,192.168.1.165:3306 |
| ...                          | ...                                                   |
| wsrep_cluster_size           | 3                                                     |
| wsrep_cluster_state_uuid     | 7e4a3888-d451-11e7-8bf9-037f76f76cc8                  |
| wsrep_cluster_status         | Primary                                               |
| wsrep_connected              | ON                                                    |
| ...                          | ...                                                   |
| wsrep_ready                  | ON                                                    |
+------------------------------+-------------------------------------------------------+
67 rows in set (0.04 sec)
```

Мы видим, что узел успешно добавлен к кластеру. Кластер содержит 3 узла, это основной компонент кластера, узел подключен к кластеру и готов к репликации данных.

3.3.6 Проверка работы репликации

Для проверки работы репликации создадим новую базу данных на втором узле, создадим таблицу в базе третьем узле и добавим данные в таблицу на первом узле.

Создадим базу данных на втором узле:

```
mysql@pxc2> CREATE DATABASE percona;
Query OK, 1 row affected (0.01 sec)
```

Создадим таблицу на третьем узле:

```
mysql@pxc3> USE percona;
Database changed

mysql@pxc3> CREATE TABLE example (node_id INT PRIMARY KEY, node_name
VARCHAR(30));
Query OK, 0 rows affected (0.05 sec)
```

Вставим запись на первом узле:

```
mysql@pxc1> INSERT INTO percona.example VALUES (1, 'percona1');
Query OK, 1 row affected (0.02 sec)
```

Проверим на втором узле, реплицирована ли запись:

```
mysql@pxc2> SELECT * FROM percona.example;
+---------+-----------+
| node_id | node_name |
+---------+-----------+
|       1 | percona1  |
+---------+-----------+
1 row in set (0.00 sec)
```

3.3.7 Установка ProxySQL

ProxySQL это приложение для проксирования SQL-запросов к базам данных для MySQL кластера. Работает как отдельный демон, все SQL-запросы, которые необходимо проксировать, обрабатываются, затем, по заранее составленным правилам, демон подключается к необходимому MySQL-серверу и выполняет запрос, и уже после этого отдает результат приложению.

Мы будем устанавливать ProxySQL на четвертый узел. Для установки воспользуемся командой (на четвертом узле):

```
sudo apt-get install proxysql
```

Для подключения к административному интерфейсу ProxySQL нам потребуется MySQL клиент. Мы можем использовать MySQL клиент, установленный на одном из узлов Percona XtraDB Cluster, либо установить его на узел с ProxySQL и подключаться локально. Воспользуемся вторым способом. Для этого установим на четвертый узел пакет Percona XtraDB Cluster:

```
sudo apt-get install percona-xtradb-cluster-client-5.7
```

Подключимся к административному модулю ProxySQL с использованием пароля/логина по умолчанию:

```
mysql -u admin -padmin -h 127.0.0.1 -P 6032
mysql: [Warning] Using a password on the command line interface can be
insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1420
Server version: 5.5.30 (ProxySQL Admin Module)

Copyright (c) 2009-2017 Percona LLC and/or its affiliates
Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Посмотрим базы данных и таблицы ProxySQL:

```
mysql@proxysql> SHOW DATABASES;
+-----+---------+-------------------------------+
| seq | name    | file                          |
+-----+---------+-------------------------------+
| 0   | main    |                               |
| 2   | disk    | /var/lib/proxysql/proxysql.db |
| 3   | stats   |                               |
| 4   | monitor |                               |
+-----+---------+-------------------------------+
4 rows in set (0.00 sec)

mysql@proxysql> SHOW TABLES;
+--------------------------------------+
| tables                               |
+--------------------------------------+
| global_variables                     |
| mysql_collations                     |
| mysql_query_rules                    |
| mysql_replication_hostgroups         |
| mysql_servers                        |
| mysql_users                          |
| runtime_global_variables             |
| runtime_mysql_query_rules            |
| runtime_mysql_replication_hostgroups |
| runtime_mysql_servers                |
| runtime_scheduler                    |
| scheduler                            |
+--------------------------------------+
12 rows in set (0.00 sec)
```

ProxySQL имеет несколько областей, где может размещаться конфигурация (рисунок 6).

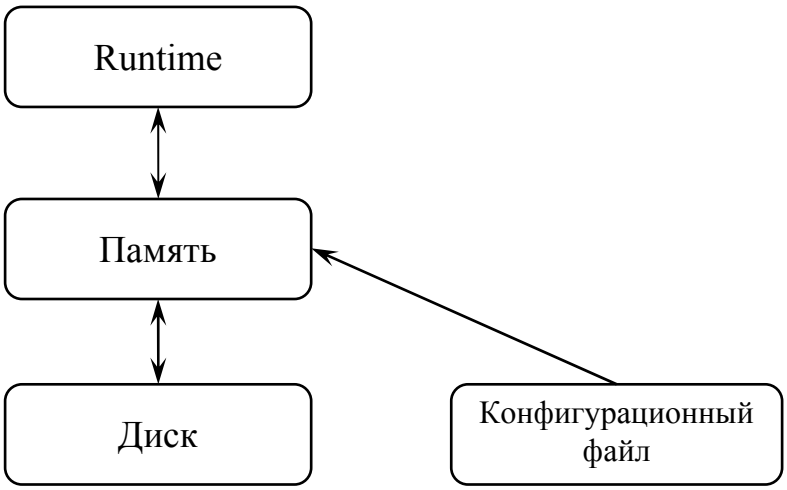

Рисунок 6 — Области хранения конфигурации ProxySQL

Область времени выполнения (runtime) — непосредственно используется демоном ProxySQL и содержит всю конфигурационную информацию для проксирования запросов.

Область памяти (memory) — представляет собой SQLite3 базу данных, которая находится в памяти и используется для внесения изменений в конфигурацию. Конфигурирование осуществляется SQL-командами через стандартный MySQL-клиент.

Область на дискового хранилища (disk) — представляет собой обычный SQLite3 базу на диске, в которую сохраняются данные внесенные через область памяти.

Конфигурационный файл — конфигурационный файл /etc/proxysql.cnf. Используется в момент инициализации, содержит информацию о нахождении SQLite3 базы данных, информацию об административном интерфейсе, а так же начальную конфигурацию демона. Конфигурационный файл при запуске демона используется только при отсутствии базы данных на диске.

Для перемещения конфигураций пользователей (MYSQL USERS) между памятью и областью времени выполнения используются команды:

```
mysql> LOAD MYSQL USERS FROM MEMORY
mysql> LOAD MYSQL USERS TO RUNTIME
```

Из области времени выполнения в память:

```
mysql> SAVE MYSQL USERS TO MEMORY
mysql> SAVE MYSQL USERS FROM RUNTIME
```

С дискового хранилища в память:

```
mysql> LOAD MYSQL USERS TO MEMORY
mysql> LOAD MYSQL USERS FROM DISK
```

Из памяти в дисковое хранилище:

```
mysql> SAVE MYSQL USERS FROM MEMORY
mysql> SAVE MYSQL USERS TO DISK
```

Из конфигурационного файла в память:

```
LOAD MYSQL USERS FROM CONFIG
```

Таким же образом перемещение можно осуществлять и для:

– MYSQL QUERY RULES (правила маршрутизации проксируемых запросов; правила также позволяют модифицировать запросы и даже кэшировать результаты);

– MYSQL SERVERS (сервера на которые проксируются запросы);

– MYSQL VARIABLES (переменные MySQL-сервера);

– ADMIN VARIABLES (переменные административных настроек, для данных переменных невозможно их чтение из конфигурационного файла);

– SCHEDULER (настройки планировщика периодических заданий, чтение их из конфигурационного файла также невозможно).

3.3.8 Добавление узлов кластера в ProxySQL

Для балансировки нагрузки на узлы кластера ProxySQL использует концепцию групп узлов (hostgroups). Балансировка нагрузки осуществляется путем маршрутизации различных типов запросов на различные группы (с

использованием правил), при этом каждый узел может быть членом нескольких групп.

```
Добавим наши три Percona XtraDB Cluster узла к группе по умолчанию (0). Для
этого вставим записи соответствующие записи в таблицу mysql_servers .
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES
(0,'192.168.1.164',3306);
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES
(0,'192.168.1.165',3306);
mysql@proxysql> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES
(0,'192.168.1.166',3306);
```

Посмотрим результат:

```
mysql@proxysql> SELECT * FROM mysql_servers;
```

3.3.9 Создание пользования для мониторинга узлов

Для мониторинга узлов кластера из-под ProxySQL создадим на узлах кластера пользователя с привилегией USAGE (без привилегий) и добавим его в конфигурацию ProxySQL

Для этого на любом из узлов кластера войдем в консоль MySQL:

```
mysql -u root –p
```

и выполним команды

```
mysql@pxc2> CREATE USER 'proxysql'@'%' IDENTIFIED BY 'ProxySQLPa55';
mysql@pxc2> GRANT USAGE ON *.* TO 'proxysql'@'%';
```

Сконфигурируем пользователя в ProxySQL (на четвертом узле):

```
mysql@proxysql> UPDATE global_variables SET variable_value='proxysql'
              WHERE variable_name='mysql-monitor_username';
mysql@proxysql> UPDATE global_variables SET variable_value='ProxySQLPa55'
              WHERE variable_name='mysql-monitor_password';
```

Скопируем конфигурацию в область runtime и сохраним ее на диск:

```
mysql@proxysql> LOAD MYSQL VARIABLES TO RUNTIME;
mysql@proxysql> SAVE MYSQL VARIABLES TO DISK;
```

Удостоверимся, что мониторинг разрешен. Для этого посмотрим лог мониторинга:

```
mysql@proxysql> SELECT * FROM monitor.mysql_server_connect_log ORDER BY
time_start_us DESC LIMIT 6;
+---------------+------+------------------+-------------------------+---------------+
| hostname      | port | time_start_us    | connect_success_time_us | connect_error |
+---------------+------+------------------+-------------------------+---------------+
| 192.168.1.166 | 3306 | 1512496502928812 | 4204                    | NULL          |
| 192.168.1.165 | 3306 | 1512496502892833 | 2252                    | NULL          |
| 192.168.1.164 | 3306 | 1512496502882723 | 1989                    | NULL          |
| 192.168.1.166 | 3306 | 1512496442902784 | 1094                    | NULL          |
| 192.168.1.165 | 3306 | 1512496442892324 | 935                     | NULL          |
| 192.168.1.164 | 3306 | 1512496442881973 | 1285                    | NULL          |
+---------------+------+------------------+-------------------------+---------------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM monitor.mysql_server_ping_log ORDER BY time_start_us DESC
LIMIT 6;
+---------------+------+------------------+----------------------+------------+
| hostname      | port | time_start_us    | ping_success_time_us | ping_error |
+---------------+------+------------------+----------------------+------------+
| 192.168.1.166 | 3306 | 1512496512961262 | 357                  | NULL       |
| 192.168.1.165 | 3306 | 1512496512958252 | 428                  | NULL       |
| 192.168.1.164 | 3306 | 1512496512955966 | 361                  | NULL       |
| 192.168.1.166 | 3306 | 1512496502960807 | 351                  | NULL       |
| 192.168.1.165 | 3306 | 1512496502957332 | 255                  | NULL       |
| 192.168.1.164 | 3306 | 1512496502955450 | 395                  | NULL       |
+---------------+------+------------------+----------------------+------------+
6 rows in set (0.00 sec)
```

Разрешим мониторинг узлов, загрузив их в runtime, и сохраним настройку узлов:

```
mysql@proxysql> LOAD MYSQL SERVERS TO RUNTIME;
mysql@proxysql> SAVE MYSQL SERVERS TO DISK;
```

3.3.10 Создание пользователя для доступа к узлам кластера

ProxySQL должен иметь пользователей, через которых будет осуществляться доступ к узлам кластера.

Добавим в таблицу mysql_users конфигурации ProxySQL логин/пароль пользователя:

```
mysql@proxysql> INSERT INTO mysql_users (username,password) VALUES
('sbuser','sbpass');
Query OK, 1 row affected (0.00 sec)
```

Загрузим пользователя в runtime область и сохраним изменения на диск:

```
mysql@proxysql> LOAD MYSQL USERS TO RUNTIME;
mysql@proxysql> SAVE MYSQL USERS TO DISK;
```

Проверим, что пользователь создан корректно. Для этого выйдем из консоли MySQL:

```
mysql@proxysql>exit
```

и войдем в нее под вновь созданным пользователем (вход осуществляется в монитор, а не административный модуль, поэтому используется другой порт):

```
mysql -u sbuser -psbpass -h 127.0.0.1 -P 6033
```

Вернемся в консоль под администратором:

```
mysql@proxysql>exit
mysql -u admin -padmin -h 127.0.0.1 -P 6032
```

Для того чтобы дать ProxySQL доступ к узлам кластера на чтение/запись, создадим этого же пользователя на кластере (любом из узлов) и дадим ему необходимые права:

```
mysql@pxc3> CREATE USER 'sbuser'@'192.168.1.167' IDENTIFIED BY 'sbpass';
mysql@pxc3> GRANT ALL ON *.* TO 'sbuser'@'192.168.1.167';
```

3.3.11 Конфигурирование поддержки Galera

По умолчанию ProxySQL не может детектировать находится ли узел кластера в состоянии Synced. Для мониторинга статуса узлов Percona XtraDB Cluster используется скрипт proxysql_galera_checker (расположен в /usr/bin/proxysql_galera_checker).

Для использования данного скрипта его необходимо загрузить в планировщик ProxySQL:

```
mysql@proxysql> INSERT INTO
scheduler(id,active,interval_ms,filename,arg1,arg2,arg3,arg4,arg5)
  VALUES (1,'1','10000','/usr/bin/proxysql_galera_checker','0','-1','0','1',
  '/var/lib/proxysql/proxysql_galera_checker.log');
```

Активируем настройки планировщика и сохраним их:

```
mysql@proxysql> LOAD SCHEDULER TO RUNTIME;
mysql@proxysql> SAVE SCHEDULER TO DISK;
```

Удостоверимся, что скрипт загружен, проверив таблицу runtime_scheduler:

```
mysql@proxysql> SELECT * FROM runtime_scheduler\G
*************************** 1. row ***************************
         id: 1
     active: 1
interval_ms: 10000
   filename: /usr/bin/proxysql_galera_checker
       arg1: 0
       arg2: -1
       arg3: 0
       arg4: 1
       arg5: /var/lib/proxysql/proxysql_galera_checker.log
    comment:
1 row in set (0.00 sec)
```

Проверим статус узлов кластера:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------+
| hostgroup_id | hostname      | port | status |
+--------------+---------------+------+--------+
| 0            | 192.168.1.164 | 3306 | ONLINE |
| 0            | 192.168.1.165 | 3306 | ONLINE |
| 0            | 192.168.1.166 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

Узел может находиться в одном из следующих состояний:

– ONLINE: узел полностью функционален;

– SHUNNED: узел временно не используется, так как за короткое время произошло слишком много ошибок подключения, или отставание репликации превысило допустимый порог;

– OFFLINE_SOFT: новые входящие соединения не принимаются, в то время как существующие соединения сохраняются до тех пор, пока они не

станут неактивными. Другими словами, соединения сохраняются до завершения текущей транзакции. Это позволяет корректно отсоединить серверный узел;

– OFFLINE_HARD: существующие соединения обрываются, и новые входящие соединения не принимаются. Это эквивалентно удалению узла из группы узлов или временному извлечению узла из группы узлов для обслуживания.

3.3.12 Тестирование узла с помощью sysbench

Установим sysbech на четвертом узле:

```
sudo apt-get install sysbench
```

На любом из узлов кластера создадим базу данных, которая будет использоваться для тестирования кластера:

```
mysql@pxc1> CREATE DATABASE sbtest;
```

Подготовим таблицу с данными для тестирования:

```
sysbench --report-interval=5 --threads=4 --time=20 --mysql-user='sbuser' --
mysql-password='sbpass' --table_size=10000 --mysql-host=127.0.0.1 --mysql-
port=6033 oltp_read_write prepare
```

Запустим тест:

```
sysbench --report-interval=5 --threads=4 --time=3 --mysql-user='sbuser' --
mysql-password='sbpass' --table_size=10000 --mysql-host=127.0.0.1 --mysql-
port=6033 oltp_read_write run
```

Собранную ProxySQL статистику можно осмотреть через схему stats (через административный монитор ProxySQL):

```
mysql@proxysql> SHOW TABLES FROM stats;
+----------------------------------+
| tables                           |
+----------------------------------+
| global_variables                 |
| stats_memory_metrics             |
| stats_mysql_commands_counters    |
| stats_mysql_connection_pool      |
| stats_mysql_connection_pool_reset |
| stats_mysql_global               |
| stats_mysql_processlist          |
| stats_mysql_query_digest         |
| stats_mysql_query_digest_reset   |
| stats_mysql_query_rules          |
| stats_mysql_users                |
| stats_proxysql_servers_checksums |
| stats_proxysql_servers_metrics   |
| stats_proxysql_servers_status    |
+----------------------------------+
14 rows in set (0.00 sec)
```

Например, можно посмотреть статистику команд выполненных на кластере:

```
mysql@proxysql> SELECT * FROM stats_mysql_commands_counters;
```

3.3.13 Автоматическое обнаружение отказов

ProxySQL автоматически определяет, что узел не доступен или не синхронизирован с кластером.

Состояние всех доступных узлов можно проверить, выполнив следующие действия:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------+
| hostgroup_id | hostname      | port | status |
+--------------+---------------+------+--------+
| 0            | 192.168.1.164 | 3306 | ONLINE |
| 0            | 192.168.1.165 | 3306 | ONLINE |
| 0            | 192.168.1.166 | 3306 | ONLINE |
+--------------+---------------+------+--------+
3 rows in set (0.00 sec)
```

Для проверки работы механизма обнаружения отказов, остановим узел 3:

```
sudo service mysql stop
```

ProxySQL обнаруживает, что узел 3 не работает и обновляет его статус на OFFLINE_SOFT:

```
mysql@proxysql> SELECT hostgroup_id,hostname,port,status FROM mysql_servers;
+--------------+---------------+------+--------------+
| hostgroup_id | hostname      | port | status       |
+--------------+---------------+------+--------------+
| 0            | 192.168.1.164 | 3306 | ONLINE       |
| 0            | 192.168.1.165 | 3306 | ONLINE       |
| 0            | 192.168.1.166 | 3306 | OFFLINE_SOFT |
+--------------+---------------+------+--------------+
3 rows in set (0.00 sec)
```

3.4 Контрольные вопросы

1 Почему при построении кластера необходимо использовать синхронную репликацию?

2 Недостатки синхронной репликации?

3 В чем заключаются особенности реализации синхронной репликации в Percona XtraDB Cluster?

4 В чем заключается проблема split-brain?

5 Для чего используется голосование и кворум?

6 Назначение ProxySQL?

Дик Дмитрий Иванович

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

Методические указания
к выполнению лабораторных работ для студентов
направлений 10.05.03 и 10.03.01

Редактор Н.Н. Погребняк

Подписано к печати 25.06.2018 Формат 60×84 1/16 Бумага 65 г/м^2
Печать цифровая Усл. печ. л. 3,25 Уч.–изд. л. 3,25
Заказ 118 Тираж 13 Не для продажи

Библиотечно-издательский центр КГУ.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.