

Arenadata™ Hadoop

Версия - v2.1.0

Руководство администратора по работе с кластером АДН

Оглавление

1	Руководство администратора по HDFS	3
1.1	HDFS Federation	3
1.2	ACL на HDFS	8
1.3	Архивные хранилища	10
1.4	Управление кэшем	13
1.5	DataNodes от Non-root	19
1.6	Примеры ACL	21
1.7	HAR. Архивы Hadoop	24
1.8	HDFS Compression	27
1.9	APIs JMX Metrics для HDFS Daemons	28
1.10	Память в качестве хранилища (техническое превью)	29
1.11	Настройка Rack Awareness	32
1.12	Режим локального чтения данных	35
1.13	Настройка WebHDFS	36
1.14	Добавление/удаление и обслуживание/декомиссия HDFS DataNode	37
1.15	POSIX ACL на HDFS	41
2	Руководство администратора по Hive	46
2.1	Tez	46
3	Руководство администратора по Spark	47
3.1	Динамическая аллокация ресурсов	47
4	Руководство администратора по YARN	48
4.1	Добавление/удаление и декомиссия YARN NodeManager	48
4.2	Высокая доступность (HA) и добавление/удаление YARN ResourceManager	51
5	Карта портов	54
5.1	Порты HDFS	54
5.2	Порты MapReduce	55
5.3	Порты YARN	56
5.4	Порты Hive	58
5.5	Порты WebHCat	59
5.6	Порты HBase	60
5.7	Порты Zookeeper	60

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

Глава 1

Руководство администратора по HDFS

В руководстве приведены сведения по HDFS, необходимые для работы с кластером.

Документ может быть полезен администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и сопровождение кластера.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

1.1 HDFS Federation

В главе представлен обзор функционала **HDFS Federation**, а также настройки и управление кластером.

- *Background*
- *Несколько Namenodes/Namespace*
- *Конфигурация*
- *Управление*

1.1.1 Background

HDFS имеет два основных слоя (Рис.1.1.):

- Namespace:
 - Состоит из каталогов, файлов и блоков;
 - Поддерживает все операции с файловой системой, связанные с пространством имен, такие как создание, удаление, изменение и просмотр файлов и каталогов;
- Block Storage Service:
 - Управление блоком (выполняется в Namenode):
 - * Обеспечивает членство в кластере Datanode путем обработки регистраций и передачи heartbeats-сообщений;
 - * Обрабатывает отчеты о блоках и поддерживает расположение блоков;
 - * Поддерживает связанные с блоком операции, такие как создание, удаление, изменение и получение местоположения блока;

- * Управляет размещением реплик, блокирует репликацию для блоков under-replicated и удаляет over-replicated блоки;
- Хранение – обеспечивается Datanodes путем хранения блоков в локальной файловой системе и предоставления доступа на чтение/запись.

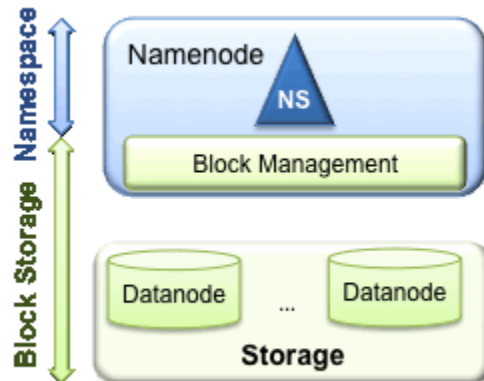


Рис.1.1.: Background

Предыдущая архитектура **HDFS** допускает только одно пространство имен для всего кластера и им управляет один Namenode. **HDFS Federation** устраняет это ограничение, добавляя поддержку нескольких Namenodes/Namespaces в **HDFS**.

1.1.2 Несколько Namenodes/Namespaces

Для горизонтального масштабирования сервиса имен **Federation** использует несколько независимых Namenodes/Namespaces. Узлы Namenodes федеративные, они независимы и не требуют координации друг с другом. Узлы Datanodes используются в качестве общего хранилища для блоков всех Namenodes, и каждая Datanode регистрируется со всеми Namenodes в кластере. Datanodes посылают периодические heartbeats-сообщения и обрабатывают команды из Namenodes (Рис.1.2).

Пользователи могут использовать ViewFs для создания персонализированных представлений пространства имен. ViewFs аналогичен mount-таблицам на стороне клиента в некоторых системах **Unix/Linux**.

Block Pool – это набор блоков, принадлежащих одному пространству имен. Узлы Datanodes хранят блоки для всех пулов в кластере. Каждый пул блоков управляется независимо, что позволяет пространству имен генерировать идентификаторы блоков для новых блоков без необходимости координации с другими пространствами имен. При этом собой Namenode не препятствует тому, чтобы Datanode обслуживал другие Namenodes в кластере.

Пространство имен и его пул блоков вместе называются **Namespace Volume**. Это самостоятельная единица управления. Когда Namenode/Namespace удаляется, удаляется и соответствующий ему пул блоков в Datanodes. И каждый Namespace Volume обновляется как единое целое во время обновления кластера.

Идентификатор **ClusterID** используется для идентификации всех узлов в кластере. При форматировании Namenode этот идентификатор либо предоставляется, либо генерируется автоматически.

Ключевые преимущества:

- **Namespace Scalability** – Federation добавляет горизонтальное масштабирование пространства имен. Большие развертывания или развертывания, использующие множество небольших файлов, выигрывают от масштабирования пространства имен, позволяя добавлять больше Namenodes в кластер;

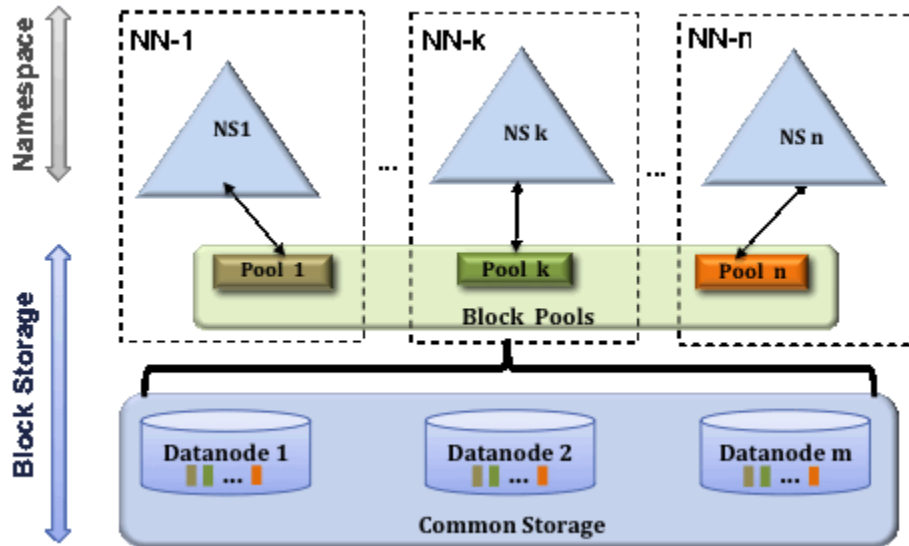


Рис.1.2.: Federation

- Performance – пропускная способность файловой системы не ограничивается одним Namenode. Добавление дополнительных Namenodes в кластер увеличивает пропускную способность чтения/записи файловой системы;
- Isolation – один Namenode не обеспечивает изоляции в многопользовательской среде. Например, экспериментальное приложение может перегрузить Namenode и замедлить работу критически важных приложений. Используя несколько Namenodes, разные категории приложений и пользователей могут быть изолированы в разных пространствах имен.

1.1.3 Конфигурация

Конфигурация **Federation** обратно совместима и позволяет существующим конфигурациям с одним Namenode работать без каких-либо изменений. Новая конфигурация разработана таким образом, что все узлы в кластере имеют одинаковую конфигурацию, не зависящую от типа узла в кластере.

Federation добавляет новую абстракцию **NameServiceID**. Namenode и соответствующие ему вторичные/резервные/контрольные узлы (secondary/backup/checkpointer) – все принадлежат NameServiceId. Для поддержки одного файла конфигурации к параметрам настроек Namenode и соответствующих ему узлов добавляется суффикс NameServiceID.

1. Добавить параметр `dfs.nameservices` в конфигурацию и настроить его с разделенным запятыми списком NameServiceID. Параметр используется Datanodes для определения Namenodes в кластере.
2. Для каждого Namenode и Secondary Namenode/BackupNode/Checkpointer добавить следующие параметры конфигурации с суффиксом соответствующего NameServiceID в общий файл конфигурации:
 - Namenode:
 - `dfs.namenode.rpc-address`
 - `dfs.namenode.servicerpc-address`
 - `dfs.namenode.http-address`
 - `dfs.namenode.https-address`
 - `dfs.namenode.keytab.file`

- *dfs.namenode.name.dir*
- *dfs.namenode.edits.dir*
- *dfs.namenode.checkpoint.dir*
- *dfs.namenode.checkpoint.edits.dir*
- Secondary Namenode:
 - *dfs.namenode.secondary.http-address*
 - *dfs.secondary.namenode.keytab.file*
- BackupNode:
 - *dfs.namenode.backup.address*
 - *dfs.secondary.namenode.keytab.file*

Пример конфигурации с двумя Namenodes:

```

<configuration>
  <property>
    <name>dfs.nameservices</name>
    <value>ns1,ns2</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns1</name>
    <value>nn-host1:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns1</name>
    <value>nn-host1:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.secondary.http-address.ns1</name>
    <value>snn-host1:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns2</name>
    <value>nn-host2:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns2</name>
    <value>nn-host2:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.secondary.http-address.ns2</name>
    <value>snn-host2:http-port</value>
  </property>
  .... Other common configuration ...
</configuration>

```

Formatting Namenodes

Форматирование Namenodes осуществляется в два шага:

1. Отформатировать Namenode, используя команду:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs namenode -format [-clusterId <cluster_id>]
```

Необходимо выбрать уникальный `cluster_id`, который не будет конфликтовать с другими кластерами в среде. Если параметр не указан, то он генерируется автоматически.

2. Отформатировать дополнительные Namenodes, используя команду:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs namenode -format -clusterId <cluster_id>
```

Важно обратить внимание, что `cluster_id` на этом шаге должен быть таким же, как в предыдущем. Если они отличаются, дополнительные Namenodes не будут частью кластера Federation.

Upgrading

В процессе обновления с предыдущего релиза и настройки **Federation** необходимо указать ClusterID следующим образом:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start namenode -upgrade -clusterId <cluster_ID>
```

Если `cluster_id` не указан, он генерируется автоматически.

Adding a new Namenode

Добавление нового Namenode в существующий кластер **HDFS** осуществляется в результате следующих действий:

- Добавить `dfs.nameservices` в конфигурацию;
- Обновить конфигурацию с помощью суффикса `NameServiceID`, чтобы использовать Federation;
- Добавить новую конфигурацию, связанную с Namenode, в файл конфигурации;
- Распространить файл конфигурации на все узлы в кластере;
- Запустить новый Namenode и Secondary/Backup;
- Обновить Datanodes, чтобы получить только что добавленный Namenode, выполнив следующую команду для всех Datanodes в кластере:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs dfsadmin -refreshNamenodes <datanode_host_name>:<datanode_rpc_port>
```

1.1.4 Управление

Команда для запуска кластера:

```
[hdfs]$ $HADOOP_HOME/sbin/start-dfs.sh
```

Команда для остановки кластера:

```
[hdfs]$ $HADOOP_HOME/sbin/stop-dfs.sh
```

Команды можно запускать с любого узла, где доступна конфигурация **HDFS**. Команда использует конфигурацию для определения Namenodes в кластере, а затем запускает процесс Namenode на этих узлах. Datanodes запускаются на узлах, указанных в файле *workers*. Скрипт можно использовать в качестве ссылки для создания собственных сценариев запуска и остановки кластера.

Balancer

Для работы с несколькими Namenodes изменен Balancer:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start balancer [-policy <policy>]
```

Параметр политики может быть любым из следующих:

- `datanode` – политика по умолчанию, уравнивает хранение на уровне Datanode. Политика похожа на политику балансировки из предыдущих выпусков;
- `blockpool` – политика балансирует хранилище на уровне пула блоков, который в свою очередь балансируется на уровне Datanode.

Important: Balancer балансирует только данные и не балансирует пространство имен

Decommissioning

Вывод из эксплуатации аналогичен предыдущим релизам – узлы, которые должны быть выведены из эксплуатации, добавляются в файл `exclude` на всех Namenodes. Каждый Namenode выводит из строя свой Block Pool. Когда все Namenodes завершают вывод из эксплуатации Datanode, узел Datanode считается списанным:

1. Команда для распространения файла `exclude` на все Namenodes:

```
[hdfs]$ $HADOOP_HOME/sbin/distribute-exclude.sh <exclude_file>
```

2. Обновление всех Namenodes для получения нового файла `exclude`:

```
[hdfs]$ $HADOOP_HOME/sbin/refresh-namenodes.sh
```

Команда использует конфигурацию **HDFS** для определения настроенных Namenodes в кластере и обновляет их, чтобы получить новый файл `exclude`.

Cluster Web Console

Подобно веб-странице статуса Namenode, при использовании **Federation** веб-консоль кластера доступна для мониторинга по адресу `http://<any_nn_host:port>/dfsclusterhealth.jsp`. Любой Namenode в кластере может быть использован для доступа к этой веб-странице.

Веб-консоль кластера предоставляет следующую информацию:

- Сводная информация о кластере, которая показывает количество файлов, количество блоков, общую настроенную емкость хранилища, а также доступное и используемое хранилище для всего кластера;
- Список Namenodes и сводку, которая включает в себя количество файлов, блоков, отсутствующих блоков и узлов живых и мертвых данных для каждого Namenode. Также предоставляется ссылка для доступа к веб-интерфейсу каждого Namenode;
- Статус декомиссии Datanodes.

1.2 ACL на HDFS

В главе описывается использование списков контроля доступа (**ACL**) в распределенной файловой системе **Hadoop – HDFS**. **ACL** расширяет модель разрешения **HDFS** для поддержки более детального доступа к файлам на основе произвольных комбинаций пользователей и групп.

1.2.1 Настройка ACL на HDFS

По умолчанию **ACL** отключены и **NameNode** отклоняет все попытки установить их. Например:

```
<property>
  <name>dfs.namenode.acls.enabled</name>
  <value>true</value>
</property>
```

Для включения **ACL** в **HDFS** необходимо в файле *hdfs-site.xml* установить свойству *dfs.namenode.acls.enabled* значение *true*.

1.2.2 Использование команд CLI для создания ACL

В **FsShell** используется две подкоманды: **setfacl** и **getfacl**. Они моделируются после одних и тех же команд **Linux**, но при этом реализуют меньше флагов. Поддержка дополнительных флагов может быть добавлена позже.

Setfacl

Устанавливает **ACL** для файлов и каталогов. Применение:

```
-setfacl [-bkr] {-m|-x} <acl_spec> <path> -setfacl --set <acl_spec> <path>
```

Функции команды описаны в таблице.

Таблица 1.1.: Функции команды setfacl

Функция	Описание
-b	Удаление всех записей, но с сохранением записей ACL. Записи для пользователей и групп сохраняются для совместимости с разрешениями
-k	Удаление ACL по умолчанию
-R	Применение операции ко всем файлам и каталогам рекурсивно
-m	Изменение ACL. Новые записи добавляются в ACL, а существующие записи сохраняются
-x	Удаление указанных записей ACL. Все остальные записи ACL сохраняются
--set	Полная замена ACL и сброс всех существующих записей. Функция «acl_spec» включает записи для пользователей и групп для совместимости с разрешениями
<acl_spec>	Список записей ACL, разделены запятыми
<path>	Путь к файлу или директории для изменения

Например:

```
hdfs dfs -setfacl -m user:hadoop:rw- /file
hdfs dfs -setfacl -x user:hadoop /file
hdfs dfs -setfacl -b /file
hdfs dfs -setfacl -k /dir
hdfs dfs -setfacl --set user::rw-,user:hadoop:rw-,group:r--,other::r-- /file
hdfs dfs -setfacl -R -m user:hadoop:r-x /dir
hdfs dfs -setfacl -m default:user:hadoop:r-x /dir
```

Код выхода: при успехе 0 и ненулевое значение при ошибке.

Getfacl

Отображает **ACL** файлов и каталогов. Если каталог имеет **ACL** по умолчанию, `getfacl` также его отображает. Применение:

```
-getfacl [-R] <path>
```

Функции команды описаны в таблице.

Таблица 1.2.: Функции команды `getfacl`

Функция	Описание
-R	Список ACL всех рекурсивных файлов и каталогов
<path>	Путь к файлу или директории списка

Например:

```
hdfs dfs -getfacl /file
hdfs dfs -getfacl -R /dir
```

Код выхода: при успехе 0 и ненулевое значение при ошибке.

1.3 Архивные хранилища

Архивные хранилища позволяют хранить данные на физических носителях с высокой плотностью хранения и низкими ресурсами обработки.

Для реализации архивного хранилища необходимо:

- Выключить `DataNode`;
- Назначить тип хранения `ARCHIVE`;
- Установить политики хранения “HOT”, “WARM” или “COLD” в файлах и каталогах `HDFS`;
- Перезапустить `DataNode`.

Для обновления параметра политики хранения в файле или каталоге необходимо использовать инструмент переноса данных **HDFS** для перемещения блоков, как указано в новой политике хранения.

1.3.1 Типы хранилищ `HDFS`

Типы хранилищ **HDFS** могут использоваться для данных, предназначенных различным типам физических носителей. Доступны следующие типы хранилищ:

- `DISK` – дисковое хранилище (тип по умолчанию);
- `ARCHIVE` – архивные хранилища (высокая плотность хранения, низкие ресурсы обработки);
- `SSD` – Solid State Drive, твердотельный накопитель;
- `RAM_DISK` – память `DataNode`.

1.3.2 Политики хранения

На дисках типа `DISK` или `ARCHIVE` можно хранить данные, используя следующие предварительно настроенные политики хранения:

- `HOT` – используется как для хранения, так и для вычислений. Данные, которые используются для обработки, остаются в этой политике. Все копии хранятся на `DISK`. Нет резервного хранилища, для хранения используется `ARCHIVE`;

- ID – 12
- Место размещения копии (для n копий) – DISK: n
- Резервное хранилище для обработки – нет
- Резервное хранилище для копий – ARCHIVE
- *WARM* – частично *HOT* и частично *COLD*. При *WARM* первая копия хранится на DISK, а остальные – в ARCHIVE. Резервным хранилищем для создания и копирования является DISK, а в случае если DISK недоступен – ARCHIVE:
- ID – 8
- Место размещения копии (для n копий) – DISK: 1, ARCHIVE: $n-1$
- Резервное хранилище для обработки – DISK, ARCHIVE
- Резервное хранилище для копий – DISK, ARCHIVE
- *COLD* – используется только для хранения, с ограниченными вычислениями. Данные, которые больше не используются или которые необходимо заархивировать, переносятся из хранилища *HOT* в *COLD*. При “COLD” все копии хранятся в ARCHIVE, и нет резервного хранилища для создания или копирования.
- ID – 4
- Место размещения копии (для n копий) – ARCHIVE: n
- Резервное хранилище для обработки – нет
- Резервное хранилище для копий – нет

Important: В настоящее время политики хранения нельзя редактировать

1.3.3 Настройка архивного хранилища

Для настройки архивного хранилища необходимо выполнить следующие действия:

1. Выключить DataNode

Закреть **DataNode** с помощью соответствующих команд.

2. Назначить тип хранения ARCHIVE

Для назначения типа хранения *ARCHIVE* для **DataNode** можно использовать свойство *dfs.name.dir* в файле */etc/hadoop/conf/hdfs-site.xml*.

Свойство *dfs.name.dir* определяет, где в локальной файловой системе **DataNode** хранит свои блоки.

Чтобы назначить **DataNode** как хранилище *DISK*, необходимо использовать путь к локальной файловой системе. Поскольку *DISK* является типом памяти по умолчанию, ничего не требуется. Например:

```
<property>
<name>dfs.data.dir</name>
<value>file:///grid/1/tmp/data_trunk</value>
</property>
```

Чтобы назначить **DataNode** как хранилище *ARCHIVE*, необходимо добавить [ARCHIVE] в начало пути локальной файловой системы. Например:

```
<property>
  <name>dfs.data.dir</name>
  <value>[ARCHIVE]file:///grid/1/tmp/data_trunk</value>
</property>
```

3. Установка и получение политики хранения

Необходимо установить политику хранения файла или каталога:

```
hdfs dfsadmin -setStoragePolicy <path> <policyName>
```

Аргументы:

- <path> – путь к каталогу или файлу;
- <policyName> – название политики хранения.

Пример:

```
hdfs dfsadmin -setStoragePolicy /cold1 COLD
```

Получение политики хранения файла или каталога осуществляется по команде:

```
hdfs dfsadmin -getStoragePolicy <path>
```

Аргументы:

- <path> – путь к каталогу или файлу.

Пример:

```
hdfs dfsadmin -getStoragePolicy /cold1
```

4. Запуск DataNode

Запустить **DataNode** с помощью соответствующих команд.

5. Использовать “mover” для применения политик хранения

При обновлении параметра политики хранения в файле или каталоге новая политика не применяется автоматически. Необходимо использовать инструмент переноса данных **HDFS** – *mover* для фактического перемещения блоков (как указано в новой политике хранения).

Средство миграции данных *mover* сканирует выбранные файлы в **HDFS** и проверяет, соответствует ли размещение блоков политике хранения. Копии блоков, нарушающих политику хранения, он перемещает в соответствующий тип хранилища для выполнения требований политики.

Команда:

```
hdfs mover [-p <files/dirs> | -f <local file name>]
```

Аргументы:

- -p <files/dirs> – список файлов/каталогов **HDFS** для переноса, разделенные пробелами;
- -f <local file> – локальный файл, содержащий список файлов/каталогов **HDFS** для миграции.

Important: Если оба параметра -p и -f опущены, путь по умолчанию является корневым каталогом

Пример:

```
hdfs mover /cold1/testfile
```

1.4 Управление кэшем

В главе приведены инструкции по настройке и использованию централизованного кэша в **HDFS**, который позволяет указать пути к каталогам или файлам для кэширования в **HDFS**, тем самым повышая производительность приложений, которые неоднократно обращаются к одним и тем же данным. Процесс осуществляется путем связывания NameNode с DataNodes, которые имеют требуемые доступные блоки на диске, и DataNodes кэшируют эти блоки.

Централизованное управление кэшем в **HDFS** дает много существенных преимуществ:

- Точный кэш предотвращает вытеснение часто используемых данных из памяти. Это особенно важно, когда размер рабочего набора превышает размер оперативной памяти, который является общим для многих рабочих нагрузок HDFS;
- Поскольку кэши данных DataNode управляются NameNode, приложения могут запрашивать набор местоположений кэшированных блоков при принятии решений о размещении задач. Совмещение задачи с кэшированной блочной копией повышает производительность чтения;
- Когда блок кэшируется с помощью DataNode, клиенты могут использовать новый, более эффективный API-интерфейс с нулевой копией. Поскольку проверка контрольных сумм кэшированных данных выполняется DataNode один раз, при использовании этого нового API клиенты могут понести практически нулевые расходы;
- Централизованное кэширование может улучшить общую эффективность использования памяти кластера. Когда используется ОС буферного кэша на каждом DataNode, повторные чтения блока приводят к тому, что все $\langle n \rangle$ копии блока перемещаются в буферный кэш. При централизованном управлении кэшем точно указывается только $\langle m \rangle$ копий $\langle n \rangle$, тем самым сохраняя память $\langle n-m \rangle$.

Централизованное управление кэшем полезно:

- Для файлов, к которым неоднократно обращаются. Например, небольшая таблица фактов в Hive, которая часто используется, является хорошим кандидатом для кэширования. И наоборот, кэширование запроса годовой отчетности менее полезно, так как подобные данные, вероятно, могут быть прочитаны только один раз;
- Для смешанной рабочей нагрузки с SLA-производительностью. Кэширование рабочего набора высокоприоритетной рабочей нагрузки гарантирует, что она не конкурирует с низкоприоритетными рабочими нагрузками для дискового ввода-вывода.

Из картинки видно, что в архитектуре NameNode отвечает за координацию всех кэш-файлов с неактивными данными в кластере. NameNode периодически получает кэш-отчет от каждого DataNode. Кэш-отчет описывает все блоки, кэшированные в DataNode. NameNode управляет кэшами DataNode с помощью кэш-копий и мгновенных команд *uncache*.

NameNode запрашивает набор Cache Directives, чтобы определить, какие контуры следует кэшировать. Cache Directives постоянно сохраняются в *fsimage* и журналах и могут быть добавлены, удалены и изменены с помощью **Java** и API-интерфейсов командной строки. В NameNode также хранится набор Cache Pools, являющийся административным объектом, который группирует Cache Directives для управления ресурсами, а также для обеспечения прав доступа.

NameNode периодически повторно сканирует пространство имен и актив Cache Directives, чтобы определить, какие блоки нужно кэшировать, а какие нет, и назначает задачи кэширования DataNodes. Повторное сканирование также может быть вызвано действиями пользователя, такими как добавление или удаление Cache Directives или удаление Cache Pools.

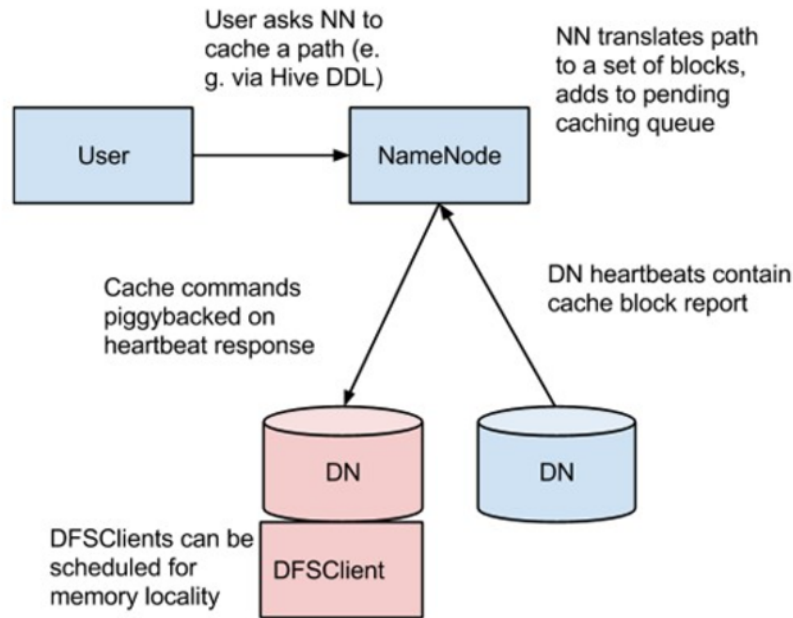


Рис.1.3.: Архитектура кэширования

Блоки кэша, находящиеся в стадии разработки, поврежденные или неполные, не кэшируются. Если Cache Directives содержит ссылку, адрес ссылки не кэшируется.

Important: В настоящее время кэширование может применяться только к каталогам и файлам

1.4.1 Терминология

Cache Directive

Cache Directive определяет контур для кэширования. Пути могут указывать либо каталоги, либо файлы. Каталоги кэшируются не рекурсивно, то есть кэшируются только файлы в листинге каталога первого уровня.

Cache Directives также указывают дополнительные параметры, такие как фактор репликации кэша и время окончания. Фактор репликации указывает количество блочных реплик в кэше. Если несколько Cache Directives относятся к одному файлу, применяется максимальный коэффициент репликации кэша.

Время окончания задается в командной строке как жизненный период (*time-to-live – TTL*), который представляет собой относительное время действия в будущем. После истечения срока действия Cache Directive больше не учитывается NameNode при принятии решений кэширования.

Cache Pool

Cache Pool – это административный объект, используемый для управления группами директив кэша. Кэш-пулы имеют UNIX-подобные разрешения, которые ограничивают доступ пользователей и групп к пулу. Разрешения на запись позволяют пользователям добавлять и удалять директивы кэша в пул. Разрешения на чтение позволяют пользователям просматривать директивы кэша в пуле и дополнительные метаданные. Execute-разрешение не используется.

Cache Pools также используются для управления ресурсами. Кэш-пулы могут обеспечить максимальный предел памяти, ограничивающий совокупное количество байтов, которые могут быть кэшированы директивами в пуле. Как правило, сумма лимитов пула приблизительно равна суммарной памяти, зарезервированной для

кэширования **HDFS** в кластере. Кэш-пулы также мониторят ряд статистических данных, чтобы помочь пользователям кластера отслеживать, что в настоящее время кэшируется, и определить, что еще нужно кэшировать.

Cache Pools также могут обеспечить максимальный жизненный период, ограничив максимальное время истечения срока действия директив, добавляемых в пул.

1.4.2 Настройка централизованного кэширования

Для отгорождения блокировки файлов в памяти DataNode использует собственный код *JNI* из *libhadoop.so*.

Important: При использовании централизованного управления кэшем HDFS обязательно должен быть включен *JNI*

Свойства конфигурации для централизованного кэширования указаны в файле *hdfs-site.xml*.

В настоящее время требуется только одно свойство:

- *dfs.datanode.max.locked.memory*.

Это свойство определяет максимальный объем памяти в байтах, который будет использовать DataNode для кэширования. Также необходимо увеличить размер заблокированного объема памяти *ulimit* (*ulimit -l*) пользователя DataNode. При настройке данного значения необходимо помнить, что пространство в памяти также требуется и для других целей (JNM, DataNode, а также страниц кэша ОС).

Пример:

```
<property>
  <name>dfs.datanode.max.locked.memory</name>
  <value>268435456</value>
</property>
```

Следующие свойства не являются обязательными, но могут быть заданы в настройках:

- *dfs.namenode.path.based.cache.refresh.interval.ms* – число миллисекунд, которое NameNode использует между последующими повторными сканированиями кэша. По умолчанию параметр установлен на *300000* (пять минут). Пример:

```
<property>
  <name>dfs.namenode.path.based.cache.refresh.interval.ms</name>
  <value>300000</value>
</property>
```

- *dfs.time.between.resending.caching.directives.ms* – NameNode использует это значение как количество миллисекунд между повторным кэшированием директивов. Пример:

```
<property>
  <name>dfs.time.between.resending.caching.directives.ms</name>
  <value>300000</value>
</property>
```

- *dfs.datanode.fsdatasetcache.max.threads.per.volume* – DataNode использует это значение как максимальное количество потоков на единицу объема для кэширования новых данных. По умолчанию параметр имеет значение *4*. Пример:

```
<property>
  <name>dfs.datanode.fsdatasetcache.max.threads.per.volume</name>
```



```
<value>4</value>
</property>
```

- *dfs.cachereport.intervalMsec* – DataNode использует это значение как число миллисекунд между отправкой отчета о состоянии кэша в NameNode. По умолчанию параметр установлен на *10000* (10 секунд). Пример:

```
<property>
  <name>dfs.cachereport.intervalMsec</name>
  <value>10000</value>
</property>
```

- *dfs.namenode.path.based.cache.block.map.allocation.percent* – процент Java-heap, распределенный по картам кэшированных блоков. Карта кэшированных блоков – это хеш-карта, которая использует связанное хэширование. Доступ к меньшим картам осуществляется медленнее, чем если количество кэшированных блоков велико; большие карты потребляют больше памяти. Значение по умолчанию равно *0,25%*. Пример:

```
<property>
  <name>dfs.namenode.path.based.cache.block.map.allocation.percent</name>
  <value>0.25</value>
</property>
```

1.4.3 Ограничения ОС

Если выдается сообщение об ошибке “*Cannot start datanode because the configured max locked memory size. . . is more than the datanode’s available RLIMIT_MEMLOCK ulimit*”, это означает, что операционная система накладывает более низкое ограничение на объем памяти, который можно заблокировать, чем настроено. Чтобы исправить это, необходимо настроить значение *ulimit -l*, с которым работает DataNode в файле */etc/security/limits.conf* (может варьироваться в зависимости от используемой ОС и дистрибутива).

Значение настроено правильно, когда при запуске *ulimit -l* выдается либо более высокое значение, чем настроенное, либо строка “*unlimited*”, что указывает на отсутствие ограничения.

Important: Для *ulimit -l* характерно выводить ограничение блокировки памяти в килобайтах, но при этом *dfs.datanode.max.locked.memory* должно быть указано в байтах

Например, значение *dfs.datanode.max.locked.memory* установлено в *128000* байт:

```
<property>
  <name>dfs.datanode.max.locked.memory</name>
  <value>128000</value>
</property>
```

Лучше установить *memlock* (максимальное адресное пространство с закрытой памятью) на несколько большее значение. Например, чтобы установить *memlock* на *130 KB* для пользователя *hdfs*, необходимо добавить следующую строку в */etc/security/limits.conf*:

```
hdfs - memlock 130
```

Important: Приведенная информация не применяется к развертыванию в Windows. Windows не имеет прямого эквивалента *ulimit -l*

1.4.4 Использование Cache Pools и Directives

Можно использовать интерфейс командной строки (CLI) для создания, изменения и перечисления Cache Pool и Cache Directives с помощью подкоманды `hdfs cacheadmin`.

Cache Directives идентифицируются уникальным не повторяющимся 64-битным ID. Идентификаторы не используются повторно, даже если Cache Directive удалена.

Cache Pools идентифицируются по уникальному имени строки.

Сначала создается Cache Pools, а затем в него добавляется Cache Directives.

Команды Cache Pools

`addPool` – команда добавления нового Cache Pool:

```
hdfs cacheadmin -addPool <name> [-owner <owner>] [-group <group>]
[-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]
```

Таблица 1.3.: Функции команды `addPool`

Функция	Описание
<name>	Имя нового Cache Pool
<owner>	Имя пользователя владельца Cache Pool. По умолчанию используется текущий пользователь
<group>	Группа, которой назначен Cache Pool. По умолчанию используется имя основной группы текущего пользователя
<mode>	Восьмеричные разрешения в стиле UNIX, назначенные Cache Pool. По умолчанию установлены <code>0755</code>
<limit>	Максимальное количество байтов, которые в совокупности могут быть кэшированы директивами в Cache Pool. По умолчанию ограничение не установлено
<maxTtl>	Максимальное допустимое время ожидания для директив, добавляемых в Cache Pool. Значение может быть указано в секундах, минутах, часах и днях, например, <code>120 s</code> , <code>30 m</code> , <code>4 h</code> , <code>2 d</code> . Допустимыми единицами являются <code>[smhd]</code> . По умолчанию максимальное значение не задано. Значение <code>never</code> указывает, что предела нет

`modifyPool` – команда изменения метаданных существующего Cache Pool:

```
hdfs cacheadmin -modifyPool <name> [-owner <owner>] [-group <group>]
[-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]
```

Таблица 1.4.: Функции команды `removePool`

Функция	Описание
<name>	Имя требующего изменения Cache Pool
<owner>	Имя пользователя владельца Cache Pool
<group>	Группа, которой назначен Cache Pool
<mode>	Восьмеричные разрешения в стиле UNIX, назначенные Cache Pool
<limit>	Максимальное количество байтов, которые в совокупности могут быть кэшированы директивами в Cache Pool
<maxTtl>	Максимальное допустимое время ожидания для директив, добавляемых в Cache Pool. Значение может быть указано в секундах, минутах, часах и днях, например, <code>120 s</code> , <code>30 m</code> , <code>4 h</code> , <code>2 d</code> . Допустимыми единицами являются <code>[smhd]</code> . По умолчанию максимальное значение не задано. Значение <code>never</code> указывает, что предела нет

`removePool` – команда удаления Cache Pool. Также удаляет пути, связанные с ним:

```
hdfs cacheadmin -removePool <name>
```

Таблица1.5.: Функции команды removePool

Функция	Описание
<name>	Имя удаляемого Cache Pool

listPools – команда отображает информацию об одном или нескольких Cache Pool, например, имя, владельца, группу, разрешения и прочее:

```
hdfs cacheadmin -listPools [-stats] [<name>]
```

Таблица1.6.: Функции команды listPools

Функция	Описание
-stats	Отображение дополнительной статистики по Cache Pool
<name>	Если параметр задан, то выдается только упомянутый Cache Pool

help – отображает подробную информацию о команде:

```
hdfs cacheadmin -help <command-name>
```

Таблица1.7.: Функции команды help

Функция	Описание
<command-name>	Отображение подробной информации по указанной команде. Если команда не указана, отображается справка по всем командам

Команды Cache Directives

addDirective – команда добавления нового Cache Directive:

```
hdfs cacheadmin -addDirective -path <path> -pool <pool-name> [-force]
[-replication <replication>] [-ttl <time-to-live>]
```

Таблица1.8.: Функции команды addDirective

Функция	Описание
<path>	Путь к каталогу кэша или файлу
<pool-name>	Cache Pool, к которому добавляется Cache Directive. Необходимо разрешение для Cache Pool на запись, чтобы добавить новые директивы
-force	Пропуск проверки ограничений ресурсов Cache Pool
-replication	Восьмеричные разрешения в стиле UNIX, назначенные Cache Pool. По умолчанию установлены <i>0755</i>
<limit>	Используемый коэффициент репликации кэша. По умолчанию установлено значение <i>1</i>
<time-to-live>	Продолжительность действия директивы. Значение может быть указано в минутах, часах и днях, например, <i>30 m, 4 h, 2 d</i> . Допустимыми единицами являются <i>[mhd]</i> . Значение <i>never</i> означает, что директива никогда не истекает. Если параметр не установлен, директива никогда не истекает

removeDirective – команда удаления Cache Directive:

```
hdfs cacheadmin -removeDirective <id>
```

Таблица 1.9.: Функции команды removeDirective

Функция	Описание
<id>	Идентификатор Cache Directive для удаления. Необходимо разрешение <i>Write</i> Cache Pool, к которому принадлежит директива. Можно использовать команду <code>-listDirectives</code> для отображения списка идентификаторов Cache Directive

removeDirectives – команда удаления всех Cache Directives по указанному пути:

```
hdfs cacheadmin -removeDirectives <path>
```

Таблица 1.10.: Функции команды removeDirectives

Функция	Описание
<path>	Путь Cache Directives для удаления. Необходимо разрешение <i>Write</i> Cache Pool, к которому относятся директивы. Можно использовать команду <code>-listDirectives</code> для отображения списка Cache Directives

listDirectives – команда возврата списка Cache Directives:

```
hdfs cacheadmin -listDirectives [-stats] [-path <path>] [-pool <pool>]
```

Таблица 1.11.: Функции команды listDirectives

Функция	Описание
<path>	Список Cache Directives данного пути. Если в пути, принадлежащему Cache Pool, нет доступа <i>Read</i> , Cache Directive не указывается
<pool>	Список Cache Directives, относящихся только к данному Cache Pool
-stats	Статистика по Cache Directive указанного пути

1.5 DataNodes от Non-root

В главе описываются правила запуска *DataNodes* от пользователя без прав *root*.

Исторически сложилось так, что часть конфигурации безопасности для **HDFS** задействовала запуск *DataNode* от пользователя *root* и привязала привилегированные порты для конечных точек сервера. Это было сделано для решения проблемы безопасности, то есть если задание **MapReduce** запущено, а *DataNode* остановился, задачу **MapReduce** можно привязать к порту *DataNode* и потенциально сделать что-то вредоносное. Решением подобных случаев стал запуск *DataNode* от пользователя *root* и использование привилегированных портов. При этом только пользователь *root* может получить доступ к привилегированным портам.

Для безопасного запуска *DataNodes* от пользователя без прав *root* можно использовать **Simple Authentication and Security Layer (SASL)**, который применяется для обеспечения безопасной связи на уровне протокола.

Important: Важно выполнить переход от использования *root* к запуску *DataNodes* с SASL в конкретной последовательности по всему кластеру. В противном случае может возникнуть риск простоя приложения

Для переноса существующего кластера, использующего аутентификацию *root*, сначала необходимо убедиться, что версия **2.6.0** (или более поздняя) развернута для всех узлов кластера, а также для любых

внешних приложений, которые необходимо подключить к кластеру. Только версии **2.6.0** + из HDFS-клиента могут подключаться к *DataNode*, использующему **SASL** для аутентификации протокола передачи данных, поэтому важно, чтобы все абоненты имели необходимую версию перед переходом.

После развертывания версии **2.6.0** (или более поздней) необходимо обновить конфигурацию любых внешних приложений, чтобы включить **SASL**. Если для **SASL** включен клиент **HDFS**, он может успешно подключиться к *DataNode*, работающему с аутентификацией *root* или аутентификацией **SASL**. Изменение конфигурации для всех клиентов гарантирует, что последующие изменения конфигурации в *DataNodes* не нарушат работу приложений. Наконец, каждый отдельный *DataNode* может быть перенесен путем изменения его конфигурации и перезапуска. Допустимо временно сочетать некоторые *DataNodes* с аутентификацией *root* и некоторые *DataNodes*, работающие с аутентификацией **SASL**, в течение периода миграции, поскольку клиент **HDFS**, подключенный для **SASL**, может подключаться к обоим.

1.5.1 Настройка DataNode SASL

Для настройки **DataNode SASL** с безопасным запуском *DataNode* от *non-root* пользователя необходимо выполнить действия:

1. Выключить **DataNode**
2. Включить **SASL**

Чтобы включить **DataNode SASL**, необходимо в файле `/etc/hadoop/conf/hdfs-site.xml` настроить свойство `dfs.data.transfer.protection`, задав одно из следующих значений:

- **authentication** – устанавливает взаимную аутентификацию между клиентом и сервером;
- **integrity** – в дополнение к аутентификации гарантирует, что man-in-the-middle не может вмешиваться в сообщения, обмен которыми осуществляется между клиентом и сервером;
- **privacy** – в дополнение к функциям **authentication** и **integrity** он также полностью шифрует сообщения, обмен которыми осуществляется между клиентом и сервером.

Также необходимо установить для свойства `dfs.http.policy` значение **HTTPS_ONLY**. При этом следует указать порты для **DataNode RPC** и HTTP-серверов.

Например:

```
<property>
  <name>dfs.data.transfer.protection</name>
  <value>integrity</value>
</property>

<property>
  <name>dfs.datanode.address</name>
  <value>0.0.0.0:10019</value>
</property>

<property>
  <name>dfs.datanode.http.address</name>
  <value>0.0.0.0:10022</value>
</property>

<property>
  <name>dfs.http.policy</name>
  <value>HTTPS_ONLY</value>
</property>
```

Important: Параметр шифрования `dfs.encrypt.data.transfer=true` похож на `dfs.data.transfer.protection=privacy`.

Эти два параметра являются взаимоисключающими, поэтому они не должны устанавливаться вместе. В случае если оба параметра установлены, `dfs.encrypt.data.transfer` не используется

3. Обновить настройки экосистемы

В файле `/etc/hadoop/conf/hadoop-env.xml` изменить параметр:

```
#On secure datanodes, user to run the datanode as after dropping privileges export HADOOP_SECURE_DN_USER=
```

Строка экспорта `HADOOP_SECURE_DN_USER=hdfs` включает устаревшую конфигурацию безопасности и, чтобы включить **SASL**, должна быть установлена на пустое значение.

4. Запустить DataNode

1.6 Примеры ACL

1.6.1 ACL & Permission Bits

До реализации списков контроля доступа – **ACL**, модель разрешения **HDFS** была эквивалентна традиционным **UNIX-разрешениям**. В этой модели разрешения для каждого файла или каталога управляются набором из трех пользовательских классов: “owner”, “group” и “others”. Для каждого пользовательского класса существует три разрешения: чтение, запись и выполнение. Когда пользователь пытается получить доступ к объекту файловой системы, **HDFS** применяет разрешения в соответствии с конкретным классом пользователя. Если пользователь является владельцем, **HDFS** проверяет разрешения класса “owner”. Если пользователь не является владельцем, но является членом группы объектов файловой системы, **HDFS** проверяет разрешения класса “group”. В противном случае **HDFS** проверяет разрешения класса “others”.

Эта модель может в достаточной степени удовлетворять большому количеству требований безопасности. Например, есть отдел продаж с требованием, чтобы один пользователь – Брюс, менеджер отдела, – контролировал все изменения данных продаж. Другим сотрудникам отдела продаж необходимо видеть данные, но без возможности их изменения. Все остальные компании (за пределами отдела продаж) не должны иметь доступа к просмотру данных. Это требование может быть реализовано путем запуска `chmod 640` в файле со следующим результатом:

```
-rw-r-----1 brucesales22K Nov 18 10:55 sales-data
```

Получается, только Брюс может изменить файл, только члены группы продаж могут прочитать файл, и никто другой не может получить доступ к файлу каким-либо образом.

Предположим, возникает новое требование, которое состоит в том, что Брюсу, Диане и Кларку должно быть разрешено вносить изменения. К сожалению, для реализации этого требования не существует возможности для разрешений, потому что может быть только один владелец и только одна группа. Типичным обходным решением является установка владельца файла на мнимую учетную запись пользователя, например, `salesmgr`, и разрешение Брюсу, Диане и Кларку использовать учетную запись `salesmgr` с помощью `sudo` или аналогичных механизмов. Недостатком обходного пути является то, что он создает сложность для конечных пользователей, требуя от них применения разных учетных записей для разных действий.

Предположим далее, что помимо сотрудников по продажам все руководители компании должны иметь возможность читать данные о продажах. Это еще одно требование, которое не может быть выражено с помощью **Permission Bits**, поскольку может быть только одна группа, и она уже используется. Типичным обходным решением является установка группы файлов в новую мнимую группу, например, `salesandexecs`, и добавление всех пользователей `sales` и `execs` к этой группе. Недостатком обходного пути является то, что он требует от администраторов создания и управления дополнительными пользователями и группами.

Таким образом, использование **Permission Bits** для удовлетворения требований к разрешениям, которые могут отличаться от естественной организационной иерархии пользователей и групп, может быть неудобно. А преимущество использования списков **ACL** заключается в том, что они позволяют более естественным образом

решать эти требования, поскольку для любого объекта файловой системы несколько пользователей и несколько групп могут иметь разные наборы разрешений.

1.6.2 Пример 1. Доступ к группе

В данном примере решается один из вопросов путем установки **ACL** для предоставления доступа на чтение данных о продажах членам группы *execs*. Для этого необходимо:

- Установить ACL:

```
> hdfs dfs -setfacl -m group:execs:r-- /sales-data
```

- Запустить `getfacl`, чтобы проверить результаты:

```
> hdfs dfs -getfacl /sales-data
# file: /sales-data
# owner: bruce
# group: sales
user::rw-
group::r--
group:execs:r--
mask::r--
other::---
```

- При помощи команды `ls` можно увидеть, что перечисленные разрешения добавлены с символом “+” для обозначения наличия ACL. Символ “+” добавляется к разрешениям любого файла или каталога с ACL:

```
> hdfs dfs -ls /sales-data
Found 1 items
-rw-r-----+ 3 bruce sales          0 2014-03-04 16:31 /sales-data
```

Новая запись **ACL** добавляется к существующим разрешениям, определенным в **Permission Bits**. Как владелец файла, Брюс имеет полный контроль. Члены группы *sales* и *execs* имеют доступ на чтение. У других пользователей доступа нет.

1.6.3 Пример 2. ACL по умолчанию

В дополнение к списку **ACL**, выполняемому проверки во время разрешений, существует также отдельная концепция – список **ACL** по умолчанию – **default ACL**, которая может применяться к каталогу, а не к файлу. **Default ACL** не имеет прямого влияния на проверку разрешений для существующих дочерних файлов и каталогов, а определяет списки **ACL**, которые будут получать новые дочерние файлы и каталоги автоматически при их создании.

Например, есть каталог “monthly-sales-data” с отдельными подкаталогами для каждого месяца. Необходимо установить список **default ACL**, чтобы гарантировать, что члены группы *execs* автоматически получают доступ к новым подкаталогам по мере их создания:

- Установить default ACL в родительский каталог:

```
> hdfs dfs -setfacl -m default:group:execs:r-x /monthly-sales-data
```

- Создать подкаталоги:

```
> hdfs dfs -mkdir /monthly-sales-data/JAN
> hdfs dfs -mkdir /monthly-sales-data/FEB
```

- Убедиться, что HDFS автоматически применил default ACL в подкаталоги:

```

> hdfs dfs -getfacl -R /monthly-sales-data
# file: /monthly-sales-data
# owner: bruce
# group: sales
user::rwx
group::r-x
other:---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other:---

# file: /monthly-sales-data/FEB
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other:---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other:---

# file: /monthly-sales-data/JAN
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other:---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other:---

```

Default ACL копируется из родительского каталога в дочерний файл или каталог при его создании. Последующие изменения **default ACL** в родительском каталоге не меняют списки **ACL** существующих дочерних элементов.

1.6.4 Пример 3. Блокировка доступа

Например, необходимо заблокировать доступ ко всему подкаталогу для конкретного пользователя (*diana*). Применение к данному пользователю списка **ACL** в корне подкаталога является самым быстрым способом без риска случайного отзыва разрешений у других пользователей. Для этого необходимо:

- Добавить запись **ACL** для блокировки всего доступа пользователя *diana* к “monthly-sales-data”:

```

> hdfs dfs -setfacl -m user:diana:--- /monthly-sales-data

```

- Запустить `getfacl` для проверки результатов:


```

> hdfs dfs -getfacl /monthly-sales-data
# file: /monthly-sales-data
# owner: bruce
# group: sales
user::rwx
user:diana:---
group::r-x
mask::r-x
other:---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other:---

```

Новая запись **ACL** добавляется к существующим разрешениям, определенным в **Permission Bits**. Брюс имеет полный контроль как владелец файла. Члены группы *sales* и *execs* имеют доступ на чтение. У других пользователей доступа нет.

Важно помнить о порядке оценки записей списка **ACL**, когда пользователь пытается получить доступ к объекту файловой системы:

- Если пользователь является владельцем файла, применяются разрешения “owner”;
- Если у пользователя есть запись в списке **ACL**, применяются соответствующие права;
- Если пользователь является членом группы файлов или любой именованной группы в **ACL**, то для всех соответствующих записей принудительно объединяются разрешения (пользователь может быть членом нескольких групп);
- Если ничто из вышеуказанного не применимо, назначаются разрешения класса “other”.

В данном примере запись **ACL**-пользователя достигла установленной цели, поскольку пользователь не является владельцем файла, а именованная пользовательская запись имеет приоритет над всеми другими записями.

1.7 HAR. Архивы Hadoop

Распределенная файловая система **Hadoop – HDFS** – предназначена для хранения и обработки больших наборов данных, но при этом **HDFS** может быть менее эффективна при хранении большого количества мелких файлов, так как они занимают большую часть пространства имен. В результате место на диске недоиспользуется из-за ограничения пространства имен.

Архивы **Hadoop – HAR** – используются для устранения ограничений пространства имен, связанных с хранением большого количества мелких файлов. Архив **Hadoop** позволяет упаковывать небольшие файлы в блоки **HDFS** более эффективно, тем самым сокращая использование памяти *NameNode*, сохраняя прозрачный доступ к файлам. **HAR** также совместимы с *MapReduce*, обеспечивая прозрачный доступ к исходным файлам.

HDFS предназначена для хранения и обработки больших массивов данных (терабайт). Например, большой продуктивный кластер может иметь *14 ПБ* дискового пространства и хранить *60* миллионов файлов.

Однако хранение большого количества мелких файлов в **HDFS** неэффективно. Обычно файл считается «маленьким», когда его размер существенно меньше размера блока **HDFS**, который в **ADH** по умолчанию равен *256 МБ*. Файлы и блоки являются объектами имен в **HDFS**, что означает, что они занимают пространство имен (место в *NameNode*). Таким образом, емкость пространства имен системы ограничена физической памятью *NameNode*.

Когда в системе хранится много мелких файлов, они занимают большую часть пространства имен. Как следствие, дисковое пространство недоиспользуется из-за ограничения пространства имен. В одном реальном

примере кластер имел 57 миллионов файлов размером менее 256 МБ, при этом каждый из этих файлов занимал один блок в *NameNode*. Эти мелкие файлы использовали 95% пространства имен, но занимали только 30% дискового пространства кластера.

Архивы **HAR** могут использоваться для устранения ограничений пространства имен, связанных с хранением большого количества мелких файлов. **HAR** упаковывает несколько мелких файлов в большой, обеспечивая прозрачный доступ к исходным файлам (без расширения файлов). Таким образом **HAR** увеличивает масштабируемость системы за счет сокращения использования пространства имен и уменьшения нагрузки на работу в *NameNode*, оптимизирует память в *NameNode* и распределяет управление пространством имен по нескольким *NameNodes*. Кроме того, **HAR** обеспечивает параллельный доступ к исходным файлам с помощью заданий **MapReduce**.

1.7.1 Компоненты HAR

Формат модели данных

Формат модели данных архивов **Hadoop** имеет вид:

```
foo.har/_masterindex //stores hashes and offsets
foo.har/_index //stores file statuses
foo.har/part-[1..n] //stores actual file data
```

Файлы данных хранятся в нескольких файлах, которые индексируются для сохранения первоначального разделения данных. Кроме того, файлы доступны параллельно с помощью *MapReduce*. В индексах файлов также записываются исходные структуры дерева каталогов и статус файла.

Файловая система HAR

Большинство архивных систем, таких как **tar**, являются инструментами для архивации и деархивации. Как правило, они не вписываются в фактический уровень файловой системы и, следовательно, не являются прозрачными для разработчика приложения, поскольку пользователь должен предварительно деархивировать (расширять) архив перед использованием.

HAR интегрируется с интерфейсом файловой системы **Hadoop**. *HarFileSystem* реализует интерфейс *FileSystem* и предусматривает доступ через *har://*. Это обеспечивает прозрачность архивных файлов и структур дерева каталогов для пользователей. Доступ к файлам в **HAR** можно получить напрямую, без его расширения.

Например, команда для копирования файла **HDFS** в локальный каталог:

```
hdfs dfs -get hdfs://namenode/foo/file-1 localdir
```

Предположив, что архив **Hadoop** *bar.har* создан из каталога *foo*, с помощью **HAR** команда для копирования исходного файла становится следующей:

```
hdfs dfs -get har://namenode/bar.har/foo/file-1 localdir
```

Пользователям следует изменить пути URL. Но при этом пользователи могут создать символическую ссылку (из *hdfs://namenode/foo* для *har://namenode/bar.har/foo* в примере выше), и тогда изменять URL не будет надобности. В любом случае, *HarFileSystem* вызывается автоматически для обеспечения доступа к файлам в **HAR**. Из-за этого прозрачного слоя **HAR** совместим с **API Hadoop**, *MapReduce*, интерфейсом командной строки оболочки **FS** и приложениями более высокого уровня, такими как **Pig**, **Zebra**, **Streaming**, **Pipes** и **DistCp**.

Инструмент архивации

Архивы **Hadoop** могут быть созданы с помощью инструмента архивации **Hadoop**, он использует *MapReduce* для эффективного параллельного создания **HAR**. Инструмент вызывается с помощью команды:

```
hadoop archive -archiveName name -p <parent> <src>* <dest>
```

Список файлов генерируется путем рекурсивного перемещения исходных каталогов, а затем список разбивается на карту входящих задач. Каждая задача создает файл (около 2 ГБ, настраивается) из подмножества исходных файлов и выводит метаданные. В итоге, *reduce task* собирает метаданные и генерирует индексные файлы.

1.7.2 Создание архива

Инструмент архивации вызывается следующей командой:

```
hadoop archive -archiveName name -p <parent> <src>* <dest>
```

Где `-archiveName` – имя создающегося архива. В имени архива должно быть указано расширение `.har`. Аргумент `<parent>` используется для указания относительного пути к папке, в которой файлы будут архивироваться в **HAR**. Например:

```
hadoop archive -archiveName foo.har -p /user/hadoop dir1 dir2 /user/zoo
```

В приведенном примере создается архив с использованием `/user/hadoop` в качестве каталога архива. Каталоги `/user/hadoop/dir1` и `/user/hadoop/dir2` будут заархивированы в архиве `/user/zoo/foo.har`.

Important: Архивирование не удаляет исходные файлы. При необходимости удаления входных файлов после создания архива (в целях сокращения пространства имен), исходные файлы должны удаляться вручную

Хотя команда архивации может быть запущена из файловой системы хоста, файл архива создается в **HDFS** из существующих каталогов. Если сослаться на каталог в файловой системе хоста, а не на **HDFS**, выдается ошибка:

```
The resolved paths set is empty. Please check whether the srcPaths exist, where srcPaths
= [</directory/path>]
```

Для создания каталогов **HDFS**, используемых в предыдущем примере, необходимо выполнить команду:

```
hdfs dfs -mkdir /user/zoo
hdfs dfs -mkdir /user/hadoop
hdfs dfs -mkdir /user/hadoop/dir1
hdfs dfs -mkdir /user/hadoop/dir2
```

1.7.3 Просмотр файлов в архивах Hadoop

Команда `hdfs dfs -ls` может использоваться для поиска файлов в архивах **Hadoop**. Используя пример архива `/user/zoo/foo.har`, созданный в предыдущем разделе, необходимо применить следующую команду для вывода списка файлов в архиве:

```
hdfs dfs -ls har:///user/zoo/foo.har/
```

Результатом будет:

```
har:///user/zoo/foo.har/dir1
har:///user/zoo/foo.har/dir2
```

Архивы были созданы с помощью команды:

```
hadoop archive -archiveName foo.har -p /user/hadoop dir1 dir2 /user/zoo
```

Если изменить команду на:

```
hadoop archive -archiveName foo.har -p /user/ hadoop/dir1 hadoop/dir2 /user/zoo
```

И затем выполнить:

```
hdfs dfs -ls -R har:///user/zoo/foo.har
```

То результатом будет:

```
har:///user/zoo/foo.har/hadoop
har:///user/zoo/foo.har/hadoop/dir1
har:///user/zoo/foo.har/hadoop/dir2
```

Следует обратить внимание, что с измененным родительским аргументом файлы архивируются относительно `/user/`, а не `/user/hadoop`.

1.7.4 Hadoop Archives и MapReduce

Для использования **HAR** с *MapReduce* необходимо ссылаться на файлы несколько иначе, чем на файловую систему по умолчанию. Если есть архив **Hadoop**, хранящийся в **HDFS** в `/user/zoo/foo.har`, следует указать каталог ввода как `har:///user/zoo/foo.har`, чтобы использовать его как *MapReduce*. Поскольку **HAR** отображаются как файловая система, *MapReduce* может использовать все логические входные файлы в архивы **Hadoop** в качестве входных данных.

1.8 HDFS Compression

В главе описывается настройка **HDFS Compression** на **Linux**.

Linux поддерживает **GzipCodec**, **DefaultCodec**, **BZip2Codec**, **LzoCodec** и **SnappyCodec**. Как правило, для **HDFS Compression** используется **GzipCodec**.

Существует два варианта использования **GzipCodec**:

1. **GzipCodec** для одноразовых заданий:

```
hadoop jar hadoop-examples-1.1.0-SNAPSHOT.jar sort sbr"-Dmapred.compress.map.output=true" sbr"-Dmapred.map.
↪output.compression.codec=org.apache.hadoop.io.compress.GzipCodec"sbr "-Dmapred.output.compress=true" sbr"-
↪Dmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec"sbr -outKey org.apache.hadoop.io.
↪Textsbr -outValue org.apache.hadoop.io.Text input output
```

2. **GzipCodec** в качестве сжатия по умолчанию:

- Отредактировать файл `core-site.xml` на главной машине NameNode:

```
<property>
  <name>io.compression.codecs</name>    <value>org.apache.hadoop.io.compress.GzipCodec,org.apache.hadoop.io.
↪compress.DefaultCodec,com.hadoop.compression.lzo.LzoCodec,org.apache.hadoop.io.compress.SnappyCodec</
↪value>
  <description>A list of the compression codec classes that can be used for compression/decompression.</
↪description>
</property>
```

- Изменить файл `mapred-site.xml` на главной машине JobTracker:

```
<property>
  <name>mapred.compress.map.output</name>
  <value>true</value>
</property>

<property>
  <name>mapred.map.output.compression.codec</name>
  <value>org.apache.hadoop.io.compress.GzipCodec</value>
</property>
```

```
<property>
  <name>mapred.output.compression.type</name>
  <value>BLOCK</value>
</property>
```

- (Опционально) Задать следующие два параметра конфигурации для включения сжатия задания. Изменить файл *mapred-site.xml* на главной машине Resource Manager:

```
<property>
  <name>mapred.output.compress</name>
  <value>>true</value>
</property>

<property>
  <name>mapred.output.compression.codec</name>
  <value>org.apache.hadoop.io.compress.GzipCodec</value>
</property>
```

- Перезапустить кластер.

1.9 APIs JMX Metrics для HDFS Daemons

Для доступа к показателям **HDFS** можно использовать методы с помощью API-интерфейсов **Java Management Extensions (JMX)**.

Доступ к метрикам **JMX** можно получить через веб-интерфейс **HDFS daemon**, что является рекомендуемым методом.

Например, для доступа к **NameNode JMX** необходимо использовать следующий формат команды:

```
curl -i http://localhost:50070/jmx
```

Для извлечения только определенного ключа можно использовать параметр `qry`:

```
curl -i http://localhost:50070/jmx?qry=Hadoop:service=NameNode,name=NameNodeInfo
```

1.9.1 Прямой доступ к удаленному агенту JMX

Метод требует, чтобы удаленный агент **JMX** был включен с опцией *JVM* при запуске сервисов **HDFS**.

Например, следующие параметры *JVM* в *hadoop-env.sh* используются для включения удаленного агента **JMX** для *NameNode*. Он работает на порту *8004* с отключенным **SSL**. Имя пользователя и пароль сохраняются в файле *mxremote.password*.

```
export HADOOP_NAMENODE_OPTS="-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.password.file=$HADOOP_CONF_DIR/jmxremote.password
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.port=8004 $HADOOP_NAMENODE_OPTS"
```

Подробности о связанных настройках можно найти [здесь](#). Также можно использовать инструмент `jmxquery` для извлечения информации через **JMX**.

Hadoop также имеет встроенный инструмент запросов **JMX** – `jmxget`. Например:

```
hdfs jmxget -server localhost -port 8004 -service NameNode
```

Important: Инструмент *jmxget* требует, чтобы аутентификация была отключена, так как она не принимает имя пользователя и пароль

Использование **JMX** может быть сложным для персонала, который не знаком с настройкой **JMX**, особенно **JMX** с **SSL** и **firewall tunnelling**. Поэтому обычно рекомендуется собирать информацию **JMX** через веб-интерфейс **HDFS daemon**, а не напрямую обращаться к удаленному агенту **JMX**.

1.10 Память в качестве хранилища (техническое превью)

В главе описывается как использовать память *DataNode* в качестве хранилища в **HDFS**.

Important: Данная возможность представлена как технический обзор и рассмотрена в рамках развития. Не следует использовать ее в продуктивной среде. При наличии вопросов относительно описанной особенности необходимо обратиться в службу поддержки – info@arenadata.io

HDFS поддерживает запись больших наборов данных в хранилище, а также обеспечивает безотказный доступ к данным, что удобно для пакетных заданий, записывающих большое количество данных.

Новые классы приложений управляют вариантами использования для записи меньшего количества временных данных. При использовании памяти *DataNode* в качестве хранилища адресов вариантов использования приложений записывается относительно небольшое количество промежуточных наборов данных с низкой задержкой. Запись данных блока в память снижает надежность, поскольку данные могут быть потеряны из-за перезагрузки процесса до их сохранения на диск. При этом **HDFS** пытается своевременно сохранить копии данных на диск, чтобы уменьшить риск потери данных.

Память *DataNode* указывается при использовании типа хранения *RAM_DISK* и политики хранения *LAZY_PERSIST*.

Для использования памяти *DataNode* в качестве хранилища **HDFS** необходимо:

- Выключить *DataNode*;
- Установить часть памяти *DataNode* для использования **HDFS**;
- Назначить *DataNode* тип хранения *RAM_DISK* и включить режим локального чтения данных;
- Установить политику хранения *LAZY_PERSIST* в файлах и каталогах **HDFS**, которые будут использовать память в качестве хранилища;
- Перезапустить *DataNode*.

При обновлении параметра политики хранения в файле или каталоге необходимо использовать инструмент переноса данных – *mover* – для фактического перемещения блоков (как указано в новой политике хранения).

Память как хранилище представляет собой один из аспектов возможностей управления ресурсами **YARN**, который включает **CPU scheduling**, **CGroups**, **node labels** и архивное хранилище.

1.10.1 Типы хранилищ HDFS

Типы хранилищ **HDFS** могут использоваться для назначения данных для различных типов физических носителей информации. Доступны следующие типы хранилища:

- *DISK* – дисковое хранилище (тип хранения по умолчанию);
- *ARCHIVE* – архивное хранилище (высокая плотность хранения, низкий ресурс обработки);
- *SSD* – твердотельный накопитель;

- *RAM_DISK* – память *DataNode*.

Если тип хранилища не назначен, по умолчанию используется тип *DISK*.

1.10.2 Политика хранения *LAZY_PERSIST*

С помощью политики хранения *LAZY_PERSIST* можно хранить данные в сконфигурированной памяти *DataNode*. При этом первая копия данных хранится в *RAM_DISK* (память *DataNode*), а остальные копии – на *DISK*. Резервным хранилищем для создания и копирования является *DISK*:

- ID политики: 15
- Название политики: *LAZY_PERSIST*
- Размещение блока (для *n* реплик): *RAM_DISK*: 1, *DISK*: *n*-1
- Резервное хранилище для генерации: *DISK*
- Резервное хранилище для копирования: *DISK*

Important: В настоящее время политики хранения нельзя редактировать

1.10.3 Настройка памяти в качестве хранилища

Для настройки памяти в качестве хранилища необходимо выполнить:

1. Выключить *DataNode*

Закрыть *DataNode*.

2. Установить часть памяти *DataNode* для *HDFS*

Для использования памяти *DataNode* в качестве хранилища необходимо сначала установить часть памяти *DataNode* для использования *HDFS*.

Например, для выделения 2 ГБ памяти для хранения *HDFS* необходимо использовать команды:

```
sudo mkdir -p /mnt/hdfsramdisk
sudo mount -t tmpfs -o size=2048m tmpfs /mnt/hdfsramdisk
Sudo mkdir -p /usr/lib/hadoop-hdfs
```

3. Назначить тип памяти *RAM_DISK* и включить режим локального чтения данных

Чтобы присвоить *DataNodes* тип памяти *RAM_DISK* и включить режим локального чтения данных, необходимо изменить следующие свойства в файле */etc/hadoop/conf/hdfs-site.xml*:

- Свойство *dfs.name.dir* – определяет, где в локальной файловой системе *DataNode* хранит свои блоки. Чтобы указать *DataNode* в качестве хранилища *RAM_DISK*, необходимо добавить *[RAM_DISK]* в начало пути локальной файловой системы и в свойство *dfs.name.dir*;
- Установить для параметра *dfs.client.read.shortcircuit* значение *true*, чтобы включить режим локального чтения данных.

Например:

```
<property>
  <name>dfs.data.dir</name>
  <value>file:///grid/3/aa/hdfs/data/, [RAM_DISK] file:///mnt/hdfsramdisk/</value>
</property>

<property>
```

```

<name>dfs.client.read.shortcircuit</name>
<value>>true</value>
</property>

<property>
  <name>dfs.domain.socket.path</name>
  <value>/var/lib/hadoop-hdfs/dn_socket</value>
</property>

<property>
  <name>dfs.checksum.type</name>
  <value>NULL</value>
</property>

```

4. Установить политику хранения `LAZY_PERSIST` в файлах или каталогах

Для установки политики хранения `LAZY_PERSIST` в файлах или каталогах необходимо выполнить:

```
hdfs dfsadmin -setStoragePolicy <path> <policyName>
```

Аргументы:

- `<path>` – путь к каталогу или файлу;
- `<policyName>` – название политики хранения.

Пример:

```
hdfs dfsadmin -setStoragePolicy /memory1 LAZY_PERSIST
```

Для возврата политики хранения файла или каталога необходимо выполнить:

```
hdfs dfsadmin -getStoragePolicy <path>
```

Аргументы:

- `<path>` – путь к каталогу или файлу.

Пример:

```
hdfs dfsadmin -getStoragePolicy /memory1 LAZY_PERSIST
```

5. Запуск `DataNode`

Запустить `DataNode`.

1.10.4 Mover для политик хранения

При обновлении параметра политики хранения в файле или каталоге новая политика не применяется автоматически. Необходимо использовать инструмент переноса данных **HDFS** – `mover` – для фактического перемещения блоков (как указано в новой политике хранения).

Средство миграции данных `mover` сканирует указанные файлы в **HDFS** и проверяет, соответствует ли размещение блоков политике хранения. Копии блоков, нарушающих политику хранения, он перемещает в соответствующий тип хранилища для выполнения требований политики.

Команда:

```
hdfs mover [-p <files/dirs> | -f <local file name>]
```

Аргументы:

- `-p <files/dirs>` – список файлов/каталогов **HDFS** для переноса, разделенные пробелами;
- `-f <local file>` – локальный файл, содержащий список файлов/каталогов **HDFS** для переноса.

Important: Если оба параметра `-p` и `-f` опущены, путь по умолчанию является корневым каталогом

Пример:

```
hdfs mover /memory1/testfile
```

1.11 Настройка Rack Awareness

Настройка **Rack Awareness** на кластере **ADH** осуществляется в несколько шагов:

- *Создание скрипта Rack Topology;*
- *Добавление свойства Script Topology в core-site.xml;*
- *Перезапуск HDFS и MapReduce;*
- *Контроль работы Rack Awareness;*

1.11.1 Создание скрипта Rack Topology

Hadoop использует скрипты топологии для определения местоположения стойки узлов и применяет эту информацию для репликации данных блока в резервные стойки.

- Создать скрипт топологии и файл данных. Скрипт топологии должен быть исполняемым.

Пример скрипта топологии. Имя файла: *rack-topology.sh*.

```
#!/bin/bash

# Adjust/Add the property "net.topology.script.file.name"
# to core-site.xml with the "absolute" path the this
# file. ENSURE the file is "executable".

# Supply appropriate rack prefix
RACK_PREFIX=default

# To test, supply a hostname as script input:
if [ $# -gt 0 ]; then

CTL_FILE=${CTL_FILE:-"rack_topology.data"}

HADOOP_CONF=${HADOOP_CONF:-"/etc/hadoop/conf"}

if [ ! -f ${HADOOP_CONF}/${CTL_FILE} ]; then
  echo -n "$RACK_PREFIX/rack "
  exit 0
fi

while [ $# -gt 0 ] ; do
  nodeArg=$1
  exec< ${HADOOP_CONF}/${CTL_FILE}
  result=""
  while read line ; do
    ar=( $line )
    if [ "${ar[0]}" = "$nodeArg" ] ; then
      result="${ar[1]}"
    fi
  done
done
```

```

shift
if [ -z "$result" ] ; then
  echo -n "$RACK_PREFIX/rack "
else
  echo -n "$RACK_PREFIX/rack_$result "
fi
done

else
  echo -n "$RACK_PREFIX/rack "
fi

```

Пример файла данных топологии. Имя файла: *rack_topology.data*.

```

# This file should be:
# - Placed in the /etc/hadoop/conf directory
#   - On the Namenode (and backups IE: HA, Failover, etc)
#   - On the Job Tracker OR Resource Manager (and any Failover JT's/RM's)
# This file should be placed in the /etc/hadoop/conf directory.

# Add Hostnames to this file. Format <host ip> <rack_location>
192.0.2.0 01
192.0.2.1 02
192.0.2.2 03

```

- Скопировать оба этих файла в каталог */etc/hadoop/conf* на всех узлах кластера;
- Запустить скрипт *rack-topology.sh*, чтобы убедиться, что он возвращает правильную информацию о стойке для каждого хоста.

1.11.2 Добавление свойства Script Topology в core-site.xml

- Остановить HDFS;
- Добавить в *core-site.xml* следующее свойство:

```

<property>
  <name>net.topology.script.file.name</name>
  <value>/etc/hadoop/conf/rack-topology.sh</value>
</property>

```

По умолчанию скрипт топологии обрабатывает до *100* заявок за запрос. Можно указать другое количество заявок в свойстве *net.topology.script.number.args*. Например:

```

<property>
  <name>net.topology.script.number.args</name>
  <value>75</value>
</property>

```

1.11.3 Перезапуск HDFS и MapReduce

Перезапустить HDFS и MapReduce.

1.11.4 Контроль работы Rack Awareness

После запуска сервисов для проверки активации **Rack Awareness** можно использовать следующие способы:

- Просмотреть журналы NameNode, расположенные в `/var/log/hadoop/hdfs/` (например: `hadoop-hdfs-namenode-sandbox.log`). Должна быть следующая запись: `:: 014-01-13 15:58:08,495 INFO org.apache.hadoop.net.NetworkTopology: Adding a new node: /rack01/<ipaddress>`
- Команда Hadoop `fsck` должна возвращать на подобии следующего (в случае двух стоек):

```
Status: HEALTHY
Total size: 123456789 B
Total dirs: 0
Total files: 1
Total blocks (validated): 1 (avg. block size 123456789 B)
Minimally replicated blocks: 1 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 3
Average block replication: 3.0
Corrupt blocks: 0
Missing replicas: 0 (0.0 %)
Number of data-nodes: 40
Number of racks: 2
FSCK ended at Mon Jan 13 17:10:51 UTC 2014 in 1 milliseconds
```

- Команда Hadoop `dfsadmin -report` возвращает отчет, содержащий имя стойки рядом с каждой машиной. Отчет должен выглядеть примерно следующим образом (частично):

```
[bsmith@hadoop01 ~]$ sudo -u hdfs hadoop dfsadmin -report
Configured Capacity: 19010409390080 (17.29 TB)
Present Capacity: 18228294160384 (16.58 TB)
DFS Remaining: 5514620928000 (5.02 TB)
DFS Used: 12713673232384 (11.56 TB) DFS Used%: 69.75%
Under replicated blocks: 181
Blocks with corrupt replicas: 0
Missing blocks: 0

-----
Datanodes available: 5 (5 total, 0 dead)

Name: 192.0.2.0:50010 (h2d1.phd.local)
Hostname: h2d1.phd.local
Rack: /default/rack_02
Decommission Status : Normal
Configured Capacity: 15696052224 (14.62 GB)
DFS Used: 314380288 (299.82 MB)
Non DFS Used: 3238612992 (3.02 GB)
DFS Remaining: 12143058944 (11.31 GB)
DFS Used%: 2.00%
DFS Remaining%: 77.36%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Last contact: Thu Jun 12 11:39:51 EDT 2014
```

1.12 Режим локального чтения данных

В HDFS чтение обычно проходит через *DataNode*. Таким образом, когда клиент запрашивает *DataNode* для чтения файла, *DataNode* считывает этот файл с диска и отправляет данные клиенту через сокет TCP. Так называемое “локальное чтение” читает в обход *DataNode*, позволяя клиенту непосредственно прочитать файл. Очевидно, что это возможно только в тех случаях, когда клиент находится в одном месте с данными. Локальное чтение обеспечивает значительное повышение производительности для многих приложений.

Для настройки локального чтения данных необходимо включить *libhadoop.so*. Подробные сведения о включении библиотеки приведены в “Native Libraries”.

Так же для настройки локального чтения данных на HDFS необходимо в файл *hdfs-site.xml* добавить свойства, приведенные далее. Локальное чтение данных должно быть настроено как для *DataNode*, так и для клиента.

- `dfs.client.read.shortcircuit`, значение *true* – включение режима локального чтения данных;
- `dfs.domain.socket.path`, значение `/var/lib/hadoop-hdfs/dn_socket` – путь к сокету домена. В сообщениях при локальном чтении данных используется сокет домена UNIX. Это особый путь в файловой системе, позволяющий связываться клиенту и *DataNodes*. Необходимо установить путь к этому сокету. *DataNode* должен иметь возможность создать этот путь. С другой стороны, создание этого пути не должно быть возможным для любого пользователя, кроме пользователя *hdfs* или *root*. По этой причине часто используются пути в `/var/run` или `/var/lib`;
- `dfs.client.domain.socket.data.traffic`, значение *false* – контролирует, будет ли обычный трафик данных передаваться через сокет домена UNIX. Функция не была сертифицирована релизами ADH, поэтому рекомендуется установить значение *false*;
- `dfs.client.use.legacy.blockreader.local`, значение *false* – установка значения *false* указывает, что используется новая версия локального чтения (на основе HDFS-347). Эта версия поддерживается и рекомендуется для использования с ADH. Значение *true* означает, что используется старый режим локального чтения;
- `dfs.datanode.hdfs-blocks-metadata.enabled`, значение *true* – логический тип данных, который обеспечивает поддержку на стороне сервера *DataNode* для экспериментального *DistributedFileSystem#getFileVBlockStorageLocations* API;
- `dfs.client.file-block-storage-locations.timeout`, значение *60* – таймаут для параллельных RPC, сделанных в *DistributedFileSystem#getFileBlockStorageLocations* (в секундах). Это свойство устарело, но по-прежнему поддерживается для обратной совместимости;
- `dfs.client.file-block-storage-locations.timeout.millis`, значение *60000* – таймаут для параллельных RPC, сделанных в *DistributedFileSystem#getFileBlockStorageLocations* (в миллисекундах). Это свойство заменяет `dfs.client.file-block-storage-locations.timeout` и предлагает более точный уровень детализации;
- `dfs.client.read.shortcircuit.skip.checksum`, значение *false* – если параметр конфигурации установлен, локальное чтение будет пропускать контрольную сумму файлов. Обычно это не рекомендуется, но может быть полезно для специальных настроек. Может пригодиться, если есть собственные контрольные суммы файлов вне HDFS;
- `dfs.client.read.shortcircuit.streams.cache.size`, значение *256* – *DFSClient* поддерживает кэш недавно открытых файловых дескрипторов. Параметр управляет размером кэша. При установке значения выше указанного используются дополнительные дескрипторы файлов, но они могут обеспечить лучшую производительность при рабочей нагрузке с большим количеством запросов;
- `dfs.client.read.shortcircuit.streams.cache.expiry.ms`, значение *300000* – контролирует минимальный промежуток времени нахождения файловых дескрипторов в контексте кэша клиента, прежде чем они могут быть закрыты (в миллисекундах).

XML для вышеуказанных записей:

```
<configuration>
  <property>
    <name>dfs.client.read.shortcircuit</name>
    <value>>true</value>
  </property>

  <property>
    <name>dfs.domain.socket.path</name>
    <value>/var/lib/hadoop-hdfs/dn_socket</value>
  </property>

  <property>
    <name>dfs.client.domain.socket.data.traffic</name>
    <value>>false</value>
  </property>

  <property>
    <name>dfs.client.use.legacy.blockreader.local</name>
    <value>>false</value>
  </property>

  <property>
    <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>
    <value>>true</value>
  </property>

  <property>
    <name>dfs.client.file-block-storage-locations.timeout.millis</name>
    <value>60000</value>
  </property>

  <property>
    <name>dfs.client.read.shortcircuit.skip.checksum</name>
    <value>>false</value>
  </property>

  <property>
    <name>dfs.client.read.shortcircuit.streams.cache.size</name>
    <value>256</value>
  </property>

  <property>
    <name>dfs.client.read.shortcircuit.streams.cache.expiry.ms</name>
    <value>300000</value>
  </property>
</configuration>
```

1.13 Настройка WebHDFS

Для настройки **WebHDFS** необходимо выполнить следующие шаги:

- Настроить WebHDFS, добавив в файл *hdfs-site.xml* свойство:

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>
```

- (Опционально) При запуске защищенного кластера выполнить следующие действия:

- Создать пользователя-принципала сервиса HTTP, используя команду:

```
kadmin: addprinc -randkey HTTP/$<Fully_Qualified_Domain_Name>@<Realm_Name>.COM
```

где `Fully_Qualified_Domain_Name` – хост, на котором развертывается NameNode; `Realm_Name` – название сферы Kerberos.

- Создать файлы *keytab* для принципалов HTTP:

```
kadmin: xst -norandkey -k /etc/security/spnego.service.keytab
HTTP/$<Fully_Qualified_Domain_Name>
```

- Убедиться, что файл *keytab* и принципал связаны с необходимым сервисом:

```
klist -k -t /etc/security/spnego.service.keytab
```

- Добавить в файл *hdfs-site.xml* свойства:

```
<property>
  <name>dfs.web.authentication.kerberos.principal</name>
  <value>HTTP/$<Fully_Qualified_Domain_Name>@<Realm_Name>.COM</value>
</property>
<property>
  <name>dfs.web.authentication.kerberos.keytab</name>
  <value>/etc/security/spnego.service.keytab</value>
</property>
```

где `Fully_Qualified_Domain_Name` – хост, на котором развертывается NameNode; `Realm_Name` – название сферы Kerberos.

- Перезапустить сервисы NameNode и DataNode.

1.14 Добавление/удаление и обслуживание/декоммиссия HDFS DataNode

Для добавления или удаления HDFS DataNode с хостов или приведения HDFS DataNode в состояние обслуживания или декоммиссии на хосте необходимо воспользоваться соответствующими кнопками выпадающего меню, доступного по нажатию на иконку в поле “*Actions*” сервиса HDFS:

Эти кнопки являются простым интерфейсом для управления состояниями DataNode, более подробно описанной в документации [Apache Hadoop](#).

Important: Описанные ниже операции не удаляют/добавляют хост из кластера – они лишь управляют компонентом HDFS DataNode на хостах. Удаление хоста из кластера возможно в разделе “*Hosts*” кластера в случаях, когда к хосту не привязан ни один компонент. Добавление хоста производится согласно инструкции.

1.14.1 Добавление HDFS DataNode

Для добавления одной или нескольких дополнительных HDFS DataNode на хосты кластера необходимо:

The screenshot displays the ARENA DATA management console for a cluster named 'Sacred Amur' (Hadoop 3.1.0.5). The interface includes a navigation sidebar on the left with options like 'Main', 'Services', 'Hosts', and 'Configuration'. The main content area shows a table of services. The 'HDFS' service is highlighted, and its 'Actions' column is expanded to show a dropdown menu with the following options: Check, Decommiss DataNodes, Add DataNodes, Maintenance DataNodes, Recommiss DataNodes, Remove DataNodes, Disable, Restart, Start, and Stop. The 'Hive' service is also visible in the table below HDFS.

Name	Version	State	Status	Actions	Config
HDFS	3.1.0	enabled			
Hive	3.1.0	created			

Рис.1.4.: Выпадающее меню “Actions” сервиса HDFS

1. нажать кнопку *“Add DataNodes”*, что приведёт к появлению окна, аналогичному разделу *“Hosts - Components”* кластера, описанному в разделе *install_components*;
2. любым из двух способов назначить добавляемому компоненту хост (компонент HDFS DataNode будет выделен белым как возможный к расширению):
 - выбрать компонент в колонке *“Components”* и назначить для него хост в колонке *“Hosts”*;
 - выбрать хост в колонке *“Hosts”* и определить для него компонент в колонке *“Components”*.
3. нажать кнопку *“Run”* в нижней части окна.

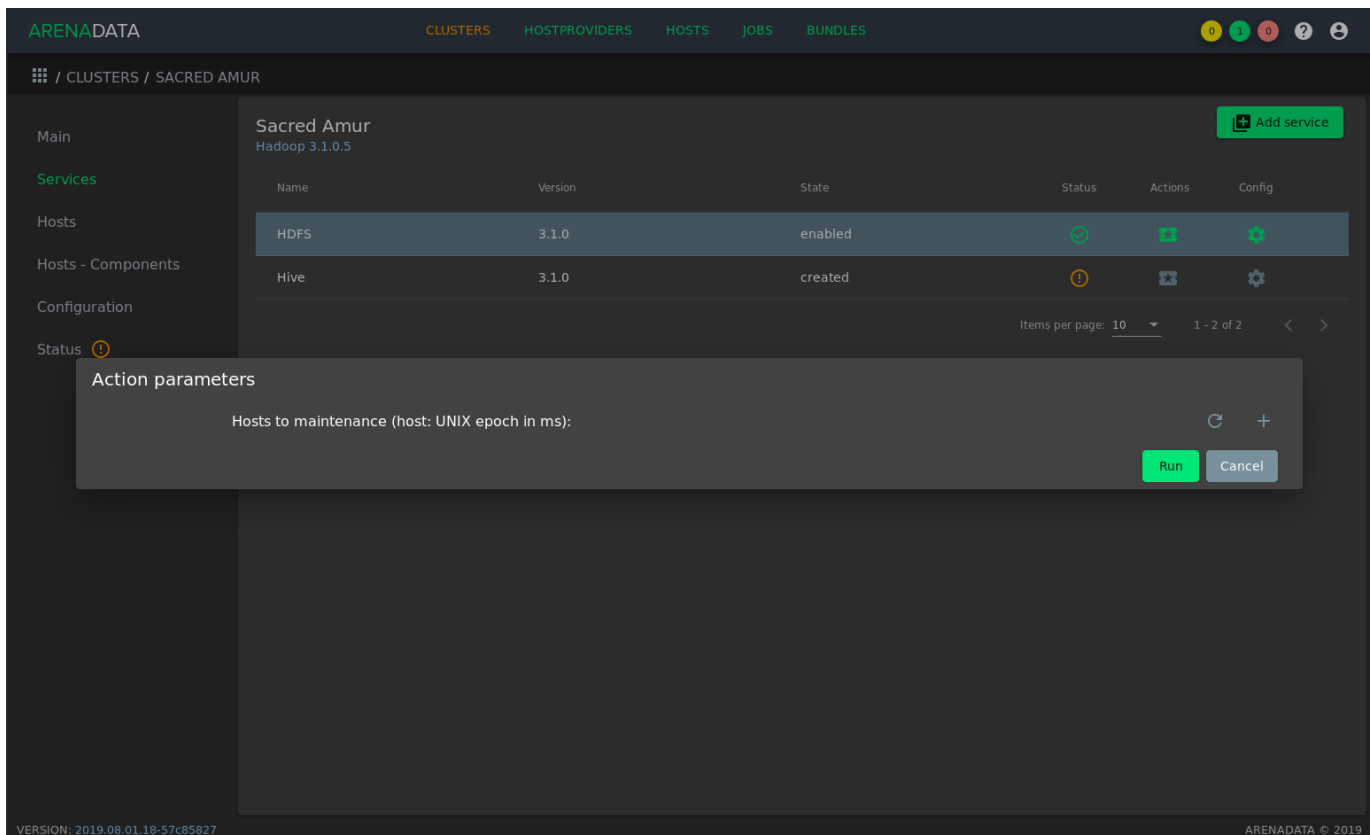
1.14.2 Обслуживание HDFS DataNode

В случае необходимости вывести хост из работы на короткий промежуток времени его возможно перевести в режим обслуживания.

В этом режиме HDFS не будет относить блоки DataNode как необходимые для репликации (under-replicated blocks), что также может быть полезно и в других ситуациях кроме недолгосрочного обслуживания.

Для перевода одной или нескольких HDFS DataNode в режим обслуживания необходимо:

1. нажать кнопку *“Maintenance DataNodes”*, что приведёт к появлению следующего окна:



2. для добавления хоста в список на обслуживание необходимо нажать *“+”* и ввести fqdn хоста и время окончания окна обслуживания в формате **UNIX-времени** в миллисекундах. Через *“+”* возможно добавить несколько хостов.
3. нажать кнопку *“Run”* в нижней части окна.

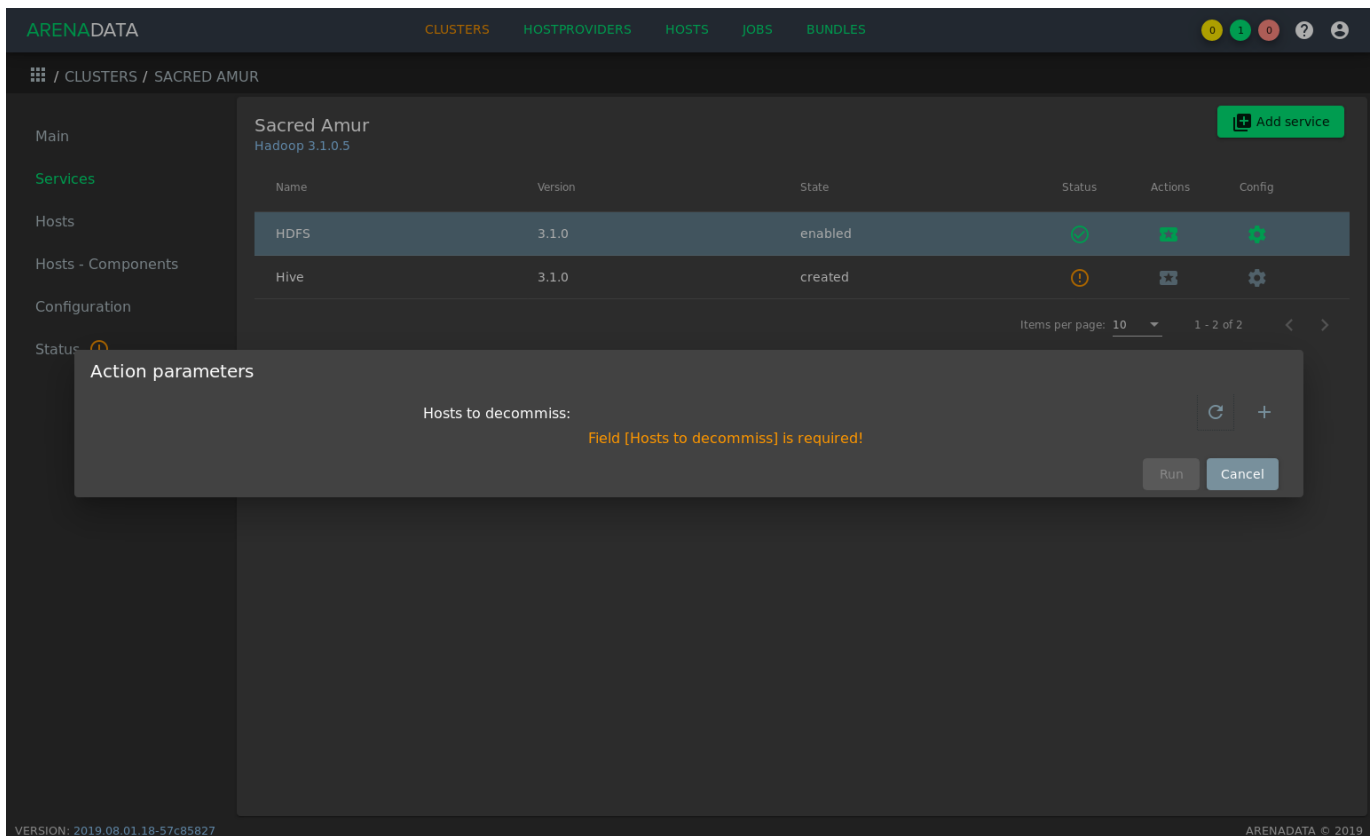
1.14.3 Декоммиссия HDFS DataNode

В случае необходимости вывести хост из работы на длительный промежуток времени или удалить из кластера его сначала необходимо декоммиссовать.

В этом режиме HDFS будет относить блоки DataNode как необходимые для репликации (under-replicated blocks).

Для декоммиссии одной или нескольких HDFS DataNode:

1. нажать кнопку “*Decommiss DataNodes*”, что приведёт к появлению следующего окна:



2. для добавления хоста в список декоммиссованных необходимо нажать “+” и ввести fqdn хоста. Через “+” возможно добавить несколько хостов.
3. нажать кнопку “*Run*” в нижней части окна.

1.14.4 Реконмиссия HDFS DataNode

Для реконмиссии (выведения хоста из состояния обслуживания или декоммиссии) одной или нескольких HDFS DataNode:

1. нажать кнопку “*Recommiss DataNodes*”, что приведёт к появлению всплывающего окна, аналогичному окну декоммиссии.
2. для добавления хоста в список реконмиссованных необходимо нажать “+” и ввести fqdn хоста. Через “+” возможно добавить несколько хостов.
3. нажать кнопку “*Run*” в нижней части окна.

1.14.5 Удаление HDFS DataNode

Для удаления одной или нескольких HDFS DataNode с хостов кластера необходимо:

1. нажать кнопку “*Remove DataNodes*”, что приведёт к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в разделе `install_components`;
2. любым из двух способов удалить привязку компонента к хосту (компонент HDFS DataNode будет выделен белым как возможный к удалению с хостов):
 - выбрать компонент в колонке “*Components*” и убрать выделение с хостов в колонке “*Hosts*”, рамки которых выделены зелёным;
 - выбрать хост в колонке “*Hosts*” и убрать выделение с компонента HDFS DataNode в колонке “*Components*” если рамка компонента HDFS DataNode выделяется зелёным.
3. нажать кнопку “*Run*” в нижней части окна.

1.15 POSIX ACL на HDFS

Списки ACL на HDFS реализуются с помощью модели ACL POSIX, которая работает так же, как и в файловой системе Linux:

- *Совместимость и применение;*
- *Доступ;*
- *Обратная связь;*
- *Обратная совместимость;*
- *Ограничения;*
- *Символические ссылки;*
- *Снапшоты;*
- *Инструментарий;*
- *Доступ нескольким пользователям;*
- *Hive Partitioned Tables;*
- *ACL по умолчанию;*
- *ACL/Permissions Only;*
- *Блокировка доступа для пользователя;*
- *Sticky Bit.*

1.15.1 Совместимость и применение

HDFS может связывать дополнительный список ACL с любым файлом или каталогом. Все операции HDFS, которые обеспечивают соблюдение разрешений, выраженных с помощью **Permission Bits**, также должны обеспечить любой ACL, который определен для файла или каталога. Так же как и любая существующая логика, которая обходит **Permission Bits**, обходит и ACL. Включая супер-пользователя HDFS и установку `false` в конфигурации `dfs.permissions`.

1.15.2 Доступ

HDFS поддерживает операции по настройке и получению **ACL**, связанного с файлом или каталогом. Эти операции доступны через несколько ориентированных на пользователя конечных точек. Эти конечные точки включают **FsShell CLI**, программную манипуляцию через классы **FileSystem** и **FileContext**, **WebHDFS** и **NFS**.

1.15.3 Обратная связь

К разрешениям любого файла или каталога с **ACL** добавляется символ + (вывод команды `ls -l`).

1.15.4 Обратная совместимость

Реализация **ACL** обратно совместима с существующими **Permission Bits**. Изменения, применяемые посредством разрешенных битов (то есть `chmod`), также отображаются как изменения в **ACL**. Аналогично, изменения, применяемые к записям **ACL** для базовых классов пользователей (“Owner”, “Group” и “Other”), отображаются в виде изменений в **Permission Bits**.

Другими словами, операции **Permission Bits** и **ACL** управляют совместно используемой моделью, и операции **Permission Bits** можно рассматривать как подмножество операций **ACL**.

1.15.5 Накладные расходы

Добавление **ACL** не приводит к негативному влиянию на потребление системных ресурсов при развертывании. Он включает в себя процессор, память, диск и пропускную способность сети. Но использование списков **ACL** влияет на производительность **NameNode**, поэтому рекомендуется использовать **Permission Bits**, прежде чем **ACL**.

1.15.6 Ограничения

Количество записей в одном списке **ACL** ограничено максимум до 32. Попытки добавить записи **ACL** сверх максимума выполняются с ошибкой, обращенной к пользователю. Это делается по двум причинам: упростить управление и ограничить потребление ресурсов.

Списки **ACL** с большим количеством записей, как правило, трудно понять и могут указывать на то, что требования лучше устраняются путем определения дополнительных групп или пользователей. Также такие списки требуют большей памяти и хранилища, и для каждой проверки разрешений требуется больше времени.

1.15.7 Символические ссылки

У символических ссылок нет собственных списков **ACL**, поэтому они рассматриваются как разрешения по умолчанию (777 в **Permission Bits**). Операции, которые изменяют список символической ссылки, вместо этого изменяют саму символическую ссылку.

1.15.8 Снапшоты

При создании снапшота все списки **ACL** блокируются. Изменения в **ACL** в момент создания снапшота не фиксируются.

1.15.9 Инструментарий

Инструментарий для **Permission Bits** не подходит для **ACL**. Списки включаются командой оболочки `cp -p` и `distcp -p`.

1.15.10 Доступ нескольким пользователям

Когда нескольким пользователям требуется доступ для чтения к файлу и при этом ни один из пользователей не является владельцем файла. Кроме того пользователи не являются членами общей группы, поэтому невозможно использовать групповые разрешения. В таком случае устанавливается ACL-доступ, содержащий несколько именованных пользовательских записей:

```
ACLs on HDFS supports the following use cases:
```

1.15.11 Доступ нескольким группам

Когда нескольким группам требуется чтение и запись в файл и при этом нет группы, объединяющей всех необходимых пользователей, поэтому невозможно использовать групповые разрешения. В таком случае устанавливается ACL-доступ, содержащий несколько именованных групповых записей:

```
group:sales:rw-
group:execs:rw-
```

1.15.12 Hive Partitioned Tables

В случае, когда **Hive** содержит секционированную таблицу данных и ключ раздела, например, – *country*. **Hive** сохраняет секционированные таблицы с помощью отдельного подкаталога для каждого определенного значения ключа, поэтому структура файловой системы выглядит так:

```
user
|-- hive
    |-- warehouse
        |-- sales
            |-- country=CN
            |-- country=GB
            |-- country=US
```

Группа *salesadmin* – группа для всех файлов. Члены группы имеют доступ на чтение и запись ко всем файлам. Отдельные группы, зависящие от конкретной страны, могут запускать запросы на использование, которые только считывают данные для конкретной страны, например, *sales_CN*, *sales_GB* и *sales_US*. У этих групп нет доступа на запись.

Такой вариант использования можно решить, установив ACL-доступ в каждом подкаталоге, содержащем запись собственной группы и именованной группы:

```
country=CN
group::rwx
group:sales_CN:r-x

country=GB
group::rwx
group:sales_GB:r-x

country=US
group::rwx
group:sales_US:r-x
```

Important: Функциональность записи ACL группы-владельца (запись группы без имени) эквивалентна установленным Permission Bits

Авторизация на основе хранилища в **Hive** в настоящее время не учитывает разрешения **ACL** в **HDFS**. Доступ проверяется с использованием традиционной модели разрешений **POSIX**.

1.15.13 ACL по умолчанию

Администратор файловой системы или владелец поддерева может определить политику доступа, применимую ко всему поддереву не только к текущему набору файлов и каталогов, но также к новым файлам и каталогам, которые будут добавляться позже.

Этот вариант использования решается установкой в каталог **ACL по умолчанию**. При этом список может содержать любую произвольную комбинацию записей. Например:

```
default:user::rwx
default:user:bruce:rw-
default:user:diana:r--
default:user:clark:rw-
default:group::r--
default:group:sales::rw-
default:group:execs::rw-
default:others:---
```

Важно отметить, что **ACL по умолчанию** копируется из каталога во вновь созданные дочерние файлы и каталоги во время их создания. Если изменить **ACL по умолчанию** для каталога, это не повлияет на списки файлов и подкаталогов, которые уже существуют. **ACL по умолчанию** никогда не учитываются при применении разрешений. Они используются только для определения списка **ACL**, который новые файлы и подкаталоги будут получать автоматически при их создании.

1.15.14 ACL/Permissions Only

Списки управления доступом **HDFS** поддерживают развертывания, в которых может потребоваться использование только битов разрешений, а не **ACL** с именованными записями пользователей и групп. **Permission Bits** эквивалентны минимальному **ACL**, содержащему только 3 записи. Например:

```
user::rw-
group::r--
others:---
```

1.15.15 Блокировка доступа для пользователя

Для примера создано поддерево файловой системы с глубоким вложением, доступное для чтения всем миром, и к которому устанавливается требование заблокировать доступ для определенного пользователя ко всем файлам в этом поддереве.

В таком случае устанавливается **ACL** в корне поддерева с именованной записью пользователя, которая лишает пользователя доступа.

```
dir1
|-- dir2
    |-- dir3
        |-- file1
        |-- file2
        |-- file3
```

Установка следующего **ACL** на *dir2* блокирует доступ Брюса к *dir3*, *file1*, *file2* и *file3*:

```
user:bruce:---
```

Удаление разрешений на *dir2* означает, что Брюс не может получить к нему доступ и, следовательно, не может видеть ни один из его дочерних элементов. Это также означает, что доступ автоматически блокируется для любых вновь добавленных файлов в *dir2*. То есть если *file4* создается под *dir3*, Брюс не сможет получить к нему доступ.

1.15.16 Sticky Bit

Когда нескольким именованным пользователям или группам требуется полный доступ к каталогу общего назначения, например, / *tmp*. Однако разрешения “Write” и “Execute” для каталога также дают пользователям возможность удаления или переименовывания любых файлов в каталоге, включая созданные другими пользователями. Такие разрешения необходимо ограничить, чтобы у пользователей был допуск на удаление или переименование созданных только ими файлов.

Этот случай можно решить, объединив **ACL** с **Sticky bit** – это существующая функциональность, которая в настоящее время работает с **Permission Bits**, и будет продолжать работать как ожидается в сочетании с **ACL**.

Глава 2

Руководство администратора по Hive

В руководстве приведены сведения по настройке сервиса Hive.

Документ может быть полезен администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и сопровождение кластера.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

2.1 Tez

2.1.1 Tez UI

Tez предоставляет собственный пользовательский интерфейс, который взаимодействует с **YARN Application Timeline Server** и отражает текущий и исторический вид приложений Tez в веб-приложении *Tez UI*.

Important: Для корректной работы веб-приложения Tez UI на хосте, с которого происходит взаимодействие с веб-приложением, необходима сетевая и DNS связность с YARN Application Timeline Server

Глава 3

Руководство администратора по Spark

В руководстве приведены сведения по настройке Spark.

Документ может быть полезен администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и сопровождение кластера.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

3.1 Динамическая аллокация ресурсов

Spark предоставляет механизм динамической регуляции ресурсов, которые требуются приложениям, на основе загрузки кластера. Это означает, что приложение может высвободить занятые им ресурсы в пользу кластера при условии, что приложение не использует эти ресурсы в данный момент. При необходимости в последующем приложение может запросить ресурсы у кластера. Эта опция особенно полезна в ситуациях, при которых несколько приложений делят ресурсы **Spark** кластера.

Более подробно динамическая аллокация описана в документации [Apache Spark](#)

3.1.1 Включение динамической аллокации

Для включения динамической аллокации ресурсов необходимо:

1. Нажать кнопку “*Switch dynamic allocation*” сервиса Spark;
2. Перезапустить сервисы YARN и Spark (последовательность важна) посредством кнопки “*Restart*” соответствующего сервиса.

Глава 4

Руководство администратора по YARN

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

4.1 Добавление/удаление и декомиссия YARN NodeManager

Для добавления или удаления *YARN NodeManager* с хостов или приведения *YARN NodeManager* в состояние декомиссии на хосте необходимо воспользоваться соответствующими кнопками выпадающего меню, доступного по нажатию на иконку в поле “*Actions*” сервиса *YARN* (Рис.4.3).

Эти кнопки являются простым интерфейсом для управления состояниями *NodeManager*, более подробно описанным в документации [Apache Hadoop](#).

Important: Описанные далее операции не удаляют/добавляют хост из кластера – они лишь управляют компонентом *YARN NodeManager* на хостах

Удаление хоста из кластера возможно в разделе “*Hosts*” кластера в случаях, когда к хосту не привязан ни один компонент.

Добавление хоста осуществляется согласно инструкции.

4.1.1 Добавление YARN NodeManager

Для добавления одной или нескольких дополнительных *YARN NodeManager* на хосты кластера необходимо:

1. Нажать кнопку “*Add NodeManagers*”, что приводит к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в `install_components`.
2. Любым из двух способов назначить добавляемому компоненту хост (компонент *YARN NodeManager* выделяется белым как возможный к расширению):
 - Выбрать компонент в колонке “*Components*” и назначить для него хост в колонке “*Hosts*”;
 - Выбрать хост в колонке “*Hosts*” и определить для него компонент в колонке “*Components*”.
3. Нажать кнопку “*Run*” в нижней части окна.

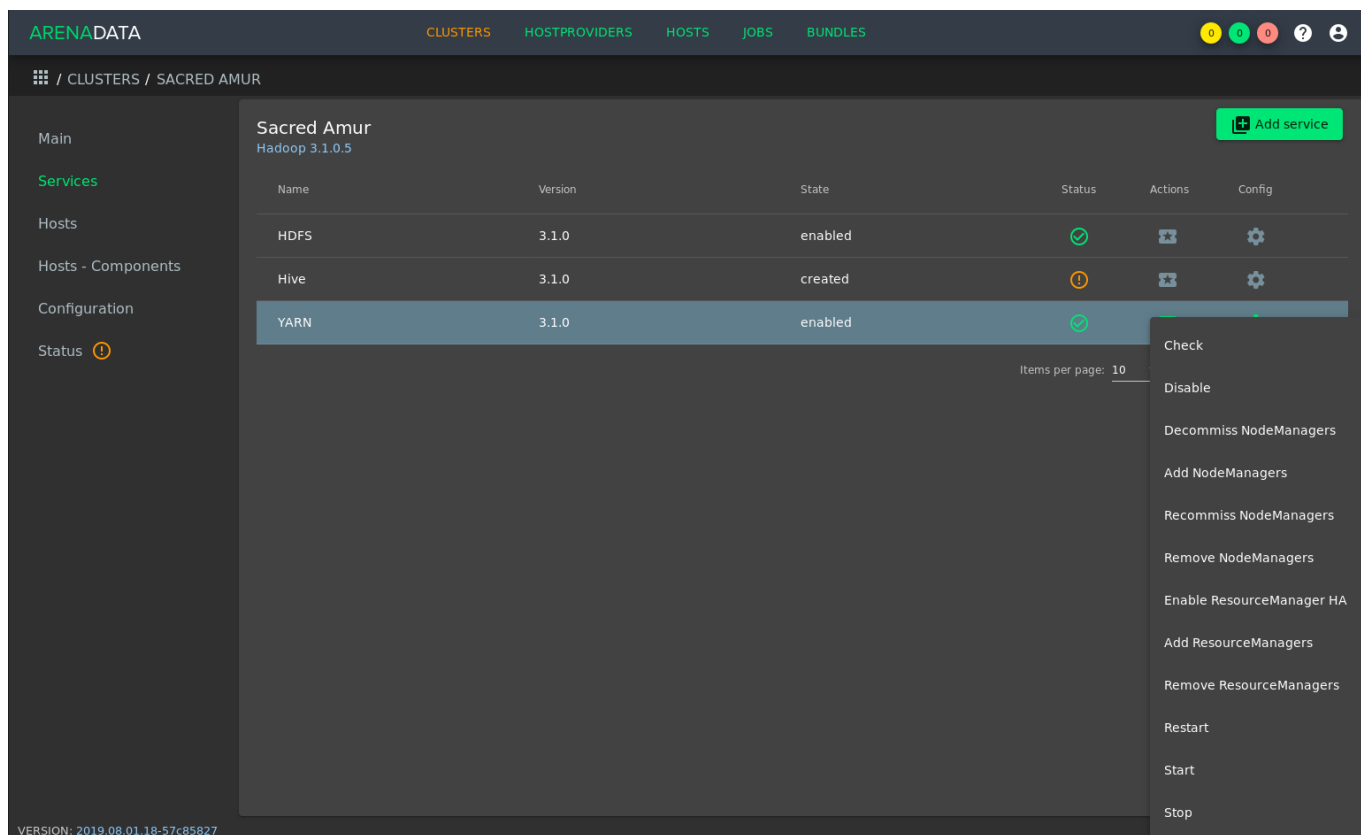


Рис.4.1.: Выпадающее меню “Actions” сервиса YARN

4.1.2 Декомиссия YARN NodeManager

В случае необходимости вывести хост из работы или удалить из кластера его сначала необходимо декомиссовать.

Существует два типа декомиссии:

- Нормальный – в этом режиме нода *YARN NodeManager* прекращает работу безусловно, не ожидая завершения работающих на ноды контейнеров;
- Изящный (*graceful*) – в этом режиме планировщик *YARN* не будет назначать новые контейнеры декомиссованной ноды *YARN NodeManager*, а работающие контейнеры продолжают работу до истечения назначенного таймаута.

Для декомиссии одной или нескольких *YARN NodeManager* необходимо выполнить действия:

1. Нажать кнопку “*Decommiss NodeManagers*”, что приводит к появлению окна “*Action parameters*” (Рис.4.2.).

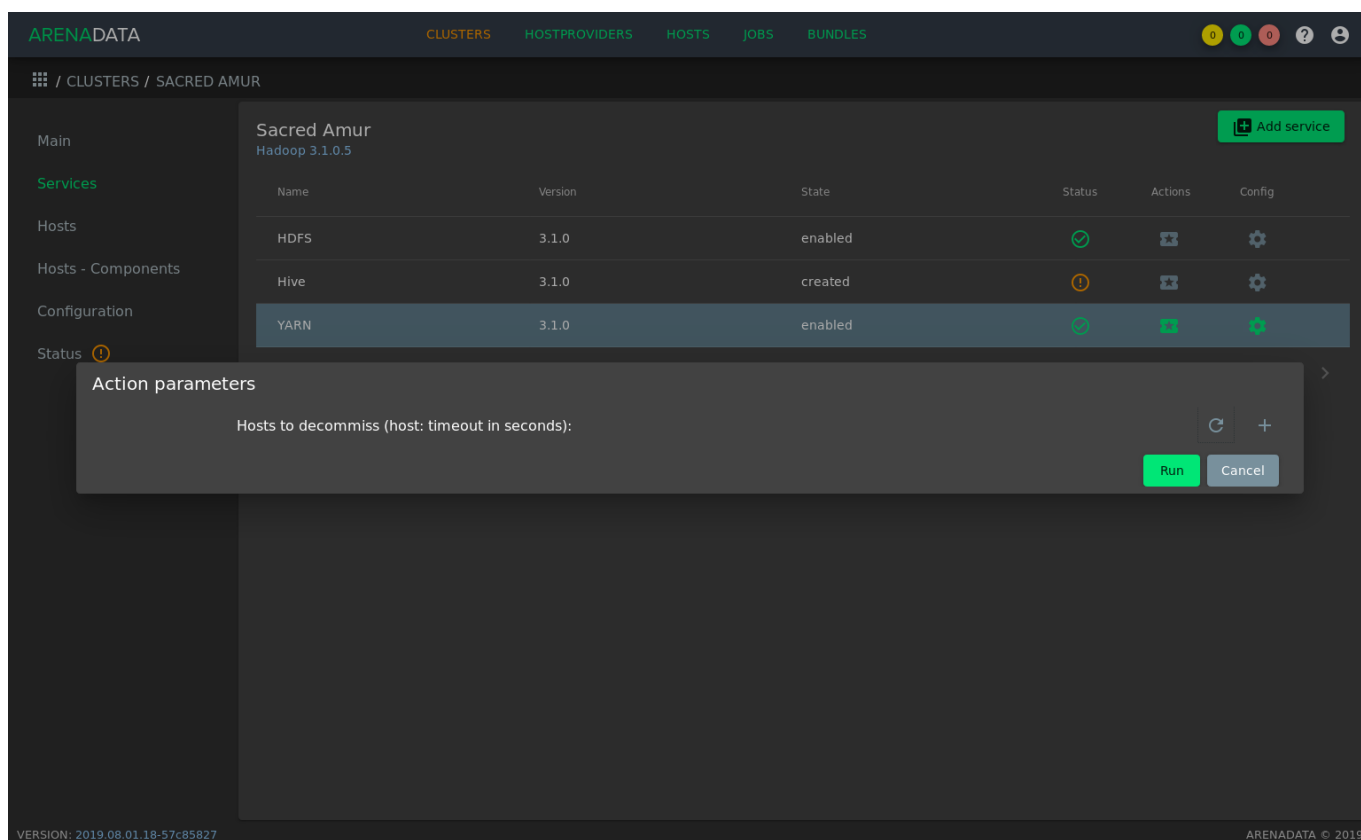


Рис.4.2.: “Action parameters”

2. Для добавления хоста в список декомиссованных необходимо нажать “+” и ввести fqdn хоста и таймаут в секундах. Через “+” можно добавить несколько хостов.
3. Нажать кнопку “*Run*” в нижней части окна.

4.1.3 Рекомиссия YARN NodeManager

Для рекомиссии (выведения хоста из состояния декомиссии) одной или нескольких *YARN NodeManager* необходимо:

1. Нажать кнопку “*Recommis NodeManagers*”, что приводит к появлению всплывающего окна, аналогичному окну декомиссии (Рис.4.2).
2. Для добавления хоста в список рекомиссованных необходимо нажать “+” и ввести fqdn хоста. Через “+” можно добавить несколько хостов.
3. Нажать кнопку “*Run*” в нижней части окна.

4.1.4 Удаление YARN NodeManager

Для удаления одной или нескольких *YARN NodeManager* с хостов кластера необходимо:

1. Нажать кнопку “*Remove NodeManagers*”, что приводит к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в `install_components`.
2. Любым из двух способов удалить привязку компонента к хосту (компонент *YARN NodeManager* выделяется белым как возможный к удалению с хостов):
 - Выбрать компонент в колонке “*Components*” и убрать выделение с хостов в колонке “*Hosts*”, рамки которых выделены зеленым;
 - Выбрать хост в колонке “*Hosts*” и убрать выделение с компонента *YARN NodeManager* в колонке “*Components*”, если рамка компонента *YARN NodeManager* выделяется зеленым.
3. Нажать кнопку “*Run*” в нижней части окна.

4.2 Высокая доступность (HA) и добавление/удаление YARN ResourceManager

При изначальной установке сервиса *YARN* в кластере **ADH** компонент *YARN ResourceManager* является **единой точкой отказа (Single point of failure)**. Это означает, что для обслуживания хоста или в случае его отказа, весь сервис *YARN* становится неработоспособным. Для решения этой проблемы существует механизм высокой доступности *YARN ResourceManager* (high availability или HA), более подробно описанный в документации **Apache Hadoop**. Этот механизм предполагает больше одного хоста с *YARN ResourceManager*, требует сервиса *ZooKeeper*, но устойчив к недоступности компонента *YARN ResourceManager* на хостах кластера. До тех пор, пока хотя бы один хост с *YARN ResourceManager* остается доступным – весь сервис *YARN* тоже остается работоспособным.

После активации HA компонент *YARN ResourceManager* становится масштабируемым, что означает возможность добавления или удаления данного компонента с хостов кластера.

Для активации HA, добавления или удаления *YARN ResourceManager* с хостов необходимо воспользоваться соответствующими кнопками выпадающего меню, доступного по нажатию на иконку в поле “*Actions*” сервиса *YARN* (Рис.4.3).

Important: Описанные далее операции не удаляют/добавляют хост из кластера – они лишь управляют компонентом *YARN ResourceManager* на хостах

Удаление хоста из кластера возможно в разделе “*Hosts*” кластера в случаях, когда к хосту не привязан ни один компонент.

Добавление хоста осуществляется согласно инструкции.

4.2.1 Активация HA YARN ResourceManager

The screenshot displays the ARENA DATA web interface. The top navigation bar includes 'CLUSTERS', 'HOSTPROVIDERS', 'HOSTS', 'JOBS', and 'BUNDLES'. The current view is for the 'Sacred Amur' cluster, showing a table of services. The 'YARN' service is selected, and its 'Actions' dropdown menu is open, listing various management options.

Name	Version	State	Status	Actions	Config
HDFS	3.1.0	enabled	✓	⌵ ⚙	⚙
Hive	3.1.0	created	⚠	⌵ ⚙	⚙
YARN	3.1.0	enabled	✓	⌵ ⚙	⚙

- Check
- Disable
- Decommiss NodeManagers
- Add NodeManagers
- Recommiss NodeManagers
- Remove NodeManagers
- Enable ResourceManager HA
- Add ResourceManagers
- Remove ResourceManagers
- Restart
- Start
- Stop

Items per page: 10

VERSION: 2019.08.01.18-57c85827

Рис.4.3.: Выпадающее меню “Actions” сервиса YARN

Important: Операция активации высокой доступности (HA) необратима. Для функционирования сервиса YARN после активации HA требуется доступный сервис ZooKeeper и два хоста с YARN ResourceManager

Перед непосредственной активацией HA необходимо убедиться в удовлетворении двух условий:

1. Сервис ZooKeeper установлен;
2. В кластер ADH добавлено больше одного хоста.

Для активации высокой доступности (HA) *YARN ResourceManager* необходимо:

1. Нажать кнопку “*Enable ResourceManagers HA*”, что приводит к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в `install_components`.
2. Любым из двух способов назначить добавляемому компоненту хост (компонент *YARN ResourceManager* выделяется белым как возможный к расширению):
 - Выбрать компонент в колонке “*Components*” и назначить для него хост в колонке “*Hosts*”;
 - Выбрать хост в колонке “*Hosts*” и определить для него компонент в колонке “*Components*”.
3. Нажать кнопку “*Run*” в нижней части окна.

4.2.2 Добавление YARN ResourceManager

Для добавления одной или нескольких дополнительных *YARN ResourceManager* на хосты кластера необходимо:

1. Нажать кнопку “*Add ResourceManagers*”, что приводит к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в `install_components`.
2. Любым из двух способов назначить добавляемому компоненту хост (компонент *YARN ResourceManager* выделяется белым как возможный к расширению):
 - Выбрать компонент в колонке “*Components*” и назначить для него хост в колонке “*Hosts*”;
 - Выбрать хост в колонке “*Hosts*” и определить для него компонент в колонке “*Components*”.
3. Нажать кнопку “*Run*” в нижней части окна.

4.2.3 Удаление YARN ResourceManager

Important: Для функционирования сервиса YARN после активации HA требуется как минимум два хоста с YARN ResourceManager

Для удаления одной или нескольких *YARN ResourceManager* с хостов кластера необходимо:

1. Нажать кнопку “*Remove ResourceManagers*”, что приводит к появлению окна, аналогичному разделу “*Hosts - Components*” кластера, описанному в `install_components`.
2. Любым из двух способов удалить привязку компонента к хосту (компонент *YARN ResourceManager* выделяется белым как возможный к удалению с хостов):
 - Выбрать компонент в колонке “*Components*” и убрать выделение с хостов в колонке “*Hosts*”, рамки которых выделены зеленым;
 - Выбрать хост в колонке “*Hosts*” и убрать выделение с компонента *YARN ResourceManager* в колонке “*Components*”, если рамка компонента *YARN ResourceManager* выделяется зеленым.
3. Нажать кнопку “*Run*” в нижней части окна.

Глава 5

Карта портов

5.1 Порты HDFS

Список портов, используемых по умолчанию для различных **HDFS**-сервисов, представлен в таблице.

Таблица 5.1.: Порты HDFS

Сервис	Характеристика
NameNode WebUI	Сервер – Master Nodes <ul style="list-style-type: none">• Порт по умолчанию – 50070<ul style="list-style-type: none">– Протокол: http– Описание: Web-интерфейс для работы с файловой системой HDFS– Доступ пользователям: Да (администратор, разработчик, поддержка)– Параметр конфигурации: dfs.http.address• Порт по умолчанию – 50470<ul style="list-style-type: none">– Протокол: https– Описание: Безопасный http-сервис– Доступ пользователям: Да (администратор, разработчик, поддержка)– Параметр конфигурации: dfs.https.address
NameNode metadata service	Сервер – Master Nodes <ul style="list-style-type: none">• Порт по умолчанию – 8020/9000<ul style="list-style-type: none">– Протокол: IPC– Описание: Взаимодействие с метаданными файловой системы– Доступ пользователям: Да (все клиенты для прямого взаимодействия с HDFS)– Параметр конфигурации: fs.default.name

Сервис	Характеристика
DataNode	<p>Сервер – Все Slave Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 50075 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для доступа к логам, статусам и пр. – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.datanode.http.address • Порт по умолчанию – 50475 <ul style="list-style-type: none"> – Протокол: https – Описание: Безопасный http-сервис – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.datanode.https.address • Порт по умолчанию – 50010 <ul style="list-style-type: none"> – Описание: Передача данных – Параметр конфигурации: dfs.datanode.address • Порт по умолчанию – 50020 <ul style="list-style-type: none"> – Протокол: IPC – Описание: Операции с метаданными – Доступ пользователям: Нет – Параметр конфигурации: dfs.datanode.ipc.address
Secondary NameNode	<p>Сервер – Secondary NameNodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 50090 <ul style="list-style-type: none"> – Протокол: http – Описание: Чекпоинт для метаданных NameNode – Доступ пользователям: Нет – Параметр конфигурации: dfs.secondary.http.address
HDFS HFTP	<p>Сервер – Master NameNodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 50470 <ul style="list-style-type: none"> – Протокол: https – Описание: Порт для доступа к HFTP файловой системы – Параметр конфигурации: dfs.https.port
JournalNode Web UI	<p>Сервер – Master NameNodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8480 <ul style="list-style-type: none"> – Протокол: http – Описание: Порт для доступа к HFTP файловой системы – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.journalnode.http-address • Порт по умолчанию – 8481 <ul style="list-style-type: none"> – Протокол: https – Описание: Безопасный http-сервис – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.journalnode.https-address

5.2 Порты MapReduce

Список портов, используемых по умолчанию для различных **MapReduce**-сервисов, представлен в таблице.

Таблица 5.2.: Порты MapReduce

Сервис	Характеристика
JobTracker WebUI	Сервер – Master Nodes <ul style="list-style-type: none"> • Порт по умолчанию – 50030 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для работы JobTracker – Доступ пользователям: Да – Параметр конфигурации: <code>mapred.job.tracker.http.address</code>
JobTracker	Сервер – Master Nodes <ul style="list-style-type: none"> • Порт по умолчанию – 8021 <ul style="list-style-type: none"> – Протокол: IPС – Описание: Для публикации заданий – Доступ пользователям: Да (все клиенты, которым требуется запуск MR, Hive, Pig и т.д.) – Параметр конфигурации: <code>mapred.job.tracker</code>
TaskTracker Web UI and Shuffle	Сервер – Все Slave Nodes <ul style="list-style-type: none"> • Порт по умолчанию – 50060 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для DataNode (логи, статус) – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: <code>mapred.task.tracker.http.address</code>
History Server WebUI	<ul style="list-style-type: none"> • Порт по умолчанию – 51111 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для истории заданий – Доступ пользователям: Да – Параметр конфигурации: <code>mapreduce.history.server.http.address</code>
MapReduce Shuffle Port	<ul style="list-style-type: none"> • Порт по умолчанию – 13562 <ul style="list-style-type: none"> – Описание: Порт, на котором работает ShuffleHandler – Доступ пользователям: Нет – Параметр конфигурации: <code>mapreduce.shuffle.port</code>

5.3 Порты YARN

Список портов, используемых по умолчанию для различных YARN-сервисов, представлен в таблице.

Таблица 5.3.: Порты YARN

Сервис	Характеристика
ResourceManager WebUI	<p>Сервер – Master Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8088 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для Resource Manager – Доступ пользователям: Да – Параметр конфигурации: yarn.resourcemanager.webapp.address
ResourceManager	<p>Сервер – Master Nodes (ResourceManager Node)</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8050 <ul style="list-style-type: none"> – Протокол: IPC – Описание: Для публикации заданий – Доступ пользователям: Да (все клиенты, которым требуется запуск YARN-приложений) – Параметр конфигурации: yarn.resourcemanager.address • Порт по умолчанию – 8025 <ul style="list-style-type: none"> – Протокол: http – Описание: Web-интерфейс для DataNode (логи, статус) – Доступ пользователям: Да (все клиенты, которым требуется запуск YARN-приложений) – Параметр конфигурации: mapred.task.tracker.http.address • Порт по умолчанию – 9099 <ul style="list-style-type: none"> – Протокол: http – Описание: Прокси для Resource Manager – Доступ пользователям: Да – Параметр конфигурации: yarn.web-proxy.address • Порт по умолчанию – 8141 <ul style="list-style-type: none"> – Протокол: http – Описание: Адрес планировщика – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.resourcemanager.admin.address
Scheduler	<p>Сервер – Master Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8030 <ul style="list-style-type: none"> – Описание: Адрес планировщика – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.resourcemanager.scheduler.address

Сервис	Характеристика
NodeManager	<p>Сервер – Master Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 45454 <ul style="list-style-type: none"> – Протокол: http – Описание: Адрес NodeManager – Параметр конфигурации: yarn.nodemanager.address <p>Сервер – Slave Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8040 <ul style="list-style-type: none"> – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.localizer.address • Порт по умолчанию – 8042 <ul style="list-style-type: none"> – Протокол: http – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.webapp.address • Порт по умолчанию – 8044 <ul style="list-style-type: none"> – Протокол: https – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.webapp.https.address
Timeline Server	<p>Сервер – Master Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 10200 <ul style="list-style-type: none"> – Протокол: http – Описание: Адрес Timeline Server – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.timeline-service.address • Порт по умолчанию – 8188 <ul style="list-style-type: none"> – Протокол: http – Описание: Адрес Timeline Server Webapp – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.timeline-service.webapp.address • Порт по умолчанию – 8190 <ul style="list-style-type: none"> – Протокол: https – Описание: Адрес Timeline Server Webapp https – Доступ пользователям: Да (администратор, разработчик, поддержка)

5.4 Порты Hive

Список портов, используемых по умолчанию для различных **Hive**-сервисов, представлен в таблице.

Таблица 5.4.: Порты Hive

Сервис	Характеристика
Hive Server2	<p>Сервер – Hive Server машина</p> <ul style="list-style-type: none"> • Порт по умолчанию – 10000 <ul style="list-style-type: none"> – Протокол: thrift – Описание: Сервис для подключения к Hive (Thrift/JDBC) – Доступ пользователям: Да (все клиенты, которым требуется подключение к Hive) – Параметр конфигурации: hive.server2.thrift.port • Порт по умолчанию – 10001 <ul style="list-style-type: none"> – Протокол: http – Описание: Сервис для подключения к Hive (http) – Доступ пользователям: Да (все клиенты, которым требуется подключение к Hive) – Параметр конфигурации: hive.server2.transport.mode
JobTracker	<p>Сервер – Master Nodes</p> <ul style="list-style-type: none"> • Порт по умолчанию – 8021 <ul style="list-style-type: none"> – Протокол: IPC – Описание: Для публикации заданий – Доступ пользователям: Да (все клиенты, которым требуется запуск MR, Hive, Pig. Задачи, использующие HCatalog)
Hive Web UI	<p>Сервер – Hive Server машина</p> <ul style="list-style-type: none"> • Порт по умолчанию – 9999 <ul style="list-style-type: none"> – Протокол: thrift – Описание: WebUI для Hive – Доступ пользователям: Да – Параметр конфигурации: hive.hwi.listen.port • Порт по умолчанию – 9933 <ul style="list-style-type: none"> – Протокол: http – Доступ пользователям: Да (все клиенты, которым требуется запуск MR, Hive, Pig) – Параметр конфигурации: hive.metastore.uris

5.5 Порты WebHCat

Список портов, используемых по умолчанию для различных **WebHCat**-сервисов, представлен в таблице.

Таблица 5.5.: Порты WebHCat

Сервис	Характеристика
WebHCat Server	<p>Сервер – WebHCat Server машина</p> <ul style="list-style-type: none"> • Порт по умолчанию – 50111 <ul style="list-style-type: none"> – Протокол: http – Описание: Web API для доступа к HCatalog и к другим сервисам Hadoop – Доступ пользователям: Да – Параметр конфигурации: templeton.port

5.6 Порты HBase

Список портов, используемых по умолчанию для различных **HBase**-сервисов, представлен в таблице.

Таблица 5.6.: Порты HBase

Сервис	Характеристика
HMaster	Сервер – Master Nodes (HBase Master Node и back-up HBase Master node) <ul style="list-style-type: none"> • Порт по умолчанию – 60000 <ul style="list-style-type: none"> – Доступ пользователям: Да – Параметр конфигурации: hbase.master.port
HMaster Info Web UI	Сервер – Master Nodes (HBase Master Node и back-up HBase Master node) <ul style="list-style-type: none"> • Порт по умолчанию – 60010 <ul style="list-style-type: none"> – Протокол: http – Описание: Порт для HBase Master UI – Доступ пользователям: Да – Параметр конфигурации: hbase.master.info.port
Region Server	Сервер – Все Slave Nodes <ul style="list-style-type: none"> • Порт по умолчанию – 60020 <ul style="list-style-type: none"> – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: hbase.regionserver.port • Порт по умолчанию – 60030 <ul style="list-style-type: none"> – Протокол: http – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: hbase.regionserver.info.port
HBase Thrift Server	Сервер – Все Thrift Server <ul style="list-style-type: none"> • Порт по умолчанию – 9090 <ul style="list-style-type: none"> – Описание: Порт, используемый HBase Thrift-сервером – Доступ пользователям: Да
HBase Thrift Server Web UI	Сервер – Все Thrift Server <ul style="list-style-type: none"> • Порт по умолчанию – 9090 <ul style="list-style-type: none"> – Описание: Web-интерфейс для HBase Thrift-сервера – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: hbase.thrift.info.port

5.7 Порты Zookeeper

Список портов, используемых по умолчанию для различных **Zookeeper**-сервисов, представлен в таблице.

Таблица 5.7.: Порты Zookeeper

Сервис	Характеристика
Zookeeper Server	<p data-bbox="418 266 784 296">Сервер – Все Zookeeper Nodes</p> <ul data-bbox="459 300 1450 705" style="list-style-type: none"><li data-bbox="459 300 1450 453">• Порт по умолчанию – 2181<ul data-bbox="516 327 1206 453" style="list-style-type: none"><li data-bbox="516 327 727 357">– Протокол: http<li data-bbox="516 361 1206 390">– Описание: Сервис доступа к Zookeeper Server/Quorum<li data-bbox="516 394 870 424">– Доступ пользователям: Да<li data-bbox="516 428 1159 457">– Параметр конфигурации: zookeeper.port/clientPort<li data-bbox="459 457 1450 579">• Порт по умолчанию – 2888<ul data-bbox="516 485 1450 579" style="list-style-type: none"><li data-bbox="516 485 1450 514">– Описание: Порт используется Zookeeper для взаимодействия компонентов<li data-bbox="516 518 881 548">– Доступ пользователям: Нет<li data-bbox="516 552 1154 581">– Параметр конфигурации: hbase.zookeeper.peerport<li data-bbox="459 583 1450 705">• Порт по умолчанию – 3888<ul data-bbox="516 611 1450 705" style="list-style-type: none"><li data-bbox="516 611 1450 640">– Описание: Порт используется Zookeeper для взаимодействия компонентов<li data-bbox="516 644 881 674">– Доступ пользователям: Нет<li data-bbox="516 678 1175 707">– Параметр конфигурации: hbase.zookeeper.leaderport