Arenadata™ Hadoop

Версия - v2.1.2

Руководство администратора по работе с кластером ADH

Оглавление

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/Namespaces*
- *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.

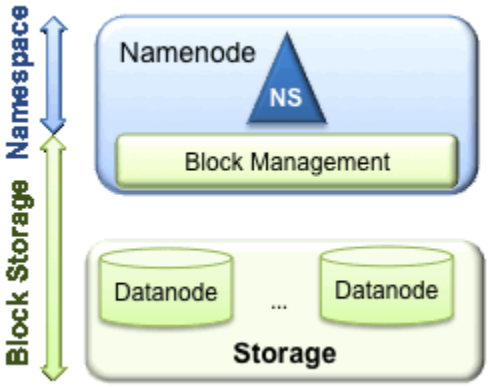


Рис.1.1.: Background

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 сбой 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;

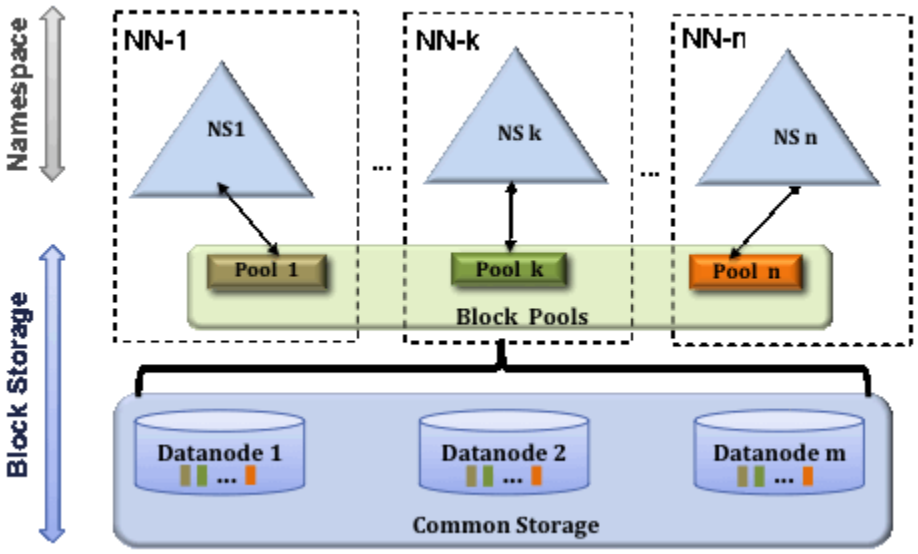


Рис.1.2.: Federation

- 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, уравновешивает 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 HDFS Erasure Coding

- *Ограничения*

1.2.1 Цель

Репликация всегда дорогостоящая – схема репликации по умолчанию (*3x*) в **HDFS** имеет *200%* накладных расходов в области хранения и других ресурсах (например, пропускная способность сети). Однако для теплых и холодных наборов данных с относительно низким уровнем операций ввода-вывода дополнительные реплики блоков редко доступны во время обычных операций, но все равно потребляют тот же объем ресурсов, что и первая реплика.

Поэтому естественным улучшением является использование Erasure Coding (EC) вместо репликации, что обеспечивает тот же уровень отказоустойчивости при гораздо меньшем объеме памяти. В типичных настройках Erasure Coding накладные расходы на хранение не превышают *50%*. Коэффициент репликации файла EC не имеет смысла, он всегда равен *1* и не может быть изменен с помощью -setrep команды.

1.2.2 Background

В системах хранения наиболее заметным использованием Erasure Coding является избыточный массив недорогих дисков (RAID). RAID реализует EC посредством чередования, которое делит логически последовательные данные (например, файл) на более мелкие единицы (например, бит, байт или блок) и сохраняет последовательные единицы на разных дисках. Далее данная единица распределения чередования называется *чередующейся ячейкой* (или просто *ячейкой*). Для каждой полосы исходных ячеек данных вычисляется и сохраняется определенное количество ячеек четности – процесс, который называется *кодированием*. Ошибка в любой чередующейся ячейке может быть исправлена путем вычисления декодирования на основе сохранившихся данных и четности ячеек.

Интеграция Erasure Coding с **HDFS** может повысить эффективность хранилища, обеспечивая при этом такую же долговечность данных, что и традиционные развертывания **HDFS** на основе репликации. Например, *3x*-реплицированный файл с *6* блоками будет занимать 6 * 3 = 18 блоков дискового пространства. Но при развертывании EC (*6* данных, *3* четности) он будет занимать только 9 блоков дискового пространства.

1.2.3 Архитектура

В контексте Erasure Coding чередование имеет несколько важных преимуществ. Во-первых, позволяется онлайн запись данных непосредственно в формате EC, избегая фазы преобразования и немедленно экономя место для хранения. Это также повышает производительность последовательного ввода-вывода за счет параллельного использования нескольких дисков, что особенно желательно в кластерах с высокопроизводительными сетями. Во-вторых, небольшие файлы естественным образом распределяются на несколько узлов данных DataNodes и устраняется необходимость объединения нескольких файлов в одну группу кодирования. Это значительно упрощает операции с файлами, такие как удаление, quota reporting и миграция между объединенными пространствами имен Namespaces.

В типичных кластерах **HDFS** небольшие файлы могут занимать более *3/4* общего объема памяти. Чтобы лучше поддерживать небольшие файлы, на этом первом этапе работы **HDFS** поддерживает EC с чередованием. В будущем **HDFS** также будет поддерживать смежную компоновку EC.

Расширения NameNode – чередующиеся файлы **HDFS**, логически состоящие из групп блоков, каждая из которых содержит определенное количество внутренних блоков. Чтобы уменьшить потребление памяти NameNode от этих дополнительных блоков, введен новый иерархический протокол именования блоков. Идентификатор группы блоков может быть выведен из идентификатора любого из ее внутренних блоков. Это позволяет управлять на уровне группы блоков, а не на уровне одного блока.

Клиентские расширения – клиентские пути чтения и записи улучшены для параллельной работы с несколькими внутренними блоками в группе блоков. На пути вывода/записи *DFSStripedOutputStream* управляет набором потоков данных, по одному для каждого DataNode, хранящего внутренний блок в текущей группе блоков. Стримеры в основном работают асинхронно. Координатор отвечает за операции над всей группой блоков, включая завершение текущей группы блоков, выделение новой группы блоков и так далее. На пути ввода/чтения

DFSStripedInputStream преобразует запрошенный логический байтовый диапазон данных в виде диапазонов во внутренние блоки, хранящиеся в DataNodes. Затем параллельно выдаются запросы на чтение. А при сбоях выдаются дополнительные запросы на чтение для декодирования.

Расширения DataNode – DataNode запускает дополнительную задачу *ErasureCodingWorker* (ECWorker) для фонового восстановления сбойных блоков Erasure Coded. Сбойные блоки EC обнаруживаются NameNode, который затем выбирает DataNode для выполнения работы по восстановлению. Задача восстановления передается как ответ на heartbeat-сообщение. Этот процесс аналогичен тому, как реплицированные блоки повторно реплицируются при сбое. Реконструкция выполняет три ключевые задачи:

- Чтение данных из исходных узлов: входные данные считываются параллельно из исходных узлов с помощью выделенного пула потоков. Основываясь на политике EC, он планирует запросы на чтение для всех исходных целей и считывает только минимальное количество входных блоков для восстановления;
- Декодирование данных и генерирование выходных данных: новые данные и блоки четности декодируются из входных данных. Все недостающие данные и блоки четности декодируются вместе;
- Передача сгенерированных блоков данных на целевые узлы: после завершения декодирования восстановленные блоки передаются на целевые DataNodes.

Политики Erasure Coding – файлам и каталогам в кластере **HDFS** разрешается использование разных политик репликации и кодирования для обеспечения разнородных рабочих нагрузок. Политика Erasure Coding инкапсулирует способ кодирования/декодирования файла. Каждая политика определяется следующей информацией:

- Схема EC – включает в себя количество блоков данных и четности в группе EC (например, *6+3*), а также codec-алгоритм (например, Reed-Solomon, XOR);
- Размер чередующейся ячейки – определяет степень детализации операций чтения и записи с чередованием, включая размеры буфера и работу кодирования.

Политики называются *codec-num data blocks-num parity blocks-cell size*. В настоящее время поддерживаются пять встроенных политик: `RS-3-2-1024k`, `RS-6-3-1024k`, `RS-10-4-1024k`, `RS-LEGACY-6-3-1024k`, `XOR-2-1- 1024k`.

Схема по умолчанию *REPLICATION* также поддерживается. Ее можно установить только для каталога, чтобы заставить каталог принимать схему репликации *3x*, а не наследовать политику erasure coding родительского каталога верхнего уровня. Политика позволяет чередовать каталог схемы репликации *3x* с каталогом erasure coding.

Политика *REPLICATION* всегда включена. Из всех политик EC по умолчанию включена *RS(6,3)*.

Подобно политикам хранения **HDFS**, политики erasure coding устанавливаются на каталог. При создании файла он наследует политику EC своего ближайшего каталога-родителя.

Политики EC уровня каталога влияют только на новые файлы, созданные в этом каталоге. Как только файл создан, его политику можно запросить, но не изменить. Если erasure coding файл переименовывается в каталог с другой политикой EC, файл принимает политику нового каталога EC. Преобразование файла в другую политику EC требует перезаписи его данных; поэтому рекомендуется копировать файл (например, через *distcp*), а не переименовывать его.

Arenadata позволяет пользователям определять свои собственные политики EC с помощью XML-файла, который должен состоять из следующих трех частей:

- layoutversion: указывает версию формата XML-файла политики EC;
- schemas: включает в себя все пользовательские схемы EC;
- policies: включает в себя все пользовательские политики EC, и каждая политика включает в себя идентификатор схемы и размер чередующейся ячейки (cellsize).

Пример XML-файла политики EC с именем `user_ec_policies.xml.template` находится в каталоге *Hadoop conf*.

Intel ISA-L расшифровывается как Intel Intelligent Storage Acceleration Library – это набор оптимизированных низкоуровневых функций с открытым исходным кодом, предназначенных для приложений хранения данных. Библиотека включает в себя быстрые блочные erasure codes типа Reed-Solomon, оптимизированные для наборов команд `Intel AVX` и `AVX2`. **HDFS** erasure coding может использовать **ISA-L** для ускорения вычислений кодирования и декодирования. **ISA-L** поддерживает большинство основных операционных систем, включая **Linux** и **Windows**. **ISA-L** не включена по умолчанию.

1.2.4 Развертывание

Конфигурация кластера и оборудования

Erasure coding предъявляет к кластеру дополнительные требования с точки зрения процессора и сети.

Работа по кодированию и декодированию требует дополнительных ресурсов ЦП как для клиентов **HDFS**, так и для узлов DataNodes.

Для Erasure coding требуется как минимум столько же DataNodes в кластере, сколько сконфигурировано блоков файловой системы EC. Для EC политики *RS (6,3)* это означает минимум *9* узлов DataNodes.

Файлы erasure coding распределяются по стойкам с целью обеспечения ее отказоустойчивости. Это означает, что при чтении и записи чередующихся файлов большинство операций выполняется вне стойки. Таким образом, пропускная способность bisection-сети очень важна.

Для отказоустойчивости стойки также важно иметь достаточное количество стоек, чтобы в среднем каждая стойка содержала количество блоков не большее, чем количество блоков четности EC. Формула для расчета получается: *(блоки данных + блоки четности) / блоки четности* с округлением в большую сторону. Для политики EC *RS (6,3)* это означает минимум *3* стойки, рассчитанные по формуле *(6 + 3) / 3 = 3*, но в идеале должно быть *9* или более для обработки запланированных и незапланированных отключений. Для кластеров с меньшим количеством стоек, чем число ячеек четности, **HDFS** не может поддерживать отказоустойчивость стойки, но при этом все равно пытается распределить чередующийся файл по нескольким узлам для сохранения отказоустойчивости на уровне узла. По этой причине рекомендуется устанавливать стойки с одинаковым количеством узлов DataNodes.

Ключи конфигурации

По умолчанию все встроенные политики erasure coding отключены, за исключением политики, определенной в `dfs.namenode.ec.system.default.policy`. Администратор кластера может включить набор политик с помощью команды `hdfs ec [-enablePolicy -policy <policyName>]` в зависимости от размера кластера и требуемых свойств отказоустойчивости. Например, для кластера с *9* стойками такая политика, как *RS-10-4-1024k*, не сохранит отказоустойчивость на уровне стойки, и *RS-6-3-1024k* или *RS-3-2-1024k* могут быть более подходящими. Если администратор заботится об отказоустойчивости только на уровне узла, политика *RS-10-4-1024k* будет по-прежнему уместной, если в кластере есть по крайней мере *14* DataNodes.

Системная политика EC по умолчанию может быть настроена через конфигурацию `dfs.namenode.ec.system.default.policy`. В этой конфигурации политика EC по умолчанию будет использоваться, когда имя политики не передается в качестве аргумента в команде `-setPolicy`.

По умолчанию `dfs.namenode.ec.system.default.policy` – *RS-6-3-1024k*.

Реализация codec для Reed-Solomon и XOR может быть настроена с помощью следующих ключей конфигурации клиента и DataNode: + `io.erasurecode.codec.rs.rawcoders` для RS codec по умолчанию; + `io.erasurecode.codec.rs-legacy.rawcoders` для предыдущих версий RS codec; + `io.erasurecode.codec.xor.rawcoders` для XOR codec.

Пользователь также может настроить самостоятельный кодек с помощью ключа конфигурации, например: `io.erasurecode.codec.self-defined-codec.rawcoders`. Значения для этого ключа являются списками имен кодеров с резервным механизмом. Эти фабрики кодеков загружаются в заданном значениями конфигурации порядке до тех пор, пока кодек не будет загружен успешно. Конфигурация кодека RS и XOR по умолчанию предпочитает нативную реализацию по сравнению с чистой *Java*. Реализация нативного кодека RS-LEGACY

отсутствует, поэтому по умолчанию используется только реализация *Java*. Все перечисленные кодеки имеют реализации на чистой *Java*. Для стандартного кодека RS и кодека XOR существует также собственная реализация, использующая библиотеку **Intel ISA-L** для повышения производительности кодека. Реализация по умолчанию для RS Legacy – это чистая *Java*, а реализации по умолчанию для RS и XOR по умолчанию – это собственные реализации, использующие библиотеку **Intel ISA-L**.

Работы по восстановлению Erasure coding background для узлов DataNodes можно настроить с помощью следующих параметров конфигурации:

- `dfs.datanode.ec.reconstruction.stripedread.timeout.millis` – тайм-аут для striped reads. Значение по умолчанию *5000 мс*;
- `dfs.datanode.ec.reconstruction.stripedread.buffer.size` – размер буфера для сервиса чтения. Значение по умолчанию *64 КБ*;
- `dfs.datanode.ec.reconstruction.threads` – количество потоков, используемых Datanode для восстановления background. Значение по умолчанию *8* потоков;
- `dfs.datanode.ec.reconstruction.xmits.weight` – относительный вес xmits, используемый задачей EC background, по сравнению с восстановлением реплицированного блока. Значение по умолчанию *0,5*. Для того, чтобы отключить вычисление весов для задач восстановления EC, необходимо установить значение *0*, тогда для задачи EC всегда определен *1* xmits. Xmits задачи восстановления erasure coding вычисляется как максимальное значение между числом потоков чтения и числом потоков записи. Например, если задаче восстановления EC нужно прочитать с *6* узлов и записать на *2* узла, она имеет xmits равный max (6, 2)*0,5 = *3*. Задача восстановления для реплицируемого файла всегда считается как *1* xmit. NameNode использует `dfs.namenode.replication.max-streams` за вычетом общего значения `xmitsInProgress` для DataNode, который объединяет xmits из реплицированного файла и файлов EC, чтобы запланировать задачи восстановления для этого DataNode.

Включение Intel ISA-L

Собственная реализация стандартного кодека RS в **HDFS** использует библиотеку **Intel ISA-L** в целях улучшения вычислений кодирования и декодирования. Чтобы включить и использовать **Intel ISA-L**, необходимо выполнить три шага:

1. Сборка библиотеки ISA-L. Подробная информация приведена на официальной странице https://github.com/01org/isa-l/.
2. Сборка Hadoop с поддержкой ISA-L.
3. Копирование содержимого каталога *isal.lib* в конечный файл *tar* с помощью `-Dbundle.isal`. Развернуть Hadoop с помощью файла *tar*. Убедиться, что ISA-L доступна на HDFS клиентах и DataNodes.

Чтобы убедиться, что **ISA-L** правильно определена в **Hadoop**, необходимо выполнить команду `hadoop checknative`.

Команды администрирования

HDFS предоставляет подкоманду `ec` для выполнения административных команд, связанных с erasure coding:

```
hdfs ec [generic options]
  [-setPolicy -path <path> [-policy <policyName>] [-replicate]]
  [-getPolicy -path <path>]
  [-unsetPolicy -path <path>]
  [-listPolicies]
  [-addPolicies -policyFile <file>]
  [-listCodecs]
  [-enablePolicy -policy <policyName>]
```

```
[-disablePolicy -policy <policyName>]
[-help [cmd ...]]
```

Подробнее о каждой команде:

- `[-setPolicy -path <path> [-policy <policyName>] [-replicate]]` – устанавливает политику erasure coding для каталога по указанному пути:
  - `path` – каталог в HDFS. Обязательный параметр. Установка политики влияет только на вновь созданные файлы и не влияет на существующие файлы;
  - `policyName` – политика erasure coding, используемая для файлов в этом каталоге. Параметр может быть пропущен, если установлена конфигурация `dfs.namenode.ec.system.default.policy`. Политика EC пути устанавливается со значением по умолчанию в конфигурации;
  - `-replicate` – применение схемы по умолчанию *REPLICATION* для каталога, принятие каталогом репликации *3x*;
  - `-replicate` и `-policy <policyName>` опциональные аргументы, и они не могут быть указаны одновременно.
- `[-getPolicy -path <path>]` – получение подробной информации о политике erasure coding файла или каталога по указанному пути;
- `[-unsetPolicy -path <path>]` – сброс политики erasure coding в каталоге, заданной вызовом `setPolicy`. Если директория наследует политику от родительского каталога верхнего уровня, то операция недопустима. Сброс политики для каталога, в котором нет явного набора политик, не возвращает ошибку;
- `[-listPolicies]` – перечисляет все (включенные, отключенные и удаленные) политики erasure coding, зарегистрированные в HDFS. Только включенные политики подходят для использования с командой `setPolicy`;
- `[-addPolicies -policyFile <file>]` – добавление списка пользовательских политик erasure coding. Пример политики приведен в файле *etc/hadoop/user_ec_policies.xml.template*. Максимальный размер ячейки определяется в свойстве `dfs.namenode.ec.policies.max.cellsize` со значением по умолчанию *4 МБ*. В настоящее время HDFS позволяет пользователю добавлять в общей сложности *64* политики, а ID добавленной политики находится в диапазоне от *64* до *127*. Если уже существует *64* политики, то добавление новой завершается ошибкой;
- `[-listCodecs]` – получение списка поддерживаемых кодеков erasure coding и кодеров в системе. Кодер – это реализация кодека. Кодек может иметь разные реализации, поэтому существуют разные кодеры. Кодеры для кодека перечисляются в обратном порядке;
- `[-removePolicy -policy <policyName>]` – удаление пользовательской политики erasure coding;
- `[-enablePolicy -policy <policyName>]` – включение политики erasure coding;
- `[-disablePolicy -policy <policyName>]` – отключение политики erasure coding.

1.2.5 Ограничения

Некоторые операции **HDFS**, такие как `hflush`, `hsync`, `concat`, `setReplication`, `truncate` и `append`, не поддерживаются файлами erasure coding из-за существенных технических проблем:

- `append()` и `truncate()` для файла erasure coding вызывают исключение `IOException`;
- `concat()` вызывает исключение `IOException`, если файлы смешаны с разными политиками erasure coding или с реплицированными файлами;
- `setReplication()` не работает для файлов erasure coding;

- `hflush()` и `hsync()` для `DFSStripedOutputStream` не используются. Таким образом, вызов `hflush()` или `hsync()` для файла erasure coding не может гарантировать сохранность данных.

Клиент может использовать StreamCapabilities API для запроса, поддерживает ли OutputStream операции `hflush()` и `hsync()`. Если клиенту требуется постоянство данных с помощью этих функций, текущим решением является создание таких файлов, как обычные файлы репликации *3x*, в каталоге без erasure coding, или использование FSDataOutputStreamBuilder#replicate() API для создания файлов репликации *3x* в каталоге erasure coding.

1.3 Квоты

Распределенная файловая система **Hadoop** (**HDFS**) позволяет администратору устанавливать квоты на количество используемых имен и объем пространства для отдельных каталогов. Квоты имен и квоты пространства работают независимо, но администрирование и реализация этих двух типов квот тесно параллельны.

1.3.1 Квоты имен

Квота имен – это жесткое ограничение на количество имен файлов и каталогов в root-дереве директории со следующими правилами:

- Создание файлов и каталогов завершается ошибкой в случае превышения квоты;
- Квоты придерживаются переименованных каталогов, но операция переименования не может быть выполнена в случае, если она приводит к нарушению квоты;
- Попытка установить квоту будет успешной, даже если каталог нарушает эту новую квоту;
- Новая созданная директория не имеет связанных квот;
- Самая большая квота – `Long.Max_Value`. Установленное значение квоты в *1* заставляет каталог оставаться пустым (получается, сам каталог учитывается квотой).

Квоты постоянны благодаря *fsimage*. При запуске, если *fsimage* нарушает квоту (возможно, при скрытном изменении *fsimage*), выдается предупреждение для каждого из таких нарушений. Установка или удаление квоты создает запись в журнале.

1.3.2 Квоты пространств

Квота пространств – это жесткое ограничение на количество байтов, используемых файлами в root-дереве директории со следующими правилами:

- Аллокация блоков не выполняется, если квота не позволяет записать полный блок;
- Каждая реплика блока подсчитывается по квоте;
- Квоты придерживаются переименованных каталогов, но операция переименования не может быть выполнена в случае, если она приводит к нарушению квоты;
- Новая созданная директория не имеет связанных квот;

- Самая большая квота – `Long.Max_Value`. Нулевая квота по-прежнему позволяет создавать файлы, но в них не могут быть добавлены блоки;
- Каталоги не используют пространство файловой системы хоста и не учитывают квоту пространства;
- Пространство файловой системы хоста, используемое для сохранения метаданных файла, не учитывается в квоте;
- Квоты начисляются с предполагаемым коэффициентом репликации для файла, при этом изменение коэффициента репликации для файла приводит к кредитным или дебетовым квотам.

Квоты постоянны благодаря *fsimage*. При запуске, если *fsimage* нарушает квоту (возможно, при скрытном изменении *fsimage*), выдается предупреждение для каждого из таких нарушений. Установка или удаление квоты создает запись в журнале.

1.3.3 Квоты типа хранилища

Квота типа хранилища – это жесткое ограничение на использование определенного типа хранилища (SSD, DISK, ARCHIVE) для файлов в root-дереве директории. Во многих аспектах она работает аналогично квоте дискового пространства, но предлагает точный контроль над использованием пространства хранения кластера. Для установки квоты в каталоге должны быть настроены политики хранения, чтобы разрешить хранение файлов в разных типах хранилища в соответствии с политикой.

Квота типа хранилища может быть объединена с квотами пространств и квотами имен для эффективного управления используемого хранилища кластера. Например:

- Для каталогов с настроенной политикой хранения администратор может установить квоты типа хранения для типов хранения с ограничением ресурсов, таких как SSD, и оставить квоты для других типов хранения и общую квоту пространств с менее ограничительными значениями или без ограничений вовсе (по умолчанию). HDFS при этом высчитывает квоты из обоих типов хранилища на основе политики хранения и общей квоты пространств;
- Для каталогов без настроенной политики хранения администратор может не настраивать квоту типа хранения. Квота может быть настроена, даже если определенный тип хранилища недоступен (или доступен, но не настроен должным образом с информацией о его типе). Однако в этом случае рекомендуется использовать общую квоту пространств, так как информация о типе хранилища либо недоступна, либо неточна для применения квоты на тип хранения;
- Квота типа хранения DISK ограничена за исключением случаев, когда DISK не является доминирующим носителем данных (например, кластер с преимущественным типом хранения ARCHIVE).

1.3.4 Административные команды

Квоты управляются набором команд, доступных только администратору:

- `hdfs dfsadmin -setQuota <N> <directory>...<directory>` – установка квоты имени в значение N для каждого каталога. Наилучшее усилие для каждого каталога с сообщениями о сбоях (файл/каталог не существует, файл/каталог превысил квоту), когда N не является положительным длинным целым числом;
- `hdfs dfsadmin -clrQuota <directory>...<directory>` – удаление любой квоты имен для каждого каталога. Это не является ошибкой, если каталог не имеет квоты имен;
- `hdfs dfsadmin -setSpaceQuota <N> <directory>...<directory>` – установка квоты пространств в значение N *байт* для каждого каталога. Это жесткое ограничение на общий размер всех файлов в дереве каталога. Квота пространств также учитывает репликацию, то есть *1 ГБ* данных с репликацией *3* потребляет *3 ГБ* квоты. Значение N также может быть указано с двоичным префиксом для удобства, например, *50g* на *50 ГБ*, *2t* на *2 ТБ* и т.д. Наилучшее усилие для каждого каталога с сообщениями о сбоях (файл/каталог не существует, файл/каталог превысил квоту), когда N не является ни нулем, ни положительным целым числом;

- `hdfs dfsadmin -clrSpaceQuota <directory>...<directory>` – удаление любой квоты пространств для каждого каталога. Это не является ошибкой, если каталог не имеет квоты пространств;
- `hdfs dfsadmin -setSpaceQuota <N> -storageType <storagetype> <directory>...<directory>` – установка квоты типа хранилища равной *N байт* для каждого каталога. Это жесткое ограничение на общее использование типа хранилища для всех файлов в дереве каталогов. Использование квоты типа хранилища отражает предполагаемое использование в соответствии с политикой хранения. Например, *1 ГБ* данных с репликацией *3* и политикой хранения ALL_SSD потребляет *3 ГБ* квоты SSD. Значение *N* также может быть указано с двоичным префиксом для удобства, например, *50g* на *50 ГБ*, *2t* на *2 ТБ* и т.д. Наилучшее усилие для каждого каталога с сообщениями о сбоях (файл/каталог не существует, файл/каталог превысил квоту), когда *N* не является ни нулем, ни положительным целым числом. Квота для конкретного типа хранилища задается, когда указана опция `-storageType`. Доступные типы хранения: RAM_DISK, DISK, SSD, ARCHIVE;
- `hdfs dfsadmin -clrSpaceQuota -storageType <storagetype> <directory>...<directory>` – удаление квоты типа хранилища, указанной для каждого каталога. Это не является ошибкой, если каталог не имеет квоты для указанного типа хранилища. Квота, относящаяся к типу хранилища, очищается, если указана опция `-storageType`. Доступные типы хранения: RAM_DISK, DISK, SSD, ARCHIVE.

1.3.5 Команда отчета

Расширение команды *count* оболочки **HDFS** сообщает о значениях квот и текущем количестве используемых имен и байт:

```
hadoop fs -count -q [-h] [-v] [-t [comma-separated list of storagetypes]]
<directory>...<directory>
```

- С помощью опции `-q` сообщается установленное для каждого каталога значение квоты имен, оставшаяся доступная квота имен, установленное значение квоты пространства и оставшаяся квота доступного пространства. Если каталог не имеет установленной квоты, сообщаются значения *none* и *inf*;
- Опция `-h` показывает размеры в удобочитаемом формате;
- Опция `-v` отображает строку заголовка;
- Опция `-t` отображает набор квот для каждого типа хранилища и оставшуюся доступную квоту для каждого каталога. Если после опции указаны конкретные типы хранения, то отображается только квота и оставшаяся квота для указанных типов. В противном случае отображается квота и оставшаяся квота всех поддерживающих ее типов хранилищ.

1.4 Снапшоты

Снапшот **HDFS** – это снимок файловой системы на момент времени, доступен только для чтения. Снапшоты могут быть сделаны в поддереве файловой системы или во всей файловой системе. Распространенные случаи использования моментальных снимков – резервное копирование данных, защита от ошибок пользователя и аварийное восстановление.

Реализация снапшотов **HDFS** очень рациональна:

- Создание снапшота происходит мгновенно: стоимость *O(1)* без учета времени поиска inode;
- Дополнительная память используется только при внесении изменений относительно снапшота: использование памяти равно *O(M)*, где *M* – количество измененных файлов/каталогов;
- Блоки в datanodes не копируются: файлы снапшота записывают список блоков и размер файла. Копирование данных не производится.

Снапшоты могут быть сделаны в любом каталоге, как только этот каталог установлен как *snapshottable*. Каталог *snapshottable* способен вместить *65 536* одновременных снимков, при этом количество директорий

snapshottable не ограничено. Администраторы могут задать для любого каталога значение *snapshottable*. Данный каталог нельзя ни удалить, ни переименовать, пока в нем находятся снапшоты.

Вложенные каталоги в *snapshottable* в настоящее время не допускаются. Другими словами, директория не может быть установлена как *snapshottable*, если хотя бы один ее каталог-родитель или каталог-потомок является *snapshottable*.

Для доступа к снапшотам каталога *snapshottable* используется компонент пути `.snapshot`. Например, */foo* – каталог *snapshottable*, тогда */foo/bar* является файлом/директорией в нем, где */foo* имеет снапшот *s0*. В результате путь */foo/.snapshot/s0/bar* ссылается на копию снимка */foo/bar*. Обычный API и CLI могут работать с путями `.snapshot`. Далее приведены некоторые примеры.

- Перечисление всех снапшотов в директории *snapshottable*:

```
hdfs dfs -ls /foo/.snapshot
```

- Перечисление файлов в снапшоте *s0*:

```
hdfs dfs -ls /foo/.snapshot/s0
```

- Копирование файла из снапшота *s0*:

```
hdfs dfs -cp -ptopax /foo/.snapshot/s0/bar /tmp
```

Important: В примере используется опция preserve для сохранения меток времени, владельца, прав доступа, списков ACL и XAttrs

1.4.1 Обновление до версии HDFS со снапшотами

Функция снапшота в **HDFS** вводит новое зарезервированное имя пути, используемое для взаимодействия со снимками: `.snapshot`. При обновлении с более старой версии **HDFS**, которая не поддерживает снапшоты, существующие пути с именем `.snapshot` необходимо сначала переименовать или удалить, чтобы избежать конфликта с зарезервированным путем в актуальной версии системы.

1.4.2 Управление снапшотами

Операции администратора

Описанные далее операции возможны только при наличии привилегий суперпользователя.

Allow – разрешение создания снапшотов. При успешном завершении операции каталог становится *snapshottable*.

Команда:

```
hdfs dfsadmin -allowSnapshot <path>
```

Аргумент:

- `path` – путь к директории *snapshottable*.

Disallow – запрещение на создание снапшотов в каталоге. Все снапшоты директории должны быть удалены перед введением запрета.

Команда:

```
hdfs dfsadmin -disallowSnapshot <path>
```

Аргумент:

- `path` – путь к директории *snapshottable*.

Операции пользователя

Далее описываются пользовательские операции над снапшотами. При этом суперпользователь **HDFS** может выполнять все операции, не удовлетворяя требованиям разрешения для отдельных операций.

Create – создание снапшота в каталоге *snapshottable*. Операция требует привилегии владельца директории.

Команда:

```
hdfs dfs -createSnapshot <path> [<snapshotName>]
```

Аргумент:

- `path` – путь к директории *snapshottable*;
- `snapshotName` – имя снапшота, необязательный аргумент. Если значение не задано, имя по умолчанию генерируется с использованием метки времени в формате 's'yyyyMMdd-HHmmss.SSS, например, `s20130412-151029.033`.

Delete – удаление снапшота из каталога *snapshottable*. Операция требует привилегии владельца директории.

Команда:

```
hdfs dfs -deleteSnapshot <path> <snapshotName>
```

Аргумент:

- `path` – путь к директории *snapshottable*;
- `snapshotName` – имя снапшота.

Rename – переименование снапшота. Операция требует привилегии владельца каталога *snapshottable*.

Команда:

```
hdfs dfs -renameSnapshot <path> <oldName> <newName>
```

Аргумент:

- `path` – путь к директории *snapshottable*;
- `oldName` – старое имя снапшота;
- `newName` – новое имя снапшота.

Get Snapshottable Directory Listing – получение списка всех каталогов *snapshottable*, где у текущего пользователя есть разрешение на создание снапшотов.

Команда:

```
hdfs lsSnapshottableDir
```

Get Snapshots Difference Report – получение различия между двумя снапшотами. Операция требует прав доступа на чтение для всех файлов/каталогов в обоих снапшотах.

Команда:

```
hdfs snapshotDiff <path> <fromSnapshot> <toSnapshot>
```

Аргумент:

- `path` – путь к директории *snapshottable*;
- `fromSnapshot` – имя начального снапшота;
- `toSnapshot` – имя конечного снапшота.

Возможные результаты проведенных операций:

- `+` – файл/каталог создан;
- `-` – файл/каталог удален;
- `M` – файл/каталог изменен;
- `R` – файл/каталог переименован.

Запись `RENAME` указывает, что файл/каталог переименован, но все еще находится в той же директории *snapshottable*. Файл/каталог считается удаленным, если он переименован за пределами директории *snapshottable*. Файл/каталог, переименованный из внешней директории *snapshottable*, считается вновь созданным.

Отчет о различиях снапшотов не гарантирует одинаковую последовательность вывода результатов операций. Например, если переименовать каталог */foo* в */foo2*, а затем добавить новые данные в файл */foo2/bar*, то отчет о различиях следующий:

```
R. /foo -> /foo2
M. /foo/bar
```

То есть об изменениях файлов/каталогов в переименованном каталоге сообщается с использованием исходного пути.

1.5 Модуль Hadoop-AWS: интеграция с Amazon Web Services

Important: В Hadoop коннекторы `s3:` и `s3n:` удалены. В качестве коннектора для данных, размещенных в S3 с Apache Hadoop, используется `s3a:`

Как перейти на клиент **S3A**:

1. Сохранить hadoop-aws JAR в classpath.
2. Добавить JAR-бандл *aws-java-sdk-bundle.jar*, который поставляется с Hadoop, в classpath.
3. Изменить ключи аутентификации:

- `fs.s3n.awsAccessKeyId` → `fs.s3a.access.key`;
- `fs.s3n.awsSecretAccessKey` → `fs.s3a.secret.key`;

Важно убедиться, что имена свойств верны. Для **S3A** это *fs.s3a.access.key* и *fs.s3a.secret.key* – нельзя просто скопировать свойства **S3N** и заменить *s3n* на *s3a*.

4. Заменить все URL, которые начинаются с `s3n://` на `s3a://`.
5. Удалить jets3t JAR, так как он больше не нужен.

Модуль **Apache Hadoop** – **hadoop-aws**, обеспечивает поддержку интеграции с **AWS** (**Amazon Web Services**).

Для включения клиента **S3A** в classpath **Apache Hadoop** по умолчанию необходимо:

1. Убедиться, что `HADOOP_OPTIONAL_TOOLS` в *hadoop-env.sh* включает hadoop-aws в свой список дополнительных модулей для добавления в classpath.
2. Для взаимодействия на стороне клиента можно объявить, что соответствующие JAR-файлы должны быть загружены в файл *~/.hadooprc*:

```
hadoop_add_to_classpath_tools hadoop-aws
```

Параметры в этом файле не распространяются на развернутые приложения, но работают для локальных клиентов, таких как команда `hadoop fs`.

Клиент **S3A** предлагает высокопроизводительный ввод-вывод по сравнению с хранилищем объектов **Amazon S3** и совместимыми реализациями:

- Непосредственно читает и пишет S3-объекты;
- Совместим со стандартными S3-клиентами;
- Совместим с файлами, созданными более старым клиентом `s3n://` и клиентом Amazon EMR `s3://`;
- Поддерживает партиционированную загрузку для объектов размером в несколько ГБ;
- Предлагает высокопроизводительный режим случайного ввода-вывода для работы со столбчатыми данными, такими как файлы Apache ORC и Apache Parquet;
- Использует Java S3 SDK от Amazon с поддержкой новейших функций S3 и схем аутентификации;
- Поддерживает аутентификацию с помощью переменных среды, свойств конфигурации Hadoop, хранилища ключей Hadoop и ролей IAM;
- Поддерживает конфигурацию для каждого сегмента;
- С помощью S3Guard добавляет высокопроизводительные и согласованные операции чтения метаданных/каталогов, что обеспечивает последовательность и скорость;
- Поддерживает S3 “Server Side Encryption” для чтения и записи: SSE-S3, SSE-KMS и SSE-C;
- Инструментирован с метриками Hadoop;
- Активно поддерживается сообществом открытого исходного кода.

Есть и другие Hadoop-коннекторы для S3, но только **S3A** активно поддерживается самим проектом **Hadoop**:

1. Оригинальный `s3://` клиент Apache Hadoop. Больше не входит в Hadoop.
2. Клиент Amazon EMR `s3://`. Из команды Amazon EMR, которая активно поддерживает его.
3. Клиент файловой системы Apache Hadoop `s3n:`. Коннектор больше недоступен.

1.5.1 Начало работы

S3A зависит от двух JAR-файлов, а также от *hadoop-common* и его зависимостей:

- hadoop-aws JAR;
- aws-java-sdk-bundle JAR.

Important: Версии hadoop-common и hadoop-aws должны быть идентичны

Для импорта библиотеки в сборку Maven, необходимо добавить JAR **hadoop-aws** и в зависимости от сборки он вытянет совместимый JAR-файл *aws-sdk*.

JAR **hadoop-aws** не декларирует никаких зависимостей, кроме AWS SDK JAR. Это упрощает исключение/настройку JAR-зависимостей **Hadoop** в имеющихся приложениях. Зависимость *hadoop-client* или *hadoop-common* должна быть объявлена.

```
<properties>
 <!-- Your exact Hadoop version here-->
  <hadoop.version>3.0.0</hadoop.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>${hadoop.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-aws</artifactId>
    <version>${hadoop.version}</version>
  </dependency>
</dependencies>
```

1.5.2 Предупреждения

Amazon S3 является примером "хранилища объектов". Чтобы добиться масштабируемости и особенно высокой доступности, **S3**, как и многие другие хранилища облачных объектов, ослабил некоторые ограничения, которые обещают классические файловые системы "POSIX".

Функция *S3Guard* пытается решить некоторые из них, но не обеспечивает этого полностью.

#1: Несогласованность модели

1. Файлы, созданные из API-интерфейсов файловой системы Hadoop, могут быть не сразу видны.
2. Операции удаления и обновления файлов могут не сразу распространяться. Старые копии файла могут существовать в течение неопределенного периода времени.
3. Операции с каталогами: `delete()` и `rename()` реализуются с помощью рекурсивных файловых операций file-by-file. Они занимают время по меньшей мере пропорциональное количеству файлов, в течение которого могут быть видны частичные обновления. Если операции прерываются, файловая система остается в промежуточном состоянии.

#2: Имитация директорий

Клиенты **S3A** имитируют каталоги:

1. Создание записи-заглушки после вызова `mkdirs`, удаление ее при добавлении файла в любом месте внизу.
2. При листинге директории выполняется поиск всех объектов, путь которых начинается с пути к каталогу, и возвращает их в виде списка.
3. При переименовании каталога берется листинг и запрашивается S3 на копирование отдельных объектов в новые объекты с назначенными именами файлов.
4. При удалении каталога берется листинг и удаляются записи в пакетном режиме.
5. При переименовании или удалении каталогов берется листинг и осуществляется работа с отдельными файлами.

Некоторые из последствий:

- В каталогах может отсутствовать время модификации. Полагающиеся на него части Hadoop могут иметь неожиданное поведение. Например, `AggregatedLogDeletionService` из YARN не удалит соответствующие лог-файлы;
- Листинг директории может быть медленным. По возможности рекомендуется использовать `listFiles(path, recursive)` для высокопроизводительных рекурсивных списков;
- Можно создать файлы под файлами, если очень постараться;
- Время переименования каталога пропорционально количеству файлов в нем (прямых и косвенных) и их размеру. Копии выполняются внутри хранилища S3, поэтому время не зависит от пропускной способности клиент-S3;
- Переименования каталога не являются атомарными: они могут частично потерпеть неудачу, и вызывающие объекты не могут безопасно полагаться на атомарные переименования как на часть алгоритма коммита;
- Удаление каталога не является атомарным и может частично завершиться ошибкой.

Последние три проблемы всплывают при использовании **S3** в качестве непосредственного места назначения работы, в отличие от **HDFS** или другой "реальной" файловой системы.

Коммиттеры **S3A** являются единственным доступным механизмом для безопасного сохранения выходных данных запросов непосредственно в хранилище объектов **S3** через файловую систему **S3A**.

#3: Разные модели авторизации у хранилищ объектов

Модель авторизации объектов **S3** сильно отличается от модели авторизации файлов **HDFS** и традиционных файловых систем. Клиент **S3A** просто сообщает информацию о заглушке от запрашивающего метаданные API:

- Владелец файла указывается как текущий пользователь;
- Файловая группа также сообщается как текущий пользователь;
- Права доступа к каталогу указываются как *777*.
- Права доступа к файлам указываются как *666*.

S3A на самом деле не применяет никаких проверок авторизации для этих заглушек. Пользователи проходят аутентификацию в S3-bucket, используя учетные данные **AWS**. Возможно, что объектные списки ACL определены для обеспечения авторизации на стороне **S3**, но это происходит полностью внутри сервиса **S3**, а не в реализации **S3A**.

#4: Ценность данных

Учетные данные **AWS** не только оплачивают сервисы, но и предоставляют доступ для чтения и записи данных. Любой пользователь с учетными данными может не только читать наборы данных, но и удалять их.

Крайне не рекомендуется распространять учетные данные целенаправленно или непреднамеренно через такие средства, как:

- Регистрация в SCM любых секретных файлов конфигурации;
- Логгирование секретных файлов конфигурации в консоли, поскольку они всегда в конечном итоге видны;
- Определение URI файловой системы с учетными данными в URL-адресе, таком как *s3a://AK0010:secret@landsat-pds/*. В итоге все оказывается в журналах и сообщениях об ошибках.

Important: Если какое-либо действие было допущено, следует немедленно изменить учетные данные

1.5.3 Аутентификация S3

За исключением случаев взаимодействия с общедоступными сегментами **S3**, клиенту **S3A** требуются учетные данные.

Клиент поддерживает несколько механизмов аутентификации и может быть настроен относительно применяемых механизмов и их порядка использования. Также можно сконфигурировать индивидуальные реализации *com.amazonaws.auth.AWSCredentialsProvider*.

Свойства аутентификации:

```
<property>
  <name>fs.s3a.access.key</name>
  <description>AWS access key ID.
   Omit for IAM role-based or provider-based authentication.</description>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <description>AWS secret key.
   Omit for IAM role-based or provider-based authentication.</description>
</property>

<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <description>
    Comma-separated class names of credential provider classes which implement
    com.amazonaws.auth.AWSCredentialsProvider.

    These are loaded and queried in sequence for a valid set of credentials.
    Each listed class must implement one of the following means of
    construction, which are attempted in order:
    1. a public constructor accepting java.net.URI and
        org.apache.hadoop.conf.Configuration,
    2. a public static method named getInstance that accepts no
       arguments and returns an instance of
       com.amazonaws.auth.AWSCredentialsProvider, or
    3. a public default constructor.

    Specifying org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider allows
    anonymous access to a publicly accessible S3 bucket without any credentials.
    Please note that allowing anonymous access to an S3 bucket compromises
    security and therefore is unsuitable for most use cases. It can be useful
    for accessing public data sets without requiring AWS credentials.

    If unspecified, then the default list of credential provider classes,
    queried in sequence, is:
    1. org.apache.hadoop.fs.s3a.BasicAWSCredentialsProvider: supports
        static configuration of AWS access key ID and secret access key.
        See also fs.s3a.access.key and fs.s3a.secret.key.
    2. com.amazonaws.auth.EnvironmentVariableCredentialsProvider: supports
        configuration of AWS access key ID and secret access key in
        environment variables named AWS_ACCESS_KEY_ID and
        AWS_SECRET_ACCESS_KEY, as documented in the AWS SDK.
    3. com.amazonaws.auth.InstanceProfileCredentialsProvider: supports use
        of instance profile credentials if running in an EC2 VM.
  </description>
</property>
```

```
<property>
  <name>fs.s3a.session.token</name>
  <description>
    Session token, when using org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider
    as one of the providers.
  </description>
</property>
```

Аутентификация через переменные среды AWS

S3A поддерживает настройку через стандартные переменные среды AWS.

Основные переменные среды предназначены для ключа доступа и связанного секрета:

```
export AWS_ACCESS_KEY_ID=my.aws.key
export AWS_SECRET_ACCESS_KEY=my.secret.key
```

Эти переменные среды могут использоваться для установки учетных данных аутентификации вместо свойств в конфигурации **Hadoop**:

```
export AWS_SESSION_TOKEN=SECRET-SESSION-TOKEN
export AWS_ACCESS_KEY_ID=SESSION-ACCESS-KEY
export AWS_SECRET_ACCESS_KEY=SESSION-SECRET-KEY
```

Если установлена переменная среды `AWS_SESSION_TOKEN`, аутентификация сессии с использованием "временных учетных данных безопасности" ("Temporary Security Credentials") включена. Идентификатор ключа и секретный ключ должны быть установлены для учетных данных этой конкретной сессии.

Important: Эти переменные среды обычно не передаются от клиента к серверу при запуске приложений YARN. Это означает, что установка переменных среды AWS при запуске приложения не позволит запущенному приложению получить доступ к ресурсам S3. Переменные среды должны каким либо образом быть установлены на хостах/процессах, где выполняется работа.

Смена провайдеров аутентификации

Стандартный способ аутентификации – с помощью ключа доступа и секретного ключа, используя свойства в файле конфигурации.

Клиент **S3A** придерживается следующей цепочки проверки подлинности:

1. Если данные для входа предоставляются в URI файловой системы, выводится предупреждение, а затем извлекаются имя пользователя и пароль для ключа и секрета AWS.
2. Файлы *fs.s3a.access.key* и *fs.s3a.secret.key* ищутся в конфигурации Hadoop XML.
3. Затем ищутся переменные среды AWS.
4. Предпринимается попытка запросить сервис Amazon EC2 Instance Metadata Service для получения учетных данных, опубликованных на виртуальных машинах EC2.

S3A можно настроить для получения провайдеров проверки подлинности клиента из классов, которые интегрируются с **AWS SDK**, путем реализации интерфейса *com.amazonaws.auth.AWSCredentialsProvider*. Это делается путем перечисления классов реализации в порядке предпочтения в параметре конфигурации `fs.s3a.aws.credentials.provider`.

Important: AWS Credential Providers отличаются от Hadoop Credential Providers. Как показано далее, Hadoop Credential Providers позволяют хранить и передавать пароли и секреты более безопасно, чем в файлах

конфигурации XML. AWS Credential Providers – это классы, которые могут использоваться Amazon AWS SDK для получения регистрации AWS из другого источника в системе, включая переменные среды, свойства JVM и файлы конфигурации

В JAR *hadoop-aws* есть три провайдера учетных данных **AWS**:

- `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` – учетные данные сессии;
- `org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider` – имя/секрет;
- `org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider` – анонимный вход.

В **Amazon SDK** также есть много провайдеров и в частности два, автоматически устанавливающихся в цепочке аутентификации:

- `com.amazonaws.auth.InstanceProfileCredentialsProvider` – учетные данные EC2 Metadata;
- `com.amazonaws.auth.EnvironmentVariableCredentialsProvider` – переменные окружения AWS.

Аутентификация EC2 IAM Metadata

Приложения, работающие в **EC2**, могут связать роль *IAM* с виртуальной машиной и запросить у EC2 Instance Metadata Service учетные данные для доступа к **S3**. В **AWS SDK** эта функциональность обеспечивается `InstanceProfileCredentialsProvider`, который применяет внутреннее принудительное использование одноэлементного инстанса для предотвращения проблемы регулирования.

Использование учетных данных сессии

Временные учетные данные безопасности (Temporary Security Credentials) можно получить в **Amazon Security Token Service**. Они состоят из ключа доступа, секретного ключа и токена сессии.

Для использования аутентификации:

1. Объявить `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` в качестве провайдера.
2. Установить ключ сессии в свойстве `fs.s3a.session.token`, а свойства доступа и секретного ключа – для свойств этой временной сессии.

Пример:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider</value>
</property>

<property>
  <name>fs.s3a.access.key</name>
  <value>SESSION-ACCESS-KEY</value>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <value>SESSION-SECRET-KEY</value>
</property>

<property>
  <name>fs.s3a.session.token</name>
  <value>SECRET-SESSION-TOKEN</value>
</property>
```

Срок действия учетных данных сессии фиксируется при их выдаче. После истечения этого срока действия приложение больше не может проходить аутентификацию в **AWS**.

Анонимный вход

Указание `org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider` разрешает анонимный доступ к общедоступным сегментам **S3** без каких-либо учетных данных:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</value>
</property>
```

Как только это будет сделано, пропадает необходимость указывать какие-либо учетные данные в конфигурации **Hadoop** или через переменные среды.

Эту опцию можно использовать для проверки того, что хранилище объектов не разрешает доступ без аутентификации: то есть, если попытка составить список сегментов осуществляется с использованием анонимного входа, то она должна завершиться неудачей (в том случае, если сегменты явно не открыты для широкого доступа).

```
hadoop fs -ls \
 -D fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider \
 s3a://landsat-pds/
```

Important: Разрешение анонимного доступа к сегменту S3 ставит под угрозу безопасность и поэтому не подходит для большинства случаев использования

Если список провайдеров учетных данных указан в `fs.s3a.aws.credentials.provider`, то *Anonymous Credential provider* должен стоять последним в перечне. В противном случае провайдеры учетных данных, перечисленные после него, игнорируются.

`SimpleAWSCredentialsProvider` – это стандартный провайдер учетных данных, который поддерживает значения секретного ключа в `fs.s3a.access.key` и токена в `fs.s3a.secret.key`. Он не поддерживает аутентификацию с учетными данными, указанными в URL-адресах.

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider</value>
</property>
```

Помимо отсутствия поддержки пользователя, сведения о пароле включаются в URL файловой системы (опасная практика, которая настоятельно не рекомендуется), этот провайдер действует точно в соответствии с базовым аутентификатором, используемым в цепочке аутентификации по умолчанию.

Это означает, что цепочка аутентификации **S3A** по умолчанию может быть определена как:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>
  org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider,
  com.amazonaws.auth.EnvironmentVariableCredentialsProvider,
  com.amazonaws.auth.InstanceProfileCredentialsProvider
  </value>
</property>
```

1.5.4 Защита учетных данных AWS

Крайне важно никогда не передавать учетные данные **AWS**. Утечка учетных данных может привести к потере всех данных. Поэтому следует:

1. Никогда не делиться секретами.
2. Никогда не передавать секреты в хранилище SCM. Помочь с этим могут git secrets.
3. Избегать использования URL-адресов s3a, в которых есть ключ и секрет. Это опасно, поскольку секреты просачиваются в логи.
4. Никогда не включать учетные данные AWS в отчеты об ошибках, прикрепленные к ним файлы и т.п.
5. При использовании переменных среды `AWS_`, список переменных среды одинаково уязвим.
6. Никогда не использовать учетные данные *root*, заместо этого использовать учетные записи пользователей IAM, причем каждый пользователь/приложение должны иметь свой собственный набор учетных данных.
7. Использовать разрешения IAM для ограничения прав доступа отдельных пользователей и приложений. Лучше всего это делать с помощью ролей, а не с помощью настройки отдельных пользователей.
8. Не передавать секреты приложениям/командам Hadoop в командной строке. Командная строка любой запущенной программы видна всем пользователям в Unix-системе (через `ps`) и сохраняется в истории команд.
9. Изучить использование предполагаемых ролей IAM для управления разрешениями: определенное соединение S3A может быть выполнено с другой предполагаемой ролью и разрешениями от основной учетной записи пользователя.
10. Рассмотреть рабочий процесс, в котором пользователям и приложениям выдаются кратковременные учетные данные сессии, с настройкой S3A для их использования через `TemporaryAWSCredentialsProvider`.
11. Иметь безопасный процесс для отмены и повторной выдачи учетных данных для пользователей и приложений. Регулярно его проверять, используя обновленные данные.

При запуске в **EC2** провайдер учетных данных инстанса IAM автоматически получает учетные данные, необходимые для доступа к сервисам **AWS** в той роли, в которой развернута виртуальная машина **EC2**. Этот провайдер включен в **S3A** по умолчанию.

Самый безопасный способ сохранить ключи входа в **AWS** в секрете от **Hadoop** – это использовать учетные данные **Hadoop**.

1.5.5 Хранение секретов с помощью Hadoop Credential Providers

Hadoop Credential Provider Framework позволяет “провайдерам учетных данных” держать секреты вне файлов конфигурации **Hadoop**, хранить их в зашифрованных файлах локально или в файловой системе **Hadoop**, включая их в запросы.

Параметры конфигурации **S3A** с конфиденциальными данными (*fs.s3a.secret.key*, *fs.s3a.access.key*, *fs.s3a.session.token* и *fs.s3a.server-side-encryption.key*) могут сохранять свои данные в двоичном файле, при этом значения считываются, когда URL-адрес файловой системы **S3A** используется для доступа к данным. Ссылка на этого поставщика учетных данных объявляется в конфигурации *hadoop*.

Следующие параметры конфигурации могут быть сохранены в хранилищах **Hadoop Credential Provider**:

```
fs.s3a.access.key
fs.s3a.secret.key
fs.s3a.session.token
fs.s3a.server-side-encryption.key
fs.s3a.server-side-encryption-algorithm
```

Первые три предназначены для аутентификации, а последние два – для шифрования. Из последних только ключ шифрования можно считать "чувствительным". Однако возможность включить алгоритм в учетные данные позволяет файлу *JCEKS* содержать все параметры, необходимые для шифрования новых данных для записи в **S3**.

Шаг 1. Создание файла учетных данных

Файл учетных данных может быть создан в любой файловой системе **Hadoop**. При создании файла в **HDFS** или **Unix** разрешения устанавливаются автоматически на сохранение конфиденциальности файла для читателя, хотя, несмотря на то, что права доступа к каталогу не затрагиваются, необходимо проверить, что содержащий файл каталог доступен для чтения только текущему пользователю.

```
hadoop credential create fs.s3a.access.key -value 123 \
    -provider jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks

hadoop credential create fs.s3a.secret.key -value 456 \
    -provider jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks
```

Можно увидеть, какие записи хранятся внутри файла учетных данных:

```
hadoop credential list -provider jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks

Listing aliases for CredentialProvider: jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks
fs.s3a.secret.key
fs.s3a.access.key
```

На этом этапе учетные данные готовы к использованию.

Шаг 2. Настройка свойства пути

URL-адрес провайдера должен быть задан в свойстве конфигурации `hadoop.security.credential.provider.path` либо в командной строке, либо в файлах конфигурации XML.

```
<property>
  <name>hadoop.security.credential.provider.path</name>
  <value>jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks</value>
  <description>Path to interrogate for protected credentials.</description>
</property>
```

Поскольку это свойство предоставляет только путь к файлу секретов, сам параметр конфигурации не является конфиденциальным элементом.

Свойство `hadoop.security.credential.provider.path` является глобальным для всех файловых систем и секретов. Есть еще одно свойство, `fs.s3a.security.credential.provider.path`, в котором перечислены только провайдеры учетных данных для файловых систем **S3A**. Эти два свойства объединяются в одно со списком провайдеров в *fs.s3a*. Свойство имеет приоритет над списком *hadoop.security *(т.е. они добавляются в общий список).

```
<property>
  <name>fs.s3a.security.credential.provider.path</name>
  <value />
  <description>
    Optional comma separated list of credential providers, a list
    which is prepended to that set in hadoop.security.credential.provider.path
  </description>
</property>
```

Это было сделано для поддержки привязки различных провайдеров учетных данных для каждого сегмента без добавления альтернативных секретов в список учетных данных. Однако некоторые приложения (например, **Hive**) не позволяют пользователям динамически обновлять список провайдеров. Поскольку теперь поддерживаются секреты для каждого сегмента, лучше включать ключи для каждого сегмента в файлы *JCEKS* и другие источники учетных данных.

Использование секретов от провайдеров учетных данных

Как только провайдер настроен в конфигурации **Hadoop**, команды *hadoop* работают точно так же, как если бы секреты были в файле XML.

```
hadoop distcp \
    hdfs://nn1.example.com:9001/user/backup/007020615 s3a://glacier1/

hadoop fs -ls s3a://glacier1/
```

Путь к провайдеру также можно указать в командной строке:

```
hadoop distcp \
    -D hadoop.security.credential.provider.path=jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks \
    hdfs://nn1.example.com:9001/user/backup/007020615 s3a://glacier1/

hadoop fs \
  -D fs.s3a.security.credential.provider.path=jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks \
  -ls s3a://glacier1/
```

Поскольку путь провайдера сам по себе не является конфиденциальным секретом, нет риска декларировать его в командной строке.

1.5.6 Общая конфигурация клиента S3A

Все параметры клиента **S3A** настроены с префиксом `fs.s3a`.

Клиент поддерживает конфигурацию для каждого сегмента, чтобы разные сегменты могли переопределять общие параметры. Это обычно используется для изменения конечной точки, механизмов шифрования и аутентификации сегментов, опций *S3Guard* и различных других мелких опций.

```
<property>
  <name>fs.s3a.connection.maximum</name>
  <value>15</value>
  <description>Controls the maximum number of simultaneous connections to S3.</description>
</property>

<property>
  <name>fs.s3a.connection.ssl.enabled</name>
  <value>true</value>
  <description>Enables or disables SSL connections to S3.</description>
</property>

<property>
  <name>fs.s3a.endpoint</name>
  <description>AWS S3 endpoint to connect to. An up-to-date list is
    provided in the AWS Documentation: regions and endpoints. Without this
    property, the standard region (s3.amazonaws.com) is assumed.
  </description>
</property>

<property>
```

```
  <name>fs.s3a.path.style.access</name>
  <value>false</value>
  <description>Enable S3 path style access ie disabling the default virtual hosting behaviour.
    Useful for S3A-compliant storage providers as it removes the need to set up DNS for virtual hosting.
  </description>
</property>

<property>
  <name>fs.s3a.proxy.host</name>
  <description>Hostname of the (optional) proxy server for S3 connections.</description>
</property>

<property>
  <name>fs.s3a.proxy.port</name>
  <description>Proxy server port. If this property is not set
    but fs.s3a.proxy.host is, port 80 or 443 is assumed (consistent with
    the value of fs.s3a.connection.ssl.enabled).</description>
</property>

<property>
  <name>fs.s3a.proxy.username</name>
  <description>Username for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.password</name>
  <description>Password for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.domain</name>
  <description>Domain for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.workstation</name>
  <description>Workstation for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.attempts.maximum</name>
  <value>20</value>
  <description>How many times we should retry commands on transient errors.</description>
</property>

<property>
  <name>fs.s3a.connection.establish.timeout</name>
  <value>5000</value>
  <description>Socket connection setup timeout in milliseconds.</description>
</property>

<property>
  <name>fs.s3a.connection.timeout</name>
  <value>200000</value>
  <description>Socket connection timeout in milliseconds.</description>
</property>

<property>
```

```
  <name>fs.s3a.paging.maximum</name>
  <value>5000</value>
  <description>How many keys to request from S3 when doing
     directory listings at a time.</description>
</property>

<property>
  <name>fs.s3a.threads.max</name>
  <value>10</value>
  <description> Maximum number of concurrent active (part)uploads,
  which each use a thread from the threadpool.</description>
</property>

<property>
  <name>fs.s3a.socket.send.buffer</name>
  <value>8192</value>
  <description>Socket send buffer hint to amazon connector. Represented in bytes.</description>
</property>

<property>
  <name>fs.s3a.socket.recv.buffer</name>
  <value>8192</value>
  <description>Socket receive buffer hint to amazon connector. Represented in bytes.</description>
</property>

<property>
  <name>fs.s3a.threads.keepalivetime</name>
  <value>60</value>
  <description>Number of seconds a thread can be idle before being
    terminated.</description>
</property>

<property>
  <name>fs.s3a.max.total.tasks</name>
  <value>5</value>
  <description>Number of (part)uploads allowed to the queue before
  blocking additional uploads.</description>
</property>

<property>
  <name>fs.s3a.multipart.size</name>
  <value>100M</value>
  <description>How big (in bytes) to split upload or copy operations up into.
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </description>
</property>

<property>
  <name>fs.s3a.multipart.threshold</name>
  <value>2147483647</value>
  <description>How big (in bytes) to split upload or copy operations up into.
    This also controls the partition size in renamed files, as rename() involves
    copying the source file(s).
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </description>
</property>

<property>
```

```
  <name>fs.s3a.multiobjectdelete.enable</name>
  <value>true</value>
  <description>When enabled, multiple single-object delete requests are replaced by
    a single 'delete multiple objects'-request, reducing the number of requests.
    Beware: legacy S3-compatible object stores might not support this request.
  </description>
</property>

<property>
  <name>fs.s3a.acl.default</name>
  <description>Set a canned ACL for newly created and copied objects. Value may be Private,
    PublicRead, PublicReadWrite, AuthenticatedRead, LogDeliveryWrite, BucketOwnerRead,
    or BucketOwnerFullControl.</description>
</property>

<property>
  <name>fs.s3a.multipart.purge</name>
  <value>false</value>
  <description>True if you want to purge existing multipart uploads that may not have been
     completed/aborted correctly</description>
</property>

<property>
  <name>fs.s3a.multipart.purge.age</name>
  <value>86400</value>
  <description>Minimum age in seconds of multipart uploads to purge</description>
</property>

<property>
  <name>fs.s3a.signing-algorithm</name>
  <description>Override the default signing algorithm so legacy
    implementations can still be used</description>
</property>

<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <description>Specify a server-side encryption algorithm for s3a: file system.
    Unset by default. It supports the following values: 'AES256' (for SSE-S3), 'SSE-KMS'
     and 'SSE-C'
  </description>
</property>

<property>
    <name>fs.s3a.server-side-encryption.key</name>
    <description>Specific encryption key to use if fs.s3a.server-side-encryption-algorithm
    has been set to 'SSE-KMS' or 'SSE-C'. In the case of SSE-C, the value of this property
    should be the Base64 encoded key. If you are using SSE-KMS and leave this property empty,
    you'll be using your default's S3 KMS key, otherwise you should set this property to
    the specific KMS key id.</description>
</property>

<property>
  <name>fs.s3a.buffer.dir</name>
  <value>${hadoop.tmp.dir}/s3a</value>
  <description>Comma separated list of directories that will be used to buffer file
    uploads to.</description>
</property>
```

```
<property>
  <name>fs.s3a.block.size</name>
  <value>32M</value>
  <description>Block size to use when reading files using s3a: file system.
  </description>
</property>

<property>
  <name>fs.s3a.user.agent.prefix</name>
  <value></value>
  <description>
    Sets a custom value that will be prepended to the User-Agent header sent in
    HTTP requests to the S3 back-end by S3AFileSystem.  The User-Agent header
    always includes the Hadoop version number followed by a string generated by
    the AWS SDK.  An example is "User-Agent: Hadoop 2.8.0, aws-sdk-java/1.10.6".
    If this optional property is set, then its value is prepended to create a
    customized User-Agent.  For example, if this configuration property was set
    to "MyApp", then an example of the resulting User-Agent would be
    "User-Agent: MyApp, Hadoop 2.8.0, aws-sdk-java/1.10.6".
  </description>
</property>

<property>
  <name>fs.s3a.impl</name>
  <value>org.apache.hadoop.fs.s3a.S3AFileSystem</value>
  <description>The implementation class of the S3A Filesystem</description>
</property>

<property>
  <name>fs.AbstractFileSystem.s3a.impl</name>
  <value>org.apache.hadoop.fs.s3a.S3A</value>
  <description>The implementation class of the S3A AbstractFileSystem.</description>
</property>

<property>
  <name>fs.s3a.readahead.range</name>
  <value>64K</value>
  <description>Bytes to read ahead during a seek() before closing and
  re-opening the S3 HTTP connection. This option will be overridden if
  any call to setReadahead() is made to an open stream.</description>
</property>

<property>
  <name>fs.s3a.list.version</name>
  <value>2</value>
  <description>Select which version of the S3 SDK's List Objects API to use.
  Currently support 2 (default) and 1 (older API).</description>
</property>
```

1.5.7 Повтор и восстановление

Клиент **S3A** прилагает все усилия для восстановления после сбоев сети.

S3A разделяет исключения, возвращаемые **AWS SDK**, на различные категории и выбирает другую политику повторных попыток в зависимости от их типа и того, является ли сбойная операция идемпотентной.

Неустранимые проблемы: Fail Fast

Следующие проблемы считаются неустранимыми, **S3A** не пытается восстановить их:

- Нет объекта/сегмента: `FileNotFoundException`;
- Нет прав доступа: `AccessDeniedException`;
- Неисправные сетевые ошибки (`UnknownHostException`, `NoRouteToHostException`, `AWSRedirectException`);
- Прерывания: `InterruptedIOException`, `InterruptedException`;
- Отклоненные HTTP-запросы: `InvalidRequestException`.

Возможные проблемы восстановления: повторная попытка

- Время соединения вышло: `ConnectTimeoutException`. Время ожидания перед настройкой соединения с конечной точкой S3 (или прокси-сервером);
- Код состояния ответа HTTP 400, “Bad Request”.

Код состояния 400, “Bad Request” обычно означает, что запрос не подлежит восстановлению. Но иногда восстановление возможно, поэтому проблема относится к данной категории, а не к неисправимым сбоям.

Сбои повторяются с фиксированным интервалом ожидания, установленным в `fs.s3a.retry.interval`, до предела, установленного в `fs.s3a.retry.limit`.

Повтор идемпотентных операций

Некоторые сетевые сбои считаются повторяемыми, если они происходят при идемпотентных операциях; при этом нет никакого способа узнать, происходят они до или после того, как запрос обрабатывается **S3**.

- `SocketTimeoutException`: общий сбой сети;
- `EOFException`: соединение разорвано во время чтения данных;
- “No response from Server” (443, 444) сервер не отвечает;
- Исключение другого AWS-клиента, сервиса или S3.

Эти сбои повторяются с фиксированным интервалом ожидания, установленным в `fs.s3a.retry.interval`, вплоть до предела, заданного в `fs.s3a.retry.limit`.

DELETE считается идемпотентным, поэтому: `FileSystem.delete()` и `FileSystem.rename()` повторяют свои запросы на удаление при любом из перечисленных сбоев.

Вопрос о том, должно ли удаление быть идемпотентным, был источником исторических противоречий в **Hadoop**:

1. При отсутствии каких-либо других изменений в хранилище объектов повторный запрос *DELETE* в конечном итоге приводит к удалению именованного объекта; и при повторной обработке он не будет работать. Как, впрочем, и `Filesystem.delete()`.
2. Если другой клиент создает файл под этим путем, он будет удален.
3. Любая файловая система, поддерживающая атомарную операцию `FileSystem.create(path, overwrite=false)` для отклонения создания файла при наличии существующего пути, *не должна* считать удаление идемпотентным, поскольку операция `create(path, false)` может стать успешной только в том случае, если первый вызов `delete()` уже успешно завершен.
4. Второй повторный вызов `delete()` может удалить новые данные.

Поскольку **S3** в конечном итоге непротиворечив *и* не поддерживает атомарную операцию создания без перезаписи (create-no-overwrite), выбор становится еще более неоднозначен.

В настоящее время **S3A** считает удаление идемпотентом, поскольку так удобней для многих рабочих процессов, включая протоколы коммитов. Поэтому важно иметь в виду, что в случае временных сбоев может быть удалено больше, чем ожидается. Для тех, кто считает это неправильным решением, есть обходной путь: необходимо перестроить модуль *hadoop-aws* с константой `S3AFileSystem.DELETE_CONSIDERED_IDEMPOTENT`, установленной со значением *false*.

Дросселированные запросы от S3 и Dynamo DB

Когда **S3A** или **Dynamo DB** возвращают ответ, указывающий, что запросы от вызывающего объекта дросселируются, происходит экспоненциальный откат с начальным интервалом и максимальным количеством запросов.

```
<property>
  <name>fs.s3a.retry.throttle.limit</name>
  <value>${fs.s3a.attempts.maximum}</value>
  <description>
    Number of times to retry any throttled request.
  </description>
</property>

<property>
  <name>fs.s3a.retry.throttle.interval</name>
  <value>1000ms</value>
  <description>
    Interval between retry attempts on throttled requests.
  </description>
</property>
```

При этом:

1. Внутри AWS SDK также происходит дросселирование, которое управляется значением `fs.s3a.attempts.maximum`.
2. События дросселирования отслеживаются в метриках и статистике файловой системы S3A.
3. Amazon KMS может дросселировать клиентов на основе общего уровня использования KMS для всех учетных записей пользователей и приложений.

Дросселирование запросов **S3** является распространенным явлением; это вызвано слишком большим количеством клиентов, пытающихся получить доступ к одному и тому же сегменту хранилища **S3**. Обычно это происходит при большом количестве операций чтения, которые наиболее распространены в приложениях **Hadoop**. Проблема усугубляется стратегией партиционирования **Hive**, используемой при хранении данных, такой как разделение по годам, а затем по месяцам. Это приводит к путям с небольшим изменением или вообще без изменений в начале, в результате чего все данные хранятся в одном и том же сегменте (сегментах).

Далее приведен перечень нескольких дорогостоящих операций. Чем больше таких событий происходит на стороне сегмента **S3**, тем большую нагрузку он испытывает:

- Большое количество клиентов пытается перечислить каталоги или вызывают `getFileStatus` по путям (запросы *LIST* и *HEAD* соответственно);
- GET-запросы, выдающиеся при чтении данных;
- Случайный ввод-вывод, используемый при чтении столбчатых данных (*ORC*, *Parquet*), что приводит к гораздо большему числу запросов *GET*, чем простое чтение по одному файлу;
- Число активных записей в часть сегмента S3.

Особый случай – когда в часть сегмента **S3** записано достаточное количество данных, и **S3** решает разделить данные на несколько сегментов: считается, что это одна за другой операция копирования, которая

может занять некоторое время. В этот момент времени обращающиеся к данным по этим путям клиенты **S3** дросселируются более, чем обычно.

Стратегия миграции:

1. Использовать отдельные сегменты для промежуточных данных/разных приложений/ролей.
2. Использовать существенно разные пути для разных наборов данных в одном сегменте.
3. Увеличить значение `fs.s3a.retry.throttle.interval` для обеспечения более длительных задержек между попытками.
4. Уменьшить параллельность запросов. Чем больше задач пытается получить доступ к данным параллельно, тем больше нагрузка.
5. Уменьшить `fs.s3a.threads.max`, чтобы сократить количество параллельных операций, выполняемых клиентами. Также можно: поднять значение `fs.s3a.readahead.range`, чтобы увеличить минимальный объем данных, запрашиваемых в каждом запросе *GET*, а также количество пропускаемых данных в существующем потоке перед его прерыванием и созданием нового потока.
6. Если таблицы *DynamoDB*, используемые *S3Guard*, дросселируются, увеличить емкость с помощью `hadoop s3guard set-capacity`.
7. KMS: “проконсультироваться с AWS по поводу увеличения емкости”.

1.5.8 Конфигурирование различных сегментов S3 с помощью посегментной настройки

Доступ к различным сегментам **S3** возможен с различными конфигурациями клиента **S3A**. Это позволяет использовать разные конечные точки, стратегии чтения и записи данных, а также данные для входа в систему.

1. Все параметры `fs.s3a`, кроме небольшого набора неизменяемых значений (в настоящее время `fs.s3a.impl`), могут быть установлены для каждого сегмента.
2. Опция для конкретного сегмента задается путем замены `fs.s3a.` префиксом опции `fs.s3a.bucket.BUCKETNAME.`, где *BUCKETNAME* – имя сегмента.
3. При подключении к сегменту все явно заданные параметры переопределяют базовые значения `fs.s3a.`.

Например, настройка может иметь базовую конфигурацию для использования информации о роли IAM, доступной при развертывании в **Amazon EC2**.

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>com.amazonaws.auth.InstanceProfileCredentialsProvider</value>
</property>
```

Это становится механизмом аутентификации по умолчанию для сегментов **S3A**.

Сегмент *s3a://nightly/* используется для ночных данных, в результате чего может быть дан ключ сессии:

```
<property>
  <name>fs.s3a.bucket.nightly.access.key</name>
  <value>AKAACCESSKEY-2</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.secret.key</name>
  <value>SESSIONSECRETKEY</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.session.token</name>
```

```
  <value>Short-lived-session-token</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider</value>
</property>
```

Наконец, общедоступный сегмент *s3a://landsat-pds/* может быть доступен анонимно:

```
<property>
  <name>fs.s3a.bucket.landsat-pds.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</value>
</property>
```

Настройка секретов S3A, хранящихся в файлах учетных данных

Секреты в файлах *JCEKS* или предоставленные другими провайдерами учетных данных **Hadoop** могут быть настроены для каждого отдельного сегмента. Клиент **S3A** смотрит секреты на каждом сегменте.

Например, файл *JCEKS* с шестью ключами:

```
fs.s3a.access.key
fs.s3a.secret.key
fs.s3a.server-side-encryption-algorithm
fs.s3a.bucket.nightly.access.key
fs.s3a.bucket.nightly.secret.key
fs.s3a.bucket.nightly.session.token
fs.s3a.bucket.nightly.server-side-encryption.key
fs.s3a.bucket.nightly.server-side-encryption-algorithm
```

При доступе к сегменту *s3a://nightly/* используются параметры конфигурации для каждого сегмента, в приведенном примере – ключи доступа и токен, включая алгоритм шифрования и ключ.

Использование Per-Bucket Configuration для доступа к данным

Сегменты **S3** находятся в разных "регионах" (по умолчанию *US-East*). Клиент **S3A** обращается к этим регионам по умолчанию, отправляя HTTP-запросы на сервер *s3.amazonaws.com*.

S3A может работать с сегментами из любого региона. Каждый регион имеет свою собственную конечную точку **S3**, описание которых приведено в документации Amazon.

1. Приложения, работающие в инфраструктуре EC2, не платят за ввод-вывод в/из локальных сегментов S3. Им выставляется счет за доступ к удаленным сегментам. Рекомендуется использовать локальные сегменты и локальные копии данных везде, где это возможно.
2. Конечная точка S3 по умолчанию может поддерживать ввод-вывод данных с любым сегментом, при условии использования протокола подписи запроса V1.
3. Когда применяется протокол подписи V4, AWS требует использования явной конечной точки региона, поэтому S3A должен быть сконфигурирован для использования конкретной конечной точки. Это делается в параметре конфигурации *fs.s3a.endpoint*.
4. Все конечные точки, кроме точки по умолчанию, поддерживают взаимодействие только с локальными для данного экземпляра S3 сегментами.

Хотя, как правило, проще использовать конечную точку по умолчанию, работая с регионами *V4-signing-only* (*Frankfurt*, *Seoul*) требуется идентификация конечной точки. Лучшая производительность предполагается от прямых подключений – *traceroute* может дать некоторое представление об этом.

В случае если используется неверная конечная точка, запрос может завершиться сбоем. И тогда об этом сообщается как ошибка *301/redirect* или *400 Bad Request*: следует принять их как подсказки для проверки настройки конечной точки сегмента.

Далее приведен список свойств, определяющих все регионы **AWS S3**, по состоянию на июнь 2017 года:

```
<!--
 This is the default endpoint, which can be used to interact
 with any v2 region.
 -->
<property>
  <name>central.endpoint</name>
  <value>s3.amazonaws.com</value>
</property>

<property>
  <name>canada.endpoint</name>
  <value>s3.ca-central-1.amazonaws.com</value>
</property>

<property>
  <name>frankfurt.endpoint</name>
  <value>s3.eu-central-1.amazonaws.com</value>
</property>

<property>
  <name>ireland.endpoint</name>
  <value>s3-eu-west-1.amazonaws.com</value>
</property>

<property>
  <name>london.endpoint</name>
  <value>s3.eu-west-2.amazonaws.com</value>
</property>

<property>
  <name>mumbai.endpoint</name>
  <value>s3.ap-south-1.amazonaws.com</value>
</property>

<property>
  <name>ohio.endpoint</name>
  <value>s3.us-east-2.amazonaws.com</value>
</property>

<property>
  <name>oregon.endpoint</name>
  <value>s3-us-west-2.amazonaws.com</value>
</property>

<property>
  <name>sao-paolo.endpoint</name>
  <value>s3-sa-east-1.amazonaws.com</value>
</property>

<property>
  <name>seoul.endpoint</name>
  <value>s3.ap-northeast-2.amazonaws.com</value>
</property>
```

```
<property>
  <name>singapore.endpoint</name>
  <value>s3-ap-southeast-1.amazonaws.com</value>
</property>

<property>
  <name>sydney.endpoint</name>
  <value>s3-ap-southeast-2.amazonaws.com</value>
</property>

<property>
  <name>tokyo.endpoint</name>
  <value>s3-ap-northeast-1.amazonaws.com</value>
</property>

<property>
  <name>virginia.endpoint</name>
  <value>${central.endpoint}</value>
</property>
```

Этот список может использоваться для указания конечной точки отдельных сегментов, например, для сегментов в *центральной* и в *EU/Ireland* конечных точках.

```
<property>
  <name>fs.s3a.bucket.landsat-pds.endpoint</name>
  <value>${central.endpoint}</value>
  <description>The endpoint for s3a://landsat-pds URLs</description>
</property>

<property>
  <name>fs.s3a.bucket.eu-dataset.endpoint</name>
  <value>${ireland.endpoint}</value>
  <description>The endpoint for s3a://eu-dataset URLs</description>
</property>
```

Зачем явно объявлять сегмент, привязанный к *центральной* конечной точке? Это гарантирует, что если конечная точка по умолчанию будет изменена на новый регион, хранилище данных *US-east* все равно будет доступно.

1.5.9 Как S3A записывает данные в S3

Оригинальный клиент **S3A** реализовал запись в файл путем буферизации всех данных на диск, как они были записаны в *OutputStream*. Загрузка при этом начинается только при вызове метода потока `close()`. В результате это делало вывод медленным, особенно при больших загрузках, и даже могло заполнять дисковое пространство небольших (виртуальных) дисков.

В **Hadoop 2.7** была добавлена альтернатива *S3AFastOutputStream*, которую в последствии **Hadoop 2.8** расширил. Теперь он считается стабильным и полностью заменил оригинальный *S3AOutputStream*, который больше не поставляется в **Hadoop**.

“Быстрый” выходной поток:

1. Загружает большие файлы в виде блоков с установленным размером в `fs.s3a.multipart.size`. То есть имеется предел, с которого начинается многочастная загрузка с идентичным размером каждой загрузки.
2. Буферизует блоки на диске (по умолчанию) или в оперативной памяти или вне нее.
3. Загружает блоки параллельно в фоновых потоках.

4. Начинает загрузку блоков, как только буферизованные данные превышают размер партиции.
5. При буферизации данных на диск используется каталог/каталоги, перечисленные в `fs.s3a.buffer.dir`. Размер данных, которые можно буферизовать, ограничен доступным дисковым пространством.
6. Генерирует выходную статистику в виде метрик в файловой системе, включая статистику активных и ожидающих загрузку блоков.
7. Время закрытия `close()` задается количеством оставшихся данных для загрузки, а не общим размером файла.

Поскольку загрузка начинается во время записи данных, она дает значительные преимущества при генерации очень больших объемов данных. Механизмы буферизации в памяти могут также обеспечивать ускорение при работе рядом с конечными точками **S3**, поскольку диски не используются для промежуточного хранения данных.

```
<property>
  <name>fs.s3a.fast.upload.buffer</name>
  <value>disk</value>
  <description>
    The buffering mechanism to use.
    Values: disk, array, bytebuffer.

    "disk" will use the directories listed in fs.s3a.buffer.dir as
    the location(s) to save data prior to being uploaded.

    "array" uses arrays in the JVM heap

    "bytebuffer" uses off-heap memory within the JVM.

    Both "array" and "bytebuffer" will consume memory in a single stream up to the number
    of blocks set by:

        fs.s3a.multipart.size * fs.s3a.fast.upload.active.blocks.

    If using either of these mechanisms, keep this value low

    The total number of threads performing work across all threads is set by
    fs.s3a.threads.max, with fs.s3a.max.total.tasks values setting the number of queued
    work items.
  </description>
</property>

<property>
  <name>fs.s3a.multipart.size</name>
  <value>100M</value>
  <description>How big (in bytes) to split upload or copy operations up into.
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </description>
</property>

<property>
  <name>fs.s3a.fast.upload.active.blocks</name>
  <value>8</value>
  <description>
    Maximum Number of blocks a single output stream can have
    active (uploading, or queued to the central FileSystem
    instance's pool of queued operations.

    This stops a single stream overloading the shared thread pool.
```

```
  </description>
</property>
```

Примечания:

- Если объем данных, записываемых в поток, меньше установленного в `fs.s3a.multipart.size`, загрузка выполняется в операции `OutputStream.close()` – как и в оригинальном выходном потоке;
- Монитор метрик Hadoop включает в себя длину очереди в реальном времени и количество операций загрузки, что позволяет определить, когда имеется отставание в работе / несоответствие между скоростью генерации данных и пропускной способностью сети. Статистика по каждому потоку также может быть записана с помощью вызова `toString()` в текущем потоке.
- Записываемые файлы остаются невидимыми до тех пор, пока запись не завершится в вызове `close()`, блокирующемся до завершения загрузки.

Буферизация загружаемых данных на диск

Когда `fs.s3a.fast.upload.buffer` установлен на `disk`, все данные перед загрузкой буферизуются на локальные жесткие диски (*disk buffer*). Это сводит к минимуму объем потребляемой памяти и, таким образом, исключает размер кучи как ограничивающий фактор при загрузке в очереди – точно так же, как и оригинальная буферизация "direct to disk".

```
<property>
  <name>fs.s3a.fast.upload.buffer</name>
  <value>disk</value>
</property>

<property>
  <name>fs.s3a.buffer.dir</name>
  <value>${hadoop.tmp.dir}/s3a</value>
  <description>Comma separated list of directories that will be used to buffer file
    uploads to.</description>
</property>
```

Это буферный механизм по умолчанию. Объем данных, которые могут быть буферизованы, ограничен объемом доступного дискового пространства.

Буферизация загружаемых данных в ByteBuffers

Когда для `fs.s3a.fast.upload.buffer` установлено значение `bytebuffer`, все данные перед загрузкой буферизируются в "Direct" *ByteBuffers* . Этот способ может оказаться быстрее, чем буферизация на диск, и к тому же, если места на диске мало (например, крошечные виртуальные машины EC2), его может не хватить.

ByteBuffers создаются в памяти JVM, а не в самой Java Heap. Объем данных, которые могут быть буферизованы, ограничивается средой выполнения **Java**, операционной системой и объемом памяти, запрашиваемым для каждого контейнера, для приложений **YARN**.

Чем медленнее пропускная способность загрузки в **S3**, тем больше риск исчерпания памяти, поэтому требуется особое внимание при настройке параметров загрузки (*Настройка загрузки потока*).

```
<property>
  <name>fs.s3a.fast.upload.buffer</name>
  <value>bytebuffer</value>
</property>
```

Буферизация загружаемых данных в массивы

Когда для `fs.s3a.fast.upload.buffer` задано значение `array`, все данные перед загрузкой буферизируются в байтовые массивы (*byte arrays*) в куче JVM. Этот способ может оказаться быстрее, чем буферизация на диск.

Объем данных, которые могут быть буферизованы, ограничивается доступным размером JVM heap. Чем медленнее пропускная способность записи в **S3**, тем больше риск переполнения кучи, но его можно сократить, настроив параметры загрузки (*Настройка загрузки потока*).

```
<property>
  <name>fs.s3a.fast.upload.buffer</name>
  <value>array</value>
</property>
```

Настройка загрузки потока

Механизмы буферизации *ByteBuffers* (*Буферизация загружаемых данных в ByteBuffers*) и *byte arrays* (*Буферизация загружаемых данных в массивы*) могут потреблять очень большие объемы памяти, как в оперативной памяти, так и во внешней. А механизм *disk buffer* (*Буферизация загружаемых данных на диск*) не занимает много памяти, но потребляет емкость жесткого диска.

Если в один процесс записывается много выходных потоков, объем используемой памяти или диска кратен объему активной памяти/диска всех потоков. Может потребоваться тщательная настройка с целью снижения риска исчерпания памяти.

Есть ряд параметров, которые могут быть настроены:

1. `fs.s3a.threads.max` – общее количество потоков, доступных в файловой системе для загрузки данных, или любые другие операции файловой системы, находящиеся в очереди.
2. `fs.s3a.max.total.tasks` – количество операций, которые могут быть поставлены в очередь на выполнение в ожидании потока.
3. `fs.s3a.fast.upload.active.blocks` – количество блоков, которые могут иметь один активный выходной поток: загрузка в поток или постановка в очередь в поток файловой системы.
4. `fs.s3a.threads.keepalivetime` – как долго неиспользуемый поток может оставаться в пуле потоков до его удаления.

При достижении максимально допустимого количества активных блоков одного потока больше никакие блоки не могут быть загружены из этого потока до тех пор, пока не завершится загрузка одного или нескольких из этих активных блоков. То есть вызов `write()`, который инициирует загрузку нового полного блока данных, блокируется до тех пор, пока в очереди не появится место.

Как это получается:

- Поскольку пул потоков, установленный в `fs.s3a.threads.max`, является общим (и предназначен для использования между всеми потоками), заданное большее значение может увеличить параллельные операции. Однако, поскольку для загрузки требуется пропускная способность сети, добавление большего количества потоков не гарантирует ускорение.
- Дополнительная очередь задач для пула потоков (`fs.s3a.max.total.tasks`) покрывает все текущие фоновые операции S3A (будущие планы включают: параллельные операции переименования, асинхронные операции с каталогами).
- При использовании буферизации памяти небольшое значение `fs.s3a.fast.upload.active.blocks` ограничивает объем памяти, который может быть использован для каждого потока.

- При использовании дисковой буферизации установленное высокое значение `fs.s3a.fast.upload.active.blocks` не занимает много памяти. Но это может привести к большому количеству блоков, чтобы конкурировать с другими операциями файловой системы.

Рекомендуется устанавливать низкое значение `fs.s3a.fast.upload.active.blocks`. Этого будет достаточно, чтобы начать фоновую загрузку без перегрузки других частей системы, а затем поэкспериментировать, чтобы увидеть, обеспечивают ли более высокие значения лучшую пропускную способность, особенно для виртуальных машин на **EC2**.

```
<property>
  <name>fs.s3a.fast.upload.active.blocks</name>
  <value>4</value>
  <description>
    Maximum Number of blocks a single output stream can have
    active (uploading, or queued to the central FileSystem
    instance's pool of queued operations.

    This stops a single stream overloading the shared thread pool.
  </description>
</property>

<property>
  <name>fs.s3a.threads.max</name>
  <value>10</value>
  <description>The total number of threads available in the filesystem for data
    uploads *or any other queued filesystem operation*.</description>
</property>

<property>
  <name>fs.s3a.max.total.tasks</name>
  <value>5</value>
  <description>The number of operations which can be queued for execution</description>
</property>

<property>
  <name>fs.s3a.threads.keepalivetime</name>
  <value>60</value>
  <description>Number of seconds a thread can be idle before being
    terminated.</description>
</property>
```

Очистка после частичных сбоев загрузки

Существует два механизма очистки после многократных загрузок:

- Команды *Hadoop s3guard CLI* для перечисления и удаления загрузок по их возрасту;
- Параметр конфигурации `fs.s3a.multipart.purge`.

Если операция записи большого потока прерывается, на **S3** могут быть загружены промежуточные разделы. Чтобы сократить расходы, можно включить `fs.s3a.multipart.purge` и установить время очистки в секундах, например, *86400* (24 часа). Когда экземпляр *S3A FileSystem* создается с временем очистки больше нуля, при запуске он удаляет все оставшиеся невыполненные запросы разделов старше этого времени.

```
<property>
  <name>fs.s3a.multipart.purge</name>
  <value>true</value>
  <description>True if you want to purge existing multipart uploads that may not have been
     completed/aborted correctly</description>
```

```
</property>

<property>
  <name>fs.s3a.multipart.purge.age</name>
  <value>86400</value>
  <description>Minimum age in seconds of multipart uploads to purge</description>
</property>
```

Если клиент **S3A** создается с помощью `fs.s3a.multipart.purge=true`, он удаляет все устаревшие загрузки во всем сегменте. То есть это влияет на все многократные загрузки в этот сегмент из всех приложений. Оставив значение `fs.s3a.multipart.purge` по умолчанию равным *false*, это означает, что клиент не будет пытаться сбросить или изменить partition rate.

Рекомендуется использовать данную опцию, чтобы отключить многосоставную очистку при обычном использовании **S3A**, и включать ее только вручную при запланированных операциях по очистке.

Поддержка политики ввода S3A “fadvise”

Клиент файловой системы **S3A** поддерживает понятие политик ввода, аналогичное понятию вызова API Posix `fadvise()`. Настраивает поведение клиента **S3A** для оптимизации запросов HTTP GET для различных вариантов использования.

1.5.10 Метрики

Метрики **S3A** можно отслеживать с помощью Hadoop-платформы **metrics2**. **S3A** создает свою собственную систему метрик **s3a-file-system**, и каждый экземпляр клиента создает собственный источник метрик, именуемый уникальным числовым идентификатором **JVM**.

В качестве простого примера в `hadoop-metrics2.properties` можно добавить следующее свойство для записи всех метрик **S3A** в файл журнала каждые *10* секунд:

```
s3a-file-system.sink.my-metrics-config.class=org.apache.hadoop.metrics2.sink.FileSink
s3a-file-system.sink.my-metrics-config.filename=/var/log/hadoop-yarn/s3a-metrics.out
*.period=10
```

Тогда строки в этом файле будут структурированы следующим образом:

```
1511208770680 s3aFileSystem.s3aFileSystem: Context=s3aFileSystem, s3aFileSystemId=892b02bb-7b30-4ffe-80ca-
↪3a9935e1d96e, bucket=bucket,
Hostname=hostname-1.hadoop.apache.com, files_created=1, files_copied=2, files_copied_bytes=10000, files_
↪deleted=5, fake_directories_deleted=3,
directories_created=3, directories_deleted=0, ignored_errors=0, op_copy_from_local_file=0, op_exists=0, op_
↪get_file_status=15, op_glob_status=0,
op_is_directory=0, op_is_file=0, op_list_files=0, op_list_located_status=0, op_list_status=3, op_mkdirs=1,␣
↪op_rename=2, object_copy_requests=0,
object_delete_requests=6, object_list_requests=23, object_continue_list_requests=0, object_metadata_
↪requests=46, object_multipart_aborted=0,
object_put_bytes=0, object_put_requests=4, object_put_requests_completed=4, stream_write_failures=0, stream_
↪write_block_uploads=0,
stream_write_block_uploads_committed=0, stream_write_block_uploads_aborted=0, stream_write_total_time=0,␣
↪stream_write_total_data=0,
s3guard_metadatastore_put_path_request=10, s3guard_metadatastore_initialization=0, object_put_requests_
↪active=0, object_put_bytes_pending=0,
stream_write_block_uploads_active=0, stream_write_block_uploads_pending=0, stream_write_block_uploads_data_
↪pending=0,
S3guard_metadatastore_put_path_latencyNumOps=0, S3guard_metadatastore_put_path_
↪latency50thPercentileLatency=0,
```

```
S3guard_metadatastore_put_path_latency75thPercentileLatency=0, S3guard_metadatastore_put_path_
↪latency90thPercentileLatency=0,
S3guard_metadatastore_put_path_latency95thPercentileLatency=0, S3guard_metadatastore_put_path_
↪latency99thPercentileLatency=0
```

В зависимости от конфигураций, метрик из других систем, контекстов и т.д. могут быть, например, следующие записи:

```
1511208770680 metricssystem.MetricsSystem: Context=metricssystem, Hostname=s3a-metrics-4.gce.cloudera.com,␣
↪NumActiveSources=1, NumAllSources=1,
NumActiveSinks=1, NumAllSinks=0, Sink_fileNumOps=2, Sink_fileAvgTime=1.0, Sink_fileDropped=0, Sink_
↪fileQsize=0, SnapshotNumOps=5,
SnapshotAvgTime=0.0, PublishNumOps=2, PublishAvgTime=0.0, DroppedPubAll=0
```

Важно обратить внимание, что низкоуровневые метрики из самого **AWS SDK** в настоящее время не включены в данные метрики.

1.5.11 Копирование данных с distcp

Инструмент `distcp` **Hadoop** часто используется для копирования данных между кластером **Hadoop** и **Amazon S3**.

Команда `distcp update` пытается выполнять инкрементное обновление данных. Нетрудно проверить, когда файлы не совпадают или имеют разную длину, но не в случаях, когда они имеют одинаковый размер. Инструмент `distcp` решает эту проблему путем сравнения контрольных сумм файлов в исходной и целевой файловых системах, что он и пытается выполнить, даже если файловые системы имеют несовместимые алгоритмы контрольных сумм.

Коннектор **S3A** может предоставить заголовок HTTP etag вызывающей стороне в качестве контрольной суммы загруженного файла, но это приводит к разрыву операций `distcp` между *hdfs* и *s3a*. По этой причине функция `etag-as-checkum` отключена по умолчанию.

```
<property>
  <name>fs.s3a.etag.checksum.enabled</name>
  <value>false</value>
  <description>
    Should calls to getFileChecksum() return the etag value of the remote
    object.
    WARNING: if enabled, distcp operations between HDFS and S3 will fail unless
    -skipcrccheck is set.
  </description>
</property>
```

Когда параметр включен, `distcp` может использовать контрольную сумму для сравнения объектов между двумя сегментами **S3**. Тогда в случае если каждый из сегментов загружен как один файл *PUT*, или, если он состоит из нескольких частей *PUT* – в блоках одинакового размера, сконфигурированного значением `fs.s3a.multipart.size` – контрольные суммы сегментов должны быть идентичными.

Для отключения проверки контрольной суммы в `distcp` следует использовать опцию `-skipcrccheck`:

```
hadoop distcp -update -skipcrccheck /user/alice/datasets s3a://alice-backup/datasets
```

1.6 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.

1.6.1 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*.

1.6.2 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.

Таблица1.1.: 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
-P	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.

Таблица1.2.: 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.

1.7 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.

1.7.1 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.

1.7.2 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

1.7.3 Настройка архивного хранилища

Для настройки архивного хранилища необходимо выполнить следующие действия:

1. Выключить DataNode

Закрыть **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
```

1.8 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 <*n*> копии блока перемещаются в буферный кэш. При централизованном управлении кэшем точно указывается только <*m*> копий <*n*>, тем самым сохраняя память <*n*-*m*>.

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.

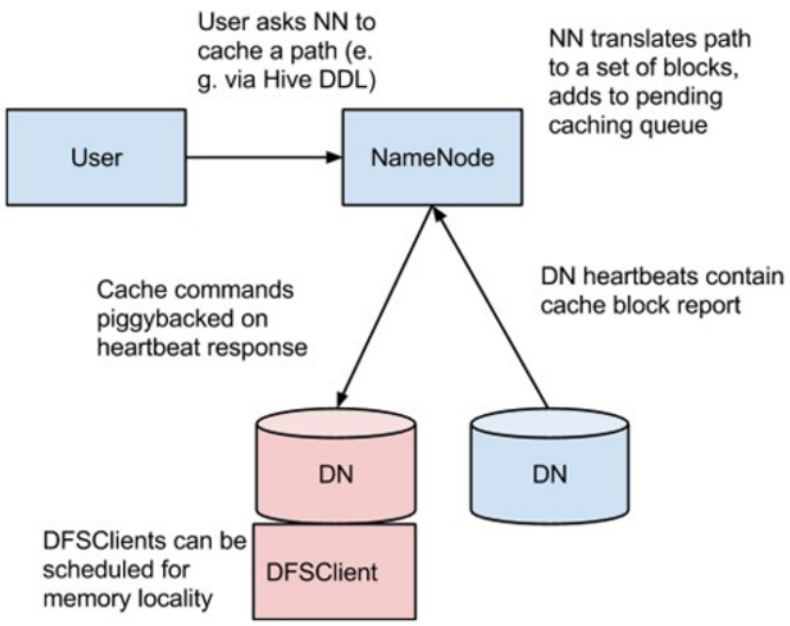


Рис.1.3.: Архитектура кэширования

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

1.8.1 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.

1.8.2 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 директив. 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>
```

1.8.3 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`

1.8.4 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>]
```

Таблица1.3.: 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 *0755*
<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, *120 s*, *30 m*, *4 h*, *2 d*. Допустимыми единицами являются *[smhd]*. По умолчанию максимальное значение не задано. Значение *never* 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>]
```

Таблица1.4.: 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, *120 s*, *30 m*, *4 h*, *2 d*. Допустимыми единицами являются *[smhd]*. По умолчанию максимальное значение не задано. Значение *never* 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 *0755*
<limit>	Используемый коэффициент репликации кэша. По умолчанию установлено значение *1*
<time-to-live>	Продолжительность действия директивы. Значение может быть указано в минутах, часах и днях, например, *30 m*, *4 h*, *2 d*. Допустимыми единицами являются *[mhd]*. Значение *never* означает, что директива никогда не истекает. Если параметр не установлен, директива никогда не истекает

removeDirective – команда удаления Cache Directive:

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

Таблица1.9.: Функции команды removeDirective

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

removeDirectives – команда удаления всех Cache Directives по указанному пути:

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

Таблица1.10.: Функции команды removeDirectives

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

listDirectives – команда возврата списка Cache Directives:

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

Таблица1.11.: Функции команды listDirectives

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

1.9 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.

1.9.1 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

1.10 Примеры ACL

1.10.1 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 отличаются 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.

1.10.2 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.

1.10.3 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.

1.10.4 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.

1.11 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 не используется полностью 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 не используется полностью 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**.

1.11.1 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 URI. Но при этом пользователи могут создать символическую ссылку (из *hdfs://namenode/foo* для *har://namenode/bar.har/foo* в примере выше), и тогда изменять URI 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.

1.11.2 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 ссылаться 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
```

1.11.3 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*.

1.11.4 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.

1.12 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.

1.13 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
```

1.13.1 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 **JXM** через веб-интерфейс **HDFS daemon**, а не напрямую обращаться к удаленному агенту **JMX**.

1.14 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.

1.14.1 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*.

1.14.2 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

1.14.3 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*.

1.14.4 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
```

1.15 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*;

1.15.1 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
```

```
  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.

1.15.2 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>
```

1.15.3 Перезапуск HDFS и MapReduce

Перезапустить **HDFS** и **MapReduce**.

1.15.4 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
```

1.16 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* – тайм-аут 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* – тайм-аут 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>
```

1.17 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.

1.18 Добавление/удаление и обслуживание/декоммиссия HDFS DataNode

Для добавления или удаления HDFS DataNode с хостов или привидения HDFS DataNode в состояние обслуживания или декоммисии 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.

1.18.1 Добавление HDFS DataNode

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

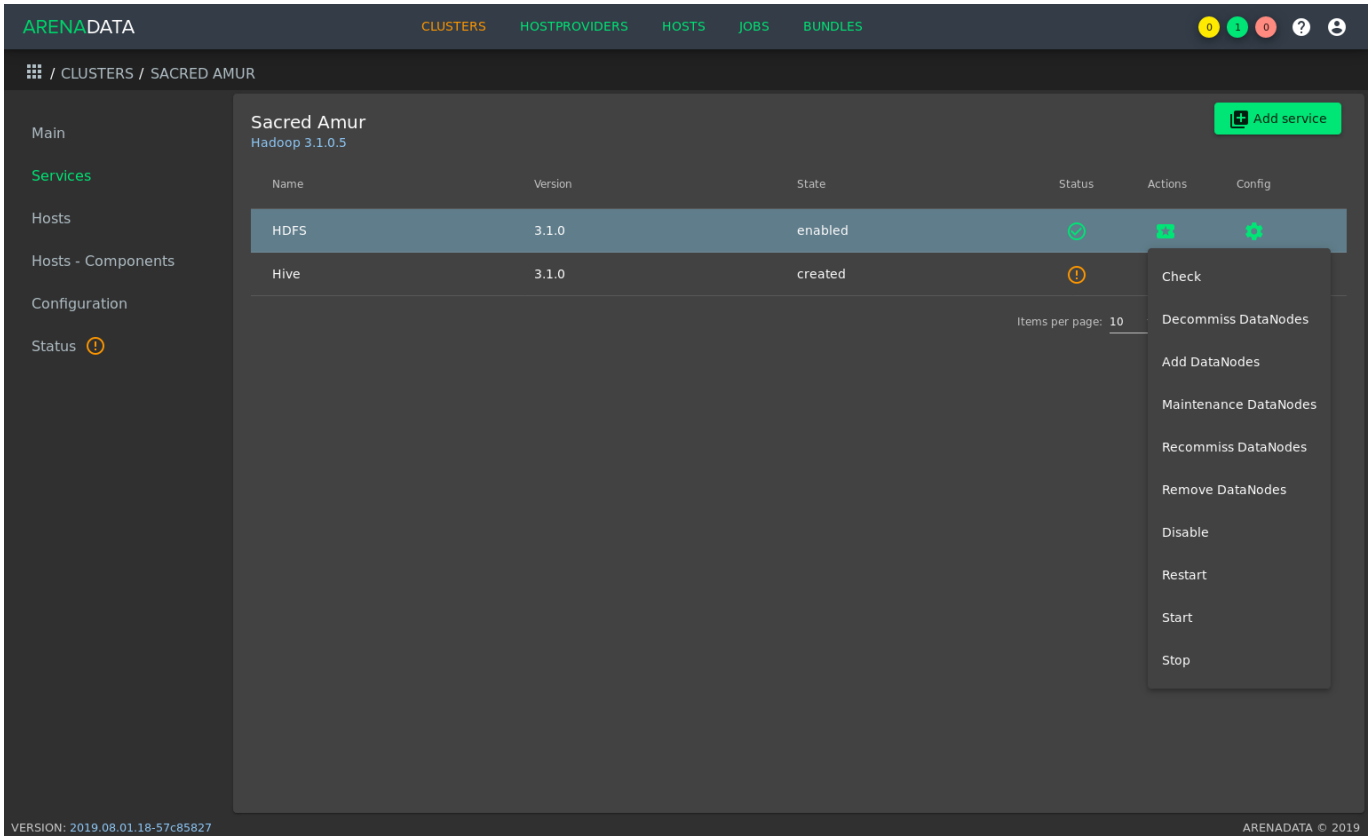


Рис.1.4.: Выпадающее меню *"Actions"* сервиса HDFS

1. нажать кнопку *“Add DataNodes”*, что приведет 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.

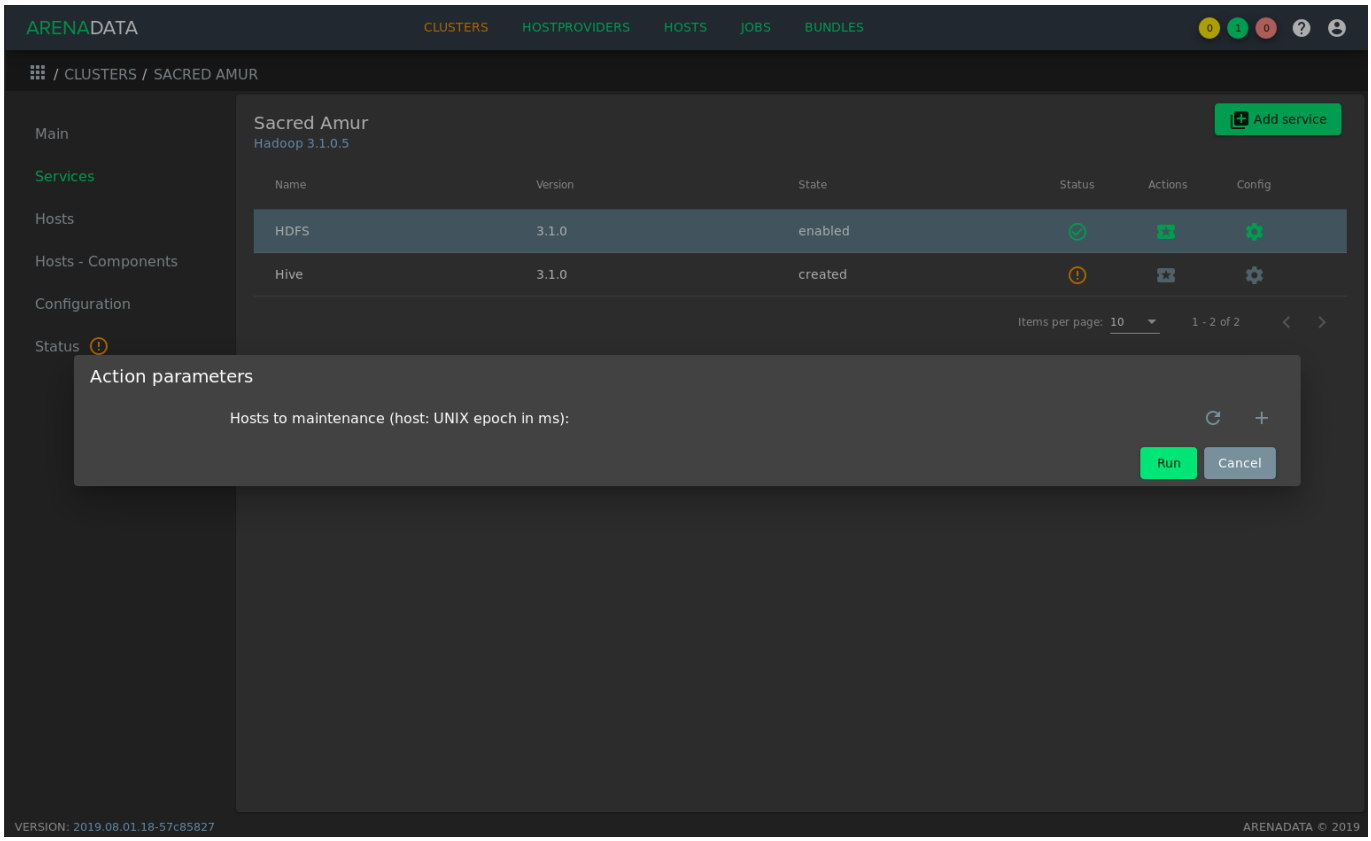
1.18.2 Обслуживание HDFS DataNode

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 переведения одной или нескольких HDFS DataNode в режим обслуживания необходимо:

1. нажать кнопку *“Maintenance DataNodes”*, что приведет 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.

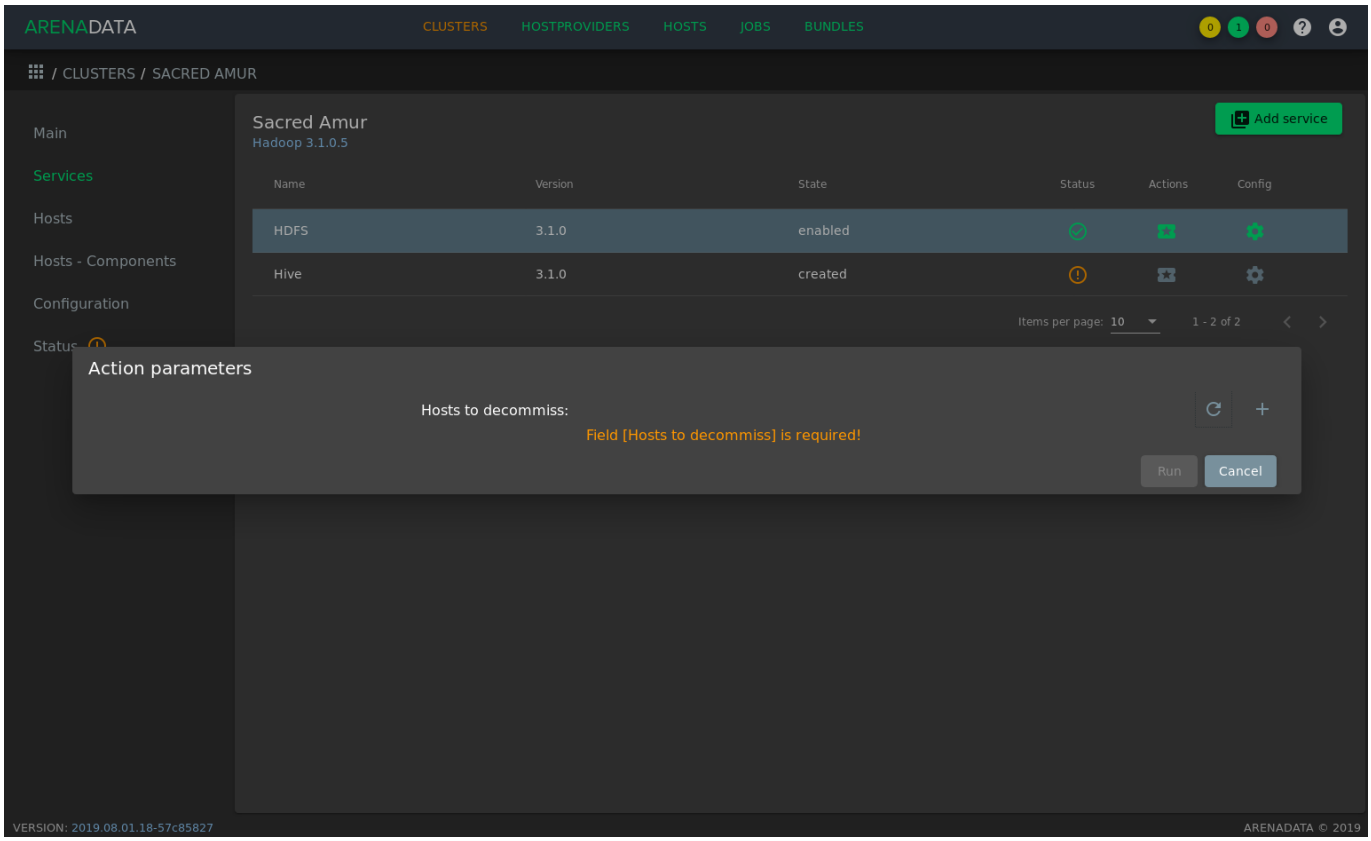
1.18.3 Декоммиссия HDFS DataNode

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 приведет 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.

1.18.4 Рекоммиссия HDFS DataNode

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

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

1.18.5 Удаление HDFS DataNode

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

1. нажать кнопку *"Remove DataNodes"*, что приведет 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 зеленым;
   - выбрать хост в колонке "Hosts" и убрать выделение с компонента HDFS DataNode в колонке "Components" если рамка компонента HDFS DataNode выделяется зеленым.
3. нажать кнопку *"Run"* в нижней части окна.

1.19 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*.

1.19.1 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*.

1.19.2 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**.

1.19.3 Обратная связь

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

1.19.4 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**.

1.19.5 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**.

1.19.6 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.

1.19.7 Символические ссылки

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

1.19.8 Снапшоты

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

1.19.9 Инструментарий

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

1.19.10 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:
```

1.19.11 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-
```

1.19.12 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**.

1.19.13 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.

1.19.14 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::---
```

1.19.15 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.

1.19.16 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 Apache Tez

Apache TEZ® позволяет создать комплексный ациклический граф задач для обработки данных. В настоящее время он реализован для Apache YARN.

Включает основные возможности:

- Гибкая модель исполнения
- Упрощение развертывания
- Повышение производительности по сравнению с Map Reduce
- Оптимальное управление ресурсами
- Планирование реконфигурации во время выполнения

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

2.2 Настройка памяти для Hive/Tez

Для корректной работы Hive/Tez, требуется задать соответствующие значения для tez.am.resource.memory.mb, hive.tez.container.size, hive.tez.java.opts.

- `tez.am.resource.memory.mb` – должно быть равно yarn.scheduler.minimum-allocation-mb
- `hive.tez.container.size` – 1x или 2x yarn.scheduler.minimum-allocation-mb, но не более yarn.scheduler.maximum-allocation-mb

Нужно учитывать также, если у вас 256GB и 16 ядер, размер контейнера не должен быть больше 16GB.

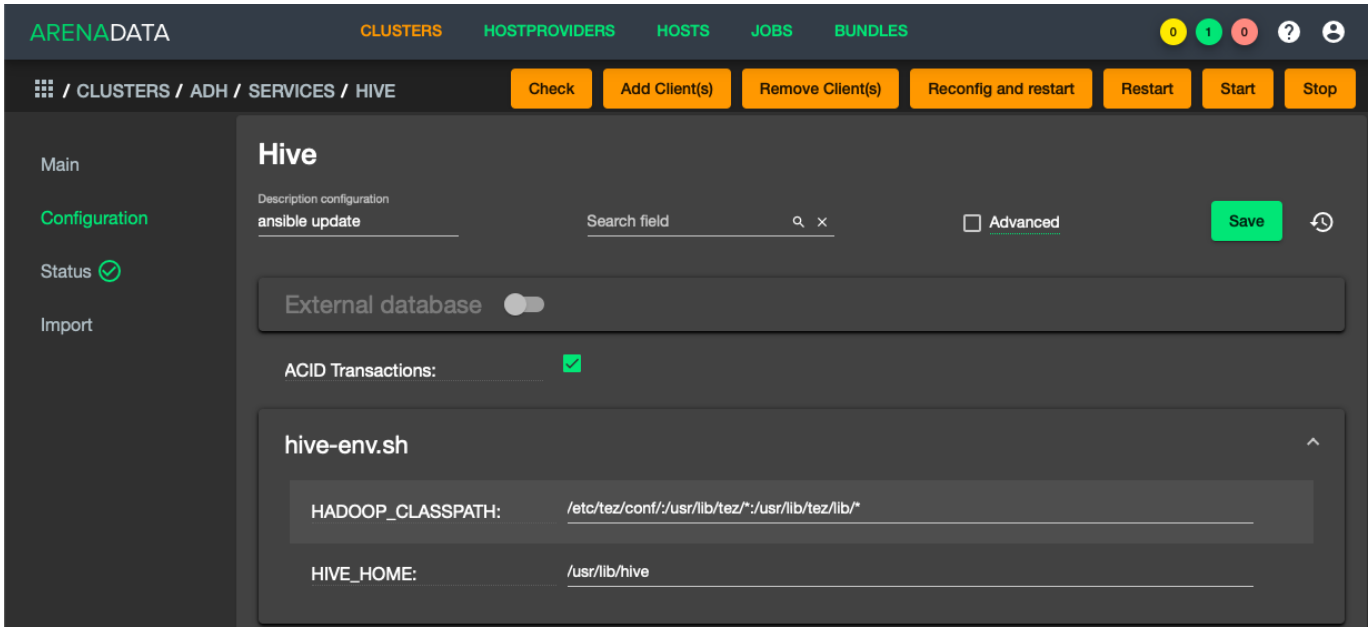
Далее задать tez.runtime.io.sort.mb, tez.runtime.unordered.output.buffer.size-mb, hive.auto.convert.join.noconditionaltask.size

- `tez.runtime.io.sort.mb` – как 40% от hive.tez.container.size. Редко более 2ГБ
- `hive.auto.convert.join.noconditionaltask` – true
- `hive.auto.convert.join.noconditionaltask.size` – 1/3 от hive.tez.container.size
- `tez.runtime.unordered.output.buffer.size-mb` – до 10% от hive.tez.container.size
- `tez.grouping.min-size=16777216` – около 16 MB минимальный сплит
- `tez.grouping.max-size=1073741824` – около 1 ГБ максимальный сплит

2.3 Hive ACID

В Hive 3 вы можете выполнять транзакции ACID (atomicity, consistency, isolation, and durability) v2 на уровне строк. Операции Hive являются атомарными на уровне строк, а не на уровне таблиц или разделов. Клиент Hive может читать из раздела, в то время как другой клиент добавляет строки в раздел.

Для включения поддержки ACID активируйте параметр ACID Transactions как указано на рисунке (Рис.2.3.).:



Important: В Hive 3 вы не можете отключить транзакции.

Как администратор, вы можете просматривать список открытых и прерванных транзакций.

Введите запрос для просмотра транзакций.

```
SHOW TRANSACTIONS
```

Следующая информация появляется в выводе:

- Transaction ID
- Transaction state
- Hive пользователь, который инициировал транзакцию
- Хост-машина, на которой была инициирована транзакция

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 регулирования 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 привидения *YARN NodeManager* в состояние декомисии 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.

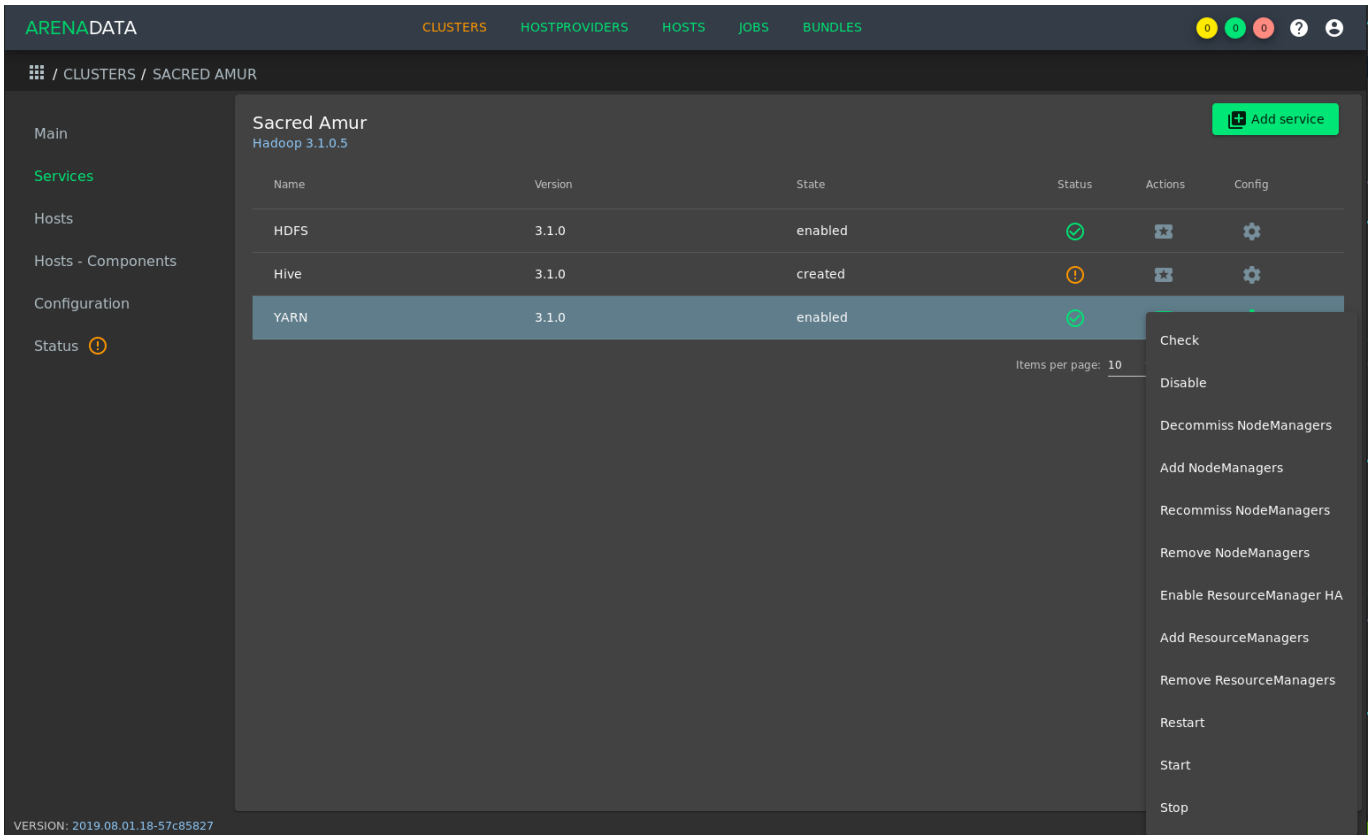


Рис.4.1.: Выпадающее меню “Actions” сервиса YARN

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 ноде контейнеров;
- Изящный (graceful) – в этом режиме планировщик *YARN* не будет назначать новые контейнеры декомиссованной ноде *YARN NodeManager*, а работающие контейнеры продолжат 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.).

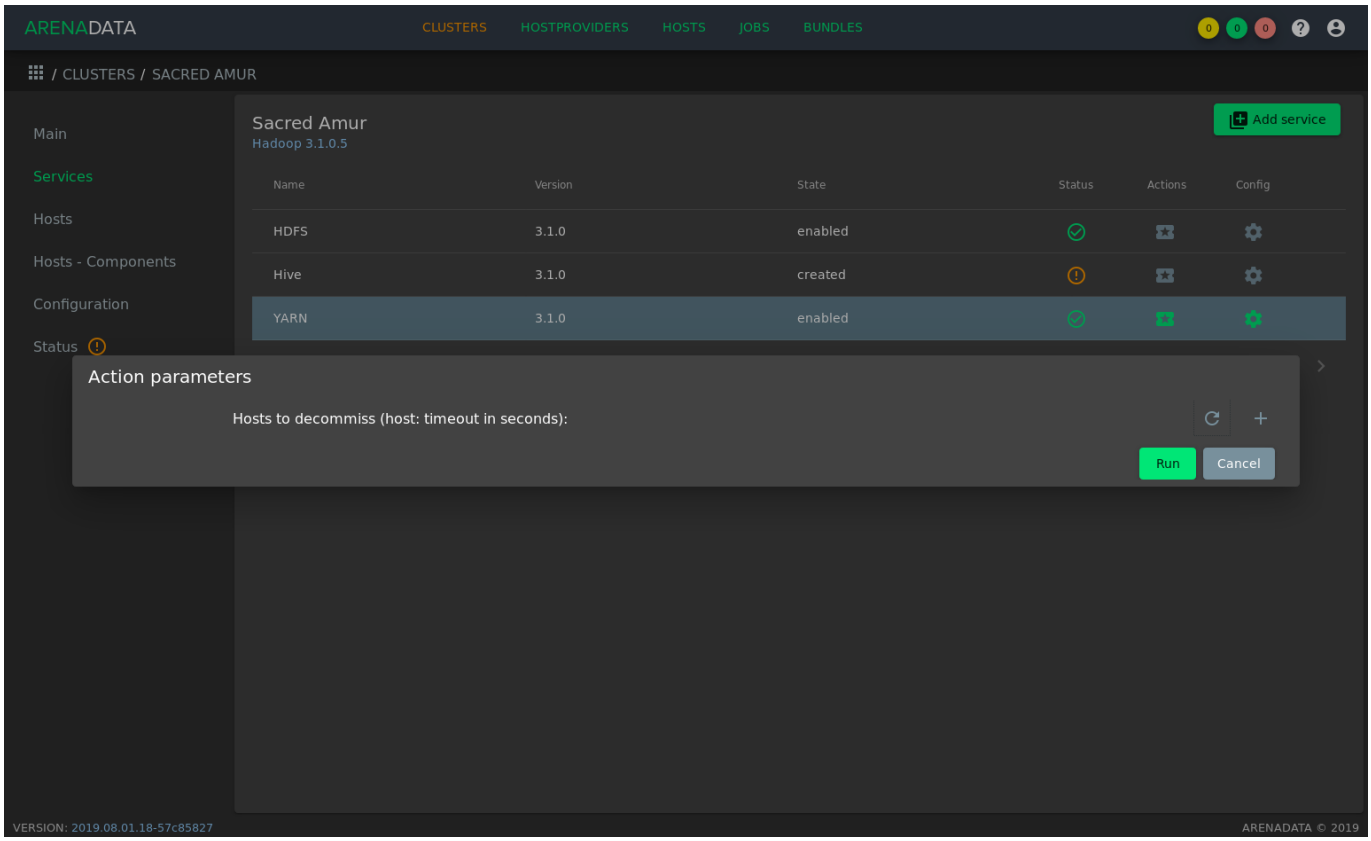


Рис.4.2.: "Action parameters"

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 *"Recommiss 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

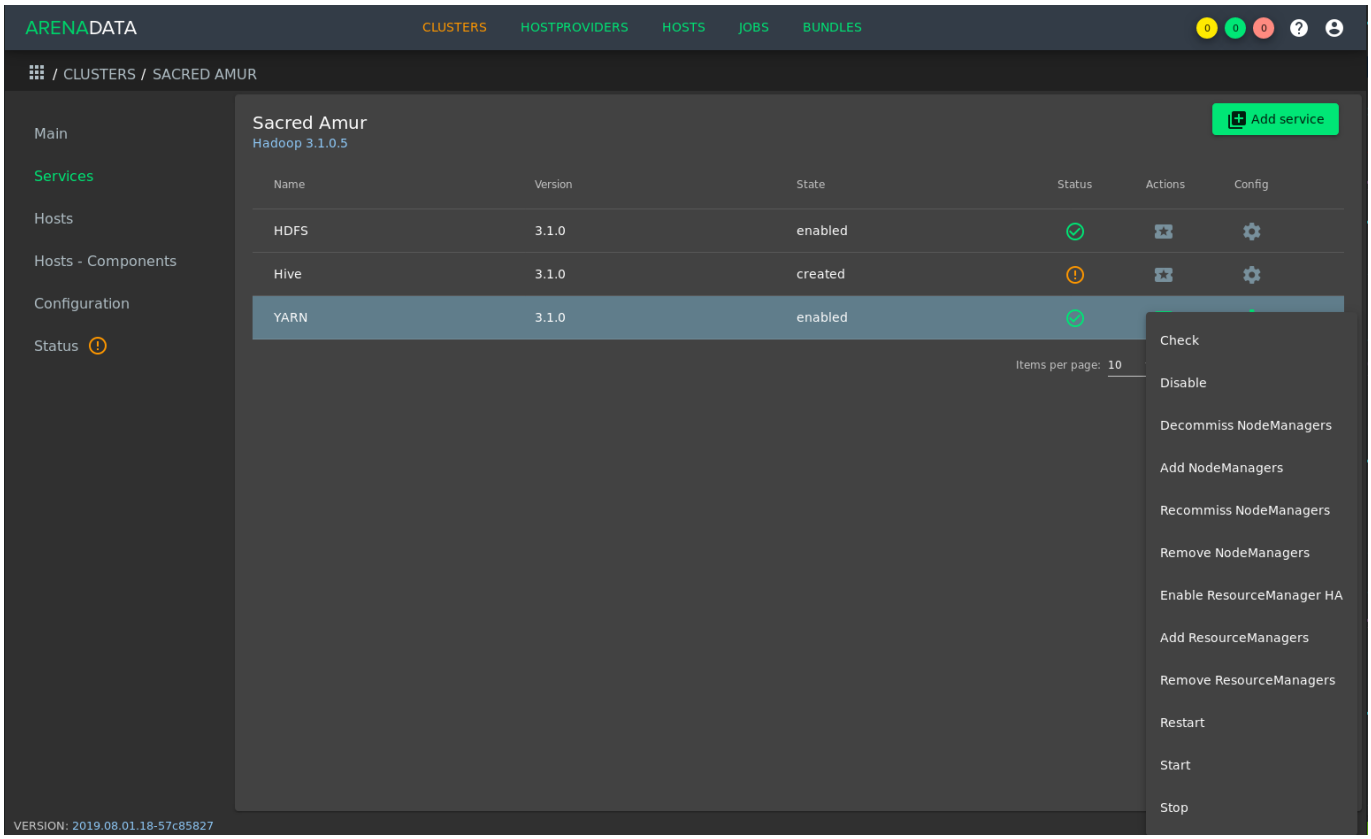


Рис.4.3.: Выпадающее меню “Actions” сервиса YARN

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.

4.3 Hadoop: Capacity Scheduler

В главе описывается **CapacityScheduler** – подключаемый планировщик для **Hadoop**, позволяющий при мультитенантности безопасно совместно использовать большой кластер таким образом, чтобы для приложений своевременно распределялись ресурсы в условиях ограниченно выделенных мощностей.

CapacityScheduler предназначен для запуска приложений **Hadoop** в виде общего мультитенантного кластера удобным для оператора способом при максимальной пропускной способности и загрузке кластера.

Традиционно каждая организация имеет свой собственный набор вычислительных ресурсов, которые имеют достаточную производительность для соответствия SLA предприятия в пиковых или около пиковых условиях. Как правило, это приводит к низкой средней загрузке и накладным расходам на управление несколькими независимыми кластерами по одному на каждую организацию. Поэтому совместное использование кластеров между несколькими организациями – это рентабельный способ запуска крупных Hadoop-инсталляций, так как это позволяет пользоваться преимуществами масштаба, не создавая частных кластеров. Однако организации обеспокоены совместным использованием кластера в вопросе использования другими предприятиями ресурсов, критически важных для их собственного SLA.

CapacityScheduler предназначен для совместного использования большого кластера, предоставляя при этом каждой организации гарантии производительности. Основная идея заключается в том, что доступные ресурсы в кластере **Hadoop** распределяются между несколькими предприятиями. Дополнительным преимуществом является то, что организация может получить доступ к любой избыточной мощности, не используемой другими. Это обеспечивает гибкость экономически эффективным образом.

Совместное использование кластеров требует строгой мультитенантности, поскольку каждому предприятию должна быть обеспечена производительность и безопасность, чтобы гарантировать, что общий кластер защищен от любого постороннего приложения или пользователя. **CapacityScheduler** предоставляет обязательный набор ограничений, гарантирующих, что отдельное приложение, пользователь или очередь не могут использовать непропорционально большое количество ресурсов в кластере. Кроме того, для обеспечения справедливости и стабильности кластера планировщик предоставляет для инициализированных и ожидающих приложений от одного пользователя очереди и ограничения.

Основной абстракцией, предоставляемой **CapacityScheduler**, является концепция очередей. Они обычно настраиваются администраторами и отражают экономику общего кластера.

С целью дополнительного контроля и предсказуемости при совместном использовании ресурсов **CapacityScheduler** также поддерживает иерархические очереди, чтобы обеспечить распределение ресурсов между под-очередями среди приложений внутри одной организации, прежде чем другим очередям будет позволено использовать свободные ресурсы.

4.3.1 Функции

CapacityScheduler поддерживает следующие функции:

- Иерархические очереди (Hierarchical Queues).

Поддерживается иерархия очередей, обеспечивающая совместное использование ресурсов между под-очередями внутри организации, прежде чем другим очередям будет позволено использовать свободные ресурсы, что обеспечивает больший контроль и предсказуемость.

- Гарантии производительности (Capacity Guarantees).

Очереди распределяются по части пропускной способности сети в том смысле, что в их распоряжении находится определенная производительность ресурсов. Все приложения, отправленные в очередь, имеют доступ к производительности, выделенной для этой конкретной очереди. Для каждой очереди ограничения пропускной способности настраиваются администраторами и могут быть как мягкими, так и жесткими.

- Безопасность (Security).

Каждая очередь имеет строгие списки ACL, которые контролируют, какие пользователи могут отправлять приложения в отдельные очереди. Кроме того, существуют средства защиты, гарантирующие, что пользователи не смогут просматривать и/или изменять приложения других пользователей. Также поддерживаются роли для каждой очереди и системного администратора.

- Эластичность (Elasticity).

Свободные ресурсы могут быть распределены на любые очереди. Когда в будущем от очередей, работающих с пониженной производительностью, возникает потребность в ресурсах, то по мере выполнения запланированных на этих ресурсах задач они назначаются требуемым приложениям (также поддерживается преимущественное право – preemption). Это гарантирует, что ресурсы доступны для очередей предсказуемо и гибко, тем самым предотвращая искусственное разделение ресурсов в кластере и помогая их использованию.

- Мультитенантность (Multi-tenancy).

Предоставляется набор ограничений для предотвращения монополизации ресурсов очереди или кластера одним приложением, пользователем или очередью, чтобы гарантировать, что кластер не перегружен.

- Работоспособность (Operability):
  - Конфигурация во время выполнения (Runtime Configuration) – определения и свойства очереди, такие как производительность и списки ACL, могут быть изменены во время выполнения безопасным способом администраторами, минимизируя неудобства для пользователей. Кроме того для пользователей и администраторов предусмотрена консоль, позволяющая просматривать текущее распределение ресурсов по различным очередям в системе. Администраторы могут добавлять дополнительные очереди во время выполнения, но очереди не могут быть удалены во время выполнения, если она не остановлена и имеет ожидающие/запущенные приложения.
  - Дренаж приложений (Drain applications) – администраторы могут останавливать очереди во время выполнения, чтобы гарантировать, что пока существующие приложения не будут завершены, новые не смогут быть направлены. Если очередь находится в состоянии *STOPPED*, новые приложения не могут быть направлены ни ей самой, ни какой-либо из ее дочерних очередей. Текущие приложения продолжают выполняться и, таким образом, очередь может быть аккуратно дренажирована. Администраторы также могут запускать остановленные очереди.
- Планирование на основе ресурсов (Resource-based Scheduling).

Поддержка ресурсоемких приложений, в которых приложение может опционально определять более высокие требования к ресурсам, чем по умолчанию, тем самым приспосабливая приложения с различными требованиями к ресурсам. В настоящее время память является поддерживаемым требованием к ресурсам.

- Маппинг очереди на основе пользователя или группы (Queue Mapping based on User or Group).

Функция позволяет пользователям сопоставлять работу с определенной очередью на основе пользователя или группы.

- Приоритетное планирование (Priority Scheduling).

Функция позволяет направлять приложения и планировать их с разными приоритетами. Более высокое целочисленное значение указывает на более высокий приоритет для приложения. В настоящее время приоритет приложения поддерживается только для политики упорядочения *FIFO*.

- Конфигурация абсолютных ресурсов (Absolute Resource Configuration).

Администраторы могут указывать абсолютные ресурсы для очереди вместо предоставления значений в процентах. Это обеспечивает лучший контроль для администраторов в целях настройки необходимого количества ресурсов для конкретной очереди.

- Динамическое автоматическое создание и управление конечными очередями (Dynamic Auto-Creation and Management of Leaf Queues).

Функция поддерживает автоматическое создание конечных очередей в сочетании с маппингом очередей, которое в настоящее время поддерживает сопоставления очередей на основе групп пользователей для размещения приложений в очереди. Планировщик также поддерживает управление производительностью для этих очередей на основе политики, настроенной в родительской очереди.

4.3.2 Конфигурация

Чтобы настроить **ResourceManager** для использования **CapacityScheduler**, необходимо установить в файле *conf/yarn-site.xml* свойство `yarn.resourcemanager.scheduler.class` со значением `org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler`.

etc/hadoop/capacity-scheduler.xml – файл конфигурации для **CapacityScheduler**.

CapacityScheduler имеет предопределенную очередь с именем *root*, все очереди в системе являются дочерними по отношению к ней. Очереди можно настроить в `yarn.scheduler.capacity.root.queues` со списком дочерних очередей, разделенных запятыми.

Конфигурация для **CapacityScheduler** для настройки иерархии очередей использует концепцию, называемую *путь к очереди* (*queue path*). Путь к очереди – это полный путь иерархии очереди, начиная с *root*, со знаком точки `.` в качестве разделителя.

Дочерние элементы очереди могут быть определены с помощью настройки `yarn.scheduler.capacity.<queue-path>.queues`. Дочерние очереди при этом не наследуют свойства напрямую от родителя, если не указано иное.

Пример с тремя дочерними очередями верхнего уровня *a*, *b* и *c* и некоторыми подпоследовательностями для *a* и *b*:

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>a,b,c</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.a.queues</name>
  <value>a1,a2</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.b.queues</name>
  <value>b1,b2,b3</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>
```

Свойства очереди

Распределение ресурсов

`yarn.scheduler.capacity.<queue-path>.capacity` – пропускная способность очереди ИЛИ минимальная пропускная способность очереди абсолютных ресурсов, указывается в процентах в виде числа с плавающей запятой (float, например, *12.5*). Сумма производительности для всех очередей на каждом уровне должна быть равна *100*. Однако, если настроен абсолютный ресурс, сумма абсолютных ресурсов дочерних очередей может быть меньше абсолютной производительности родительского ресурса. Приложения в очереди могут потреблять

больше ресурсов, чем пропускная способность очереди, если есть свободные ресурсы, обеспечивающие эластичность.

`yarn.scheduler.capacity.<queue-path>.maximum-capacity` – максимальная пропускная способность очереди ИЛИ максимальная пропускная способность очереди абсолютных ресурсов, указывается в процентах в виде числа с плавающей запятой (float). Параметр ограничивает эластичность для приложений в очереди: 1) Значение находится в диапазоне от *0* до *100*; 2) Администратор должен убедиться, что абсолютная максимальная производительность больше или равна абсолютной производительности для каждой очереди. Кроме того, установка значения в *-1* задает максимальную производительность в *100%*.

`yarn.scheduler.capacity.<queue-path>.minimum-user-limit-percent` – каждая очередь устанавливает ограничение на процент ресурсов, выделяемых пользователю в любой момент времени при потребности в ресурсах. Пользовательское ограничение может варьироваться между минимальным и максимальным значением. Минимум устанавливает данное свойство, а максимум зависит от количества отправивших приложение пользователей. Например, значение свойства равно *25*. Тогда если два пользователя отправляют приложения в очередь, ни один из них не может использовать более *50%* ресурсов очереди. Если третий пользователь отправляет приложение, то ни один пользователь не может использовать более *33%* ресурсов очереди. При наличии *4 или более* пользователей ни один из них не может использовать более *25%* ресурсов очереди. Значение *100* подразумевает, что ограничения для пользователей не вводятся. По умолчанию устанавливается значение *100*. Значение указывается как целое число (integer).

`yarn.scheduler.capacity.<queue-path>.user-limit-factor` – множество пропускной способности очереди, которое может быть настроено так, чтобы позволить пользователю получить больше ресурсов. По умолчанию значение равно *1*, что гарантирует, что пользователь никогда не сможет получить больше, чем настроенная производительность очереди, независимо от того, насколько простаивает кластер. Значение указывается как число с плавающей запятой (float).

`yarn.scheduler.capacity.<queue-path>.maximum-allocation-mb` – максимальный лимит памяти для каждой очереди, выделяемый каждому запросу контейнера в Resource Manager. Параметр переопределяет конфигурацию кластера `yarn.scheduler.maximum-allocation-mb`. Значение должно быть меньше или равно максимуму кластера.

`yarn.scheduler.capacity.<queue-path>.maximum-allocation-vcores` – максимальный лимит виртуальных ядер для каждой очереди, выделяемый каждому запросу контейнера в Resource Manager. Параметр переопределяет конфигурацию кластера `yarn.scheduler.maximum-allocation-vcores`. Значение должно быть меньше или равно максимуму кластера.

`yarn.scheduler.capacity.<queue-path>.user-settings.<user-name>.weight` – это значение с плавающей запятой (float), которое используется для вычисления предельных значений ресурсов пользователя среди пользователей в очереди. Значение определяет по весу каждого пользователя в большей или меньшей степени, относительно других пользователей в очереди. Например, если пользователь *A* должен получить на *50%* больше ресурсов в очереди, чем пользователи *B* и *C*, это свойство должно быть установлено равным *1,5* для пользователя *A*. При этом для пользователей *B* и *C* должно оставаться значение по умолчанию *1.0*.

Распределение ресурсов с помощью Absolute Resources

CapacityScheduler поддерживает настройку абсолютных ресурсов вместо предоставления процентной пропускной способности очереди. Как упоминается в конфигурации для `yarn.scheduler.capacity.<queue-path>.capacity` и `yarn.scheduler.capacity.<queue-path>.max-capacity`, администратор может указать значение абсолютного ресурса, например, `[memory=10240,vcores=12]`. Это допустимая конфигурация, указывающая *10 ГБ* памяти и *12 VCores*.

Ограничения для запущенных и ожидающих приложений

Для управления запущенными и ожидающими приложениями **CapacityScheduler** поддерживает следующие параметры:

`yarn.scheduler.capacity.maximum-applications / yarn.scheduler.capacity.<queue-path>.maximum-applicatio` – максимальное количество приложений в системе, которые могут быть одновременно активными (как запущенными, так и ожидающими). Ограничения в каждой очереди прямо пропорциональны пропускной способности очереди и пользовательским лимитам. Это жесткое ограничение, и любые поданные при его достижении приложения отклоняются. По умолчанию значение равно *10000*. Параметр может быть установлен для всех очередей с помощью `yarn.scheduler.capacity.maximum-applications`, а также может быть переопределен для каждой очереди путем задания `yarn.scheduler.capacity.<queue-path>.maximum-applications`. Параметр должен представлять собой целочисленное значение (integer).

`yarn.scheduler.capacity.maximum-am-resource-percent / yarn.scheduler.capacity.<queue-path>.maximum-am-` – максимальный процент ресурсов в кластере, которые могут быть использованы для запуска мастера (application masters), контролирующего количество одновременно работающих приложений. Ограничения в каждой очереди прямо пропорциональны пропускной способности очереди и пользовательским лимитам. Указывается как число с плавающей запятой (float), то есть значение *0.5* равно *50%*. По умолчанию задается *10%*. Значение может быть установлено для всех очередей с помощью параметра `yarn.scheduler.capacity.maximum-am-resource-percent`, а также может быть переопределено для каждой очереди путем задания `yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent`.

Администрирование и разрешения очереди

`yarn.scheduler.capacity.<queue-path>.state` – статус очереди: *RUNNING* или *STOPPED*. Если очередь находится в состоянии *STOPPED*, новые приложения не могут быть отправлены ни ей самой, ни какой-либо из ее дочерних очередей. Таким образом, если очередь *root* остановлена, никакие приложения не могут быть переданы всему кластеру, текущие приложения продолжают выполняться, и очередь может быть аккуратно дренажирована. Значение указывается в виде именованной константы (enumeration).

`yarn.scheduler.capacity.root.<queue-path>.acl_submit_applications` – список ACL, который контролирует, кто может подавать приложения в конкретную очередь. Если у пользователя/группы есть необходимые списки управления доступом в очереди или в одной из ее родительских очередей в иерархии, то пользователь/группа может подаваться. Списки ACL для этого свойства наследуются из родительской очереди, если не указано иное.

`yarn.scheduler.capacity.root.<queue-path>.acl_administer_queue` – список ACL, который контролирует, кто может администрировать приложения в конкретной очереди. Если у пользователя/группы есть необходимые ACL в очереди или в одной из ее родительских очередей в иерархии, то пользователь/группа может администрировать приложения. Списки ACL для этого свойства наследуются из родительской очереди, если не указано иное.

ACL имеет форму *user1,user2 space group1,group2*. Особое значение `*` подразумевает *все*. Особое значение `space` подразумевает *никто*. Значение по умолчанию `*` для очереди *root*, если не указано иное.

Маппинг очереди на основе пользователя или группы

`yarn.scheduler.capacity.queue-mappings` – конфигурация определяет маппинг пользователя или группы в определенную очередь. Можно сопоставить одного пользователя или список пользователей с очередями. Синтаксис: `[u or g]:[name]:[queue_name][,next_mapping]`. Обозначение *u* или *g* указывает, предназначено ли сопоставление для пользователя или группы соответственно; *name* указывает имя пользователя или имя группы. Чтобы указать пользователя, отправившего приложение, можно использовать *%user*. Обозначение *queue_name* указывает имя очереди, для которой должно маппироваться приложение. Чтобы указать имя очереди, совпадающее с именем пользователя, можно использовать *%user*. Чтобы указать имя очереди, совпадающее с именем основной группы, к которой принадлежит пользователь, можно использовать *%primary_group*.

`yarn.scheduler.capacity.queue-mappings-override.enable` – функция используется для задания возможности переопределения указанных пользователем очередей. Это логическое значение (boolean), и значением по умолчанию является *false*.

Пример:

```
<property>
  <name>yarn.scheduler.capacity.queue-mappings</name>
  <value>u:user1:queue1,g:group1:queue2,u:%user:%user,u:user2:%primary_group</value>
  <description>
    Here, <user1> is mapped to <queue1>, <group1> is mapped to <queue2>,
    maps users to queues with the same name as user, <user2> is mapped
    to queue name same as <primary group> respectively. The mappings will be
    evaluated from left to right, and the first valid mapping will be used.
  </description>
</property>
```

Срок приложений в очереди

`yarn.scheduler.capacity.<queue-path>.maximum-application-lifetime` – максимальное время жизни отправленного в очередь приложения, задается в секундах. Любое меньшее или равное нулю значение считается как отключенное и является жестким лимитом времени для всех приложений в этой очереди. Если задано положительное значение параметра, любое приложение, отправленное в данную очередь, уничтожается после превышения настроенного срока. Пользователь также может указать срок для каждого приложения в контексте. Срок пользователя переопределяется, если он превышает максимальное время жизни очереди. Это конфигурация на определенный момент времени. Настройка слишком низкого значения приводит к быстрому уничтожению приложения. Функция применима только для leaf-очереди.

`yarn.scheduler.capacity.root.<queue-path>.default-application-lifetime` – время жизни отправленного в очередь приложения по умолчанию, задается в секундах. Любое меньшее или равное нулю значение считается как отключенное. Если пользователь отправляет приложение с незаданным значением срока, то оно задается автоматически. Это конфигурация на определенный момент времени. Примечание: время жизни по умолчанию не может превышать максимальное время жизни. Функция применима только для leaf-очереди.

Настройка приоритета приложения

Приоритет приложения работает только совместно с политикой упорядочения по умолчанию *FIFO*.

Приоритет по умолчанию для приложения может быть на уровне кластера и очереди:

- Приоритет на уровне кластера – у любого приложения, отправленного с приоритетом, превышающим приоритет *cluster-max*, происходит сброс приоритета до *cluster-max*. Файл конфигурации для приоритета *cluster-max* – *$HADOOP_HOME/etc/hadoop/yarn-site.xml*. Параметр `yarn.cluster.max-application-priority` определяет максимальный приоритет приложения в кластере.
- Приоритет на уровне leaf-очереди – каждой leaf-очереди предоставляется приоритет администратора по умолчанию. Приоритет очереди по умолчанию используется для любого приложения, отправленного без заданного приоритета. Файл конфигурации для приоритета на уровне очереди – *$HADOOP_HOME/etc/hadoop/capacity-scheduler.xml*. Параметр `yarn.scheduler.capacity.root.<leaf-queue-path>.default-application-priority` определяет приоритет приложения по умолчанию в leaf-очереди.

Important: Приоритет приложения не изменяется при перемещении приложения в другую очередь

Преимущественное право в Capacity Scheduler

CapacityScheduler поддерживает возможность преимущественного права (preemption) контейнера от очередей, чье использование ресурсов превышает их гарантированную производительность. Для этого

следующие параметры конфигурации должны быть включены в *yarn-site.xml*:

`yarn.resourcemanager.scheduler.monitor.enable` – включение набора периодического мониторинга (periodic monitors, указанных в `yarn.resourcemanager.scheduler.monitor.policies`), влияющих на планировщик. Значением по умолчанию является *false*.

`yarn.resourcemanager.scheduler.monitor.policies` – список классов *SchedulingEditPolicy*, взаимодействующих с планировщиком. Настроенные политики должны быть совместимы с планировщиком. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.monitor.capacity.ProportionalCapacityPreem* что совместимо с CapacityScheduler.

Следующие параметры конфигурации могут быть настроены в *yarn-site.xml* для управления преимущественным правом контейнеров, когда класс *ProportionalCapacityPreemptionPolicy* задан для *yarn.resourcemanager.scheduler.monitor.policies*:

`yarn.resourcemanager.monitor.capacity.preemption.observe_only` – если установлено значение *true*, следует запустить политику, но не влиять на кластер событиями preemption и kill. Значением по умолчанию является *false*.

`yarn.resourcemanager.monitor.capacity.preemption.monitoring_interval` – время между вызовами политики *ProportionalCapacityPreemptionPolicy* (в миллисекундах). Значение по умолчанию *3000*.

`yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_kill` – время между запросом preemption из приложения и уничтожением контейнера (в миллисекундах). Значение по умолчанию *15000*.

`yarn.resourcemanager.monitor.capacity.preemption.total_preemption_per_round` – максимальный процент ресурсов для вытеснения по преимущественному праву за один раунд. Управляя этим значением, можно регулировать скорость, с которой контейнеры извлекаются из кластера. После вычисления общего желаемого преимущественного права политика сокращает его в пределах этого лимита. Значение по умолчанию *0.1*.

`yarn.resourcemanager.monitor.capacity.preemption.max_ignored_over_capacity` – максимальное количество ресурсов, превышающих по преимущественному праву заданную пропускную способность. Параметр определяет мертвую зону вокруг назначенной пропускной способности, которая помогает предотвратить колебания вокруг вычисленного заданного баланса. Высокие значения замедляют производительность и (при отсутствии *natural.completions*) могут препятствовать конвергенции к гарантированной производительности. Значение по умолчанию *0.1*.

`yarn.resourcemanager.monitor.capacity.preemption.natural_termination_factor` – учитывая вычисленное заданное преимущественное право, следует учесть контейнеры с истекающим сроком и выгрузить только этот процент дельты. Параметр определяет скорость геометрической конвергенции в мертвую зону (*MAX_IGNORED_OVER_CAPACITY*). Например, фактор высвобождения (termination factor) *0.5* восстанавливает почти *95%* ресурсов в пределах `5 * #WAIT_TIME_BEFORE_KILL`, даже при отсутствии естественного завершения (natural termination). Значение по умолчанию составляет *0.2*.

CapacityScheduler поддерживает следующие конфигурации в *capacity-scheduler.xml* для управления преимущественным правом контейнеров приложений, отправляемых в очередь:

`yarn.scheduler.capacity.<queue-path>.disable_preemption` – конфигурацию можно установить в значение *true* для того, чтобы выборочно отключить преимущественное право контейнеров приложений, отправленных в указанную очередь. Свойство применяется только в том случае, если право preemption в масштабе всей системы включено путем настройки `yarn.resourcemanager.scheduler.monitor.enable` на *true* и `yarn.resourcemanager.scheduler.monitor.policies` на *ProportionalCapacityPreemptionPolicy*. Если данное свойство не установлено, то значение наследуется от родителя очереди. Значением по умолчанию является *false*.

`yarn.scheduler.capacity.<queue-path>.intra-queue-preemption.disable_preemption` – конфигурация может быть установлена в значение *true* для того, чтобы выборочно отключить внутри очереди преимущественное право контейнеров приложений, отправленных в указанную очередь. Свойство применяется только в том случае, если право preemption в масштабе всей системы включено путем настройки `yarn.resourcemanager.scheduler.monitor.enable` в значение

true, `yarn.resourcemanager.scheduler.monitor.policies` на *ProportionalCapacityPreemptionPolicy* и `yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled` в значение *true*. Если данное свойство не установлено, то значение наследуется от родителя очереди. Значением по умолчанию является *false*.

Свойства резервирования

CapacityScheduler поддерживает параметры для управления созданием, удалением, обновлением и списком резервирований. Важно обратить внимание, что любой пользователь может обновлять, удалять или перечислять свои собственные резервирования. Если списки ACL-резервирования включены, но не определены, доступ будет иметь каждый. В приведенных далее примерах *<queue>* – это имя очереди. Например, чтобы настроить ACL для управления резервированиями в очереди по умолчанию, следует использовать свойство `yarn.scheduler.capacity.root.default.acl_administer_reservations`.

`yarn.scheduler.capacity.root.<queue>.acl_administer_reservations` – ACL, который контролирует, кто может управлять резервированием для указанной очереди. Если у данного пользователя/группы есть необходимые ACL в этой очереди, то он/она может отправлять, удалять, обновлять и составлять список всех резервирований. ACL для свойства не наследуются.

`yarn.scheduler.capacity.root.<queue>.acl_list_reservations` – ACL, который контролирует, кто может составлять список резервирований для указанной очереди. Если у данного пользователя/группы есть необходимые ACL в этой очереди, то он/она может составлять список всех приложений. ACL для свойства не наследуются.

`yarn.scheduler.capacity.root.<queue>.acl_submit_reservations` – ACL, который контролирует, кто может отправлять резервирования в указанную очередь. Если у данного пользователя/группы есть необходимые ACL в этой очереди, то он/она может отправлять резервирование. ACL для свойства не наследуются.

Настройка ReservationSystem с помощью CapacityScheduler

CapacityScheduler поддерживает систему **ReservationSystem**, которая позволяет пользователям резервировать ресурсы заблаговременно. Таким образом приложение может запросить зарезервированные ресурсы во время выполнения, указав *reservationId*. Для этого могут быть настроены следующие параметры конфигурации в *yarn-site.xml*:

`yarn.resourcemanager.reservation-system.enable` – обязательный параметр: включить **ReservationSystem** в **ResourceManager**. Значение может быть только логическим (boolean), по умолчанию является *false*, то есть **ReservationSystem** не включена.

`yarn.resourcemanager.reservation-system.class` – необязательный параметр: имя класса **ReservationSystem**. Значение по умолчанию выбирается на основе настроенного планировщика, то есть если настроен **CapacityScheduler**, то классом является *CapacityReservationSystem*.

`yarn.resourcemanager.reservation-system.plan.follower` – необязательный параметр: имя класса *PlanFollower*, который запускается по таймеру и синхронизирует **CapacityScheduler** с *Plan* и наоборот. Значение по умолчанию выбирается на основе настроенного планировщика, то есть если настроен **CapacityScheduler**, то классом является *CapacitySchedulerPlanFollower*.

`yarn.resourcemanager.reservation-system.planfollower.time-step` – необязательный параметр: частота таймера *PlanFollower* (в миллисекундах). Значением по умолчанию является *1000*.

ReservationSystem интегрирована с иерархией очереди **CapacityScheduler** и может быть настроена для любой *LeafQueue*. Для этого в **CapacityScheduler** поддерживаются следующие параметры:

`yarn.scheduler.capacity.<queue-path>.reservable` – обязательный параметр: указывает **ReservationSystem**, что ресурсы очереди доступны для резервирования пользователями. Значение может быть только логическим (boolean), по умолчанию является *false*, то есть резервирование в *LeafQueue* не включено.

`yarn.scheduler.capacity.<queue-path>.reservation-agent` – необязательный параметр: имя класса для использования в целях определения реализации *ReservationAgent*, который принимает попытки разместить запрос пользователя на резервирование в *Plan*. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.reservation.planning.AlignedPlannerWithGreedy*.

`yarn.scheduler.capacity.<queue-path>.reservation-move-on-expiry` – необязательный параметр, который указывает **ReservationSystem**, следует ли перемещать или уничтожать приложения в родительской резервируемой очереди (настроенной выше) по истечении срока действия соответствующего резервирования. Значение может быть только логическим (boolean), по умолчанию является *true*, означающее, что приложение будет перемещено в резервируемую очередь.

`yarn.scheduler.capacity.<queue-path>.show-reservations-as-queues` – необязательный параметр для отображения или скрытия очередей резервирования в пользовательском интерфейсе планировщика. Значение может быть только логическим (boolean), по умолчанию является *false*, то есть очереди резервирования скрываются.

`yarn.scheduler.capacity.<queue-path>.reservation-policy` – необязательный параметр: имя класса для использования в целях определения реализации *SharingPolicy* для проверки новых резервирований на предмет нарушения каких-либо инвариантов. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.reservation.CapacityOverTimePolicy*.

`yarn.scheduler.capacity.<queue-path>.reservation-window` – необязательный параметр, представляющий время в миллисекундах, в течение которого *SharingPolicy* проверяет соблюдение ограничений в *Plan*. Значение по умолчанию составляет один день.

`yarn.scheduler.capacity.<queue-path>.instantaneous-max-capacity` – необязательный параметр: максимальная пропускная способность в процентах в виде числа с плавающей запятой (float), которую *SharingPolicy* позволяет зарезервировать одному пользователю. Значение по умолчанию равно *1*, то есть *100%*.

`yarn.scheduler.capacity.<queue-path>.average-capacity` – необязательный параметр: средняя допустимая пропускная способность, агрегируемая в *ReservationWindow* в процентах в виде числа с плавающей запятой (float), которую *SharingPolicy* позволяет зарезервировать одному пользователю. Значение по умолчанию равно *1*, то есть *100%*.

`yarn.scheduler.capacity.<queue-path>.reservation-planner` – необязательный параметр: имя класса для использования в целях определения реализации *Planner*, вызываемой при падении производительности *Plan* ниже зарезервированных пользователем ресурсов (из-за планового обслуживания или сбоев узла). Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.reservation.planning.SimpleCapacityReplanner*, сканирующее *Plan* и жадно удаляющее резервирования в обратном порядке (*LIFO*) до тех пор, пока зарезервированные ресурсы не оказываются в пределах пропускной способности *Plan*.

`yarn.scheduler.capacity.<queue-path>.reservation-enforcement-window` – необязательный параметр, представляющий время в миллисекундах, в течение которого *Planner* проверяет соблюдение ограничений в *Plan*. Значение по умолчанию составляет один час.

Динамическое автосоздание и управление leaf-очередями

CapacityScheduler поддерживает автоматическое создание наследуемых leaf-очередей, настроенных с включенной данной функцией.

- Настройка при помощи маппинга

user-group queue mapping(s), перечисленные в `yarn.scheduler.capacity.queue-mappings`, должны содержать дополнительный параметр очереди, в которую будет осуществляться автосоздание leaf-очередей. Свойства описаны выше в подразделе “Queue Mapping based on User or Group”. Так же важно обратить внимание, что в таких родительских очередях необходимо включить автосоздание дочерних очередей, как указано далее.

Пример:

```
<property>
   <name>yarn.scheduler.capacity.queue-mappings</name>
   <value>u:user1:queue1,g:group1:queue2,u:user2:%primary_group,u:%user:parent1.%user</value>
   <description>
     Here, u:%user:parent1.%user mapping allows any <user> other than user1,
     user2 to be mapped to its own user specific leaf queue which
     will be auto-created under <parent1>.
   </description>
 </property>
```

- Конфигурация родительской очереди

Функция *Dynamic Queue Auto-Creation and Management* интегрирована с иерархией очереди **CapacityScheduler** и может быть настроена для *ParentQueue* для автоматического создания leaf-очередей. Такие родительские очереди не поддерживают возможность сосуществования автосозданных очередей вместе с другими предварительно сконфигурированными очередями. Свойства:

`yarn.scheduler.capacity.<queue-path>.auto-create-child-queue.enabled` – обязательный параметр: указывает для **CapacityScheduler**, что для заданной родительской очереди необходимо включить автосоздание leaf-очереди. Значение может быть только логическим (boolean), по умолчанию является *false*, то есть автосоздание leaf-очереди в *ParentQueue* не включено.

`yarn.scheduler.capacity.<queue-path>.auto-create-child-queue.management-policy` – необязательный параметр: имя класса для использования с целью определения реализации *AutoCreatedQueueManagementPolicy*, которая динамически управляет leaf-очередями и их производительностью в данной родительской очереди. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.queuemanagement.GuaranteedOrZeroCapacityOverTimePolicy*. Пользователи или группы могут отправлять приложения в автосозданные leaf-очереди в течение ограниченного времени и прекращать их использование. Следовательно, число leaf-очередей, автосозданных в родительской очереди, может быть больше, чем ее гарантированная пропускная способность. Текущая реализация политики позволяет либо настроить, либо обнулить производительность, исходя из доступности пропускной способности в родительской очереди и порядка отправки приложения через leaf-очереди.

- Настройка при помощи CapacityScheduler

Родительская очередь для автосоздания leaf-очередей поддерживает настройку параметров их шаблона. Автосозданные очереди поддерживают все параметры конфигурации leaf-очереди, за исключением *Queue ACL*, *Absolute Resource*. Списки ACL очереди в настоящее время не настраиваются в шаблоне, но наследуются от родительской очереди. Свойства:

`yarn.scheduler.capacity.<queue-path>.leaf-queue-template.capacity` – обязательный параметр: указывает минимальную гарантированную пропускную способность для автосоздаваемых leaf-очередей. В настоящее время конфигурации Absolute Resource не поддерживаются в автоматически созданных leaf-очередях.

`yarn.scheduler.capacity.<queue-path>.leaf-queue-template.<leaf-queue-property>` – необязательный параметр: для других параметров очереди, которые могут быть настроены в автосоздаваемых leaf-очередях, таких как *maximum-capacity*, *user-limit-factor*, *maximum-am-resource-percent* и прочие (*Свойства очереди*).

Пример:

```
<property>
   <name>yarn.scheduler.capacity.root.parent1.auto-create-child-queue.enabled</name>
   <value>true</value>
 </property>
 <property>
    <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.capacity</name>
    <value>5</value>
 </property>
```

```
<property>
   <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.maximum-capacity</name>
   <value>100</value>
</property>
<property>
   <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.user-limit-factor</name>
   <value>3.0</value>
</property>
<property>
   <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.ordering-policy</name>
   <value>fair</value>
</property>
<property>
   <name>yarn.scheduler.capacity.root.parent1.GPU.capacity</name>
   <value>50</value>
</property>
<property>
    <name>yarn.scheduler.capacity.root.parent1.accessible-node-labels</name>
    <value>GPU,SSD</value>
  </property>
<property>
    <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.accessible-node-labels</name>
    <value>GPU</value>
 </property>
<property>
   <name>yarn.scheduler.capacity.root.parent1.leaf-queue-template.accessible-node-labels.GPU.capacity</
↪name>
   <value>5</value>
</property>
```

- Управление конфигурацией Scheduling Edit Policy

Администраторы должны указать дополнительную политику редактирования *org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.QueueManagementDynamicEditPolicy* со списком текущих политик в виде строки и разделенные запятыми в конфигурации `yarn.resourcemanager.scheduler.monitor.policies`.

`yarn.resourcemanager.monitor.capacity.queue-management.monitoring-interval` – время между вызовами политики *QueueManagementDynamicEditPolicy* (в миллисекундах). Значение по умолчанию *1500*.

Другие свойства

- Калькулятор ресурсов:

`yarn.scheduler.capacity.resource-calculator` – реализация **ResourceCalculator** для использования в целях сравнения ресурсов в планировщике. По умолчанию, *org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator*, используется только память, тогда как **DominantResourceCalculator** использует *Dominant-resource* для сравнения многомерных ресурсов, таких как память, процессор и пр. Значением должно быть имя класса Java ResourceCalculator.

- Расположение данных

Capacity Scheduler использует **Delay Scheduling** для соблюдения ограничений месторасположения задач. Существует 3 уровня: node-local, rack-local и off-switch. Планировщик учитывает количество упущенных возможностей, когда локальность не может быть удовлетворена, и ждет, пока это число достигнет порогового значения, прежде чем ослабить ограничение положения до следующего уровня. Порог можно настроить в следующих свойствах:

`yarn.scheduler.capacity.node-locality-delay` – число упущенных возможностей, после которых

CapacityScheduler пытается запланировать rack-local контейнеры. Как правило, значение должно быть установлено на количество узлов в кластере. По умолчанию устанавливается приблизительное количество узлов в одной стойке, которое составляет *40*. Должно быть положительное целое число.

`yarn.scheduler.capacity.rack-locality-additional-delay` – число дополнительных упущенных возможностей относительно node-locality-delay, после чего **CapacityScheduler** пытается запланировать off-switch контейнеры. По умолчанию значение равно *-1*, тогда в этом случае количество упущенных возможностей для назначения off-switch контейнеров рассчитывается по формуле `L * C / N`, где L – количество мест (узлов или стоек), указанных в запрос ресурса, C – количество запрошенных контейнеров, а N – размер кластера.

Важно обратить внимание, что эту функцию следует отключить, если *YARN* развертывается отдельно от файловой системы, поскольку локальность в таком случае не имеет смысла. Для этого необходимо установить `yarn.scheduler.capacity.node-locality-delay` в значение *-1*, тогда ограничение на местоположение запроса игнорируется.

- Распределение контейнеров по NodeManager Heartbeat:

`yarn.scheduler.capacity.per-node-heartbeat.multiple-assignments-enabled` – допуск контейнеров нескольких назначений в одном heartbeat-сообщении NodeManager. По умолчанию устанавливается значение *true*.

`yarn.scheduler.capacity.per-node-heartbeat.maximum-container-assignments` – максимальное количество контейнеров, которое может быть назначено в одном heartbeat-сообщении NodeManager при заданном параметре multiple-assignments-enabled на *true*. Значение по умолчанию равно *100*, что ограничивает максимальное количество назначений контейнеров от 1 до 100. Установка значения в *-1* отключает ограничение.

`yarn.scheduler.capacity.per-node-heartbeat.maximum-offswitch-assignments` – максимальное количество off-switch контейнеров, которое может быть назначено в одном heartbeat-сообщении NodeManager при заданном параметре multiple-assignments-enabled на *true*. Значение по умолчанию равно *1*, что означает выделение только одного off-switch на heartbeat-сообщение.

Проверка конфигурации CapacityScheduler

Конфигурацию **CapacityScheduler** можно проверить после завершения установки и настройки путем запуска кластера *YARN* через веб-интерфейс:

- Запустить кластер YARN обычным способом;
- Открыть веб-интерфейс ResourceManager;
- Веб-страница */scheduler* должна показывать использование ресурсов отдельными очередями.

4.3.3 Изменение конфигурации очереди

Изменение конфигурации очереди через файл осуществляется путем редактирования *conf/capacity-scheduler.xml* и запуска `yarn rmadmin -refreshQueues`:

```
$ vi $HADOOP_CONF_DIR/capacity-scheduler.xml
$ $HADOOP_YARN_HOME/bin/yarn rmadmin -refreshQueues
```

Удаление очереди через файл реализуется в два шага:

- Остановка очереди: перед удалением leaf-очереди она не должна иметь запущенных/ожидающих приложений и должна быть в статусе STOPPED путем изменения `yarn.scheduler.capacity.<queue-path>.state` (*Администрирование и разрешения очереди*). Перед удалением родительской очереди все ее дочерние очереди не должны иметь запущенных/ожидающих приложений и должны быть в статусе STOPPED. Родительская очередь также должна быть STOPPED;
- Удаление очереди: удалить конфигурации очереди из файла и запустить обновление.

Изменение конфигурации очереди через API осуществляется путем использования резервного хранилища для конфигурации планировщика. Для этого могут быть настроены параметры в *yarn-site.xml*:

`yarn.scheduler.configuration.store.class` – тип используемого резервного хранилища;

`yarn.scheduler.configuration.mutation.acl-policy.class` – политика ACL может быть настроена для ограничения того, какие пользователи могут изменять какие очереди. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.scheduler.DefaultConfigurationMutationACLPolicy*, что позволяет только YARN-администраторам вносить изменения в конфигурацию. Другим значением является *org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.conf.QueueAdminConfigurationMutationACLPolicy*, что позволяет вносить изменения очереди только в том случае, если вызывающий объект является администратором очереди;

`yarn.scheduler.configuration.store.max-logs` – изменения конфигурации регистрируются в бэк-хранилище, если используется leveldb или zookeeper. Эта конфигурация контролирует максимальное количество журналов аудита для хранения, удаляя старые журналы при превышении значения. По умолчанию *1000*;

`yarn.scheduler.configuration.leveldb-store.path` – путь к хранилищу конфигурации при использовании leveldb. Значением по умолчанию является *${hadoop.tmp.dir}/yarn/system/confstore*;

`yarn.scheduler.configuration.leveldb-store.compaction-interval-secs` – интервал сжатия конфигурации при использовании leveldb (в секундах). Значение по умолчанию *86400* (один день);

`yarn.scheduler.configuration.zk-store.parent-path` – путь к root-узлу zookeeper для хранения связанной с конфигурацией информации при использовании zookeeper. Значением по умолчанию является */confstore*.

При включении конфигурации планировщика через `yarn.scheduler.configuration.store.class`, отключается *yarn rmadmin -refreshQueues*, то есть исключается возможность обновления конфигурации через файл.

Important: Функция изменения конфигурации очереди через API находится в альфа-фазе и может быть изменена

4.3.4 Обновление контейнера

Экспериментально. API может измениться в будущем

Как только **Application Master** получает контейнер от **Resource Manager**, Master может запросить у Manager обновить некоторые атрибуты контейнера. В настоящее время поддерживаются только два типа обновлений контейнера:

- Resource Update: когда Application Master может запросить Resource Manager обновить ресурсный размер контейнера. Например: изменить контейнер *2GB, 2 vcore* на контейнер *4GB, 2 vcore*.
- ExecutionType Update: когда Application Master может запросить Resource Manager обновить ExecutionType контейнера. Например: изменить тип выполнения с *GUARANTEED* на *OPPORTUNISTIC* или наоборот.

Этому способствует **Application Master**, заполняющий поле *updated_containers*, представляющее собой список типа *UpdateContainerRequestProto* в *AllocateRequestProto*. Master может сделать несколько запросов на обновление контейнера в одном вызове.

Схема *UpdateContainerRequestProto* выглядит следующим образом:

```
message UpdateContainerRequestProto {
  required int32 container_version = 1;
  required ContainerIdProto container_id = 2;
```

```
  required ContainerUpdateTypeProto update_type = 3;
  optional ResourceProto capability = 4;
  optional ExecutionTypeProto execution_type = 5;
}
```

ContainerUpdateTypeProto является перечислением:

```
enum ContainerUpdateTypeProto {
  INCREASE_RESOURCE = 0;
  DECREASE_RESOURCE = 1;
  PROMOTE_EXECUTION_TYPE = 2;
  DEMOTE_EXECUTION_TYPE = 3;
}
```

В соответствии с приведенным перечислением планировщик в настоящее время поддерживает изменение типа обновлений контейнера Resource Update либо ExecutionType Update в одном запросе.

Application Master также должен предоставить последнюю версию *ContainerProto*, полученную от **Resource Manager** – это контейнер, который Manager запрашивает на обновление.

Если **Resource Manager** может обновить запрошенный контейнер, то тогда обновленный контейнер возвращается в поле списка *updated_containers* типа *UpdatedContainerProto* в возвращаемом значении *AllocateResponseProto* того же самого вызова или одного из последующих.

Схема *UpdatedContainerProto* выглядит следующим образом:

```
message UpdatedContainerProto {
  required ContainerUpdateTypeProto update_type = 1;
  required ContainerProto container = 2;
}
```

Здесь указывается тип выполненного обновления для контейнера и объект обновленного контейнера, содержащий обновленный токен.

Затем токен контейнера может использоваться **Application Master** для запроса соответствующего **NodeManager** либо запуска контейнера, если он еще не запущен, либо для обновления контейнера с использованием обновленного токена.

Обновления контейнера *DECREASE_RESOURCE* и *DEMOTE_EXECUTION_TYPE* выполняются автоматически – **Application Master** не должен явно запрашивать **NodeManager**, чтобы уменьшить ресурсы контейнера. Другие типы обновлений требуют, чтобы Master явно запрашивал об обновлении.

Если для параметра конфигурации `yarn.resourcemanager.auto-update.containers` задано значение *true* (по умолчанию *false*), **Resource Manager** обеспечивает автоматическое обновление всех контейнеров.

4.4 Hadoop: Fair Scheduler

В главе описывается **FairScheduler** – подключаемый планировщик для **Hadoop**, позволяющий YARN-приложениям справедливо распределять ресурсы в больших кластерах.

Справедливое планирование – это метод распределения ресурсов между приложениями таким образом, чтобы все приложения в среднем получали равную долю ресурсов с течением времени. **Hadoop NextGen** способен планировать несколько типов ресурсов. По умолчанию в **FairScheduler** планирование решений основывается только на памяти. Но он также может быть сконфигурирован для планирования, основанного на памяти вместе с процессором, используя понятие Dominant Resource Fairness, разработанное **Ghodsi et al**.

Когда запущено одно приложение, оно использует весь кластер. А при добавлении новых приложений, им назначаются освобождающиеся ресурсы, таким образом каждое приложение в конечном итоге получает

примерно одинаковый объем ресурсов. В отличие от стандартного планировщика **Hadoop**, который формирует очередь приложений, **FairScheduler** позволяет коротким приложениям завершать работу в разумные сроки, не оставляя ждать при этом долговременные приложения. Это также разумный способ разделить кластер между несколькими пользователями. Наконец, справедливое распределение может также работать с приоритетностью приложений – приоритеты используются в качестве веса для определения доли ресурсов от их совокупности, которую должны получить приложения.

Далее планировщик организует приложения в "очереди" и справедливо распределяет ресурсы между этими очередями. По умолчанию все пользователи имеют общую очередь с именем *default*. Если приложение специально указывает очередь в запросе ресурса контейнера, запрос отправляется в эту очередь. Также можно назначать очереди на основе имени пользователя, включенного в запрос, через конфигурацию. В каждой очереди имеется политика планирования для совместного использования ресурсов между запущенными приложениями. По умолчанию устанавливается распределение ресурсов на основе памяти, но также могут быть настроены *FIFO* и мультиресурсность с Dominant Resource Fairness. Очереди могут быть организованы в иерархию для разделения ресурсов и настроены с весами для совместного использования кластера в определенных пропорциях.

Кроме этого, **FairScheduler** позволяет назначать гарантированные минимальные доли в очереди, что полезно для обеспечения определенных пользователей, групп или рабочих приложений достаточными ресурсами. Когда очередь содержит приложения, она получает по крайней мере свою минимальную долю, но когда очередь не нуждается в полной гарантированной доле, избыток распределяется между другими запущенными приложениями. Это позволяет планировщику гарантировать пропускную способность очередей при эффективном использовании ресурсов, когда эти очереди не содержат приложений.

FairScheduler позволяет запускать все приложения по умолчанию, но также через файл конфигурации можно ограничить количество запущенных приложений на пользователя и на очередь. Это может быть полезно, когда пользователь должен одновременно отправить сотни приложений, или в целом для повышения производительности, если запуск слишком большого количества приложений может привести к созданию слишком большого объема промежуточных данных или слишком частому переключению контекста. Ограничение приложений не приводит к сбою каких-либо последующих отправленных приложений, а только к ожиданию в очереди планировщика, пока не завершатся более ранние приложения пользователя.

4.4.1 Иерархия очередей и политика

FairScheduler поддерживает иерархию очередей. Все очереди происходят из очереди с именем *root*. Доступные ресурсы распределяются между дочерними элементами root-очереди типичным способом справедливого планирования. Затем дочерние очереди таким же образом распределяют выделенные им ресурсы по своим дочерним очередям. Приложения могут быть запланированы только в leaf-очередях. Очереди можно указывать как дочерние элементы других очередей, помещая их как подэлементы в файл распределения.

Имя очереди начинается с перечисления имен ее родителей с точками в качестве разделителей. Таким образом, очередь с именем *queue1* в root-очереди называется "root.queue1", а очередь с именем *queue2* в очереди с именем "parent1" называется "root.parent1.queue2". При обращении к очередям часть имени *root* необязательна, поэтому *queue1* может называться просто "queue1", а *queue2* – "parent1.queue2".

Кроме того, **FairScheduler** позволяет устанавливать различные индивидуальные политики для каждой очереди, чтобы разрешить совместное использование ресурсов любым удобным для пользователя способом. Политика может быть построена путем расширения `org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.SchedulingPolicy`. *FifoPolicy*, *FairSharePolicy* (по умолчанию) и *DominantResourceFairnessPolicy* являются встроенными и могут быть легко использованы.

Некоторые дополнения еще не поддерживаются в оригинальном (**MR1**) **Fair Scheduler**. Среди них – использование индивидуальных политик, управляющих приоритетом "boosting" над определенными приложениями.

4.4.2 Размещение приложений в очередях

Планировщик **Fair Scheduler** позволяет администраторам настраивать политики, которые автоматически помещают отправленные приложения в соответствующие очереди. Размещение может зависеть от пользователя и групп отправителя и запрашиваемой очереди. Политика состоит из набора правил, которые применяются последовательно для классификации входящего приложения. Каждое правило либо помещает приложение в очередь, либо отклоняет его, либо переходит к следующему правилу. Далее в документации приведен *Формат файла распределения* для настройки этих политик.

4.4.3 Установка

Для использования **Fair Scheduler** сначала необходимо назначить соответствующий класс планировщика в *yarn-site.xml*:

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
```

4.4.4 Конфигурация

Настройка планировщика **Fair Scheduler** обычно включает в себя изменение двух файлов. Во-первых, можно настроить параметры всего планировщика, добавив свойства конфигурации в файл *yarn-site.xml* в существующую директорию. Во-вторых, в большинстве случаев пользователи желают создать список файлов распределения, в котором указываются существующие очереди и соответствующий им вес и пропускная способность. Файл распределения перезагружается каждые *10* секунд, позволяя вносить изменения на лету.

Свойства в файле yarn-site.xml

`yarn.scheduler.fair.allocation.file` – путь к файлу распределения. Файл распределения – это XML-манифест, описывающий очереди и их свойства в дополнение к определенным параметрам политики по умолчанию. Этот файл должен быть в формате XML, описанном в следующем параграфе. Если указан относительный путь, то поиск файла осуществляется по classpath (который обычно включает в себя каталог *Hadoop conf*). По умолчанию используется *fair-scheduler.xml*.

`yarn.scheduler.fair.user-as-default-queue` – определяет, следует ли использовать имя пользователя, связанное с распределением, в качестве имени очереди по умолчанию, если другого не указано. Если для параметра установлено значение *false* или не задано вовсе, все задачи имеют общую очередь по умолчанию с именем “default”. По умолчанию значение параметра устанавливается на *true*. Если в файле распределения указывается политика размещения в очереди, то данное свойство игнорируется.

`yarn.scheduler.fair.preemption` – определяет, следует ли использовать преимущественное право preemption. По умолчанию устанавливается на *false*.

`yarn.scheduler.fair.preemption.cluster-utilization-threshold` – порог использования, после которого вступает в действие преимущественное право preemption. Вычисляется как максимальное отношение использования к пропускной способности среди всех ресурсов. По умолчанию задается *0,8f*.

`yarn.scheduler.fair.sizebasedweight` – определяет, следует ли назначать общие ресурсы отдельным приложениям, основываясь на их размере, вместо того, чтобы предоставлять равные ресурсы всем приложениям независимо от их размера. При значении *true* приложения взвешиваются по натуральному логарифму – единица плюс вся запрашиваемая память приложения, поделенная на натуральный логарифм *2*. По умолчанию значение *false*.

`yarn.scheduler.fair.assignmultiple` – определяет, разрешить ли назначение нескольких контейнеров в одном heartbeat-сообщении. По умолчанию *false*.

`yarn.scheduler.fair.dynamic.max.assign` – устанавливает, следует ли динамически определять количество ресурсов, которое может быть назначено за одно heartbeat-сообщение, если для атрибута *assignmultiple* задано значение *true*. При включенном параметре около половины нераспределенных ресурсов на узле распределяются по контейнерам за одинарное heartbeat-сообщение. По умолчанию *true*.

`yarn.scheduler.fair.max.assign` – максимальное количество контейнеров, которое может быть назначено за одно heartbeat-сообщение, при условии: значение *assignmultiple* задано *true*, а для *dynamic.max.assign* равно *false*. По умолчанию параметр устанавливается в *-1*, что не задает никаких ограничений.

`yarn.scheduler.fair.locality.threshold.node` – число возможностей планирования для приложений, которые запрашивают контейнеры на определенных узлах, с момента последнего назначения контейнера в ожидании перед принятием размещения на другом узле. Выражается в виде числа с плавающей запятой (float) от *0* до *1*, которое в виде доли от размера кластера представляет собой количество возможностей планирования, которые необходимо упустить. Значение по умолчанию *-1.0* означает, что никаких возможностей планирования упускаться не будет.

`yarn.scheduler.fair.locality.threshold.rack` – число возможностей планирования для приложений, запрашивающих контейнеры на определенных стойках, с момента последнего назначения контейнера для ожидания перед принятием размещения на другой стойке. Выражается в виде числа с плавающей запятой (float) от *0* до *1*, которое в виде доли от размера кластера представляет собой количество возможностей планирования, которые необходимо пропустить. Значение по умолчанию *-1.0* означает, что никаких возможностей планирования упускаться не будет.

`yarn.scheduler.fair.allow-undeclared-pools` – если параметр установлен на *true*, то во время отправки приложения могут быть созданы новые очереди, так как они указаны отправителем в качестве очереди приложения либо благодаря свойству *user-as-default-queue property*. Если значение установлено на *false*, то каждый раз, когда приложение помещается в очередь, которая не определена в файле распределения, оно помещается в очередь *default*. По умолчанию параметр установлен на *true*. Если в файле распределения указывается политика размещения в очереди, то данное свойство игнорируется.

`yarn.scheduler.fair.update-interval-ms` – интервал, в течение которого можно заблокировать планировщик и пересчитать справедливые доли и спрос и проверить, не требуется ли что-либо для преимущественного права preemption. По умолчанию устанавливается *500 мс*.

`yarn.resource-types.memory-mb.increment-allocation` – инкремент памяти. Если отправить задачу с запросом ресурса, не кратным параметру, запрос округляется до ближайшего инкремента. По умолчанию *1024 МБ*.

`yarn.resource-types.vcores.increment-allocation` – инкремент vcores. Если отправить задачу с запросом ресурса, не кратным параметру, запрос округляется до ближайшего инкремента. По умолчанию *1*.

`yarn.resource-types.<resource>.increment-allocation` – инкремент <resource>. Если отправить задачу с запросом ресурса, не кратным параметру, запрос округляется до ближайшего инкремента. Если свойство не указано для ресурса, округление не применяется. Если единица измерения не указана, принимается единица измерения ресурса по умолчанию.

`yarn.scheduler.increment-allocation-mb` – инкремент памяти. По умолчанию *1024 МБ*. Вместо данного параметра предпочитается `yarn.resource-types.memory-mb.increment-allocation`.

`yarn.scheduler.increment-allocation-vcores` – инкремент CPU vcores. По умолчанию *1*. Вместо данного параметра предпочитается `yarn.resource-types.vcores.increment-allocation`.

Формат файла распределения

Файл распределения должен быть в формате XML.

Элементы очереди. Элементы очереди могут принимать необязательный атрибут *type*, который при установке на *parent* делает очередь родительской. Это полезно в случаях, когда необходимо создать

родительскую очередь без настройки каких-либо leaf-очередей. Каждый элемент очереди может содержать следующие свойства:

- *minResources*: минимальные ресурсы, на которые имеет право очередь, в форме “X mb, Y vcores”. При политике *single-resource* значение vcores игнорируется. Если минимальная общая доля очереди не удовлетворяется, ей предлагаются доступные ресурсы прежде, чем любой другой очереди с тем же родителем. В соответствии с политикой *single-resource* очередь считается неудовлетворенной, если ее использование памяти ниже минимального совместно используемого объема памяти. В соответствии с принципом *dominant resource* очередь считается неудовлетворенной, если ее использование в качестве основного ресурса относительно производительности кластера ниже минимальной доли для этого ресурса. Если в этой ситуации не удовлетворяется несколько очередей, ресурсы попадают в очередь с наименьшим соотношением между соответствующим использованием ресурсов и минимальным. Важно обратить внимание, что существует вероятность того, что очередь, которая находится ниже своего минимума, может не сразу достичь этого минимума при отправке приложения, поскольку ресурсы могут использоваться уже выполняющимися заданиями.

- *maxResources*: максимальное количество ресурсов, выделяемых очереди, выраженное либо в абсолютных значениях (“X mb, Y vcores”), либо в процентах от ресурсов кластера (“X% memory, Y% cpu”). Очередь не назначается контейнеру, превышающему данный предел ее совокупного использования.

- *maxChildResources*: максимальное количество ресурсов, выделяемых специально для дочерней очереди, выраженное либо в абсолютных значениях (“X mb, Y vcores”), либо в процентах от ресурсов кластера (“X% memory, Y% cpu”). Очередь не назначается контейнеру, превышающему данный предел ее совокупного использования.

- *maxRunningApps*: лимит на количество приложений из очереди для одновременного запуска.

- *maxAMShare*: лимит на долю очереди, который может быть использован для запуска Application Masters. Свойство можно использовать только для leaf-очередей. Например, если задается значение *1.0f*, то Masters в leaf-очереди могут занимать до *100%* от общей доли памяти и ЦПУ. Значение *-1.0f* отключает функцию, и тогда amShare не проверяется. Значением по умолчанию является *0,5f*.

- *weight*: возможность делить кластер непропорционально с другими очередями. Вес очереди задается по умолчанию *1*, в таком случае очередь с установленным весом *2* получает примерно в два раза больше ресурсов.

- *schedulingPolicy*: установка политики планирования для любой очереди. Допустимыми значениями являются: *fifo* / *fair* / *drf* или любой класс, который расширяет *org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.SchedulingPolicy*. Значение по умолчанию *fair*. Если устанавливается *fifo*, приложениям с более ранним временем отправки отдается предпочтение для контейнеров, но при этом одновременно могут выполняться отправленные позже приложения при условии наличия пространства в кластере после удовлетворения запросов ранних приложений.

- *aclSubmitApps*: список пользователей и/или групп, которые могут отправлять приложения в очередь. Более подробно о формате списка и как работают ACL очереди приведено в *Списки контроля доступа к очереди*.

- *aclAdministerApps*: список пользователей и/или групп, которые могут управлять очередью. В настоящее время единственным административным действием является уничтожение приложения. Более подробно о формате списка и как работают ACL очереди приведено в *Списки контроля доступа к очереди*.

- *minSharePreemptionTimeout*: количество секунд, в течение которых очередь находится под минимальным общим ресурсом, прежде чем предпринять попытки зарезервировать контейнеры для получения ресурсов из других очередей. Если не установлено, очередь наследует значение от своей родительской очереди. Значением по умолчанию является *Long.MAX_VALUE*, что означает, что контейнеры резервироваться не будут, пока не будет задано полноценное значение параметра.

- *fairSharePreemptionTimeout*: количество секунд, в течение которых очередь находится ниже порогового значения fairShare, прежде чем предпринять попытки зарезервировать контейнеры для получения ресурсов из других очередей. Если не установлено, очередь наследует значение от своей родительской очереди.

Значением по умолчанию является *Long.MAX_ VALUE*, что означает, что контейнеры резервироваться не будут, пока не будет задано полноценное значение параметра.

- *fairSharePreemptionThreshold*: порог преимущественного права fairShare для очереди. Если очередь ожидает *fairSharePreemptionTimeout*, не получая ресурсы fairSharePreemptionThreshold*fairShare, то допускается резервирование контейнеров для получения ресурсов из других очередей. Если не установлено, очередь наследует значение от своей родительской очереди. По умолчанию задается *0.5f*.
- *allowPreemptionFrom*: определяет, разрешено ли планировщику резервировать ресурсы из очереди. По умолчанию устанавливается *true*. При значении *false* свойство рекурсивно применяется ко всем дочерним очередям.
- *reservation*: указывает ReservationSystem, что ресурсы очереди доступны для бронирования пользователями. Относится только к leaf-очередям. При этом leaf-очередь не может быть забронирована, если свойство не настроено.

Элементы пользователя. Представляют собой настройки, управляющие поведением отдельных пользователей. Они могут содержать одно свойство: *maxRunningApps* – ограничение количества запущенных приложений для конкретного пользователя.

Элемент userMaxAppsDefault. Устанавливает лимит запуска приложения по умолчанию для всех пользователей, у которых не указано иное ограничение.

Элемент defaultFairSharePreemptionTimeout. Задает тайм-аут преимущественного права preemption для root-очереди. Переопределяется элементом *fairSharePreemptionTimeout* в root-очереди. По умолчанию значение *Long.MAX_ VALUE*.

Элемент defaultMinSharePreemptionTimeout. Устанавливает минимальное время ожидания преимущественного права preemption для root-очереди. Переопределяется элементом *minSharePreemptionTimeout* в root-очереди. По умолчанию значение *Long.MAX_ VALUE*.

Элемент defaultFairSharePreemptionThreshold. Задает порог преимущественного права для root-очереди. Переопределяется элементом *fairSharePreemptionThreshold* в root-очереди. По умолчанию значение *0.5f*.

Элемент queueMaxAppsDefault. Устанавливает ограничение по умолчанию для запущенного приложения для очередей. Переопределяется элементом *maxRunningApps* в каждой очереди.

Элемент queueMaxResourcesDefault. Задает максимальный лимит ресурсов по умолчанию для очереди. Переопределяется элементом *maxResources* в каждой очереди.

Элемент queueMaxAMShareDefault. Устанавливает ограничение ресурса Application Master по умолчанию для очереди. Переопределяется элементом *maxAMShare* в каждой очереди.

Элемент defaultQueueSchedulingPolicy. Устанавливает политику планирования по умолчанию для очередей. Переопределяется элементом *schedulingPolicy* в каждой очереди, если указан. По умолчанию *fair*.

Элемент reservation-agent. Задает имя класса для реализации ReservationAgent, который пытается разместить запрос пользователя на резервирование в Plan. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.reservation.planning.AlignedPlannerWithGreedy*.

Элемент reservation-policy. Задает имя класса реализации SharingPolicy, который проверяет, не нарушает ли новое резервирование какие-либо инварианты. Значением по умолчанию является *org.apache.hadoop.yarn.server.resourcemanager.reservation.CapacityOverTimePolicy*.

Элемент reservation-planner. Задает имя класса для реализации Planner, который вызывается, если пропускная способность Plan падает ниже зарезервированных пользователем ресурсов (из-за планового обслуживания или отказов узла). Значение по умолчанию *org.apache.hadoop.yarn.server.resourcemanager.reservation.planning.SimpleCapacityReplanner*, что приводит к сканированию Plan и жадному удалению резервирований в обратном порядке (*LIFO*) до тех пор, пока зарезервированные ресурсы не оказываются в пределах производительности Plan.

Элемент queuePlacementPolicy. Содержит список элементов правил, которые сообщают планировщику, как в очередях размещать входящие приложения. Правила применяются в том порядке, в котором они перечислены. Правила могут принимать аргументы. Все правила принимают аргумент *create*, который указывает, может ли правило создавать новую очередь. *Create* по умолчанию имеет значение *true*. Если установлено значение *false* и правило помещает приложение в очередь, которая не настроена в файле распределения, осуществляется переход к следующему правилу. Последнее правило должно быть заключительным, не вызывающим продолжения. Допустимые правила:

- **specified**: приложение помещается в запрашиваемую очередь. Допустимо, если приложение не запрашивает никакой очереди, то есть значение *default*. Если приложение запрашивает имя очереди, начинающееся или заканчивающееся точкой, то есть такие имена, как ".q1" или "q1.", запрос отклоняется.
- **user**: приложение помещается в очередь с именем отправившего его пользователя. Знак точки в имени пользователя заменяется на `_dot_`, то есть, например, "first.last" превращается в "first_dot_last".
- **primaryGroup**: приложение помещается в очередь с именем основной группы отправившего его пользователя. Знак точки в имени группы заменяется на `_dot_`, то есть, например, "one.two" превращается в "one_dot_two".
- **secondaryGroupExistingQueue**: приложение помещается в очередь с именем вторичной группы отправившего его пользователя. Выбирается первая вторичная группа, соответствующая настроенной очереди. Знак точки в имени группы заменяется на `_dot_`, то есть, например, пользователь с "one.two" в качестве одной из его вторичных групп помещается в очередь "one_dot_two", если такая очередь существует.
- **nestedUserQueue**: приложение помещается в очередь с именем пользователя под очередью, предложенной вложенным правилом. Похоже на правило *user*, различие заключается в том, что при *nestedUserQueue* пользовательские очереди могут создаваться в любой родительской очереди, в то время как правило *user* создает пользовательские очереди только в root-очереди. Важно обратить внимание, что правило *nestedUserQueue* применяется только в том случае, если вложенное правило возвращает родительскую очередь. Поэтому можно настроить родительскую очередь, установив атрибут *type* для очереди *parent* либо настроив по крайней мере одну leaf-очередь, что сделает ее родительской (приведено далее в примере распределенного файла).
- **default**: приложение помещается в очередь, указанную в правиле по умолчанию в атрибуте *queue*. Если атрибут не указан, приложение помещается в очередь *root.default*.
- **reject**: приложение отклоняется.

Пример распределенного файла:

```
<?xml version="1.0"?>
<allocations>
  <queue name="sample_queue">
    <minResources>10000 mb,0vcores</minResources>
    <maxResources>90000 mb,0vcores</maxResources>
    <maxRunningApps>50</maxRunningApps>
    <maxAMShare>0.1</maxAMShare>
    <weight>2.0</weight>
    <schedulingPolicy>fair</schedulingPolicy>
    <queue name="sample_sub_queue">
      <aclSubmitApps>charlie</aclSubmitApps>
      <minResources>5000 mb,0vcores</minResources>
    </queue>
    <queue name="sample_reservable_queue">
      <reservation></reservation>
    </queue>
  </queue>
```

```
  <queueMaxAMShareDefault>0.5</queueMaxAMShareDefault>
  <queueMaxResourcesDefault>40000 mb,0vcores</queueMaxResourcesDefault>

  <!-- Queue 'secondary_group_queue' is a parent queue and may have
       user queues under it -->
  <queue name="secondary_group_queue" type="parent">
  <weight>3.0</weight>
  <maxChildResources>4096 mb,4vcores</maxChildResources>
  </queue>

  <user name="sample_user">
    <maxRunningApps>30</maxRunningApps>
  </user>
  <userMaxAppsDefault>5</userMaxAppsDefault>

  <queuePlacementPolicy>
    <rule name="specified" />
    <rule name="primaryGroup" create="false" />
    <rule name="nestedUserQueue">
        <rule name="secondaryGroupExistingQueue" create="false" />
    </rule>
    <rule name="default" queue="sample_queue"/>
  </queuePlacementPolicy>
</allocations>
```

Important: Для обратной совместимости с исходным FairScheduler элементы *queue* могут быть названы как элементы *pool*

Списки контроля доступа к очереди

Списки контроля доступа к очереди (Queue Access Control Lists, Queue ACL) позволяют администраторам контролировать, кто может выполнять действия в определенных очередях. Они настраиваются с помощью свойств *aclSubmitApps* и *aclAdministerApps*, которые можно установить для каждой очереди. В настоящее время единственным поддерживаемым административным действием является уничтожение приложения. Администратор также может отправлять приложения на уничтожение. Свойства принимают значения в формате “user1,user2 group1,group2” или ” group1,group2” (с учетом пробела). Действия в очереди разрешены, если пользователь/группа является членом Queue ACL самой очереди или любой из ее родителей. Таким образом, если *queue2* находится в *queue1*, а *user1* находится в ACL *queue1*, а *user2* находится в ACL *queue2*, тогда оба пользователя могут отправиться в *queue2*.

Important: Пробел является разделителем. Для того, чтобы указать только группы ACL, значение должно начинаться с символа пробела

Important: По умолчанию списки ACL для root-очереди имеют значение *, что означает, что по причине того, что списки ACL передаются, любой пользователь может отправлять и уничтожать приложения из любой очереди. Для ограничения доступа необходимо изменить ACL root-очереди на что-то отличное от указанного символа

Списки контроля доступа к резервированию

Списки контроля доступа к резервированию (Reservation Access Control Lists, Reservation ACL) позволяют администраторам контролировать, кто может выполнять действия по резервированию в определенных очередях. Они настраиваются с помощью свойств *aclAdministerReservations*, *aclListReservations* и *aclSubmitReservations*, которые можно установить для каждой очереди. В настоящее время поддерживаемые административные действия – это обновление и удаление резервирований. Администратор также может отправлять и перечислять все резервирования в очереди. Свойства принимают значения в формате “user1,user2 group1,group2” или ” group1,group2” (с учетом пробела). Действия в очереди разрешены, если пользователь/группа является членом Reservation ACL. Важно обратить внимание, что любой пользователь может обновлять, удалять или перечислять свои собственные резервирования.

Important: Если Reservation ACL включены, но не определены, доступ будет иметь каждый пользователь

Настройка ReservationSystem

Fair Scheduler поддерживает **ReservationSystem**, позволяющую пользователям резервировать ресурсы заблаговременно. Таким образом приложение может запросить зарезервированные ресурсы во время выполнения, указав *reservationId*. Для этого могут быть настроены следующие параметры конфигурации в *yarn-site.xml*:

`yarn.resourcemanager.reservation-system.enable` – обязательный параметр: включить **ReservationSystem** в **ResourceManager**. Значение может быть только логическим (boolean), по умолчанию является *false*, то есть **ReservationSystem** не включена.

`yarn.resourcemanager.reservation-system.class` – необязательный параметр: имя класса **ReservationSystem**. Значение по умолчанию выбирается на основе настроенного планировщика, то есть если настроен **FairScheduler**, то классом является *FairReservationSystem*.

`yarn.resourcemanager.reservation-system.plan.follower` – необязательный параметр: имя класса *PlanFollower*, который запускается по таймеру и синхронизирует **FairScheduler** с *Plan* и наоборот. Значение по умолчанию выбирается на основе настроенного планировщика, то есть если настроен **FairScheduler**, то классом является *FairSchedulerPlanFollower*.

`yarn.resourcemanager.reservation-system.planfollower.time-step` – необязательный параметр: частота таймера *PlanFollower* (в миллисекундах). Значением по умолчанию является *1000*.

ReservationSystem интегрирована с иерархией очереди **Fair Scheduler** и может быть настроена только для leaf-очередей.

4.4.5 Администрирование

Изменение конфигурации на лету

Есть возможность изменения минимальных долей, лимитов, веса, тайм-аута преимущественного права preemtion и политики планирования очередей на лету во время выполнения посредством редактирования файла распределения. Планировщик перезагружает файл через *10-15* секунд после того, как увидит, что он изменен.

Мониторинг через веб-интерфейс

Текущие приложения, очереди и общие ресурсы можно просмотреть через веб-интерфейс **ResourceManager** по адресу `http://*ResourceManager URL*/cluster/scheduler`. Для каждой очереди в веб-интерфейсе можно просмотреть следующие поля:

- *Used Resources* – сумма ресурсов, выделенных контейнерам в очереди;
- *Num Active Applications* – количество приложений в очереди, которые получили хотя бы один контейнер;

- *Num Pending Applications* – количество приложений в очереди, которые еще не получили ни одного контейнера;
- *Min Resources* – настроенные минимальные ресурсы, которые гарантированы для очереди;
- *Max Resources* – настроенные максимальные ресурсы, которые разрешены в очереди;
- *Instantaneous Fair Share* – мгновенная справедливая доля ресурсов. Эти общие ресурсы учитывают только активные очереди (с запущенными приложениями) и используются для планирования решений. Очередям могут быть выделены ресурсы за пределами их общих ресурсов в случаях, когда другие очереди в них не нуждаются. На очередь, потребление ресурсов которой находится на уровне или ниже ее справедливой доли, не влияет преимущественное право preemption контейнеров;
- *Steady Fair Share* – постоянная справедливая доля ресурсов в очереди. Эти общие ресурсы учитывают все очереди независимо от того, активны ли они (имеют запущенные приложения). Они вычисляются реже и изменяются только при изменении конфигурации или пропускной способности. Предназначены для обеспечения видимости ресурсов, которые пользователь может ожидать, и поэтому отображаются в веб-интерфейсе.

Перемещение приложений между очередями

Fair Scheduler поддерживает возможность перемещения запущенного приложения в другую очередь. Это может быть полезно для перемещения важного приложения в очередь с более высоким приоритетом или для перемещения неважного приложения в очередь с более низким приоритетом. Приложения можно перемещать, запустив `yarn application -movetoqueue appID -queue targetQueueName`.

Когда приложение перемещается в очередь, его существующие распределения пересчитываются с распределениями новой очереди для целей определения справедливости. Если добавление ресурсов приложения приводит к нарушению ограничения *maxRunningApps* или *maxResources* в новой очереди, то попытка переместить приложение в нее завершается неудачей.

Дамп состояния Fair Scheduler

Fair Scheduler может периодически сбрасывать свое состояние. По умолчанию данная функция отключена, но администратор может включить ее, установив для уровня `org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler.statedump` значение *DEBUG*.

Журналы **Fair Scheduler** по умолчанию отправляются в лог-файл **Resource Manager**. Но дампы состояния планировщика потенциально могут генерировать большой объем данных, поэтому для того, чтобы вывести состояние в отдельный файл, необходимо раскомментировать раздел *Fair scheduler state dump* в *log4j.properties*.

4.5 YARN Timeline Service v.2

YARN Timeline Service v.2 – это следующая крупная итерация **Timeline Server** после *v.1* и *v.1.5*. Версия *v.2* создана с целью решения двух основных задач *v.1*.

Масштабируемость

Версия *v.1* ограничивается одним экземпляром устройства записи/чтения и хранения и не может масштабироваться далеко за пределы небольших кластеров. Версия *v.2* использует более масштабируемую распределенную архитектуру записи и масштабируемое backend-хранилище.

YARN Timeline Service v.2 отделяет сбор (запись) данных от обслуживания (чтения) данных. Он использует распределенные коллекторы, и по существу для каждого приложения **YARN** выделяется один коллектор. Читатели – это отдельные экземпляры, предназначенные для обслуживания запросов через REST API.

В качестве основного резервного хранилища **YARN Timeline Service v.2** выбирает СУБД **Apache HBase**, поскольку она хорошо масштабируется до большого размера, сохраняя при этом хорошее время отклика для чтения и записи.

Улучшения юзабилити

В большинстве случаев пользователи интересуются информацией на уровне “потоков” (flows) или логических групп приложений **YARN**. Гораздо более распространенным является запуск набора или серии приложений YARN для завершения логического приложения. **Timeline Service v.2** поддерживает понятие потоков в явном виде. Кроме того, он поддерживает агрегирование метрик на flow-уровне.

К тому же, такая информация, как конфигурация и метрики, обрабатывается и поддерживается как объекты первого класса.

Диаграмма иллюстрирует взаимосвязь между различными сущностями **YARN**, моделирующими потоки (Рис.4.4.).

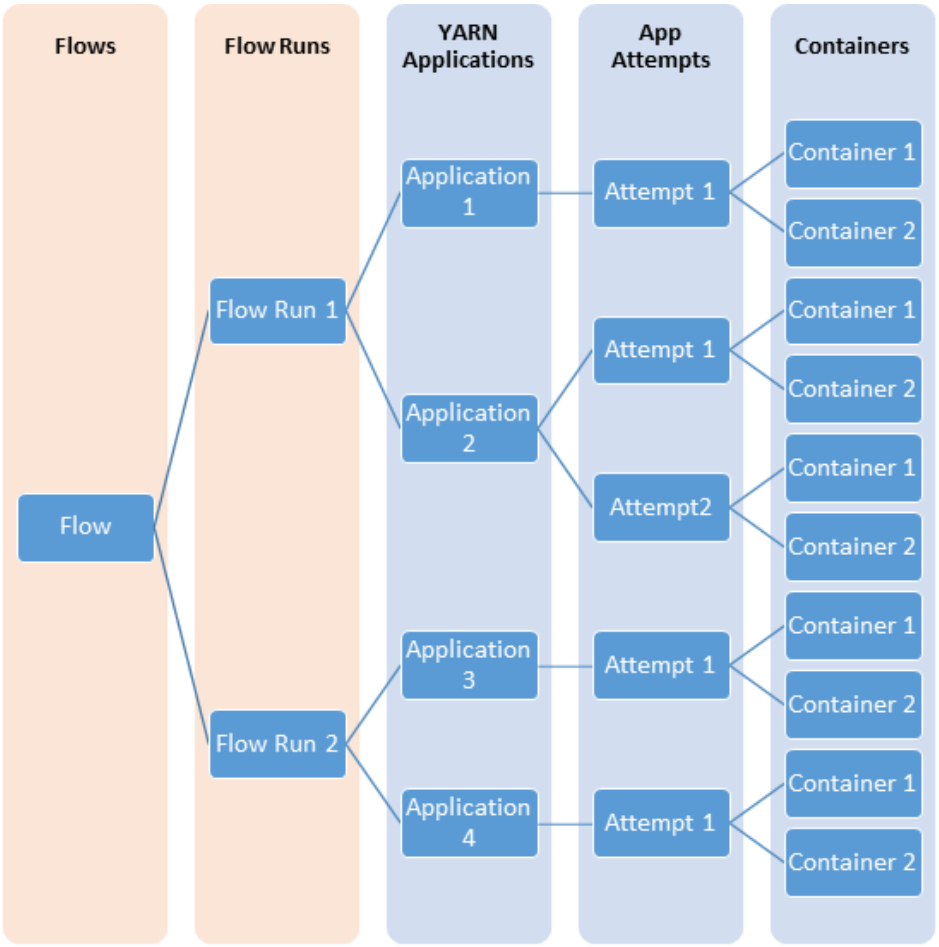


Рис.4.4.: Взаимосвязь между сущностями YARN

4.5.1 Архитектура

YARN Timeline Service v.2 использует набор коллекторов (писателей) для записи данных в backend-хранилище. Коллекторы распределяются и размещаются совместно с **Application Masters** (AM), которым они предназначены. Все данные, принадлежащие приложению, отправляются timeline-коллекторам уровня приложения, за исключением timeline-коллектора уровня **Resource Manager** (RM).

Для такого приложения **Application Master** может записывать данные в совместно расположенные timeline-коллекторы (которые являются вспомогательным сервисом **NodeManager** в этом выпуске). Кроме того, **NodeManagers** других узлов с выполняющимися контейнерами для приложения, также записывают данные в timeline-коллектор на узле, на котором выполняется **Application Master**.

Resource Manager тоже поддерживает свой собственный timeline-коллектор. Он генерирует только события жизненного цикла, характерные для **YARN**, чтобы поддерживать разумный объем записей.

Timeline-читатели – это отделенные от timeline-коллекторов демоны, предназначенные для обслуживания запросов через REST API (Рис.4.5.).

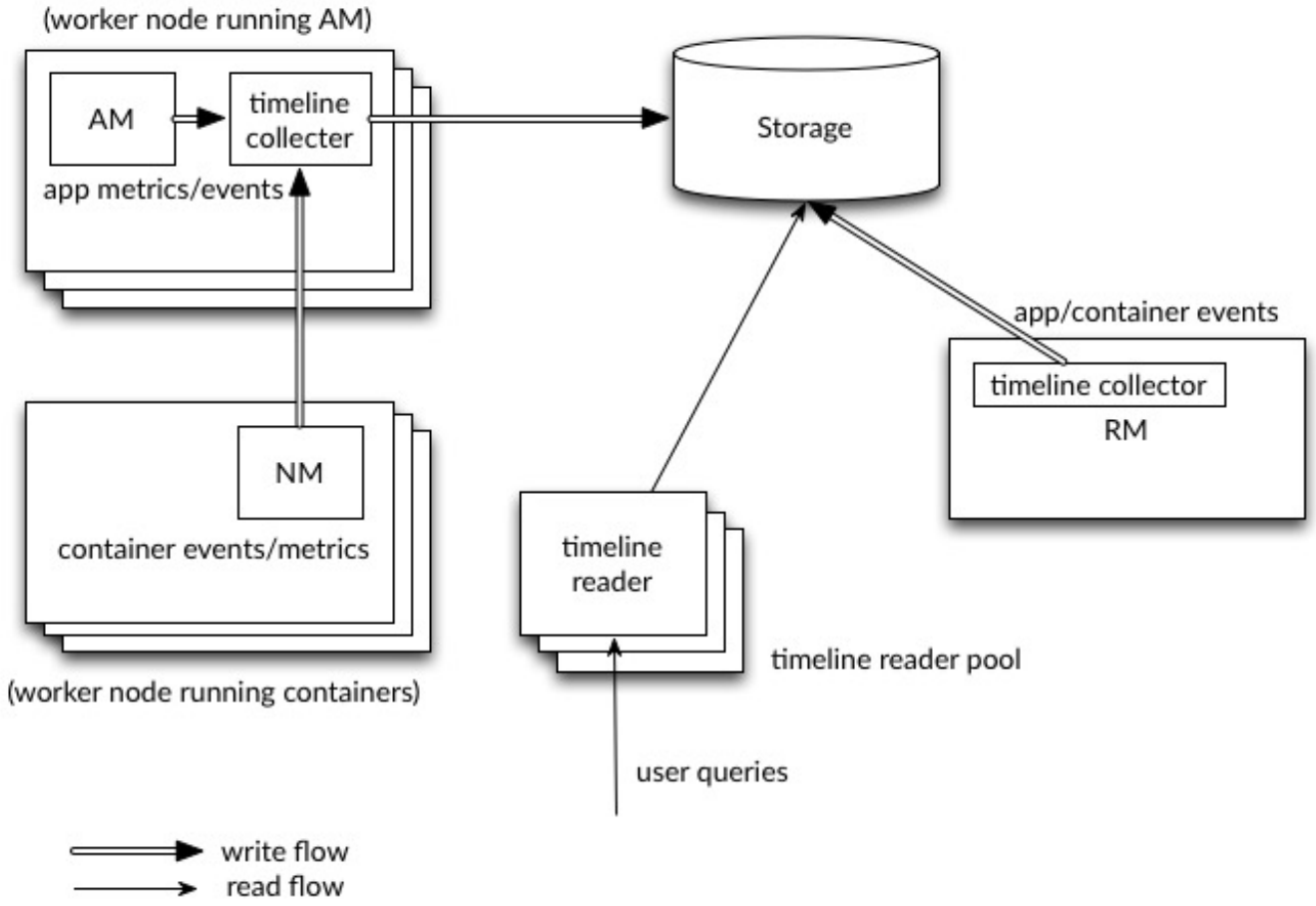


Рис.4.5.: Архитектура на высоком уровне

Текущее состояние и планы на будущее

YARN Timeline Service v.2 в настоящее время находится в альфа-версии (“alpha 2”). Разработка части функционала находится в процессе, и многие вещи могут и будут быстро меняться.

Полный сквозной поток операций записи и чтения является функциональным с **Apache HBase** в качестве серверной части. При включении сервиса публикуются все общие для **YARN** события, а также системные метрики **YARN**, такие как процессор и память. Кроме того, некоторые приложения, в том числе **Distributed Shell** и **MapReduce**, могут записывать в **YARN Timeline Service v.2** данные для каждой платформы.

Основным способом доступа к данным является REST. Поэтому REST API поставляется с большим количеством полезных и гибких шаблонов запросов (*REST API*). К тому же в настоящее время отсутствует

поддержка доступа к командной строке.

Коллекторы (писатели) в настоящее время встроены в **Node Managers** в качестве вспомогательных сервисов. **Resource Manager** также имеет свой специальный внутрипроцессный коллектор. Читатель в настоящее время является единственным экземпляром. Также в текущий период невозможно выполнить запись в **Timeline Service** вне контекста приложения **YARN** (то есть вне кластерного клиента).

Начиная с *alpha2*, **Timeline Service v.2** поддерживает простую авторизацию в виде настраиваемого белого списка пользователей и групп, которые могут читать timeline-данные. Администраторам кластера по умолчанию разрешено читать эти данные.

Отключенный **YARN Timeline Service v.2** никак не влияет на любую другую существующую функциональность.

Работа, чтобы сделать сервис действительно готовым к production-ready, продолжается. Некоторые ключевые элементы включают в себя:

- Более надежная отказоустойчивость хранилища;
- Поддержка внекластерных клиентов;
- Улучшенная поддержка для долгоработающих приложений;
- Поддержка ACL;
- Автономное (периодическое по времени) агрегирование потоков, пользователей и очередей для отчетов и анализа;
- Коллекторы timeline как отдельные экземпляры от Node Managers;
- Кластеризация читателей;
- Миграция и совместимость с v.1.

4.5.2 Конфигурация. Basic

`yarn.timeline-service.enabled` – указывает клиентам, включен ли сервис Timeline. При включенном параметре используемая приложениями библиотека *TimelineClient* публикует сущности и события на сервер Timeline. Значение по умолчанию *false*;

`yarn.timeline-service.version` – указывает текущую версию запущенного Timeline Service. Например, если значение параметра равно *1,5*, а `yarn.timeline-service.enabled` установлен на *true*, то это означает, что кластер будет и должен запускать Timeline Service версии *v.1.5*. На стороне клиента, если он использует такую же версию сервера, результат будет успешным. В случае если клиент выбирает меньшую версию, несмотря на то, насколько надежна история совместимости между версиями, результаты могут отличаться. По умолчанию значение параметра *1.0f*.

Новые параметры, введенные в версии *v.2*:

`yarn.timeline-service.writer.class` – класс операции записи backend-хранилища. Значение по умолчанию *HBase*;

`yarn.timeline-service.reader.class` – класс операции чтения backend-хранилища. Значение по умолчанию *HBase*;

`yarn.system-metrics-publisher.enabled` – определяет, публикуются ли системные метрики YARN в сервисе Timeline (от Resource Manager и Node Manager). Значение по умолчанию *false*;

`yarn.timeline-service.schema.prefix` – префикс схемы для hbase-таблиц. По умолчанию `prod.`.

4.5.3 Конфигурация. Advanced

`yarn.timeline-service.hostname` – имя хоста веб-приложения сервиса Timeline. Значение по умолчанию *0.0.0.0*;

`yarn.timeline-service.reader.webapp.address` – http-адрес веб-приложения Timeline Reader. По умолчанию *${yarn.timeline-service.hostname}:8188*;

`yarn.timeline-service.reader.webapp.https.address` – https-адрес веб-приложения Timeline Reader. По умолчанию *${yarn.timeline-service.hostname}:8190*;

`yarn.timeline-service.reader.bind-host` – фактический адрес, к которому привязывается timeline-читатель. Если параметр установлен, сервер читателя связывается с этим адресом и портом, указанным в `yarn.timeline-service.reader.webapp.address`. Наиболее полезно в целях прослушивания сервисом всех интерфейсов, задав значение параметра *0.0.0.0*.

Новые параметры, введенные в версии *v.2*:

`yarn.timeline-service.hbase.configuration.file` – необязательный URL-адрес файла конфигурации *hbase-site.xml*, используемый для подключения кластера timeline-service hbase. Если значение параметра пусто или не указано, конфигурация HBase загружается из *classpath*. Указанное значение параметра переопределяет *classpath*. По умолчанию установлено пустое значение;

`yarn.timeline-service.writer.flush-interval-seconds` – определяет частоту сброса записи timeline. Значение по умолчанию *60*;

`yarn.timeline-service.app-collector.linger-period.ms` – период времени, в течение которого коллектор приложений активен в Node Manager после завершения работы Application Master. Значение по умолчанию *60000* (60 секунд);

`yarn.timeline-service.timeline-client.number-of-async-entities-to-merge` – количество попыток клиента timeline V2 для объединения многочисленных асинхронных сущностей (если они доступны), после чего вызывает REST ATS V2 API для отправки. Значение по умолчанию *10*;

`yarn.timeline-service.hbase.coprocessor.app-final-value-retention-milliseconds` – определяет, как долго сохраняется финальное значение метрики завершенного приложения до объединения с суммой потока. По умолчанию *259200000* (3 дня). Значение должно быть установлено в кластере HBase;

`yarn.rm.system-metrics-publisher.emit-container-events` – определяет, публикуется ли метрика контейнера yarn на сервере timeline (от Resource Manager). Параметр конфигурации предназначен для ATS V2. Значение по умолчанию *false*.

4.5.4 Конфигурация. Security

Безопасность можно включить, установив для `yarn.timeline-service.http-authentication.type` значение *kerberos*, после чего станут доступны следующие параметры конфигурации:

`yarn.timeline-service.http-authentication.type` – определяет аутентификацию, используемую для конечной точки HTTP timeline-сервера (коллектор/читатель). Поддерживаемые значения: *simple* / *kerberos* / *#AUTHENTICATION_HANDLER_CLASSNAME#*. Значение по умолчанию *simple*;

`yarn.timeline-service.http-authentication.simple.anonymous.allowed` – указывает, разрешены ли анонимные запросы timeline-сервером при использовании аутентификации *simple*. По умолчанию *true*;

`yarn.timeline-service.http-authentication.kerberos.principal` – принципал Kerberos, используемый для конечной точки HTTP timeline-сервера (коллектор/читатель);

`yarn.timeline-service.http-authentication.kerberos.keytab` – keytab-файл Kerberos, используемый для конечной точки HTTP timeline-сервера (коллектор/читатель);

`yarn.timeline-service.principal` – принципал Kerberos для timeline-читателя. Для timeline-коллектора используется принципал Node Manager, поскольку он работает в качестве вспомогательного сервиса внутри Node Manager;

`yarn.timeline-service.keytab` – keytab-файл Kerberos для timeline-читателя. Для timeline-коллектора используется keytab-файл ключей Node Manager, поскольку он работает в качестве вспомогательного сервиса внутри Node Manager;

`yarn.timeline-service.delegation.key.update-interval` – значение по умолчанию *86400000* (1 день);

`yarn.timeline-service.delegation.token.renew-interval` – значение по умолчанию *86400000* (1 день);

`yarn.timeline-service.delegation.token.max-lifetime` – значение по умолчанию *604800000* (7 дней);

`yarn.timeline-service.read.authentication.enabled` – включает или отключает проверку авторизации для чтения данных timeline service v2. По умолчанию установлено *false* – отключена;

`yarn.timeline-service.read.allowed.users` – разделенный запятыми список пользователей и после пробела разделенный запятыми список групп. Функция позволяет введенному списку пользователей и групп читать данные и отклонять остальных пользователей и группы. По умолчанию установлено значение *none*. Если авторизация включена, то данный параметр обязателен.

4.5.5 Включение поддержки CORS

Для включения поддержки совместного использования ресурсов (Cross-origin resource sharing, CORS) в **Timeline Service v.2** необходимо установить следующие параметры конфигурации:

- В *yarn-site.xml* параметр `yarn.timeline-service.http-cross-origin.enabled` установить на *true*;
- В *core-site.xml* добавить `org.apache.hadoop.security.HttpCrossOriginFilterInitializer` к `hadoop.http.filter.initializers`.

Важно обратить внимание, что параметр `yarn.timeline-service.http-cross-origin.enabled`, установленный на *true*, переопределяет `hadoop.http.cross-origin.enabled`.

4.5.6 Включение Timeline Service v.2

Подготовка кластера **Apache HBase** к **Timeline Service v.2** заключается в выполнении нескольких шагов:

- *Настройка кластера HBase*;
- *Включение сопроцессора*;
- *Создание схемы для Timeline Service v.2*.

Настройка кластера HBase

Первый шаг заключается в настройке или выборе **Apache HBase** для использования в качестве кластера хранения. Версия **Timeline Service v.2** поддерживает **Apache HBase 1.2.6**. Ранние версии **Apache HBase** (*1.0.x*) не работают с **Timeline Service v.2**, а более поздние не протестированы.

HBase имеет разные режимы развертывания. При намерении создания простого профиля для кластера **Apache HBase** со слабой загрузкой данных, но с сохранением их при входе и выходе с узла, подходит режим развертывания “Standalone HBase over HDFS”.

Это полезный вариант автономной настройки **HBase**, когда все демоны **HBase** работают внутри одной JVM, и вместо того, чтобы сохраняться в локальной файловой системе, сохраняются в экземпляре **HDFS**. Для настройки такого автономного варианта необходимо отредактировать файл *hbase-site.xml*, указав `hbase.rootdir` на каталог в экземпляре **HDFS**, а затем установить для `hbase.cluster.distributed` значение *false*. Например:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://namenode.example.org:8020/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>false</value>
  </property>
</configuration>
```

Включение сопроцессора

В этой версии осуществляется динамическая загрузка сопроцессора (табличный сопроцессор для flowrun-таблицы). Для этого необходимо скопировать jar-файл сервиса timeline в **HDFS**, откуда **HBase** сможет его загрузить. Это требуется для создания flowrun-таблицы в schema creator. По умолчанию расположение в **HDFS** – */hbase/coprocessor*. Например:

```
hadoop fs -mkdir /hbase/coprocessor
hadoop fs -put hadoop-yarn-server-timelineservice-hbase-3.0.0-alpha1-SNAPSHOT.jar
       /hbase/coprocessor/hadoop-yarn-server-timelineservice.jar
```

Также можно воспользоваться параметром yarn-конфигурации – `yarn.timeline-service.hbase.coprocessor.jar.hd`
Например:

```
<property>
  <name>yarn.timeline-service.hbase.coprocessor.jar.hdfs.location</name>
  <value>/custom/hdfs/path/jarName</value>
</property>
```

Создание схемы для Timeline Service v.2

Подготовка кластера **Apache HBase** к **Timeline Service v.2** завершается запуском инструмента schema creator для создания необходимых таблиц:

```
bin/hadoop org.apache.hadoop.yarn.server.timelineservice.storage.TimelineSchemaCreator -create
```

Инструмент **TimelineSchemaCreator** поддерживает несколько опций, которые могут пригодиться, особенно при тестировании. Например, можно использовать `-skipExistingTable` (сокращенно `-s`), чтобы пропустить существующие таблицы и продолжить создание других таблиц, не прерывая создания схемы. Если параметр или `-help` (сокращенно `-h`) не задан, отображается command usage и продолжается создание других таблиц без сбоя создания схемы. По умолчанию таблицы имеют префикс схемы `prod.`.

4.5.7 Основные конфигурации Timeline Service v.2

Основные конфигурации для запуска **Timeline service v.2**:

```
<property>
  <name>yarn.timeline-service.version</name>
  <value>2.0f</value>
</property>

<property>
  <name>yarn.timeline-service.enabled</name>
  <value>true</value>
</property>
```

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle,timeline_collector</value>
</property>

<property>
 <name>yarn.nodemanager.aux-services.timeline_collector.class</name>
  <value>org.apache.hadoop.yarn.server.timelineservice.collector.PerNodeTimelineCollectorsAuxService</value>
</property>

<property>
  <description>The setting that controls whether yarn system metrics is
  published on the Timeline service or not by RM And NM.</description>
  <name>yarn.system-metrics-publisher.enabled</name>
  <value>true</value>
</property>

<property>
  <description>The setting that controls whether yarn container events are
  published to the timeline service or not by RM. This configuration setting
  is for ATS V2.</description>
  <name>yarn.rm.system-metrics-publisher.emit-container-events</name>
  <value>true</value>
</property>
```

Кроме того, для имени кластера **YARN** можно установить уникальное значение (удобно при использовании нескольких кластеров для хранения данных в одном и том же хранилище **Apache HBase**):

```
<property>
  <name>yarn.resourcemanager.cluster-id</name>
  <value>my_research_test_cluster</value>
</property>
```

Также можно добавить файл *hbase-site.xml* в конфигурацию кластера **Hadoop** клиента, чтобы он мог записывать данные в используемый кластер **Apache HBase**, или установить `yarn.timeline-service.hbase.configuration.file` в URL файла на *hbase-site.xml*. Например:

```
<property>
  <description> Optional URL to an hbase-site.xml configuration file to be
  used to connect to the timeline-service hbase cluster. If empty or not
  specified, then the HBase configuration will be loaded from the classpath.
  When specified the values in the specified configuration file will override
  those from the ones that are present on the classpath.
  </description>
  <name>yarn.timeline-service.hbase.configuration.file</name>
  <value>file:/etc/hbase/hbase-ats-dc1/hbase-site.xml</value>
</property>
```

4.5.8 Запуск Timeline Service v.2

Для того, чтобы выбрать новую конфигурацию, необходимо перезапустить **Resource Manager**, а также **Node Managers**. Коллекторы запускаются в рамках **Resource Manager** и **Node Managers**.

Timeline Service reader – это отдельный демон **YARN**, который можно запустить, используя следующий синтаксис:

```
$ yarn-daemon.sh start timelinereader
```

4.5.9 Включение MapReduce

Для записи данных **MapReduce** в **Timeline Service v.2** необходимо включить следующую конфигурацию в *mapred-site.xml*:

```
<property>
  <name>mapreduce.job.emit-timeline-data</name>
  <value>true</value>
</property>
```

4.5.10 Обновление с alpha1 до alpha2

При использовании **Timeline Service v.2** версии *alpha1* рекомендуется:

- Очистить существующие данные в таблицах (truncate tables), так как ключ строки для *AppToFlow* изменился;
- Сопроцессор теперь является динамически загружаемым сопроцессором уровня таблицы в *alpha2*. Рекомендуется удалить таблицу, заменить jar-файл сопроцессора на hdfs на *alpha2*, перезапустить серверы *Region* и воссоздать flowrun-таблицу.

4.5.11 Публикация определенных данных приложения

Глава предназначена для разработчиков приложений **YARN**, которые хотят интегрироваться с **Timeline Service v.2**.

Разработчикам необходимо использовать *TimelineV2Client* API для публикации данных для каждой платформы в **Timeline Service v.2**, поскольку API сущности/объекта для *v.2* значительно изменилось по отношению к *v.1*, в части объектной модели. Класс сущности в *v.2* – `org.apache.hadoop.yarn.api.records.timelineservice.TimelineEntity`.

Метод `putEntities` в **Timeline Service v.2** бывает двух видов: `putEntities` и `putEntitiesAsync`. Первый – это операция блокировки, используемая для записи наиболее важных данных (например, событий жизненного цикла). Последний является неблокирующей операцией. Важно обратить внимание, что ни один из методов не имеет возвращаемого значения.

Создание *TimelineV2Client* включает передачу идентификатора приложения статическому методу *TimelineV2Client.createTimelineClient*.

```
// Create and start the Timeline client v.2
TimelineV2Client timelineClient =
    TimelineV2Client.createTimelineClient(appId);
timelineClient.init(conf);
timelineClient.start();

try {
  TimelineEntity myEntity = new TimelineEntity();
  myEntity.setType("MY_APPLICATION");
  myEntity.setId("MyApp1");
  // Compose other entity info

  // Blocking write
  timelineClient.putEntities(myEntity);

  TimelineEntity myEntity2 = new TimelineEntity();
```

```
  // Compose other info

  // Non-blocking write
  timelineClient.putEntitiesAsync(myEntity2);

} catch (IOException | YarnException e) {
  // Handle the exception
} finally {
  // Stop the Timeline client
  timelineClient.stop();
}
```

Как показано в примере, следует указать идентификатор приложения **YARN**, чтобы иметь возможность записи в **Timeline Service v.2**. Также важно обратить внимание, что при текущей версии необходимо находиться в кластере, чтобы иметь возможность записи в сервис. Например, **Application Master** или код в контейнере могут выполнять запись в **Timeline Service**, в то время как отправитель задания (job submitter) **MapReduce** вне кластера – нет.

После создания клиента *timeline v2* пользователь также должен установить информацию timeline-коллектора, содержащую его адрес и токен (только в безопасном режиме) для приложения. Если используется *AMRMClient*, то достаточно зарегистрировать timeline-клиент, вызвав `AMRMClient#registerTimelineV2Client`.

```
amRMClient.registerTimelineV2Client(timelineClient)
```

Еще один адрес должен быть извлечен из распределенного отклика от **Application Master** и должен быть явно установлен в timeline-клиенте:

```
timelineClient.setTimelineCollectorInfo(response.getCollectorInfo());
```

Создавать и публиковать собственные сущности, события и метрики можно также, как и в предыдущих версиях.

Объекты *TimelineEntity* имеют следующие поля для хранения timeline-данных:

- *events* – набор TimelineEvents, упорядоченный по метке времени событий в порядке убывания. Каждое событие связано с одной меткой времени и содержит один идентификатор и карту для хранения связанной информации;
- *configs* – сопоставление из строки (config name) в строку (config value), представляющее все настройки, связанные с сущностью. Пользователи могут публиковать весь конфиг или его часть в поле конфигурации. Поддерживается для приложений и общих сущностей;
- *metrics* – набор метрик, связанных с сущностью. Бывает два типа метрик: метрика одного значения (single value) и метрика временного ряда (time series). Каждый элемент метрики содержит имя метрики (id), значение и тип операции агрегирования, которая должна выполняться в этой метрике (по умолчанию *noop*). Поддерживается для потока, приложения и общих сущностей;
- *info* – сопоставление из строки (info key name) в объект (info value) для хранения связанной информации для сущности. Поддерживается для приложений и общих сущностей;
- *isrelatedtoEntities and relatestoEntities* – каждая сущность содержит поля *relatedtoEntities* и *isrelatedtoEntities* для представления взаимосвязей с другими сущностями. Оба поля представляют собой сопоставление от строки (name of the relationship) до timeline-сущности. Таким образом, взаимосвязи между сущностями могут быть представлены как DAG.

Важно обратить внимание, что при публикации timeline-метрик можно выбрать способ агрегирования каждой метрики с помощью метода `TimelineMetric#setRealtimeAggregationOp()`. Слово “aggregate” здесь означает применение одной из операций *TimelineMetricOperation* для набора сущностей. **Timeline service**

v2 обеспечивает встроенную агрегацию на уровне приложения, что означает агрегирование метрик из разных timeline-сущностей в одном YARN-приложении. В настоящее время в *TimelineMetricOperation* поддерживается два вида операций:

- *MAX* – получение максимального значения среди всех объектов TimelineMetric;
- *SUM* – получение суммы всех объектов TimelineMetric.

По умолчанию задается *NOP* – в реальном времени никакая операция агрегирования не выполняется.

Платформы приложений по возможности должны устанавливать "flow context", чтобы воспользоваться преимуществами поддержки потока **Timeline Service v.2**. Контекст потока состоит из:

- *Flow name* – строка, идентифицирующая поток высокого уровня (например, "distributed grep" или любое имя, которое может уникально представлять приложение);
- *Flow run id* – возрастающая последовательность чисел, отличающая разные серии одного и того же потока;
- *Flow version*, опционально – строковый идентификатор, обозначающий версию потока. Версия потока может использоваться для определения изменений в потоках, таких как изменения кода или сценариев.

Если контекст потока не указан, по умолчанию предоставляется:

- *Flow name* – имя приложения YARN (или идентификатор приложения, если имя не задано);
- *Flow run id* – время запуска приложения в Unix time (миллисекунды);
- *Flow version* – "1".

Можно предоставить контекст потока через теги YARN-приложения:

```
ApplicationSubmissionContext appContext = app.getApplicationSubmissionContext();

// set the flow context as YARN application tags
Set<String> tags = new HashSet<>();
tags.add(TimelineUtils.generateFlowNameTag("distributed grep"));
tags.add(Timelineutils.generateFlowVersionTag("3df8b0d6100530080d2e0decf9e528e57c42a90a"));
tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));

appContext.setApplicationTags(tags);
```

Important: Resource Manager преобразует теги приложения YARN в нижний регистр перед их сохранением. Следовательно, необходимо преобразовать имена и версии потоков в нижний регистр, прежде чем использовать их в запросах REST API

4.5.12 REST API

Запросы **Timeline Service v.2** в настоящее время поддерживается только через REST API; в библиотеках **YARN** не реализован API-клиент.

REST API в версии *v.2* осуществляется по пути */ws/v2/timeline/* в веб-сервисе **Timeline Service**.

Root path:

```
GET /ws/v2/timeline/
```

Возвращает объект JSON, описывающий экземпляр сервиса и информацию о версии.

```
{
  "About":"Timeline Reader API",
```

```
  "timeline-service-version":"3.0.0-alpha1-SNAPSHOT",
  "timeline-service-build-version":"3.0.0-alpha1-SNAPSHOT from fb0acd08e6f0b030d82eeb7cbfa5404376313e60 by␣
↪sjlee source checksum be6cba0e42417d53be16459e1685e7",
  "timeline-service-version-built-on":"2016-04-11T23:15Z",
  "hadoop-version":"3.0.0-alpha1-SNAPSHOT",
  "hadoop-build-version":"3.0.0-alpha1-SNAPSHOT from fb0acd08e6f0b030d82eeb7cbfa5404376313e60 by sjlee␣
↪source checksum ee968fd0aedcc7384230ee3ca216e790",
  "hadoop-version-built-on":"2016-04-11T23:14Z"
}
```

Далее описываются поддерживаемые запросы в REST API:

- *Query Flows*
- *Query Flow Runs*
- *Query Flow Run*
- *Query Apps for a flow*
- *Query Apps for a flow run*
- *Query app*
- *Query generic entities with in the scope of Application*
- *Query generic entities*
- *Query generic entity with in the scope of Application*
- *Query generic entity*
- *Query generic entity types*

Query Flows

С помощью Query Flows API можно получить список активных потоков, запущенных за последнее время. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если ни один из потоков не соответствует предикатам, возвращается пустой список.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/flows/

or

GET /ws/v2/timeline/flows/
```

Поддерживаемые параметры запроса:

`limit` – определяет количество возвращаемых потоков. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`daterange` – формат значения `[startdate]-[enddate]`, то есть начальная и конечная даты, разделенные дефисом, или одна дата. Даты интерпретируются в формате *yyyyMMdd* и допускаются в формате UTC. Если указана одна дата, возвращаются все потоки, активные в этот день. Если задано начальное и конечное значение, возвращаются все активные потоки в указанный период. Если задана только начальная дата, возвращаются активные потоки на указанный день и все последующие. Если задана только конечная дата, возвращаются потоки, активные на указанный день и все предшествующие. Например:

- `daterange=20150711` – возвращает активные потоки на дату 11.07.2015;

- `daterange=20150711-20150714` – возвращает активные потоки на период 11.07.2015-14.07.2015;
- `daterange=20150711-` – возвращает активные потоки на дату 11.07.2015 и все последующие;
- `daterange=-20150711` – возвращает активные потоки на дату 11.07.2015 и все предшествующие;

`fromid` – возвращение набора потоков из заданного *fromid*, включая набор сущностей. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе.

Пример ответа JSON:


```
[
  {
    "metrics": [],
    "events": [],
    "id": "test-cluster/1460419200000/sjlee@ds-date",
    "type": "YARN_FLOW_ACTIVITY",
    "createdtime": 0,
    "flowruns": [
      {
        "metrics": [],
        "events": [],
        "id": "sjlee@ds-date/1460420305659",
        "type": "YARN_FLOW_RUN",
        "createdtime": 0,
        "info": {
          "SYSTEM_INFO_FLOW_VERSION": "1",
          "SYSTEM_INFO_FLOW_RUN_ID": 1460420305659,
          "SYSTEM_INFO_FLOW_NAME": "ds-date",
          "SYSTEM_INFO_USER": "sjlee"
        },
        "isrelatedto": {},
        "relatesto": {}
      },
      {
        "metrics": [],
        "events": [],
        "id": "sjlee@ds-date/1460420587974",
        "type": "YARN_FLOW_RUN",
        "createdtime": 0,
        "info": {
          "SYSTEM_INFO_FLOW_VERSION": "1",
          "SYSTEM_INFO_FLOW_RUN_ID": 1460420587974,
          "SYSTEM_INFO_FLOW_NAME": "ds-date",
          "SYSTEM_INFO_USER": "sjlee"
        },
        "isrelatedto": {},
        "relatesto": {}
      }
    ],
    "info": {
      "SYSTEM_INFO_CLUSTER": "test-cluster",
      "UID": "test-cluster!sjlee!ds-date",
      "FROM_ID": "test-cluster!1460419200000!sjlee!ds-date",
      "SYSTEM_INFO_FLOW_NAME": "ds-date",
      "SYSTEM_INFO_DATE": 1460419200000,
      "SYSTEM_INFO_USER": "sjlee"
    },
    "isrelatedto": {},
    "relatesto": {}
```

```
  }
]
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query Flow Runs

С помощью Query Flow Runs API можно углубиться в детали и получить запуски (runs) потока (конкретные экземпляры). Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если ни один из запусков потока не соответствует предикатам, возвращается пустой список.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/

or

GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/
```

Поддерживаемые параметры запроса:

`limit` – определяет количество возвращаемых потоков. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`createdtimestart` – возвращаются runs потока, запущенные после указанной временной метки;

`createdtimeend` – возвращаются runs потока, запущенные до указанной временной метки;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Если параметр не задан, в ответе возвращаются поля *id*, *type*, *createdtime* и *info*. Для выполнения запроса flow runs доступны только поля *ALL* и *METRICS*, другие поля приводят к ответу HTTP 400 (Bad Request);

`fromid` – возвращение набора flow run из заданного *fromid*, включая набор сущностей. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе.

Пример ответа JSON:

```
[
  {
    "metrics": [],
    "events": [],
    "id": "sjlee@ds-date/1460420587974",
    "type": "YARN_FLOW_RUN",
```

```
    "createdtime": 1460420587974,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460420587974",
      "FROM_ID": "test-cluster!sjlee!ds-date!1460420587974",
      "SYSTEM_INFO_FLOW_RUN_ID": 1460420587974,
      "SYSTEM_INFO_FLOW_NAME": "ds-date",
      "SYSTEM_INFO_FLOW_RUN_END_TIME": 1460420595198,
      "SYSTEM_INFO_USER": "sjlee"
    },
    "isrelatedto": {},
    "relatesto": {}
  },
  {
    "metrics": [],
    "events": [],
    "id": "sjlee@ds-date/1460420305659",
    "type": "YARN_FLOW_RUN",
    "createdtime": 1460420305659,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460420305659",
      "FROM_ID": "test-cluster!sjlee!ds-date!1460420305659",
      "SYSTEM_INFO_FLOW_RUN_ID": 1460420305659,
      "SYSTEM_INFO_FLOW_NAME": "ds-date",
      "SYSTEM_INFO_FLOW_RUN_END_TIME": 1460420311966,
      "SYSTEM_INFO_USER": "sjlee"
    },
    "isrelatedto": {},
    "relatesto": {}
  }
]
```


Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса или указано недопустимое для запроса поле;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query Flow Run

С помощью данного API можно запросить определенный flow run, идентифицированный кластером, пользователем, именем потока или run-идентификатором. Так же при этом по умолчанию возвращаются метрики потока. Если используется конечная точка REST без имени кластера, берется кластер, указанный в `configuration yarn.resourcemanager.cluster-id` в *yarn-site.xml*.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/{run id}

or

GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}
```

Поддерживаемые параметры запроса:

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный

запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы.

Пример ответа JSON:


```
{
  "metrics": [
    {
      "type": "SINGLE_VALUE",
      "id": "org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter:BYTES_READ",
      "aggregationOp": "NOP",
      "values": {
        "1465246377261": 118
      }
    },
    {
      "type": "SINGLE_VALUE",
      "id": "org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter:BYTES_WRITTEN",
      "aggregationOp": "NOP",
      "values": {
        "1465246377261": 97
      }
    }
  ],
  "events": [],
  "id": "varun@QuasiMonteCarlo/1465246348599",
  "type": "YARN_FLOW_RUN",
  "createdtime": 1465246348599,
  "isrelatedto": {},
  "info": {
    "UID":"yarn-cluster!varun!QuasiMonteCarlo!1465246348599",
    "FROM_ID":"yarn-cluster!varun!QuasiMonteCarlo!1465246348599",
    "SYSTEM_INFO_FLOW_RUN_END_TIME":1465246378051,
    "SYSTEM_INFO_FLOW_NAME":"QuasiMonteCarlo",
    "SYSTEM_INFO_USER":"varun",
    "SYSTEM_INFO_FLOW_RUN_ID":1465246348599
  },
  "relatesto": {}
}
```


Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 404 (Not Found) – запуск потока для данного flow run id не может быть найден;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query Apps for a flow

С помощью данного API можно запрашивать все приложения **YARN**, которые являются частью определенного потока. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если количество совпадающих приложений

превышает установленный лимит, возвращаются последние приложения до достижения предела. Если ни одно из приложений не соответствует предикатам, возвращается пустой список.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/apps

or

GET /ws/v2/timeline/users/{user name}/flows/{flow name}/apps
```

Поддерживаемые параметры запроса:

`limit` – определяет количество возвращаемых потоков. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`createdtimestart` – возвращаются приложения, созданные после указанной временной метки;

`createdtimeend` – возвращаются приложения, созданные до указанной временной метки;

`relatesto` – определяет, должны ли совпадающие приложения относиться к заданным сущностям. Представляется как выражение вида:

```
(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>
↪:<entityid>...,<entitytype>:<entityid>:<entityid>...)
```

Если выражение имеет тип сущности (взаимосвязь идентификатора(-ов) сущности, указанная в скобках, последующих за знаком `!`) это означает, что приложения с этими взаимосвязями не возвращаются. Для выражений или подвыражений без знака `!` возвращаются все приложения, имеющие указанные отношения в своем поле *relatesto*. Оператор `op` является логическим и может быть *AND* или *OR*. Тип сущности может сопровождаться любым числом идентификаторов сущностей. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`isrelatedto` – определяет, должны ли совпадающие приложения быть связаны с данными сущностями. Представляется так же, как выражение `relatesto`;

`infofilters` – определяет, должны ли совпадающие приложения иметь точное совпадение с данным информационным ключом и должны ли быть равны его значению. Информационный ключ (info key) – это строка, значением которой может быть любой объект. Инфофильтры представляются в виде выражения: `(<key> <compareop> <value>) <op> (<key> <compareop> <value>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает “равно”), *ne* (означает “не равно” и наличие ключа для совпадения не требуется) или *ene* (означает “не равно”, но наличие ключа необходимо). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))`. Если *value* является объектом, значение может быть задано в форме JSON-формата без пробелов. Например: `(infokey1 eq {“<key>”:“<value>”,“<key>”:“<value>”...})`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`conffilters` – определяет, должны ли совпадающие приложения иметь точное совпадение с данным именем конфигурации и должны ли быть равны ее значению. Имя и значение конфигурации должны быть строками. Представляется так же, как выражение `infofilters`;

`metricfilters` – определяет, должны ли совпадающие приложения иметь точные совпадения с данной метрикой и удовлетворять указанной связи со значением метрики. Идентификатор метрики должен быть строкой, а значение метрики должно быть целочисленным (integral). Параметр представляется в выражении вида: `(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает “равно”), *ne* (означает “не равно” и наличие метрики

для совпадения не требуется), *ene* (означает "не равно", но наличие метрики необходимо), *gt* (означает "больше, чем"), *ge* (означает "больше или равно"), *lt* (означает "меньше, чем") и *le* (означает "меньше или равно"). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))`. По сути, это выражение эквивалентно `(metric1 == 50 AND metric2 > 40) OR (metric1 < 20)`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`eventfilters` – определяет, должны ли совпадающие приложения содержать данные события. Параметр представляется в выражении вида: `(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)`. Здесь `!` означает, что ни один из перечисленных через запятую списков событий в скобках со знаком `!` не должен существовать для того, чтобы произошло совпадение. Если `!` не указано, события в скобках должны существовать. Оператор `op` может быть *AND* или *OR*. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((event1,event2) AND !(event4)) OR (event3,event7,event5))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-приложения, тип (эквивалент *YARN_APPLICATION*), время создания приложения и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени;

`fromid` – возвращение набора сущностей приложения из заданного *fromid*. Набор сущностей включает указанный *fromid*. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе потока сущности.

Пример ответа JSON:


```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION",
    "id": "application_1465246237936_0001",
    "createdtime": 1465246348599,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001"
      "FROM_ID": "yarn-cluster!varun!QuasiMonteCarlo!1465246348599!application_1465246237936_0001",
    },
    "relatesto": { }
  },
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION",
    "id": "application_1464983628730_0005",
    "createdtime": 1465033881959,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1464983628730_0005"
      "FROM_ID": "yarn-cluster!varun!QuasiMonteCarlo!1465246348599!application_1464983628730_0005",
    },
    "relatesto": { }
  }
]
```


Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query Apps for a flow run

С помощью данного API можно запрашивать все приложения **YARN**, которые являются частью определенного flow run. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если количество совпадающих приложений превышает установленный лимит, возвращаются последние приложения до достижения предела. Если ни одно из приложений не соответствует предикатам, возвращается пустой список.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/{run id}/apps

or

GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}/apps/
```

Поддерживаемые параметры запроса:

`limit` – определяет количество возвращаемых приложений. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`createdtimestart` – возвращаются приложения, созданные после указанной метки времени;

`createdtimeend` – возвращаются приложения, созданные до указанной метки времени;

`relatesto` – определяет, должны ли совпадающие приложения относиться к заданным сущностям. Представляется как выражение вида:

```
(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>
↪:<entityid>...,<entitytype>:<entityid>:<entityid>...)
```

Если выражение имеет тип сущности (взаимосвязь идентификатора(-ов) сущности, указанная в скобках, последующих за знаком `!`) это означает, что приложения с этими взаимосвязями не возвращаются. Для выражений или подвыражений без знака `!` возвращаются все приложения, имеющие указанные отношения в своем поле *relatesto*. Оператор `op` является логическим и может быть *AND* или *OR*. Тип сущности может сопровождаться любым числом идентификаторов сущностей. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`isrelatedto` – определяет, должны ли совпадающие приложения быть связаны с данными сущностями и их типом. Представляется так же, как выражение `relatesto`;

`infofilters` – определяет, должны ли совпадающие приложения иметь точное совпадение с данным информационным ключом и должны ли быть равны его значению. Информационный ключ (info key) – это строка, значением которой может быть любой объект. Инфофильтры представляются в виде выражения: `(<key> <compareop> <value>) <op> (<key> <compareop> <value>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает “равно”), *ne* (означает “не равно” и наличие ключа для совпадения не требуется) или *ene* (означает “не равно”, но наличие ключа необходимо). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))`. Если *value* является объектом, значение может быть задано в форме JSON-формата без пробелов. Например: `(infokey1 eq {“<key>”:“<value>”,“<key>”:“<value>”...})`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`conffilters` – определяет, должны ли совпадающие приложения иметь точное совпадение с данным именем конфигурации и должны ли быть равны ее значению. Имя и значение конфигурации должны быть строками. Представляется так же, как выражение `infofilters`;

`metricfilters` – определяет, должны ли совпадающие приложения иметь точные совпадения с данной метрикой и удовлетворять указанной связи со значением метрики. Идентификатор метрики должен быть строкой, а значение метрики должно быть целочисленным (integral). Параметр представляется в выражении вида: `(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает “равно”), *ne* (означает “не равно” и наличие метрики для совпадения не требуется), *ene* (означает “не равно”, но наличие метрики необходимо), *gt* (означает “больше, чем”), *ge* (означает “больше или равно”), *lt* (означает “меньше, чем”) и *le* (означает “меньше или равно”). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))`. По сути, это выражение эквивалентно `(metric1 == 50 AND metric2 > 40) OR (metric1 < 20)`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`eventfilters` – определяет, должны ли совпадающие приложения содержать данные события. Параметр представляется в выражении вида: `(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)`. Здесь `!` означает, что ни один из перечисленных через запятую списков событий в скобках со знаком `!` не должен существовать для того, чтобы произошло совпадение. Если `!` не указано, события в скобках

должны существовать. Оператор `op` может быть *AND* или *OR*. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((event1,event2) AND !(event4)) OR (event3,event7,event5))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-приложения, тип (эквивалент *YARN_APPLICATION*), время создания приложения и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS/ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени;

`fromid` – возвращение набора сущностей приложения из заданного *fromid*. Набор сущностей включает указанный *fromid*. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе потока сущности.

Пример ответа JSON:


```
[
  {
    "metrics": [],
    "events": [],
    "id": "application_1460419579913_0002",
    "type": "YARN_APPLICATION",
    "createdtime": 1460419580171,
    "info": {
      "UID": "test-cluster!sjlee!ds-date!1460419580171!application_1460419579913_0002"
      "FROM_ID": "test-cluster!sjlee!ds-date!1460419580171!application_1460419579913_0002",
    },
```

```
    "configs": {},
    "isrelatedto": {},
    "relatesto": {}
  }
]
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query app

С помощью данного API можно запрашивать одно приложение **YARN**, идентифицированное кластером ID-приложения. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Информация о контексте потока, то есть пользователь, имя потока и run id, не являются обязательными, но если они указаны в параметре запроса, это может исключить необходимость в дополнительной операции для получения информации о контексте потока на основе id кластера и приложения.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}

or

GET /ws/v2/timeline/apps/{app id}
```

Поддерживаемые параметры запроса:

`userid` – возвращает приложения, принадлежащие данному пользователю. Параметр запроса должен быть указан вместе с параметрами `flowname` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowname` – возвращает приложения, принадлежащие данному имени потока. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowrunid` – возвращает приложения, принадлежащие данному идентификатору flow run. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowname`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-приложения, тип (эквивалент *YARN_APPLICATION*), время создания приложения и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени.

Пример ответа JSON:

```
{
  "metrics": [],
  "events": [],
  "id": "application_1460419579913_0002",
  "type": "YARN_APPLICATION",
  "createdtime": 1460419580171,
  "info": {
    "UID": "test-cluster!sjlee!ds-date!1460419580171!application_1460419579913_0002"
  },
  "configs": {},
  "isrelatedto": {},
  "relatesto": {}
}
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 404 (Not Found) – информация о контексте потока не может быть получена или приложение для данного id приложения не может быть найдено;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query generic entities with in the scope of Application

С помощью данного API можно запрашивать общие сущности, идентифицируемые по ID-кластера и приложения и типу сущности для каждой платформы. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Информация о контексте потока, то есть пользователь, имя потока и run id, не являются обязательными, но

если они указаны в параметре запроса, это может исключить необходимость в дополнительной операции для получения информации о контексте потока на основе id кластера и приложения. Если количество совпадающих сущностей превышает установленный лимит, возвращаются последние сущности до достижения предела.

Эта конечная точка может использоваться для запроса контейнеров, приложения или любой другой общей сущности, которую клиенты помещают в серверную часть. Например, можно запросить контейнеры, указав тип сущности как *YARN_CONTAINER* и *YARN_APPLICATION_ATTEMPT*. Если ни одна из сущностей не соответствует предикатам, возвращается пустой список.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}

or

GET /ws/v2/timeline/apps/{app id}/entities/{entity type}
```

Поддерживаемые параметры запроса:

`userid` – возвращает сущности, принадлежащие данному пользователю. Параметр запроса должен быть указан вместе с параметрами `flowname` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowname` – возвращает сущности, принадлежащие данному имени потока. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowrunid` – возвращает сущности, принадлежащие данному идентификатору flow run. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowname`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`limit` – определяет количество возвращаемых сущностей. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`createdtimestart` – возвращаются сущности, созданные после указанной метки времени;

`createdtimeend` – возвращаются сущности, созданные до указанной метки времени;

`relatesto` – определяет, должны ли совпадающие сущности относиться к заданным сущностям. Представляется как выражение вида:

```
(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>
↪:<entityid>...,<entitytype>:<entityid>:<entityid>...)
```

Если выражение имеет тип сущности (взаимосвязь идентификатора(-ов) сущности, указанная в скобках, последующих за знаком `!`) это означает, что сущности с этими взаимосвязями не возвращаются. Для выражений или подвыражений без знака `!` возвращаются все сущности, имеющие указанные отношения в своем поле *relatesto*. Оператор `op` является логическим и может быть *AND* или *OR*. Тип сущности может сопровождаться любым числом идентификаторов сущностей. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`isrelatedto` – определяет, должны ли совпадающие сущности быть связаны с данными сущностями и их типом. Представляется так же, как выражение `relatesto`;

`infofilters` – определяет, должны ли совпадающие сущности иметь точное совпадение с данным информационным ключом и должны ли быть равны его значению. Информационный ключ (info key) – это строка, значением которой может быть любой объект. Инфофильтры представляются в виде выражения: `(<key> <compareop> <value>) <op> (<key> <compareop> <value>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает "равно"), *ne* (означает "не равно" и наличие ключа для совпадения не требуется) или *ene* (означает "не равно", но наличие ключа необходимо). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))`. Если *value* является объектом, значение может быть задано в форме JSON-формата без пробелов. Например: `(infokey1 eq {"<key>":"<value>","<key>":"<value>"...})`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`conffilters` – определяет, должны ли совпадающие сущности иметь точное совпадение с данным именем конфигурации и должны ли быть равны ее значению. Имя и значение конфигурации должны быть строками. Представляется так же, как выражение `infofilters`;

`metricfilters` – определяет, должны ли совпадающие сущности иметь точные совпадения с данной метрикой и удовлетворять указанной связи со значением метрики. Идентификатор метрики должен быть строкой, а значение метрики должно быть целочисленным (integral). Параметр представляется в выражении вида: `(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает "равно"), *ne* (означает "не равно" и наличие метрики для совпадения не требуется), *ene* (означает "не равно", но наличие метрики необходимо), *gt* (означает "боольше, чем"), *ge* (означает "больше или равно"), *lt* (означает "меньше, чем") и *le* (означает "меньше или равно"). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))`. По сути, это выражение эквивалентно `(metric1 == 50 AND metric2 > 40) OR (metric1 < 20)`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`eventfilters` – определяет, должны ли совпадающие сущности содержать данные события. Параметр представляется в выражении вида: `(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)`. Здесь `!` означает, что ни один из перечисленных через запятую списков событий в скобках со знаком `!` не должен существовать для того, чтобы произошло совпадение. Если `!` не указано, события в скобках должны существовать. Оператор `op` может быть *AND* или *OR*. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((event1,event2) AND !(event4)) OR (event3,event7,event5))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан,

конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-сущности и ее тип, время создания и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени;

`fromid` – возвращение набора общих сущностей из заданного *fromid*. Набор сущностей включает указанный *fromid*. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе потока сущности.

Пример ответа JSON:

```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION_ATTEMPT",
    "id": "appattempt_1465246237936_0001_000001",
    "createdtime": 1465246358873,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT!appattempt_1465246237936_
↪0001_000001"
      "FROM_ID": "yarn-cluster!sjlee!ds-date!1460419580171!application_1465246237936_0001!YARN_APPLICATION_
↪ATTEMPT!0!appattempt_1465246237936_0001_000001"
    },
    "relatesto": { }
  },
  {
    "metrics": [ ],
    "events": [ ],
    "type": "YARN_APPLICATION_ATTEMPT",
    "id": "appattempt_1465246237936_0001_000002",
    "createdtime": 1465246359045,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT!appattempt_1465246237936_
↪0001_000002"
      "FROM_ID": "yarn-cluster!sjlee!ds-date!1460419580171!application_1465246237936_0001!YARN_APPLICATION_
↪ATTEMPT!0!appattempt_1465246237936_0001_000002"
    },
    "relatesto": { }
  }
]
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 404 (Not Found) – информация о контексте потока не может быть получена;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query generic entities

С помощью данного API можно запрашивать общие сущности для каждого пользователя, идентифицируемые по ID-кластера, *doAsUser* и типу сущности. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если количество совпадающих сущностей превышает установленный лимит, возвращаются последние сущности до достижения предела.

Эта конечная точка может использоваться для запроса общей сущности, которую клиенты помещают в серверную часть. Например, можно запросить пользовательские сущности, указав тип сущности как *TEZ_DAG_ID*. Если ни одна из сущностей не соответствует предикатам, возвращается пустой список. Примечание: на данный момент можно запрашивать только те сущности, которые опубликованы с помощью *doAsUser*, отличного от владельца приложения.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{userid}/entities/{entitytype}

or

GET /ws/v2/timeline/users/{userid}/entities/{entitytype}
```

Поддерживаемые параметры запроса:

`limit` – определяет количество возвращаемых сущностей. Максимально возможное значение лимита – максимальное значение *Long*. Если значение не указано или меньше *0*, то лимит считается равным *100*;

`createdtimestart` – возвращаются сущности, созданные после указанной метки времени;

`createdtimeend` – возвращаются сущности, созданные до указанной метки времени;

`relatesto` – определяет, должны ли совпадающие сущности относиться к заданным сущностям. Представляется как выражение вида:

```
(<entitytype>:<entityid>:<entityid>...,<entitytype>:<entityid>:<entityid>...) <op> !(<entitytype>:<entityid>
↪:<entityid>...,<entitytype>:<entityid>:<entityid>...)
```

Если выражение имеет тип сущности (взаимосвязь идентификатора(-ов) сущности, указанная в скобках, последующих за знаком `!`) это означает, что сущности с этими взаимосвязями не возвращаются. Для выражений или подвыражений без знака `!` возвращаются все сущности, имеющие указанные отношения в своем поле *relatesto*. Оператор `op` является логическим и может быть *AND* или *OR*. Тип сущности может сопровождаться любым числом идентификаторов сущностей. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((type1:id1:id2:id3,type3:id9) AND !(type2:id7:id8)) OR (type1:id4))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`isrelatedto` – определяет, должны ли совпадающие сущности быть связаны с данными сущностями и их типом. Представляется так же, как выражение `relatesto`;

`infofilters` – определяет, должны ли совпадающие сущности иметь точное совпадение с данным информационным ключом и должны ли быть равны его значению. Информационный ключ (info key)

– это строка, значением которой может быть любой объект. Инфофильтры представляются в виде выражения: `(<key> <compareop> <value>) <op> (<key> <compareop> <value>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает "равно"), *ne* (означает "не равно" и наличие ключа для совпадения не требуется) или *ene* (означает "не равно", но наличие ключа необходимо). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((infokey1 eq value1) AND (infokey2 ne value1)) OR (infokey1 ene value3))`. Если *value* является объектом, значение может быть задано в форме JSON-формата без пробелов. Например: `(infokey1 eq {"<key>":"<value>","<key>":"<value>"...})`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`conffilters` – определяет, должны ли совпадающие сущности иметь точное совпадение с данным именем конфигурации и должны ли быть равны ее значению. Имя и значение конфигурации должны быть строками. Представляется так же, как выражение `infofilters`;

`metricfilters` – определяет, должны ли совпадающие сущности иметь точные совпадения с данной метрикой и удовлетворять указанной связи со значением метрики. Идентификатор метрики должен быть строкой, а значение метрики должно быть целочисленным (integral). Параметр представляется в выражении вида: `(<metricid> <compareop> <metricvalue>) <op> (<metricid> <compareop> <metricvalue>)`. Оператор `op` может быть *AND* или *OR*; `compareop` – *eq* (означает "равно"), *ne* (означает "не равно" и наличие метрики для совпадения не требуется), *ene* (означает "не равно", но наличие метрики необходимо), *gt* (означает "боольше, чем"), *ge* (означает "больше или равно"), *lt* (означает "меньше, чем") и *le* (означает "меньше или равно"). Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((metric1 eq 50) AND (metric2 gt 40)) OR (metric1 lt 20))`. По сути, это выражение эквивалентно `(metric1 == 50 AND metric2 > 40) OR (metric1 < 20)`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`eventfilters` – определяет, должны ли совпадающие сущности содержать данные события. Параметр представляется в выражении вида: `(<eventid>,<eventid>) <op> !(<eventid>,<eventid>,<eventid>)`. Здесь `!` означает, что ни один из перечисленных через запятую списков событий в скобках со знаком `!` не должен существовать для того, чтобы произошло совпадение. Если `!` не указано, события в скобках должны существовать. Оператор `op` может быть *AND* или *OR*. Можно комбинировать любое количество *AND* и *OR* для создания сложных выражений. Для объединения выражений можно использовать скобки. Например: `(((event1,event2) AND !(event4)) OR (event3,event7,event5))`. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим

образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-сущности и ее тип, время создания и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени;

`fromid` – возвращение набора общих сущностей из заданного *fromid*. Набор сущностей включает указанный *fromid*. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе потока сущности.

Пример ответа JSON:


```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "TEZ_DAG_ID",
    "id": "dag_1465246237936_0001_000001",
    "createdtime": 1465246358873,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!sjlee!TEZ_DAG_ID!0!dag_1465246237936_0001_000001"
      "FROM_ID": "sjlee!yarn-cluster!TEZ_DAG_ID!0!dag_1465246237936_0001_000001"
    },
    "relatesto": { }
  },
  {
    "metrics": [ ],
    "events": [ ],
    "type": "TEZ_DAG_ID",
    "id": "dag_1465246237936_0001_000002",
    "createdtime": 1465246359045,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!sjlee!TEZ_DAG_ID!0!dag_1465246237936_0001_000002!userX"
      "FROM_ID": "sjlee!yarn-cluster!TEZ_DAG_ID!0!dag_1465246237936_0001_000002!userX"
    },
    "relatesto": { }
  }
]
```


Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query generic entity with in the scope of Application

С помощью данного API можно запрашивать определенную общую сущность, идентифицированную по ID кластера и приложения, типу сущности для каждой платформы и ID-сущности. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Информация о контексте потока, то есть пользователь, имя потока и run id, не являются обязательными, но если они указаны в параметре запроса, это может исключить необходимость в дополнительной операции для получения информации о контексте потока на основе id кластера и приложения.

Эта конечная точка может использоваться для запроса отдельного контейнера, приложения или любой другой общей сущности, что клиенты помещают в серверную часть. Например, можно запросить определенный YARN-контейнер, указав тип сущности как *YARN_CONTAINER* и задав идентификатор сущности как ID контейнера. Аналогично, приложение может быть запрошено путем указания типа сущности как *YARN_APPLICATION_ATTEMPT*, a application attempt ID в виде идентификатора сущности.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}/{entity id}

or

GET /ws/v2/timeline/apps/{app id}/entities/{entity type}/{entity id}
```

Поддерживаемые параметры запроса:

`userid` – возвращает сущности, принадлежащие данному пользователю. Параметр запроса должен быть указан вместе с параметрами `flowname` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowname` – возвращает сущности, принадлежащие данному имени потока. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowrunid` – возвращает сущности, принадлежащие данному идентификатору flow run. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowname`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае

извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-сущности и ее тип, время создания и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени.

`entityidprefix` – задает id-префикс для извлекаемой сущности. При указанном параметре извлечение сущности ускоряется.

Пример ответа JSON:

```
{
  "metrics": [ ],
  "events": [ ],
  "type": "YARN_APPLICATION_ATTEMPT",
  "id": "appattempt_1465246237936_0001_000001",
  "createdtime": 1465246358873,
  "isrelatedto": { },
  "configs": { },
  "info": {
    "UID": "yarn-cluster!application_1465246237936_0001!YARN_APPLICATION_ATTEMPT!0!appattempt_1465246237936_
↪0001_000001"
    "FROM_ID": "yarn-cluster!sjlee!ds-date!1460419580171!application_1465246237936_0001!YARN_APPLICATION_
↪ATTEMPT!0!appattempt_1465246237936_0001_000001"
  },
  "relatesto": { }
}
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 404 (Not Found) – информация о контексте потока не может быть получена или сущность для данного id-сущности не может быть найдена;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query generic entity

С помощью данного API можно запрашивать общую сущность для каждого пользователя, идентифицируемую по ID-кластера, *doAsUser* и типу сущности и ее ID. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если количество совпадающих сущностей превышает установленный лимит, возвращаются последние сущности до достижения предела.

Эта конечная точка может использоваться для запроса общей сущности, которую клиенты помещают в серверную часть. Например, можно запросить пользовательские сущности, указав тип сущности как *TEZ_DAG_ID*. Если ни одна из сущностей не соответствует предикатам, возвращается пустой список. Примечание: на данный момент можно запрашивать только те сущности, которые опубликованы с помощью *doAsUser*, отличного от владельца приложения.

HTTP:

```
GET /ws/v2/timeline/clusters/{cluster name}/users/{userid}/entities/{entitytype}/{entityid}

or

GET /ws/v2/timeline/users/{userid}/entities/{entitytype}/{entityid}
```

Поддерживаемые параметры запроса:

`metricstoretrieve` – определяет, какие метрики извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – разделенный запятыми список id-префиксов метрики. В таком случае извлекаются все соответствующие указанным префиксам метрики. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<metricprefix>,<metricprefix>,<metricprefix>,<metricprefix>...)` – что тоже указывает на разделенный запятыми список id-префиксов метрики, но в таком случае извлекаются только не соответствующие ни одному из префиксов метрики. Если параметр задан, метрики извлекаются независимо от того, указаны ли они в полях *METRICS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`confstoretrieve` – определяет, какие конфигурации извлекать, и отправляет обратно в ответе. Может быть выражением вида: `(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>..` – разделенный запятыми список префиксов имени конфигурации. В таком случае извлекаются все соответствующие указанным префиксам конфигурации. Для простого выражения скобки необязательны. Альтернативно, выражения могут иметь такую форму: `!(<config_name_prefix>,<config_name_prefix>,<config_name_prefix>,<config_name_prefix>...)` – что тоже указывает на разделенный запятыми список префиксов имени конфигурации, но в таком случае извлекаются только не соответствующие ни одному из префиксов конфигурации. Если параметр задан, конфигурации извлекаются независимо от того, указаны ли они в полях *CONFIGS* параметра запроса или нет. Важно обратить внимание, что небезопасные символы URL, такие как пробелы, должны быть соответствующим образом закодированы;

`fields` – определяет поля для извлечения. Возможные значения для полей: *EVENTS*, *INFO*, *CONFIGS*, *METRICS*, *RELATES_TO*, *IS_RELATED_TO* и *ALL*. Если указано *ALL*, извлекаются все поля. Может быть указано несколько полей в виде списка через запятую. Если ни одно поле не указано, в ответе возвращается id-сущности и ее тип, время создания и UID из поля *info*;

`metricslimit` – определяет количество возвращаемых метрик. Учитывается только в случае, если поля содержат *METRICS*/*ALL* или указан `metricstoretrieve`. В иных случаях игнорируется. Максимально возможным значением может быть максимальное значение Integer. Если параметр не указан или имеет значение меньше *1*, и при этом метрики должны быть получены, то `metricslimit` рассматривается как *1*, и возвращает последнее значение метрики (метрик);

`metricstimestart` – возвращаются метрики для сущности после указанной метки времени;

`metricstimeend` – возвращаются метрики для сущности до указанной метки времени;

`fromid` – возвращение набора общих сущностей из заданного *fromid*. Набор сущностей включает указанный *fromid*. Значение *fromid* должно быть взято из информационного ключа *FROM_ID* в отправленном ранее ответе потока сущности.

Пример ответа JSON:

```
[
  {
    "metrics": [ ],
    "events": [ ],
    "type": "TEZ_DAG_ID",
    "id": "dag_1465246237936_0001_000001",
    "createdtime": 1465246358873,
    "isrelatedto": { },
    "configs": { },
    "info": {
      "UID": "yarn-cluster!sjlee!TEZ_DAG_ID!0!dag_1465246237936_0001_000001!userX"
      "FROM_ID": "sjlee!yarn-cluster!TEZ_DAG_ID!0!dag_1465246237936_0001_000001!userX"
    },
    "relatesto": { }
  }
]
```


Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

Query generic entity types

С помощью данного API можно запрашивать набор доступных типов сущностей для данного идентификатора приложения. Если используется конечная точка REST без имени кластера, берется кластер, указанный в конфигурации `yarn.resourcemanager.cluster-id` в *yarn-site.xml*. Если идентификатор пользователя, имя потока и идентификатор потока выполнения, которые являются необязательными параметрами запроса, не указаны, они будут запрашиваться на основе идентификатора приложения и идентификатора кластера из информации о контексте потока, хранящейся в базовой реализации хранилища.

HTTP:

```
GET /ws/v2/timeline/apps/{appid}/entity-types

or

GET /ws/v2/timeline/clusters/{clusterid}/apps/{appid}/entity-types
```

Поддерживаемые параметры запроса:

`userid` – возвращает сущности, принадлежащие данному пользователю. Параметр запроса должен быть указан вместе с параметрами `flowname` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowname` – возвращает сущности, принадлежащие данному имени потока. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowrunid`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения;

`flowrunid` – возвращает сущности, принадлежащие данному идентификатору flow run. Параметр запроса должен быть указан вместе с параметрами `userid` и `flowname`, в противном случае он игнорируется. Если все три параметра не заданы, то извлекать информацию о контексте потока приходится при выполнении запроса на основе id кластера и приложения.

Пример ответа JSON:

```
{
  YARN_APPLICATION_ATTEMPT,
  YARN_CONTAINER,
  MAPREDUCE_JOB,
  MAPREDUCE_TASK,
  MAPREDUCE_TASK_ATTEMPT
}
```

Код ответа:

- HTTP 200 (OK) – успех;
- HTTP 400 (Bad Request) – какая-либо проблема при синтаксическом анализе запроса;
- HTTP 404 (Not Found) – информация о контексте потока не может быть получена или сущность для данного id-сущности не может быть найдена;
- HTTP 500 (Internal Server Error) – неустранимые ошибки при возвращении данных.

4.6 Hadoop: YARN Federation

4.6.1 Архитектура

Известно, что **YARN** масштабируется до тысяч узлов. Масштабируемость **YARN** определяется **Resource Manager**, и она пропорциональна количеству узлов, активных приложений и контейнеров, а так же частоте heartbeat-сообщений (как узлов, так и приложений). Снижение heartbeat-сообщений может обеспечить увеличение масштабируемости, однако это отрицательно сказывается на использовании.

В данной главе описан подход на основе Federation для масштабирования одного кластера **YARN** до десятков тысяч узлов путем интеграции нескольких подкластеров **YARN**. Предлагаемый метод заключается в разделении большого кластера (10-100 тысяч узлов) на более мелкие блоки, называемые субкластерами (sub-cluster), каждый из которых имеет свой собственный **YARN Resource Manager** и вычислительные узлы. Система Federation объединяет эти субкластеры и делает их одним большим кластером **YARN** для приложений. Приложения при этом видят один массивный кластер **YARN** и могут планировать задачи на любом его узле, в то время как в рамках системы Federation ведутся переговоры с **Resource Manager** субкластеров для предоставления ресурсов приложению. Цель состоит в том, чтобы позволить отдельной задаче бесшовно “охватить” субкластеры.

Такая конструкция является структурно масштабируемой, поскольку она связывает количество узлов, за которые отвечает каждый **Resource Manager**, а соответствующие политики пытаются обеспечить, чтобы большинство приложений находилось в одном субкластере, таким образом, число видимых приложений для каждого **Resource Manager** также ограничено. Это означает, что масштабирумость может быть почти линейной, просто добавляя субкластеры (поскольку для них требуется очень небольшая координация).

Такая архитектура может обеспечить очень строгое соблюдение инвариантов планирования в каждом субкластере (просто наследуется от **YARN**), в то время как непрерывная перебалансировка по субкластеру обеспечивает (менее строго) то, что эти свойства также соблюдаются на глобальном уровне (например, если субкластер теряет большое количество узлов, можно переназначить очереди на другие субкластеры, чтобы обеспечить исключение несправедливого воздействия на пользователей, работающих в поврежденном субкластере).

Federation спроектирована как “слой” поверх существующей кодовой базы **YARN** с ограниченными изменениями в основных механизмах (Рис.4.6.).

YARN Sub-cluster

Субкластер (sub-cluster) – это кластер **YARN** размером до нескольких тысяч узлов. Точный размер субкластера определяется с учетом простоты развертывания/обслуживания, согласования с сетевыми зонами и

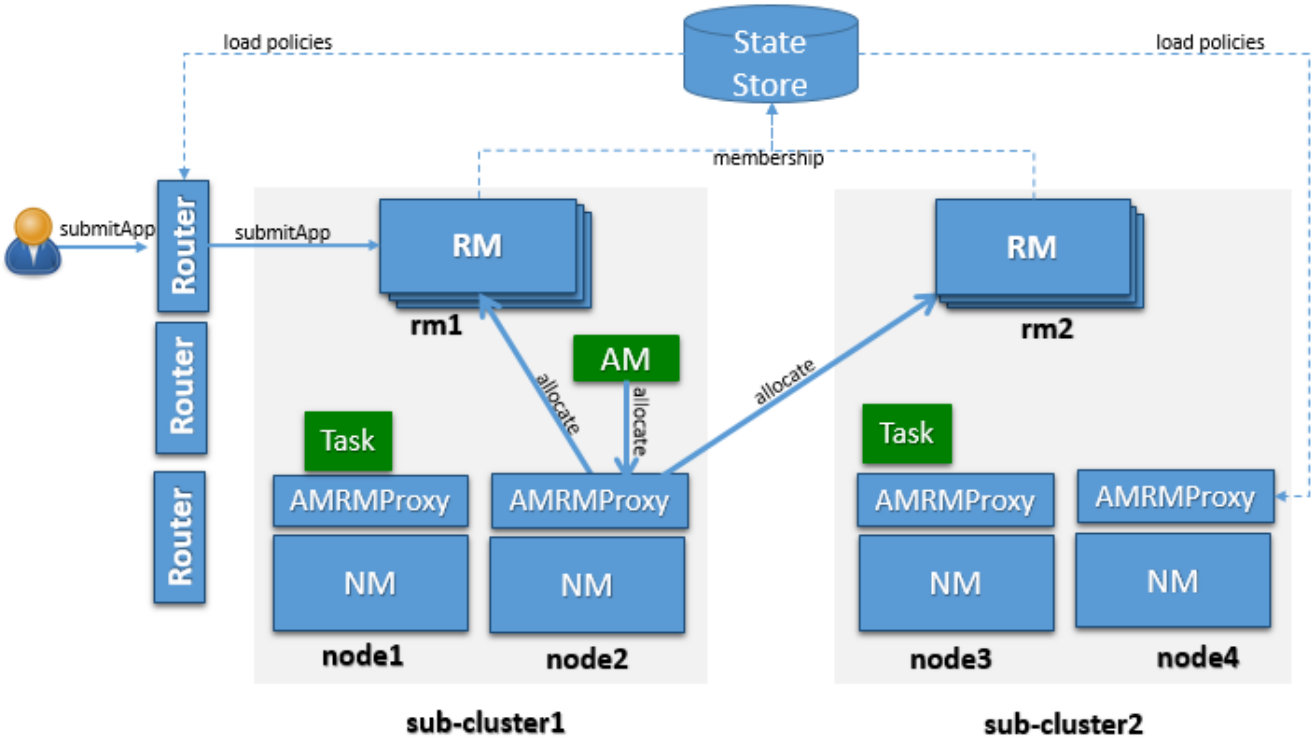


Рис.4.6.: Основные компоненты, составляющие кластер Federation

их доступности, а также общими рекомендациями.

Resource Manager субкластера **YARN** работает с высоким уровнем доступности (HA) с сохранением работоспособности, то есть необходимо быть в состоянии к сбоям **Resource Manager** и **Node Manager** с минимальными нарушениями. Если весь субкластер скомпрометирован, внешние механизмы обеспечивают повторную передачу заданий в отдельный субкластер (в дальнейшем это может быть включено в Federation).

Субкластер также является единицей масштабируемости в среде Federation – ее можно расширить, добавив один или несколько субкластеров.

По своей структуре каждый субкластер является полностью функциональным **Resource Manager**, и его вклад в Federation может быть установлен лишь на долю его общей емкости, т.е. субкластер может иметь "частичное" обязательство перед Federation, сохраняя при этом способность выдавать часть своих возможностей локальным способом.

Router

Приложения **YARN** отправляются на один из маршрутизаторов (Router), который, в свою очередь, применяет политику маршрутизации (полученную из Policy Store), запрашивает в State Store URL-адрес субкластера и перенаправляет запрос на отправку приложения в соответствующий **Resource Manager** субкластера. Субкластер, в котором запускается задание, называется "домашним субкластером" (home sub-cluster), а "вторичными" (secondary sub-clusters) называются все остальные субкластеры, на которые распространяется задание.

Маршрутизатор предоставляет *ApplicationClientProtocol* внешнему миру, прозрачно скрывая присутствие нескольких **Resource Manager**. Для этого маршрутизатор также сохраняет соответствие между приложением и его домашним субкластером в State Store. Это позволяет маршрутизаторам быть в мягком состоянии, недорого поддерживая при этом запросы пользователей, так как любой маршрутизатор может восстановить приложение для маппинга домашнего субкластера и направить запросы к нужному **Resource Manager**. Целесообразно для кэширования производительности и балансировки нагрузки. Состояние Federation (включая приложения и узлы) отображается через веб-интерфейс.

AMRMProxy

AMRMProxy является ключевым компонентом, позволяющим приложению масштабироваться и работать в субкластерах. *AMRMProxy* работает на всех машинах **Node Manager** и действует как прокси-сервер для **YARN Resource Manager** для **Application Master**, реализуя *ApplicationMasterProtocol*. Приложениям не разрешается напрямую связываться с **Resource Manager** субкластера. Система принудительно подключает их только к конечной точке *AMRMProxy*, что обеспечивает прозрачный доступ к нескольким **YARN Resource Manager** (путем динамической маршрутизации / разделения / маппинга коммуникаций). В любой момент времени задание может охватывать один домашний и несколько вторичных субкластеров, но работающие в *AMRMProxy* политики пытаются ограничить площадь каждого задания, чтобы минимизировать накладные расходы на инфраструктуру планирования (Рис.4.7.).

Роль *AMRMProxy*:

- Защита субкластера YARN Resource Manager от некорректно работающих Application Master. AMRMProxy может предотвратить DDOS-атаки, дросселируя/уничтожая требующих слишком много ресурсов Application Master;
- Маскировка нескольких YARN Resource Manager в кластере и прозрачный допуск Application Master к распределению по субкластерам. Все распределения контейнеров выполняются инфраструктурой YARN Resource Manager, которая состоит из AMRMProxy, выходящего в домашний и другие субкластера Resource Manager;
- Перехват всех запросов, поэтому может принудительно применять квоты приложений, которые не могут быть выполнены субкластером Resource Manager (поскольку каждый из них видит только часть запросов Application Master);

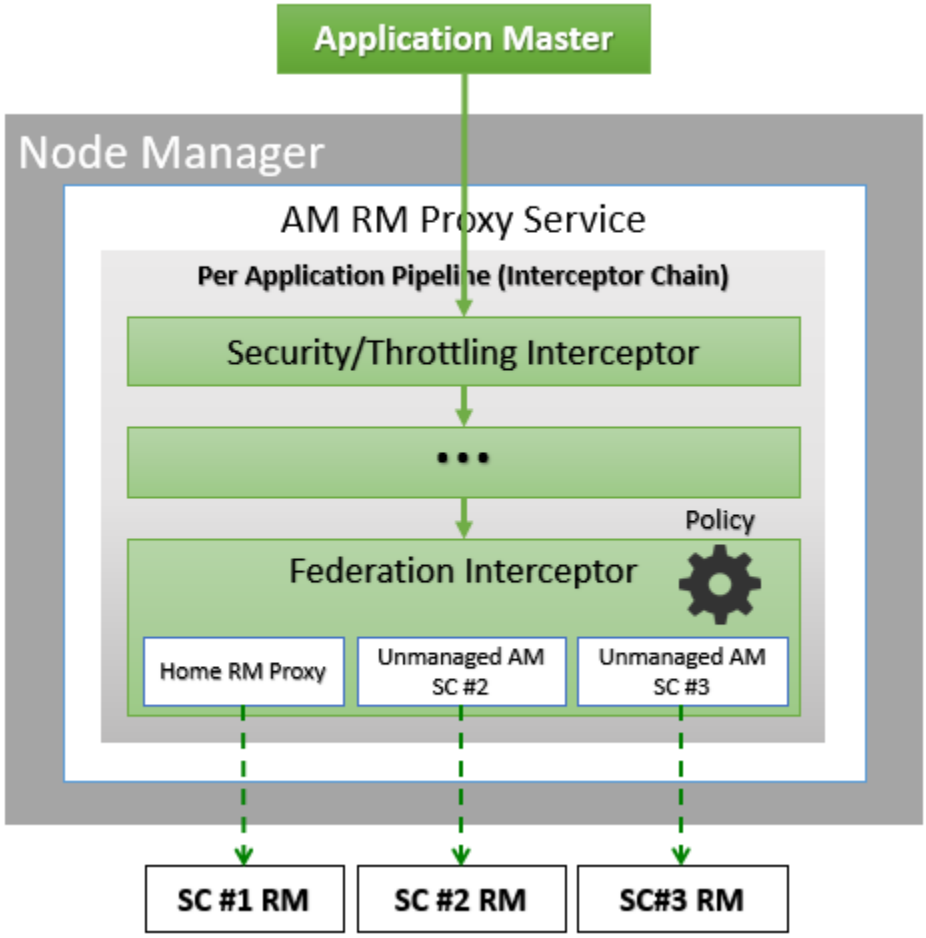


Рис.4.7.: Архитектура цепочки перехватчиков AMRMProxy

- Может применять политики балансировки нагрузки / переполнения.

Global Policy Generator

Global Policy Generator (*GPG*) контролирует всю Federation и гарантирует, что система все время настроена должным образом. Ключевым моментом идеи является то, что доступность кластера не зависит от постоянно включенного *GPG*. При этом *GPG* работает непрерывно, но вне зоны действия всех операций кластера, и предоставляет уникальную точку обзора, которая позволяет применять глобальные инварианты, влиять на балансировку нагрузки, инициировать дренаж субкластеров, которые будут подвергаться техническому обслуживанию, и т.д. *GPG* точнее обновляет маппинг распределения пропускной способности пользователя субкластеру и реже меняет политики, выполняющиеся в Routers, *AMRMProxy* (и возможных **Resource Manager**).

В случае если *GPG* недоступен, операции кластера продолжаются с момента последней публикации политик *GPG*, и хотя долгосрочная недоступность может означать, что некоторые из желательных свойств баланса, оптимального использования кластера и глобальных инвариантов могут исчезнуть, вычисления и доступ к данным не будут скомпрометированы.

В текущей реализации *Global Policy Generator* представляет собой процесс ручной настройки, представленный через CLI (YARN-3657).

Эта часть системы Federation является частью будущей работы в YARN-5597.

Federation State-Store

Federation State определяет дополнительное состояние, которое необходимо поддерживать для свободного объединения нескольких отдельных субкластеров в один большой кластер Federation. Включает в себя:

- Sub-cluster Membership

Члены **YARN Resource Manager** непрерывно передают heartbeat-сообщения в State Store для keep-alive и публикации своей текущей мощности/загрузке. Эта информация используется *GPG* для принятия необходимых политических решений. Также эта информация может использоваться маршрутизаторами для выбора лучшего домашнего субкластера. Этот механизм позволяет динамически увеличивать/уменьшать "кластерный парк", добавляя или удаляя субкластеры, а также позволяет легко обслуживать каждый из них. Это новая функциональность, которую необходимо добавить в **YARN Resource Manager**, при этом механизмы между собой хорошо понятны, поскольку функциональность аналогична индивидуальной высокой доступности (HA) **YARN Resource Manager**.

- Application's Home Sub-cluster

Субкластер, в котором выполняется **Application Master**, называется "домашним субкластером приложения" (home sub-cluster). При этом **Application Master** не ограничивается ресурсами только домашнего субкластера и может запрашивать ресурсы из других, называемых "вторичными" (secondary sub-clusters). Среда Federation настраивается и периодически налаживается таким образом, чтобы при размещении **Application Master** в субкластере он мог найти большую часть ресурсов в домашнем субкластере и только в определенных случаях запрашивал ресурсы у других субкластеров.

Federation Policy Store

Federation Policy Store – это логически отдельное хранилище (хотя оно может поддерживаться одним и тем же физическим компонентом), которое содержит информацию о том, как приложения и запросы ресурсов направляются в разные субкластеры. Текущая реализация предоставляет несколько политик – от случайных/ хэширующих/ циклических/ приоритетных до более сложных, которые учитывают нагрузку субкластера и запрашивают потребности в локальности.

4.6.2 Запуск приложений через субкластеры

При отправке приложения система определяет наиболее подходящий субкластер для его запуска, и он становится домашним. Все коммуникации от **Application Master** к **Resource Manager** осуществляются через *AMRMProxy*, работающий локально на машине **Application Master**. *AMRMProxy* предоставляет ту же конечную точку протокола *ApplicationMasterService*, что и **YARN Resource Manager**. **Application Master** может запрашивать контейнеры, используя информацию о местоположении, предоставляемую уровнем хранения.

В идеальном случае приложение размещается в субкластере, где доступны все ему необходимые ресурсы и данные, но если ему нужны контейнеры на узлах в других субкластерах, *AMRMProxy* прозрачно согласовывает с их **Resource Manager** и предоставляет ресурсы, что позволяет приложению рассматривать всю среду Federation как один массивный кластер **YARN**. *AMRMProxy*, *Global Policy Generator* и *Router* работают вместе для бесшовной реализации (Рис.4.8.).

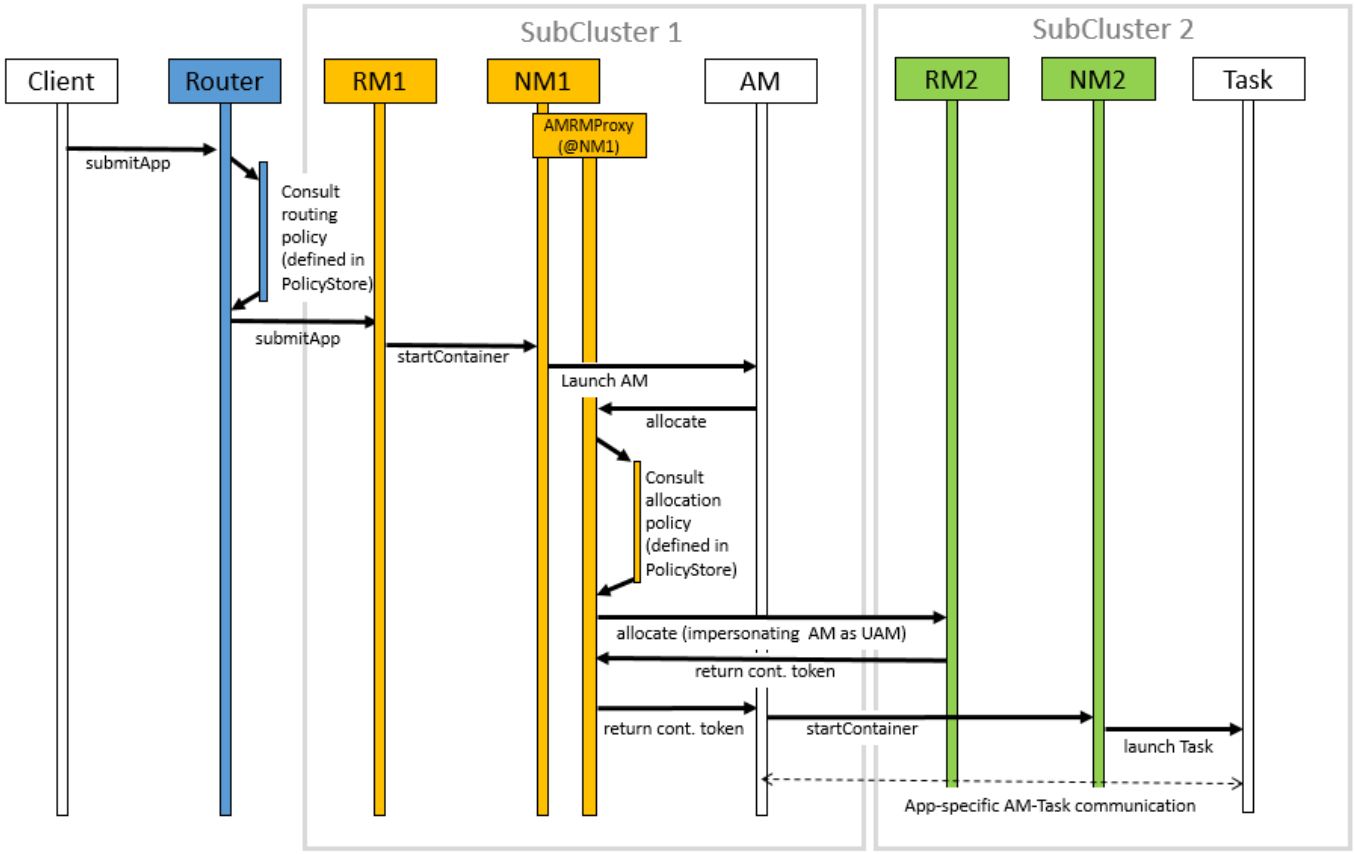


Рис.4.8.: Диаграмма последовательности

На рисунке показана диаграмма последовательности для следующего потока выполнения задания:

1. Router получает запрос на отправку приложения, являющийся жалобой на YARN Application Client Protocol.
2. Маршрутизатор опрашивает таблицу/политику маршрутизации, чтобы выбрать домашний Resource Manager для задания (конфигурация политики принимается из State Store по heartbeat-сообщению).
3. Маршрутизатор запрашивает состояние membership, чтобы определить конечную точку домашнего Resource Manager.
4. Затем маршрутизатор перенаправляет запрос на отправку приложения в домашний Resource Manager.

5. Маршрутизатор обновляет состояние приложения с помощью идентификатора домашнего субкластера.
6. Как только приложение отправляется в домашний Resource Manager, запускается поток YARN, то есть приложение добавляется в очередь планировщика, и его Application Master запускается в домашнем субкластере в первом NodeManager с доступными ресурсами.

- Во время этого процесса среда Application Master изменяется, указывая адрес AMRMProxy в качестве YARN Resource Manager для связи;
- Токены безопасности также изменяются NodeManager при запуске Application Master, так что Application Master может общаться только с AMRMProxy. Любые дальнейшие коммуникации от Application Master до YARN Resource Manager осуществляются посредством AMRMProxy.

7. Затем Application Master запрашивает контейнеры, используя информацию о местонахождении, предоставляемую HDFS.
8. На основе политики AMRMProxy может олицетворять Application Master в других субкластерах, отправляя Unmanaged Application Master и перенаправляя heartbeats-сообщения Application Master соответствующим субкластерам.

- Federation поддерживает несколько попыток приложения с помощью AMRMProxy HA. Контейнеры Application Master имеют разные идентификаторы попыток в домашнем субкластере, но один и тот же Unmanaged Application Master во вторичных;
- Когда AMRMProxy HA включен, токен Unmanaged Application Master хранится в Yarn Registry. При вызове `registerApplicationMaster` от каждой попытки приложения AMRMProxy извлекает существующие токены Unmanaged Application Master из реестра (если таковые имеются) и повторно подключается к существующим Unmanaged Application Master.

9. AMRMProxy использует как информацию о местонахождении, так и подключаемую политику, настроенную в State Store, чтобы решить, следует ли перенаправлять полученные от Application Master запросы ресурсов в домашний Resource Manager или во вторичный (один или более). На рисунке отображен случай, когда AMRMProxy решает переслать запрос на вторичный Resource Manager.
10. Вторичный Resource Manager предоставляет AMRMProxy актуальные токены контейнера для запуска нового контейнера на узле в его субкластере. Такой механизм гарантирует, что каждый субкластер использует свои собственные токены безопасности и избегает необходимости общего секрета кластера для создания токенов.
11. AMRMProxy пересылает ответ распределения обратно в Application Master.
12. Application Master запускает контейнер на целевом NodeManager (в субкластере 2), используя стандартные протоколы YARN.

4.6.3 Конфигурация

Настройка **YARN** для использования Federation осуществляется через ряд свойств в файле *conf/yarn-site.xml*.

Общие для всех

`yarn.federation.enabled` – включена Federation или нет. Пример значения:

```
true
```

`yarn.resourcemanager.cluster-id` – уникальный идентификатор субкластера для данного Resource Manager (такой же, что используется для HA). Пример значения:

```
<unique-subcluster-id>
```

State Store:

В настоящее время поддерживается реализации State Store на основе **ZooKeeper** и **SQL**.

Обязательные настройки **ZooKeeper** для **Hadoop**:

`yarn.federation.state-store.class` – тип State Store. Пример значения:

```
org.apache.hadoop.yarn.server.federation.store.impl.ZookeeperFederationStateStore
```

`hadoop.zk.address` – адрес для ансамбля ZooKeeper. Пример значения:

```
host:port
```

Обязательные параметры **SQL**:

`yarn.federation.state-store.class` – тип State Store. Пример значения:

```
org.apache.hadoop.yarn.server.federation.store.impl.SQLFederationStateStore
```

`yarn.federation.state-store.sql.url` – имя базы данных для SQLFederationStateStore, в которой хранится состояние. Пример значения:

```
jdbc:mysql://<host>:<port>/FederationStateStore
```

`yarn.federation.state-store.sql.jdbc-class` – используемый класс jdbc для SQLFederationStateStore. Пример значения:

```
com.mysql.jdbc.jdbc2.optional.MysqlDataSource
```

`yarn.federation.state-store.sql.username` – имя пользователя для соединения с БД для SQLFederationStateStore. Пример значения:

```
<dbuser>
```

`yarn.federation.state-store.sql.password` – пароль для подключения к БД для SQLFederationStateStore. Пример значения:

```
<dbpass>
```

Для **MySQL** и **Microsoft SQL Server** предоставляются скрипты.

Для **MySQL** необходимо загрузить последнюю версию *jar 5.x* из MVN Repository и добавить ее в *CLASSPATH*. Затем схема БД создается путем выполнения следующих скриптов SQL в базе данных:

- sbin/FederationStateStore/MySQL/FederationStateStoreDatabase.sql
- sbin/FederationStateStore/MySQL/FederationStateStoreUser.sql
- sbin/FederationStateStore/MySQL/FederationStateStoreTables.sql
- sbin/FederationStateStore/MySQL/FederationStateStoreStoredProcs.sql

В том же каталоге предоставляются скрипты для удаления хранимых процедур, таблиц, пользователя и базы данных.

Important: FederationStateStoreUser.sql определяет для БД пользователя/пароль по умолчанию, для которого настоятельно рекомендуется установить собственный надежный пароль

Для SQL-сервера процесс аналогичен, но драйвер *jdbc* уже включен. Скрипты SQL-сервера находятся в каталоге *sbin/FederationStateStore/SQLServer/*.

Optional:

`yarn.federation.failover.enabled` – следует ли повторить попытку, учитывая отказоустойчивость Resource Manager в каждом субкластере. Пример значения:

```
true
```

`yarn.federation.blacklist-subclusters` – список черных списков субкластеров, используемых для отключения субкластера. Пример значения:

```
<subcluster-id>
```

`yarn.federation.policy-manager` – выбор диспетчера политик, определяющий как Applications и ResourceRequests маршрутизируются через систему. Пример значения:

```
org.apache.hadoop.yarn.server.federation.policies.manager.WeightedLocalityPolicyManager
```

`yarn.federation.policy-manager-params` – полезная нагрузка, которая настраивает политику. В примере набор весов для политик маршрутизатора и AMRMProxy. Обычно генерируется путем сериализации policymanager, который был сконфигурирован программно, или путем заполнения State Store его сериализованной формой `.json`. Пример значения:

```
<binary>
```

`yarn.federation.subcluster-resolver.class` – класс, используемый для определения, к какому субкластеру принадлежит узел, и к какому субкластеру(ам) принадлежит стойка. Пример значения:

```
org.apache.hadoop.yarn.server.federation.resolver.DefaultSubClusterResolverImpl
```

`yarn.federation.machine-list` – путь к файлу со списком машин, используемых SubClusterResolver. Каждая строка файла представляет собой узел с информацией о субкластере и стойке (например: node1, subcluster1, rack1 / node2, subcluster2, rack1 / node3, subcluster3, rack2 / node4, subcluster3, rack2). Пример значения:

```
<path of machine-list file>
```

Resource Manager

Дополнительная конфигурация, которая должна отображаться в файле *conf/yarn-site.xml* в каждом **Resource Manager**.

`yarn.resourcemanager.epoch` – начальное значение для ряда идентификаторов контейнеров, гарантирующих уникальность container-IDs, генерируемых различными Resource Manager. Поэтому значение параметра должно быть уникальным среди субкластеров и быть достаточно разнесенным, чтобы учитывать сбои. Приращения *1000* допускают большое количество субкластеров и гарантируют практически нулевую вероятность коллизий (коллизия может произойти только в том случае, если контейнер все еще жив при 1000 перезапусках одного Resource Manager, в то время как следующий Resource Manager никогда не перезапускается, и приложение запрашивает больше контейнеров). Пример значения:

```
<unique-epoch>
```

Опционально:

`yarn.federation.state-store.heartbeat-interval-secs` – интервал частоты, с которой Resource Manager сообщают о своем membership в Federation центральному State Store. Пример значения:

```
60
```

Router

Дополнительные конфигурации, которые должны отображаться в файле *conf/yarn-site.xml* в каждом *Router*.

`yarn.router.bind-host` – IP-адрес хоста для привязки маршрутизатора. Фактический адрес, к которому привязывается сервер. Если этот адрес установлен, серверы RPC и webapp привязываются к нему и к указанному в yarn.router.*.address порту. Для того, чтобы маршрутизатор слушал все интерфейсы, рекомендуется значение:

```
0.0.0.0
```

`yarn.router.clientrm.interceptor-class.pipeline` – разделенный запятыми список классов перехватчиков, которые должны запускаться на маршрутизаторе при взаимодействии с клиентом. Последним этапом этого конвейера должен быть Federation Client Interceptor. Пример значения:

```
org.apache.hadoop.yarn.server.router.clientrm.FederationClientInterceptor
```

Опционально:

`yarn.router.hostname` – имя хоста маршрутизатора. Пример значения:

```
0.0.0.0
```

`yarn.router.clientrm.address` – адрес клиента маршрутизатора. Пример значения:

```
0.0.0.0:8050
```

`yarn.router.webapp.address` – адрес веб-приложения на маршрутизаторе. Пример значения:

```
0.0.0.0:8089
```

`yarn.router.admin.address` – админ-адрес на маршрутизаторе. Пример значения:

```
0.0.0.0:8052
```

`yarn.router.webapp.https.address` – безопасный адрес веб-приложения на маршрутизаторе. Пример значения:

```
0.0.0.0:8091
```

`yarn.router.submit.retry` – количество попыток в маршрутизаторе, перед отказом. Пример значения:

```
3
```

`yarn.federation.statestore.max-connections` – максимальное количество параллельных подключений, которые каждый маршрутизатор устанавливает в State Store. Пример значения:

```
10
```

`yarn.federation.cache-ttl.secs` – маршрутизатор кеширует информацию, и это время, чтобы уйти до того, как кеш становится недействительным. Пример значения:

```
60
```

`yarn.router.webapp.interceptor-class.pipeline` – разделенный запятыми список классов перехватчиков, которые должны запускаться на маршрутизаторе при взаимодействии с клиентом через интерфейс REST. Последним этапом этого конвейера должен быть Federation Interceptor REST. Пример значения:

```
org.apache.hadoop.yarn.server.router.webapp.FederationInterceptorREST
```

NodeManager

Дополнительные конфигурации, которые должны отображаться в файле *conf/yarn-site.xml* в каждом **NodeManager**.

`yarn.nodemanager.amrmproxy.enabled` – определяет, включен ли AMRMProxy. Пример значения:

```
true
```

`yarn.nodemanager.amrmproxy.interceptor-class.pipeline` – разделенный запятыми список перехватчиков, которые необходимо запустить в AMRMProxy. Для Federation последним этапом этого конвейера должен быть FederationInterceptor. Пример значения:

```
org.apache.hadoop.yarn.server.nodemanager.amrmproxy.FederationInterceptor
```

Опционально:

`yarn.nodemanager.amrmproxy.ha.enable` – определяет, включен ли AMRMProxy HA для поддержки нескольких попыток приложения. Пример значения:

```
true
```

`yarn.federation.statestore.max-connections` – максимальное количество параллельных подключений от каждого AMRMProxy к State Store. Это значение обычно ниже, чем у маршрутизатора, поскольку всегда есть много AMRMProxy, которые могут быстро прожечь многие соединения с БД. Пример значения:

```
1
```

`yarn.federation.cache-ttl.secs` – время для кэша AMRMProxy. Это значение обычно больше, чем у маршрутизатора, так как количество AMRMProxy велико, и целесообразно ограничить нагрузку центральным State Store. Пример значения:

```
300
```

4.6.4 Запуск тестового задания

Для отправки заданий в кластер Federation необходимо создать отдельный набор конфигураций для клиента, из которого будут отправляться задания. В них *conf/yarn-site.xml* должен иметь следующие дополнительные конфигурации:

`yarn.resourcemanager.address` – перенаправляет запущенные на клиенте задания на клиентский порт маршрутизатора Resource Manager. Пример значения:

```
<router_host>:8050
```

`yarn.resourcemanager.scheduler.address` – перенаправляет задания на порт федерации AMRMProxy. Пример значения:

```
localhost:8049
```

Любые задания **YARN** для кластера могут быть отправлены из описанных выше конфигураций клиента. Чтобы запустить задание через Federation, сначала необходимо запустить все участвующие в ней кластеры. Затем выполнить старт маршрутизатора на компьютере маршрутизатора с помощью команды:

```
$HADOOP_HOME/bin/yarn --daemon start router
```

Теперь, когда *$HADOOP_CONF_DIR* указывает на папку конфигураций клиента, необходимо запустить задание обычным способом. Конфигурации направляют задание на клиентский порт маршрутизатора **Resource Manager**, где Router должен прослушиваться после запуска. Пример запуска задания *Pi* на кластере Federation с клиента:

```
$HADOOP_HOME/bin/yarn jar hadoop-mapreduce-examples-3.0.0.jar pi 16 1000
```

Задание передается на маршрутизатор, который использует сгенерированную политику из *GPG*, чтобы выбрать домашний **Resource Manager** для задания.

Выходные данные приведенного примера задания должны быть примерно такими:

```
2017-07-13 16:29:25,055 INFO mapreduce.Job: Job job_1499988226739_0001 running in uber mode : false
2017-07-13 16:29:25,056 INFO mapreduce.Job:  map 0% reduce 0%
2017-07-13 16:29:33,131 INFO mapreduce.Job:  map 38% reduce 0%
2017-07-13 16:29:39,176 INFO mapreduce.Job:  map 75% reduce 0%
2017-07-13 16:29:45,217 INFO mapreduce.Job:  map 94% reduce 0%
2017-07-13 16:29:46,228 INFO mapreduce.Job:  map 100% reduce 100%
2017-07-13 16:29:46,235 INFO mapreduce.Job: Job job_1499988226739_0001 completed successfully
.
.
.
Job Finished in 30.586 seconds
Estimated value of Pi is 3.14250000......
```

Состояние задания также можно отслеживать в веб-интерфейсе маршрутизатора по адресу *routerhost:8089*.

Важно обратить внимание, что для использования Federation не потребовалось никаких изменений в коде или перекомпиляции входного jar. Кроме того, выходные данные приведенного задания такие же, как и при запуске без Federation. Чтобы получить все преимущества Federation, рекомендуется использовать большее количество mappers, чем того требует кластер. Для приведенного примера это число составляет *16*.

4.7 Запуск приложений с использованием Docker-контейнеров

Important: Включение функции и запуск Docker-контейнеров в кластере имеет последствия для безопасности. Учитывая интеграцию Docker со многими мощными функциями ядра, крайне важно, чтобы администраторы понимали безопасность Docker, прежде чем включать данный функционал

Docker сочетает в себе простой в использовании интерфейс с контейнерами **Linux** и простые в создании image-файлы для них. В общем получается, что Docker позволяет пользователям создавать бандлы (bundle) – то есть связывать приложение с его предпочтительной средой выполнения для исполнения на целевой машине. Дополнительные сведения о Docker приведены в документации.

Механизм **Linux Container Executor** (**LCE**) позволяет **YARN NodeManager** приводить в действие YARN-контейнеры для запуска непосредственно на хост-машине либо внутри Docker-контейнеров. Приложение, запрашивающее ресурсы, может указать для каждого контейнера, как оно должно выполняться. **LCE** также обеспечивает повышенную безопасность, и поэтому он требуется при развертывании кластера. Когда **LCE**

запускает YARN-контейнер для выполнения в Docker, приложение может указать используемый образ (image) Docker.

Docker-контейнеры предоставляют кастомную среду выполнения, в которой запускается код приложения, изолированный от среды выполнения **NodeManager** и других приложений. Эти контейнеры могут включать специальные необходимые для приложения библиотеки, и они могут иметь различные версии собственных инструментов и библиотек, включая **Perl**, **Python** и **Java**. Docker-контейнеры могут даже работать с другим типом **Linux**, нежели тот, что работает на **NodeManager**.

Docker для **YARN** обеспечивает как согласованность (все YARN-контейнеры имеют одинаковую программную среду), так и изоляцию (без вмешательства в то, что установлено на физической машине).

4.7.1 Конфигурация кластера

LCE требует, чтобы бинарный файл container-executor принадлежал *root:hadoop* и имел разрешения *6050*. Для запуска Docker-контейнеров, демон (daemon) Docker должен быть запущен на всех хостах **NodeManager**, где планируют запускаться docker-контейнеры. Docker-клиент также должен быть установлен на всех хостах **NodeManager**, на которых планируют запускаться docker-контейнеры, должен быть запущен и иметь возможность старта Docker-контейнеров.

Для предотвращения тайм-аутов при запуске заданий любые большие Docker-образы, которые планируют использоваться приложением, уже должны быть загружены в кэш docker-демона на хостах **NodeManager**. Простой способ загрузить образ – выполнить pull-запрос. Например:

```
sudo docker pull library/openjdk:8
```

Следующие свойства должны быть установлены в *yarn-site.xml*:

```
<configuration>
  <property>
    <name>yarn.nodemanager.container-executor.class</name>
    <value>org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor</value>
    <description>
      This is the container executor setting that ensures that all applications
      are started with the LinuxContainerExecutor.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.linux-container-executor.group</name>
    <value>hadoop</value>
    <description>
      The POSIX group of the NodeManager. It should match the setting in
      "container-executor.cfg". This configuration is required for validating
      the secure access of the container-executor binary.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.linux-container-executor.nonsecure-mode.limit-users</name>
    <value>false</value>
    <description>
      Whether all applications should be run as the NodeManager process' owner.
      When false, applications are launched instead as the application owner.
    </description>
  </property>

  <property>
```

```
  <name>yarn.nodemanager.runtime.linux.allowed-runtimes</name>
  <value>default,docker</value>
  <description>
    Comma separated list of runtimes that are allowed when using
    LinuxContainerExecutor. The allowed values are default, docker, and
    javasandbox.
  </description>
</property>

<property>
  <name>yarn.nodemanager.runtime.linux.docker.allowed-container-networks</name>
  <value>host,none,bridge</value>
  <description>
    Optional. A comma-separated set of networks allowed when launching
    containers. Valid values are determined by Docker networks available from
    `docker network ls`
  </description>
</property>

<property>
  <name>yarn.nodemanager.runtime.linux.docker.default-container-network</name>
  <value>host</value>
  <description>
    The network used when launching Docker containers when no
    network is specified in the request. This network must be one of the
    (configurable) set of allowed container networks.
  </description>
</property>

<property>
  <name>yarn.nodemanager.runtime.linux.docker.host-pid-namespace.allowed</name>
  <value>false</value>
  <description>
    Optional. Whether containers are allowed to use the host PID namespace.
  </description>
</property>

<property>
  <name>yarn.nodemanager.runtime.linux.docker.privileged-containers.allowed</name>
  <value>false</value>
  <description>
    Optional. Whether applications are allowed to run in privileged
    containers. Privileged containers are granted the complete set of
    capabilities and are not subject to the limitations imposed by the device
    cgroup controller. In other words, privileged containers can do almost
    everything that the host can do. Use with extreme care.
  </description>
</property>

<property>
  <name>yarn.nodemanager.runtime.linux.docker.delayed-removal.allowed</name>
  <value>false</value>
  <description>
    Optional. Whether or not users are allowed to request that Docker
    containers honor the debug deletion delay. This is useful for
    troubleshooting Docker container related launch failures.
  </description>
</property>
```

```
  <property>
    <name>yarn.nodemanager.runtime.linux.docker.stop.grace-period</name>
    <value>10</value>
    <description>
      Optional. A configurable value to pass to the Docker Stop command. This
      value defines the number of seconds between the docker stop command sending
      a SIGTERM and a SIGKILL.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.runtime.linux.docker.privileged-containers.acl</name>
    <value></value>
    <description>
      Optional. A comma-separated list of users who are allowed to request
      privileged contains if privileged containers are allowed.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.runtime.linux.docker.capabilities</name>
    <value>CHOWN,DAC_OVERRIDE,FSETID,FOWNER,MKNOD,NET_RAW,SETGID,SETUID,SETFCAP,SETPCAP,NET_BIND_SERVICE,
↪SYS_CHROOT,KILL,AUDIT_WRITE</value>
    <description>
      Optional. This configuration setting determines the capabilities
      assigned to docker containers when they are launched. While these may not
      be case-sensitive from a docker perspective, it is best to keep these
      uppercase. To run without any capabilites, set this value to
      "none" or "NONE"
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.runtime.linux.docker.enable-userremapping.allowed</name>
    <value>true</value>
    <description>
      Optional. Whether docker containers are run with the UID and GID of the
      calling user.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.runtime.linux.docker.userremapping-uid-threshold</name>
    <value>1</value>
    <description>
      Optional. The minimum acceptable UID for a remapped user. Users with UIDs
      lower than this value will not be allowed to launch containers when user
      remapping is enabled.
    </description>
  </property>

  <property>
    <name>yarn.nodemanager.runtime.linux.docker.userremapping-gid-threshold</name>
    <value>1</value>
    <description>
      Optional. The minimum acceptable GID for a remapped user. Users belonging
      to any group with a GID lower than this value will not be allowed to
```

```
      launch containers when user remapping is enabled.
    </description>
  </property>

</configuration>
```

Кроме того, файл *container-executer.cfg* должен существовать и содержать настройки для исполнителя контейнера. Файл должен принадлежать пользователю *root* с разрешениями *0400*. Формат файла – это стандартный формат файла свойств **Java**, например:

```
`key=value`
```

Для включения поддержки Docker требуются следующее свойство:

`yarn.nodemanager.linux-container-executor.group` – группа Unix для NodeManager. Должно соответствовать `yarn.nodemanager.linux-container-executor.group` в файле *yarn-site.xml*.

Файл *container-executor.cfg* должен содержать раздел, чтобы определить возможности, которые разрешены для контейнеров. Содержит следующие свойства:

`module.enabled` – значение должно быть *true* или *false*, чтобы включить или отключить запуск Docker-контейнеров. Значение по умолчанию *0*;

`docker.binary` – бинарный файл, используемый для запуска docker-контейнеров. По умолчанию */usr/bin/docker*;

`docker.allowed.capabilities` – разделенные запятыми возможности, которые могут добавлять контейнеры. По умолчанию никакие возможности не могут быть ими добавлены;

`docker.allowed.devices` – разделенные запятыми устройства, для которых разрешено устанавливаться к контейнерам. По умолчанию никакие устройства не могут быть добавлены;

`docker.allowed.networks` – разделенные запятыми сети, которые разрешено использовать контейнерам. Если при запуске контейнера сеть не указана, используется сеть Docker по умолчанию;

`docker.allowed.ro-mounts` – разделенные запятыми каталоги, которые контейнеры могут устанавливать в режиме только для чтения. По умолчанию никакие каталоги не разрешено монтировать;

`docker.allowed.rw-mounts` – разделенные запятыми каталоги, которые контейнеры могут устанавливать в режиме чтения-записи. По умолчанию никакие каталоги не разрешено монтировать;

`docker.allowed.volume-drivers` – разделенный запятыми список драйверов тома Docker, которые разрешено использовать. По умолчанию никакие драйверы тома не разрешены;

`docker.host-pid-namespace.enabled` – значение должно быть *true* или *false*, чтобы включить или отключить использование хостом PID namespace. Значением по умолчанию является *false*;

`docker.privileged-containers.enabled` – значение должно быть *true* или *false*, чтобы включить или отключить запуск привилегированных контейнеров. Значением по умолчанию является *false*;

`docker.trusted.registries` – разделенный запятыми список реестров доверенного докера для запуска доверенных привилегированных docker-контейнеров. По умолчанию реестры не определены;

`docker.inspect.max.retries` – значение Integer для проверки готовности docker-контейнера. Каждая проверка устанавливается с отсрочкой *3* секунды. Значение по умолчанию, равное *10*, ожидает *30* секунд, пока Docker-контейнер не станет готов, прежде чем пометить его как сбой;

`docker.no-new-privileges.enabled` – значение должно быть *true* или *false*, чтобы включить или отключить флаг *no-new-privileges* для запуска Docker. Значением по умолчанию является *false*;

`docker.allowed.runtimes` – разделенные запятыми среды выполнения, которые разрешено использовать контейнерам. По умолчанию никакая среда выполнения не может быть добавлена.

Important: При необходимости запуска Docker-контейнеров, которым требуется доступ к локальным каталогам YARN, следует добавить их в список *docker.allowed.rw-mounts*

Кроме того, контейнерам не разрешается устанавливать любого родителя каталога *container-executor.cfg* в режиме чтения-записи.

Следующие свойства являются опциональными:

`min.user.id` – минимальный UID, разрешенный для запуска приложений. По умолчанию минимум не установлен;

`banned.users` – разделенный запятыми список имен пользователей, которым нельзя разрешать запуск приложений. Значение по умолчанию: *yarn*, *mapred*, *hdfs* и *bin*;

`allowed.system.users` – разделенный запятыми список имен пользователей, которым следует разрешать запуск приложений, даже если их UID ниже настроенного минимума. Если пользователь указан в `allowed.system.users` и `banned.users`, он считается забаненным;

`feature.tc.enabled` – значение должно быть *true* или *false*, чтобы включить или отключить команды управления движением (traffic control commands).

Фрагмент *container-executor.cfg*, который позволяет запускать docker-контейнеры:

```
yarn.nodemanager.linux-container-executor.group=yarn
[docker]
  module.enabled=true
  docker.privileged-containers.enabled=true
  docker.trusted.registries=centos
  docker.allowed.capabilities=SYS_CHROOT,MKNOD,SETFCAP,SETPCAP,FSETID,CHOWN,AUDIT_WRITE,SETGID,NET_RAW,
↪FOWNER,SETUID,DAC_OVERRIDE,KILL,NET_BIND_SERVICE
  docker.allowed.networks=bridge,host,none
  docker.allowed.ro-mounts=/sys/fs/cgroup
  docker.allowed.rw-mounts=/var/hadoop/yarn/local-dir,/var/hadoop/yarn/log-dir
```

4.7.2 Требования Docker-образа

Для работы с **YARN** существует два требования к docker-образам.

Во-первых, docker-контейнер явно запускается с владельцем приложения в качестве пользователя контейнера. Если владелец приложения не является валидным пользователем в docker-образе, приложение завершается ошибкой. Пользователь контейнера указывается в UID. Если UID пользователя в **NodeManager** и в docker-образе отличается, контейнер может быть запущен как неправильный пользователь или может не запуститься вовсе, так как UID не существует (подробнее в \T2A\CYRU\T2A\cyrp\T2A\cyrr\T2A\cyra\T2A\cyrv\T2A\cyrl\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyre\T2A\cyrp\T2A\cyro\T2A\cyrl\T2A\cyrsftsn\T2A\cyrz\T2A\cyro\T2A\cyrv\T2A\cyra\T2A\cyrt\T2A\cyre\T2A\cyrl\T2A\cyrya\T2A\cyrm\T2A\cyri).

Во-вторых, docker-образ должен иметь то, что ожидает приложение для выполнения. В случае **Hadoop** (**MapReduce** или **Spark**) Docker-образ должен содержать библиотеки *JRE* и *Hadoop* и иметь набор необходимых переменных окружения: *JAVA_HOME*, *HADOOP_COMMON_PATH*, *HADOOP_HDFS_HOME*, *HADOOP_MAPRED_HOME*, *HADOOP_YARN_HOME*, *HADOOP_CONF_DIR*. Важно обратить внимание, что доступные в docker-образе версии компонентов **Java** и **Hadoop** должны быть совместимы с тем, что установлено в кластере, и с любыми другими docker-образами, используемыми для других задач того же задания. В противном случае компоненты **Hadoop**, запущенные в docker-контейнере, не смогут взаимодействовать с внешними компонентами **Hadoop**.

Если docker-образ имеет набор команд command, поведение зависит от того, установлено ли значение параметра `YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE` в *true*. Если это так, то команда переопределяется, когда **LCE** запускает образ с помощью скрипта запуска контейнера **YARN**.

Если для docker-образа задана точка входа и для параметра `YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE` установлено значение *true*, команда *launch_command* передается в программу *ENTRYPOINT* в качестве параметров CMD в Docker. Формат *launch_command* выглядит следующим образом: `param1,param2`, что приводит к CMD `[ “param1”,“param2” ]` в Docker.

Если приложение запрашивает docker-образ, который еще не загружен Docker-демоном на хосте, где он должен выполняться, Docker-демон неявно выполняет pull-команду. **MapReduce** и **Spark** предполагают, что задачи, для отчета о которых требуется более 10 минут, остановились, поэтому указание большого Docker-образа может привести к сбою приложения.

4.7.3 Отправка приложения

Перед запуском docker-контейнера необходимо убедиться, что конфигурация **LCE** работает для приложений, запрашивающих обычные YARN-контейнеры. Если после включения **LCE** не удается запустить один или несколько **NodeManager**, скорее всего причина в том, что владение и/или разрешения для бинарного файла *container-executer* неверны. Тогда следует проверить журналы.

Для запуска приложения в docker-контейнере необходимо установить следующие переменные среды в среде приложения (первые два обязательны, остальная часть может быть установлена по мере необходимости):

`YARN_CONTAINER_RUNTIME_TYPE` – определяет, будет ли приложение запущено в Docker-контейнере. Если значение установлено на *docker*, приложение запускается в Docker-контейнере. В противном случае используется обычный контейнер дерева процессов;

`YARN_CONTAINER_RUNTIME_DOCKER_IMAGE` – имена, образ которых используется для запуска Docker-контейнера. Можно использовать любое имя образа, которое можно передать команде запуска Docker-клиента. Имя образа может включать repo-префикс;

`YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE` – управляет переопределением команды по умолчанию Docker-контейнера. При установленном значении *true* команда Docker-контейнера является `bash path_to_launch_script`. Если параметр не задан или установлено *false*, используется команда по умолчанию;

`YARN_CONTAINER_RUNTIME_DOCKER_CONTAINER_NETWORK` – устанавливает тип сети для использования Docker-контейнером. Это должно быть валидное значение, определенное свойством `yarn.nodemanager.runtime.linux.docker.allowed-container-networks`;

`YARN_CONTAINER_RUNTIME_DOCKER_PORTS_MAPPING` – позволяет пользователю указать маппинг портов для сетевого моста Docker-контейнера. Значением переменной среды должен быть разделенный запятыми список портов. Аналогично опции `-p` для команды запуска Docker. Если значение не установлено, указывается `-P`;

`YARN_CONTAINER_RUNTIME_DOCKER_CONTAINER_PID_NAMESPACE` – определяет, какое пространство имен PID используется Docker-контейнером. По умолчанию каждый Docker-контейнер имеет свое собственное пространство имен PID. Для совместного использования пространства имен на хосте необходимо установить для свойства `yarn.nodemanager.runtime.linux.docker.host-pid-namespace.allowed` значение *true*. При разрешенном пространстве имен PID на хосте и заданном значении для данной переменной среды *host* Docker-контейнер использует пространство имен PID хоста. Другие значения не допускаются, поэтому при необходимости переменную следует оставить неустановленной, а не задавать ей значение *false*;

`YARN_CONTAINER_RUNTIME_DOCKER_RUN_PRIVILEGED_CONTAINER` – определяет, является ли Docker-контейнер привилегированным контейнером. Чтобы использовать привилегированные контейнеры, для свойства `yarn.nodemanager.runtime.linux.docker.privileged-containers.allowed` должно быть установлено значение *true*, а владелец приложения должен отображаться в значении `yarn.nodemanager.runtime.linux.docker.privileged-containers.acl`. Если для данной переменной среды задано значение *true*, то используется привилегированный Docker-контейнер, если это разрешено. Другие значения не допускаются, поэтому при необходимости переменную окружения следует оставить неустановленной, а не задавать ей значение *false*;

`YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS` – добавляет дополнительный том в Docker-контейнер. Значением переменной среды должен быть список точек монтирования через запятую. Все такие монтирования

должны быть указаны как `source:dest[:mode]`, а чтобы указать тип запрашиваемого доступа режим должен быть `ro` (read-only) или `rw` (read-write). Если ни то, ни другое не указано, принимается режим чтение-запись. Режим может включать опцию распространения привязки (bind propagation). В этом случае режим должен иметь вид `[option]`, `rw+[option]` или `ro+[option]`. Допустимые варианты распространения привязки: *shared*, *rshared*, *slave*, *rslave*, *private* и *rprivate*. Запрашиваемые монтирования проверяются на основе значений, установленных в *container-executor.cfg* для `docker.allowed.ro-mounts` и `docker.allowed.rw-mounts`;

`YARN_CONTAINER_RUNTIME_DOCKER_TMPFS_MOUNTS` – добавляет дополнительные *tmpfs* в docker-контейнер. Значением переменной среды должен быть список разделенных запятыми абсолютных точек монтирования в контейнере;

`YARN_CONTAINER_RUNTIME_DOCKER_DELAYED_REMOVAL` – позволяет пользователю запрашивать отложенное удаление Docker-контейнера на основе каждого контейнера. Если установлено значение *true*, docker-контейнеры не удаляются до тех пор, пока не истечет время, определенное параметром `yarn.nodemanager.delete.debug-delay-sec`. Администраторы могут отключить эту функцию через yarn-site свойство `yarn.nodemanager.runtime.linux.docker.delayed-removal.allowed`. По умолчанию функция отключена. При отключенной функции или при значении параметра *false* контейнер удаляется, как только он выйдет.

Когда приложение отправляется для запуска в Docker-контейнере, оно ведет себя точно так же, как и любое другое приложение **YARN**. Журналы агрегируются и сохраняются на соответствующем сервере истории. Жизненный цикл приложения остается таким же, как и для приложения, не являющегося Docker.

4.7.4 Использование Docker Bind Mounted Volumes

Important: Включение доступа к каталогам, таким как /, /etc, /run или /home, не рекомендуется и может привести к тому, что контейнеры негативно повлияют на хост или приведут к утечке конфиденциальной информации

Файлы и каталоги с хоста обычно необходимы в Docker-контейнерах, которые Docker предоставляет через тома volumes. Примеры включают локализованные ресурсы, бинарные файлы **Apache Hadoop** и сокеты. Чтобы облегчить эту потребность, в *YARN-6623* добавлена возможность для администраторов устанавливать белый список каталогов хоста, который выполняет *bind mounted* в виде томов в контейнерах. В *YARN-5534* добавлена возможность предоставления пользователям списка монтирований в контейнеры, если это разрешено белым списком администратора.

Для использования этой функции необходимо:

- Администратор должен определить белый список томов в *container-executor.cfg*, установив `docker.allowed.ro-mounts` и `docker.allowed.rw-mounts` в список родительских каталогов, которые могут быть монтированы;
- Отправитель приложения запрашивает необходимые тома во время отправки приложения, используя переменную среды `YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS`.

Предоставленный администратором белый список определяется как разделенный запятыми список каталогов, которые разрешено монтировать в контейнеры. Исходный каталог, предоставленный пользователем, должен совпадать или быть дочерним по отношению к указанному каталогу.

Предоставленный пользователем список монтирования определяется как разделенный запятыми список в форме `source:destination` или `source:destination:mode`. Источником является файл или каталог на хосте. Пункт назначения – это путь через контейнер, по которому осуществляется *bind mounted*. Режим определяет режим, который пользователь ожидает для монтирования, который может быть `ro` (read-only) или `rw` (read-write). Если ни то, ни другое не указано, принимается режим чтение-запись. Режим может включать опцию распространения привязки (bind propagation). Допустимые варианты: *shared*, *rshared*, *slave*, *rslave*, *private* и *rprivate*. В этом случае режим должен иметь вид `option`, `rw+option` или `ro+option`.

Далее показано, как использовать эту функцию для монтирования обычно необходимого каталога */sys/fs/cgroup* в запущенный на **YARN** контейнер.

Администратор устанавливает `docker.allowed.ro-mounts` в *container-executor.cfg* в *"/sys/fs/cgroup"*. После этого приложения могут запросить монтирование *"/sys/fs/cgroup"* с хоста в контейнер в read-only режиме. Во время отправки приложения можно установить переменную среды `YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS` для запроса этого монтирования. В данном примере переменная окружения устанавливается на *"/sys/fs/cgroup:/sys/fs/cgroup:ro"*, при этом путь назначения не ограничен, и *"/sys/fs/cgroup:/cgroup:ro"* также валиден с учетом приведенного в пример белого списка администратора.

4.7.5 Управление пользователями

Поддержка docker-контейнеров **YARN** запускает контейнерные процессы с использованием идентификатора пользователя *uid:gid*, определенного на хосте **NodeManager**. Несоответствие имени пользователя и группы между хостом **NodeManager** и контейнером может привести к проблемам с разрешениями, неудачным запускам контейнера или даже к пробелам в безопасности. Централизованное управление пользователями и группами как для хостов, так и для контейнеров значительно снижает эти риски. При запуске контейнерных приложений в **YARN** необходимо понимать, какая пара *uid:gid* будет использоваться для запуска процесса контейнера.

По умолчанию в небезопасном режиме non-secure **YARN** запускает процессы как пользователь *nobody*. В системах на базе **CentOS** uid пользователя *nobody* равен *99*, и группы *nobody* тоже *99*. В результате **YARN** вызывает `docker run` с `--user 99:99`. Если у пользователя *nobody* нет *uid 99* в контейнере, запуск может завершиться неудачей или привести к неожиданным результатам.

Единственным исключением из этого правила является использование Docker-контейнеров Privileged. Привилегированные контейнеры не устанавливают пару *uid:gid* при запуске контейнера и учитывают записи *USER* и *GROUP* в Dockerfile. Это позволяет запускать привилегированные контейнеры как любой пользователь, но имеет последствия для безопасности.

Есть много способов управления пользователями и группами. По умолчанию Docker аутентифицирует пользователей по */etc/passwd* (и */etc/shadow*) внутри контейнера. Но использование */etc/passwd*, представленного в docker-образе, вряд ли содержит соответствующие записи пользователя и скорее приведет к сбоям запуска. Поэтому настоятельно рекомендуется централизовать управление пользователями и группами. Несколько подходов к управлению пользователями и группами описано далее.

Статическое управление

Основным подходом к управлению пользователями и группами является изменение пользователя и группы в Docker-образе. Этот подход возможен только в небезопасном режиме, когда все контейнерные процессы запускаются как один известный пользователь, например, *nobody*. В этом случае единственным требованием является соответствие пары *uid:gid* пользователя и группы *nobody* между хостом и контейнером. В системе на базе **CentOS** это означает, что пользователю *nobody* в контейнере нужен *UID 99*, а группе *nobody* в контейнере нужен *GID 99*.

Один из подходов к изменению UID и GID заключается в использовании *usermod* и *groupmod*. Установка правильных UID и GID для пользователя/группы *nobody*:

```
usermod -u 99 nobody
groupmod -g 99 nobody
```

Данный подход не рекомендуется использовать после тестирования, учитывая негибкость добавления пользователей.

Bind mounting

Когда в компании уже имеется автоматизация для создания локальных пользователей в каждой системе, может быть целесообразно выполнить bind mount */etc/passwd* и */etc/group* в контейнер в качестве

альтернативы непосредственному изменению образа контейнера. Для подключения возможности bind mount */etc/passwd* и */etc/group* необходимо обновить `docker.allowed.ro-mounts` в *container-executor.cfg*, чтобы включить эти пути. Затем при отправке приложения `YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS` должен содержать */etc/passwd:/etc/passwd:ro* и */etc/group:/etc/group:ro*.

У данного подхода bind mount есть пара проблем, которые необходимо учитывать:

- Любые пользователи и группы, определенные в образе, перезаписываются пользователями и группами хоста;
- После запуска контейнера нельзя добавлять пользователей и группы, так как */etc/passwd* и */etc/group* неизменны в контейнере. Не рекомендуется монтировать эти файлы для чтения и записи, так как это может привести к неработоспособности хоста.

Данный подход не рекомендуется использовать после тестирования, учитывая негибкость модификации запущенных контейнеров.

SSSD

Альтернативным подходом, позволяющим централизованно управлять пользователями и группами, является SSSD – System Security Services Daemon, предоставляющий доступ к различным поставщикам аутентификации, таким как **LDAP** и **Active Directory**.

Традиционная схема для аутентификации **Linux**:

```
application -> libpam -> pam_authenticate -> pam_unix.so -> /etc/passwd
```

При использовании SSSD для user-lookup схема принимает вид:

```
application -> libpam -> pam_authenticate -> pam_sss.so -> SSSD -> pam_unix.so -> /etc/passwd
```

Можно выполнить bind-mount UNIX-сокетов к контейнеру через SSSD коммуникации. Это позволяет библиотекам на стороне клиента SSSD проходить аутентификацию на SSSD, запущенном на хосте. В результате пользовательская информация не должна существовать в */etc/passwd* docker-образа, а вместо этого обслуживается SSSD.

Пошаговая настройка хоста и контейнера:

1. Конфигурация хоста:

- Установка пакетов:

```
# yum -y install sssd-common sssd-proxy
```

- Создание PAM-сервиса для контейнера:

```
# cat /etc/pam.d/sss_proxy
auth required pam_unix.so
account required pam_unix.so
password required pam_unix.so
session required pam_unix.so
```

- Создание концигурационного файла SSSD */etc/sssd/sssd.conf*. Важно обратить внимание, что разрешения должны быть *0600*, а файл должен принадлежать пользователю *root:root*:

```
# cat /etc/sssd/sssd/conf
[sssd]
services = nss,pam
config_file_version = 2
domains = proxy
```

```
[nss]
[pam]
[domain/proxy]
id_provider = proxy
proxy_lib_name = files
proxy_pam_target = sss_proxy
```

- Запуск SSSD:

```
# systemctl start sssd
```

- Проверка, что пользователь может быть извлечен с помощью SSSD:

```
# getent passwd -s sss localuser
```

2. Настройка контейнера. Важно выполнить bind-mount каталога */var/lib/sss/pipes* от хоста к контейнеру, так как SSSD UNIX сокеты находятся там:

```
-v /var/lib/sss/pipes:/var/lib/sss/pipes:rw
```

3. Конфигурация контейнера. Все шаги выполняются на самом контейнере:

- Установка только клиентских библиотек sss:

```
# yum -y install sssd-client
```

- Проверка, что sss настроена для баз данных *passwd* и *group*:

```
/etc/nsswitch.conf
```

- Настройка PAM-сервиса, используемого приложением для вызова в SSSD:

```
# cat /etc/pam.d/system-auth
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        sufficient    pam_unix.so try_first_pass nullok
auth        sufficient    pam_sss.so forward_pass
auth        required      pam_deny.so

account     required      pam_unix.so
account     [default=bad success=ok user_unknown=ignore] pam_sss.so
account     required      pam_permit.so

password    requisite     pam_pwquality.so try_first_pass local_users_only retry=3 authtok_type=
password    sufficient    pam_unix.so try_first_pass use_authtok nullok sha512 shadow
password    sufficient    pam_sss.so use_authtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session     optional      pam_systemd.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
session     optional      pam_sss.so
```

- Сохранение docker-образа и последующее использование его в качестве базового образа приложений.

- Тестирование Docker-образа, запущенного в среде YARN:

```
$ id
uid=5000(localuser) gid=5000(localuser) groups=5000(localuser),1337(hadoop)
```

4.7.6 Безопасность привилегированного контейнера

Docker-контейнер Privileged может взаимодействовать с устройствами хост-системы, но без надлежащей осторожности это может нанести вред операционной системе хоста. Чтобы снизить риск запуска привилегированного контейнера в кластере **Hadoop**, внедрен управляемый процесс для песочницы неавторизованных привилегированных Docker-образов.

Поведение по умолчанию запрещает любые docker-контейнеры Privileged. Когда для `docker.privileged-containers.enabled` установлено значение *enabled*, docker-образ может запускаться с правами *root* в docker-контейнере, но доступ к устройствам уровня хоста отключен. Это позволяет разработчикам и тестировщикам запускать docker-образы из Интернета, не причиняя вреда операционной системе хоста.

В случае когда docker-образы сертифицированы разработчиками и тестировщиками как заслуживающие доверия, такие образы могут быть переведены в реестр доверенных докеров (trusted docker registry). И системный администратор может определить `docker.trusted.registries` и настроить частный сервер docker-registry для поддержки таких доверенных образов.

Доверенные образы могут монтироваться к внешним устройствам, таким как **HDFS**, через протокол **NFS gateway** или конфигурацию **Hadoop** на уровне хоста. Если системные администраторы разрешают запись на внешние тома с помощью директивы `docker.allow.rw-mounts`, docker-контейнер Privileged может иметь полный контроль над файлами уровня хоста в предопределенных томах.

4.7.7 Требования к перезапуску контейнера

При рестарте **NodeManager**, как часть процесса восстановления, удостоверяет, что контейнер все еще запущен, проверив наличие PID-каталога контейнера в файловой системе */proc*. В целях безопасности администратор операционной системы может включить параметр монтирования *hidepid* для файловой системы */proc*. Если опция включена, основную YARN-группу пользователя необходимо внести в белый список, установив флаг монтирования *gid*, как показано далее. Иначе повторное получение контейнера (container reacquisition) завершается неудачей, и контейнер уничтожается при перезапуске **NodeManager**.

```
proc     /proc     proc     nosuid,nodev,noexec,hidepid=2,gid=yarn     0 0
```

4.7.8 Подключение к безопасному Docker-репозиторию

Клиентская команда Docker извлекает свою конфигурацию из местоположения по умолчанию *$HOME/.docker/config.json* на хосте **NodeManager**. В конфигурации Docker хранятся учетные данные репозитория Secure, поэтому использование **LCE** совместно с безопасными репозиториями Docker не рекомендуется.

В YARN-5428 добавлена поддержка **Distributed Shell** для безопасного предоставления конфигурации Docker-клиента.

В качестве обходного пути можно вручную зарегистрировать Docker-демон на каждом хосте **NodeManager** в безопасном репозитории, используя команду входа:

```
docker login [OPTIONS] [SERVER]

Register or log in to a Docker registry server, if no server is specified
"https://index.docker.io/v1/" is the default.
```

```
-e, --email=""       Email
-p, --password=""    Password
-u, --username=""    Username
```

Important: При данном подходе все пользователи имеют доступ к безопасному репозиторию

4.7.9 Пример: MapReduce

В примере предполагается, что **Hadoop** установлен в */usr/local/hadoop*. А так же `docker.allowed.ro-mounts` в *container-executor.cfg* обновлен и содержит каталоги: */usr/local/hadoop*, */etc/passwd*, */etc/group*.

Чтобы отправить задание *pi* для запуска в docker-контейнерах, необходимо выполнить команды:

```
HADOOP_HOME=/usr/local/hadoop
YARN_EXAMPLES_JAR=$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar
MOUNTS="$HADOOP_HOME:$HADOOP_HOME:ro,/etc/passwd:/etc/passwd:ro,/etc/group:/etc/group:ro"
IMAGE_ID="library/openjdk:8"

export YARN_CONTAINER_RUNTIME_TYPE=docker
export YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=$IMAGE_ID
export YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS=$MOUNTS

yarn jar $YARN_EXAMPLES_JAR pi \
  -Dmapreduce.map.env.YARN_CONTAINER_RUNTIME_TYPE=docker \
  -Dmapreduce.map.env.YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS=$MOUNTS \
  -Dmapreduce.map.env.YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=$IMAGE_ID \
  -Dmapreduce.reduce.env.YARN_CONTAINER_RUNTIME_TYPE=docker \
  -Dmapreduce.reduce.env.YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS=$MOUNTS \
  -Dmapreduce.reduce.env.YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=$IMAGE_ID \
  1 40000
```

Важно обратить внимание, что application master, map tasks и reduce tasks настраиваются независимо. В данном примере используется образ *openjdk:8* для всех трех.

4.7.10 Пример: Spark

В примере предполагается, что **Hadoop** установлен в */usr/local/hadoop*, а **Spark** – в */usr/local/spark*. А так же `docker.allowed.ro-mounts` в *container-executor.cfg* обновлен и содержит каталоги: */usr/local/hadoop*, */etc/passwd*, */etc/group*.

Чтобы запустить оболочку **Spark** в docker-контейнерах, необходимо выполнить команды:

```
HADOOP_HOME=/usr/local/hadoop
SPARK_HOME=/usr/local/spark
MOUNTS="$HADOOP_HOME:$HADOOP_HOME:ro,/etc/passwd:/etc/passwd:ro,/etc/group:/etc/group:ro"
IMAGE_ID="library/openjdk:8"

$SPARK_HOME/bin/spark-shell --master yarn \
  --conf spark.yarn.appMasterEnv.YARN_CONTAINER_RUNTIME_TYPE=docker \
  --conf spark.yarn.appMasterEnv.YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=$IMAGE_ID \
  --conf spark.yarn.appMasterEnv.YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS=$MOUNTS \
  --conf spark.executorEnv.YARN_CONTAINER_RUNTIME_TYPE=docker \
  --conf spark.executorEnv.YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=$IMAGE_ID \
  --conf spark.executorEnv.YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS=$MOUNTS
```

Важно обратить внимание, что application master и executors настраиваются независимо. В данном примере используется образ *openjdk:8* для обоих.

4.7.11 Поддержка ENTRYPOINT Docker-контейнера

В **Hadoop 2.x** введена поддержка Docker, платформа разработана для запуска существующих программ **Hadoop** внутри docker-контейнера, перенаправление журнала и настройка среды интегрированы с **Node Manager**. В **Hadoop 3.x** поддержка Docker выходит за рамки выполнения рабочей нагрузки **Hadoop** и поддерживает Docker-контейнер в собственной форме Docker, используя *ENTRYPOINT* из *dockerfile*. Приложение может принять поддержку режима *YARN* или режима *Docker* по умолчанию, задав переменную среды `YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE`. Системный администратор также может установить для кластера настройку по умолчанию, чтобы сделать *ENTRY_POINT* в качестве режима работы по умолчанию.

В *yarn-site.xml* необходимо добавить `YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE` в белый список среды **Node Manager**:

```
<property>
       <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,HADOOP_YARN_HOME,HADOOP_MAPRED_
↪HOME,YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE</value>
</property>
```

В *yarn-env.sh* определить:

```
export YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE=true
```

4.8 YARN on GPU

YARN поддерживает **NVIDIA GPU** в качестве ресурса.

Important: На данный момент YARN поддерживает только графические процессоры Nvidia. На все YARN NodeManagers должны быть предварительно установлены драйверы Nvidia. В случае использования Docker, необходимо установить nvidia-docker 1.0.

Fair Scheduler не поддерживает **Dominant Resource Calculator**. Политика *fairshare*, которую использует **Fair Scheduler**, учитывает только память для расчета *fairShare* и *minShare*, поэтому устройства **GPU** выделяются из общего пула.

Для включения поддержки **GPU** необходимо активировать раздел *advanced* и в нем параметр `GPU on YARN` (Рис.4.9.).

4.8.1 Конфигурация

GPU scheduling

В *resource-types.xml* необходимо добавить следующие свойства:

```
<configuration>
  <property>
     <name>yarn.resource-types</name>
     <value>yarn.io/gpu</value>
  </property>
</configuration>
```

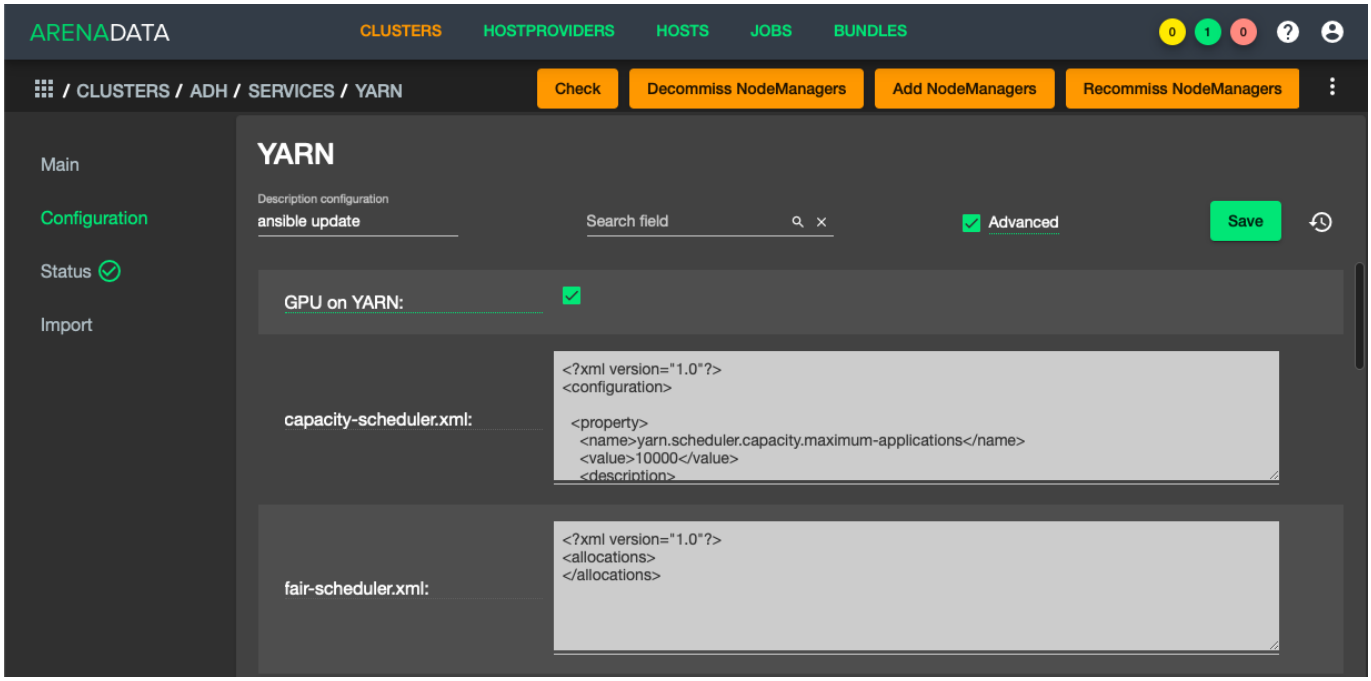


Рис.4.9.: Активация GPU on YARN

В *yarn-site.xml* параметр `DominantResourceCalculator` должен быть настроен на включение scheduling/isolation графического процессора.

Для **Capacity Scheduler** необходимо использовать свойство для настройки `DominantResourceCalculator` (в *capacity-scheduler.xml*):

- свойство `yarn.scheduler.capacity.resource-calculator`, значение по умолчанию *org.apache.hadoop.yarn.util.resource.DominantResourceCalculator*.

GPU Isolation

В *yarn-site.xml* для включения модуля **GPU isolation** на стороне **NodeManager**:

```
<property>
  <name>yarn.nodemanager.resource-plugins</name>
  <value>yarn.io/gpu</value>
</property>
```

По умолчанию **YARN** автоматически определяет и настраивает графические процессоры, когда установлен вышеуказанный конфиг. Следующие настройки необходимо устанавливать в *yarn-site.xml*, только если у администратора есть особые требования для этого.

1. Разрешение GPU Devices.

- `yarn.nodemanager.resource-plugins.gpu.allowed-gpu-devices`, значение по умолчанию *auto*.

Необходимо указать устройства **GPU**, которыми можно управлять с помощью **YARN NodeManager** (через запятую). Количество устройств с графическим процессором сообщается **Resource Manager** для принятия решений о планировании. Значение по умолчанию *auto* приводит к автоматическому обнаруживанию **YARN** ресурса **GPU** из системы.

В случае если автоопределение не удалось или администратору необходимо подмножество устройств с графическим процессором, управляемых **YARN**, устройства **GPU** указываются вручную. Устройство **GPU**

идентифицируется по его минорному номеру и индексу. Распространенным подходом для получения минорного номера устройства на графических процессорах – это использование `nvidia-smi -q` и поиск по *Minor Number*.

Когда минорные номера задаются вручную, администратор должен также включать индекс графических процессоров: `index:minor_number[,index:minor_number...]`. Например: `0:0,1:1,2:2,3:4` – позволяет **YARN NodeManager** управлять устройствами **GPU** с индексами 0/1/2/3 и минорными номерами 0/1/2/4.

2. Исполняемый файл для обнаружения GPU.

- `yarn.nodemanager.resource-plugins.gpu.path-to-discovery-executables`, значение */absolute/path/to/nvidia-smi*.

Когда указано `yarn.nodemanager.resource-plugins.gpu.allowed-gpu-devices=auto`, **YARN NodeManager** должен запустить бинарный файл обнаружения **GPU** (поддерживает только *nvidia-smi*), чтобы получить связанную с **GPU** информацию. Когда значение параметра пустое (по умолчанию), **YARN NodeManager** самостоятельно пытается найти исполняемый файл обнаружения. Пример значения конфигурации: */usr/local/bin/nvidia-smi*.

3. Подключаемые модули Docker.

Следующие конфигурации могут быть настроены, когда пользователю необходимо запустить приложения **GPU** внутри Docker-контейнера. Данные действия не требуются, если администратор следует установке и настройке *nvidia-docker* по умолчанию.

- `yarn.nodemanager.resource-plugins.gpu.docker-plugin`, значение по умолчанию *nvidia-docker-v1*.

Необходимо указать плагин команды docker для **GPU**. По умолчанию используется **Nvidia docker V1.0**, для *V2.x* доступен *nvidia-docker-v2*.

- `yarn.nodemanager.resource-plugins.gpu.docker-plugin.nvidia-docker-v1.endpoint`, значение по умолчанию *http://localhost:3476/v1.0/docker/cli*.

  Необходимо указать конечную точку *nvidia-docker-plugin* (подробнее в документации NVIDIA).

4. CGroups mount.

GPU isolation использует контроллер устройств CGroup для выполнения изоляции устройства для каждого графического процессора. Следующий параметр должен быть добавлен в *yarn-site.xml* для автоматического монтирования подустройств **CGroup**, в противном случае администратору необходимо вручную создать подпапку для устройств, чтобы использовать эту функцию.

- `yarn.nodemanager.linux-container-executor.cgroups.mount`, значение по умолчанию *true*.

В *container-executor.cfg* для включения модуля **GPU isolation** должна быть добавлена следующая конфигурация:

```
[gpu]
module.enabled=true
```

Когда пользователю необходимо запустить приложения с графическим процессором в среде, отличной от Docker:

```
[cgroups]
# This should be same as yarn.nodemanager.linux-container-executor.cgroups.mount-path inside yarn-site.xml
root=/sys/fs/cgroup
# This should be same as yarn.nodemanager.linux-container-executor.cgroups.hierarchy inside yarn-site.xml
yarn-hierarchy=yarn
```

Когда пользователю необходимо запустить приложения с графическим процессором в среде Docker:

1. Добавить связанные с GPU устройства в docker-раздел (разделенные запятой значения, которые можно получить, запустив */dev/nvidia*):

```
[docker]
docker.allowed.devices=/dev/nvidiactl,/dev/nvidia-uvm,/dev/nvidia-uvm-tools,/dev/nvidia1,/dev/nvidia0
```

2. Добавить nvidia-docker в белый список драйверов томов:

```
[docker]
...
docker.allowed.volume-drivers
```

3. Добавить *nvidia_ driver_ <version>* в белый список монтирования только для чтения:

```
[docker]
...
docker.allowed.ro-mounts=nvidia_driver_375.66
```

4. Если в качестве плагина gpu docker используется *nvidia-docker-v2*, необходимо добавить *nvidia* в белый список выполнения:

```
[docker]
...
docker.allowed.runtimes=nvidia
```

Important: В настоящее время распределенная оболочка поддерживает задание дополнительных типов ресурсов, кроме памяти и vcores

4.8.2 Distributed-shell + GPU without Docker

Для запуска распределенной оболочки без использования docker-контейнера (запрашивается 2 задачи, каждая имеет 3 ГБ памяти, 1 vcore, 2 устройства GPU):

```
yarn jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
  -jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
  -shell_command /usr/local/nvidia/bin/nvidia-smi \
  -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=2 \
  -num_containers 2
```

Вывод запущенного контейнера:

```
Tue Dec  5 22:21:47 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.66                 Driver Version: 375.66                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 0000:04:00.0     Off |                    0 |
| N/A   30C    P0    24W / 250W |      0MiB / 12193MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE...  Off  | 0000:82:00.0     Off |                    0 |
| N/A   34C    P0    25W / 250W |      0MiB / 12193MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
```

```
| Processes:                                                                  GPU Memory |
|  GPU       PID  Type  Process name                                          Usage      |
|=============================================================================|
|  No running processes found                                                            |
+-----------------------------------------------------------------------------+
```

4.8.3 Distributed-shell + GPU with Docker

Запуск распределенной оболочки с использованием docker-контейнера. Для этого должны быть заданы параметры `YARN_CONTAINER_RUNTIME_TYPE` и `YARN_CONTAINER_RUNTIME_DOCKER_IMAGE`:

```
yarn jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
       -jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
       -shell_env YARN_CONTAINER_RUNTIME_TYPE=docker \
       -shell_env YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=<docker-image-name> \
       -shell_command nvidia-smi \
       -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=2 \
       -num_containers 2
```

Глава 5

Карта портов

5.1 Порты HDFS

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

Таблица5.1.: Порты HDFS

Сервис	Характеристика
NameNode WebUI	Сервер – Master Nodes • Порт по умолчанию – 50070 – 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 – Протокол: https – Описание: Безопасный http-сервис – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.https.address
NameNode metadata service	Сервер – Master Nodes • Порт по умолчанию – 8020/9000 – 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	Сервер – Все Slave Nodes • Порт по умолчанию – 50075 – 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 – Протокол: https – Описание: Безопасный http-сервис – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: dfs.datanode.https.address • Порт по умолчанию – 50010 – Описание: Передача данных – Параметр конфигурации: dfs.datanode.address • Порт по умолчанию – 50020 – Протокол: IPC – Описание: Операции с метаданными – Доступ пользователям: Нет – Параметр конфигурации: dfs.datanode.ipc.address
Secondary NameNode	Сервер – Secondary NameNodes • Порт по умолчанию – 50090 – Протокол: http – Описание: Чекпоинт для метаданных NameNode – Доступ пользователям: Нет – Параметр конфигурации: dfs.secondary.http.address
HDFS HFTP	Сервер – Master NameNodes • Порт по умолчанию – 50470 – Протокол: https – Описание: Порт для доступа к HFTP файловой системы – Параметр конфигурации: dfs.https.port
JournalNode Web UI	Сервер – Master NameNodes • Порт по умолчанию – 8480 – 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 – 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.

Таблица5.2.: Порты MapReduce

Сервис	Характеристика
JobTracker WebUI	Сервер – Master Nodes • Порт по умолчанию – 50030 – Протокол: http – Описание: Web-интерфейс для работы JobTracker – Доступ пользователям: Да – Параметр конфигурации: mapred.job.tracker.http.address
JobTracker	Сервер – Master Nodes • Порт по умолчанию – 8021 – Протокол: IPC – Описание: Для публикации заданий – Доступ пользователям: Да (все клиенты, которым требуется запуск MR, Hive, Pig и т.д.) – Параметр конфигурации: mapred.job.tracker
TaskTracker Web UI and Shuffle	Сервер – Все Slave Nodes • Порт по умолчанию – 50060 – Протокол: http – Описание: Web-интерфейс для DataNode (логи, статус) – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: mapred.task.tracker.http.address
History Server WebUI	• Порт по умолчанию – 51111 – Протокол: http – Описание: Web-интерфейс для истории заданий – Доступ пользователям: Да – Параметр конфигурации: mapreduce.history.server.http.address
MapReduce Shuffle Port	• Порт по умолчанию – 13562 – Описание: Порт, на котором работает ShuffleHandler – Доступ пользователям: Нет – Параметр конфигурации: mapreduce.shuffle.port

5.3 Порты YARN

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

Таблица5.3.: Порты YARN

Сервис	Характеристика
ResourceManager WebUI	Сервер – Master Nodes • Порт по умолчанию – 8088 – Протокол: http – Описание: Web-интерфейс для Resource Manager – Доступ пользователям: Да – Параметр конфигурации: yarn.resourcemanager.webapp.address
ResourceManager	Сервер – Master Nodes (ResourceManager Node) • Порт по умолчанию – 8050 – 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 – 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 – Протокол: http – Описание: Прокси для Resource Manager – Доступ пользователям: Да – Параметр конфигурации: yarn.web-proxy.address • Порт по умолчанию – 8141 – Протокол: http – Описание: Адрес планировщика – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.resourcemanager.admin.address
Scheduler	Сервер – Master Nodes • Порт по умолчанию – 8030 – Описание: Адрес планировщика – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.resourcemanager.scheduler.address

Сервис	Характеристика
NodeManager	Сервер – Master Nodes • Порт по умолчанию – 45454 – Протокол: http – Описание: Адрес NodeManager – Параметр конфигурации: yarn.nodemanager.address Сервер – Slave Nodes • Порт по умолчанию – 8040 – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.localizer.address • Порт по умолчанию – 8042 – Протокол: http – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.webapp.address • Порт по умолчанию – 8044 – Протокол: https – Описание: NodeManager – Параметр конфигурации: yarn.nodemanager.webapp.https.address
Timeline Server	Сервер – Master Nodes • Порт по умолчанию – 10200 – Протокол: http – Описание: Адрес Timeline Server – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.timeline-service.address • Порт по умолчанию – 8188 – Протокол: http – Описание: Адрес Timeline Server Webapp – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: yarn.timeline-service.webapp.address • Порт по умолчанию – 8190 – 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.

Таблица5.4.: Порты Hive

Сервис	Характеристика
Hive Server2	Сервер – Hive Server машина • Порт по умолчанию – 10000 – 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 – 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	Сервер – Master Nodes • Порт по умолчанию – 8021 – Протокол: IPC – Описание: Для публикации заданий – Доступ пользователям: Да (все клиенты, которым требуется запуск MR, Hive, Pig. Задачи, использующие HCatalog)
Hive Web UI	Сервер – Hive Server машина • Порт по умолчанию – 9999 – Протокол: thrift – Описание: WebUI для Hive – Доступ пользователям: Да – Параметр конфигурации: hive.hwi.listen.port • Порт по умолчанию – 9933 – 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.

Таблица5.5.: Порты WebHCat

Сервис	Характеристика
WebHCat Server	Сервер – WebHCat Server машина • Порт по умолчанию – 50111 – 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.

Таблица5.6.: Порты HBase

Сервис	Характеристика
HMaster	Сервер – Master Nodes (HBase Master Node и back-up HBase Master node) • Порт по умолчанию – 16000 – Доступ пользователям: Да – Параметр конфигурации: hbase.master.port
HMaster Info Web UI	Сервер – Master Nodes (HBase Master Node и back-up HBase Master node) • Порт по умолчанию – 16010 – Протокол: http – Описание: Порт для HBase Master UI – Доступ пользователям: Да – Параметр конфигурации: hbase.master.info.port
Region Server	Сервер – Все Slave Nodes • Порт по умолчанию – 16020 – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: hbase.regionserver.port • Порт по умолчанию – 16030 – Протокол: http – Доступ пользователям: Да (администратор, разработчик, поддержка) – Параметр конфигурации: hbase.regionserver.info.port
HBase Thrift Server	Сервер – Все Thrift Server • Порт по умолчанию – 9090 – Описание: Порт, используемый HBase Thrift-сервером – Доступ пользователям: Да
HBase Thrift Server Web UI	Сервер – Все Thrift Server • Порт по умолчанию – 9090 – 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.

Таблица5.7.: Порты Zookeeper

Сервис	Характеристика
Zookeeper Server	Сервер – Все Zookeeper Nodes • Порт по умолчанию – 2181 – Протокол: http – Описание: Сервис доступа к Zookeeper Server/Quorum – Доступ пользователям: Да – Параметр конфигурации: zookeeper.port/clientPort • Порт по умолчанию – 2888 – Описание: Порт используется Zookeeper для взаимодействия компонентов – Доступ пользователям: Нет – Параметр конфигурации: hbase.zookeeper.peerport • Порт по умолчанию – 3888 – Описание: Порт используется Zookeeper для взаимодействия компонентов – Доступ пользователям: Нет – Параметр конфигурации: hbase.zookeeper.leaderport