**ИНСТРУКЦИЯ АДМИНИСТРАТОРА**

**RT DATALAKE**

2021

ОГЛАВЛЕНИЕ

[1 ОПИСАНИЕ RT DATALAKE 6](#_Toc96429447)

[2 АДМИНИСТРИРОВАНИЕ HDFS 7](#_Toc96429448)

[2.1 HDFS Federation 7](#_Toc96429449)

[2.1.1 Background 7](#_Toc96429450)

[2.1.2 Несколько Namenodes/Namespaces 7](#_Toc96429451)

[2.1.3 Конфигурация 8](#_Toc96429452)

[2.1.4 Управление 11](#_Toc96429453)

[2.2 HDFS Erasure Coding 12](#_Toc96429454)

[2.2.1 Цель 12](#_Toc96429455)

[2.2.2 Background 13](#_Toc96429456)

[2.2.3 Архитектура 13](#_Toc96429457)

[2.2.4 Развёртывание 16](#_Toc96429458)

[2.2.5 Ограничения 20](#_Toc96429459)

[2.3 Квоты 20](#_Toc96429460)

[2.3.1 Квоты имён 21](#_Toc96429461)

[2.3.2 Квоты пространств 21](#_Toc96429462)

[2.3.3 Квоты типа хранилища 22](#_Toc96429463)

[2.3.4 Административные команды 22](#_Toc96429464)

[2.3.5 Команда отчёта 23](#_Toc96429465)

[2.4 Снапшоты 24](#_Toc96429466)

[2.4.1 Обновление до версии HDFS со снапшотами 25](#_Toc96429467)

[2.4.2 Управление снапшотами 25](#_Toc96429468)

[2.5 ACL на HDFS 27](#_Toc96429469)

[2.5.1 Настройка ACL на HDFS 27](#_Toc96429470)

[2.5.2 Использование команд CLI для создания ACL 27](#_Toc96429471)

[2.6 Архивные хранилища 28](#_Toc96429472)

[2.6.1 Типы хранилищ HDFS 29](#_Toc96429473)

[2.6.2 Политики хранения 29](#_Toc96429474)

[2.6.3 Настройка архивного хранилища 30](#_Toc96429475)

[2.7 Управление кэшем 31](#_Toc96429476)

[2.7.1 Терминология 33](#_Toc96429477)

[2.7.2 Настройка централизованного кэширования 34](#_Toc96429478)

[2.7.3 Ограничения ОС 36](#_Toc96429479)

[2.7.4 Использование Cache Pools и Directives 36](#_Toc96429480)

[2.8 DataNodes от Non-root 39](#_Toc96429481)

[2.8.1 Настройка DataNode SASL 40](#_Toc96429482)

[2.9 Примеры ACL 42](#_Toc96429483)

[2.9.1 ACL & Permission Bits 42](#_Toc96429484)

[2.9.2 Пример 1. Доступ к группе 43](#_Toc96429485)

[2.9.3 Пример 2. ACL по умолчанию 44](#_Toc96429486)

[2.9.4 Пример 3. Блокировка доступа 45](#_Toc96429487)

[2.10 HAR. Архивы Hadoop 46](#_Toc96429488)

[2.10.1 Компоненты HAR 47](#_Toc96429489)

[2.10.2 Создание архива 48](#_Toc96429490)

[2.10.3 Просмотр файлов в архивах Hadoop 49](#_Toc96429491)

[2.10.4 Hadoop Archives и MapReduce 50](#_Toc96429492)

[2.11 HDFS Compression 50](#_Toc96429493)

[2.12 APIs JMX Metrics для HDFS Daemons 51](#_Toc96429494)

[2.12.1 Прямой доступ к удалённому агенту JMX 51](#_Toc96429495)

[2.13 Настройка Rack Awareness 52](#_Toc96429496)

[2.13.1 Создание скрипта Rack Topology 52](#_Toc96429497)

[2.13.2 Добавление свойства Script Topology в core-site.xml 54](#_Toc96429498)

[2.13.3 Перезапуск HDFS и MapReduce 54](#_Toc96429499)

[2.13.4 Контроль работы Rack Awareness 54](#_Toc96429500)

[2.14 Режим локального чтения данных 55](#_Toc96429501)

[2.15 Настройка WebHDFS 58](#_Toc96429502)

[2.16 POSIX ACL на HDFS 59](#_Toc96429503)

[2.16.1 Совместимость и применение 59](#_Toc96429504)

[2.16.2 Доступ 59](#_Toc96429505)

[2.16.3 Обратная связь 59](#_Toc96429506)

[2.16.4 Обратная совместимость 59](#_Toc96429507)

[2.16.5 Накладные расходы 60](#_Toc96429508)

[2.16.6 Ограничения 60](#_Toc96429509)

[2.16.7 Символические ссылки 60](#_Toc96429510)

[2.16.8 Снапшоты 60](#_Toc96429511)

[2.16.9 Инструментарий 60](#_Toc96429512)

[2.16.10 Доступ нескольким пользователям 60](#_Toc96429513)

[2.16.11 Доступ нескольким группам 61](#_Toc96429514)

[2.16.12 Hive Partitioned Tables 61](#_Toc96429515)

[2.16.13 ACL по умолчанию 62](#_Toc96429516)

[2.16.14 ACL/Permissions Only 62](#_Toc96429517)

[2.16.15 Блокировка доступа для пользователя 63](#_Toc96429518)

[2.16.16 Sticky Bit 63](#_Toc96429519)

[2.17 Shell-команды 63](#_Toc96429520)

[2.17.1 [balancer] Балансировка кластера 64](#_Toc96429521)

[2.17.2 [cacheadmin] Администрирование кэша 66](#_Toc96429522)

[2.17.3 [crypto] Управление шифрованием 70](#_Toc96429523)

[2.17.4 [datanode] Запуск DataNode 71](#_Toc96429524)

[2.17.5 [dfsadmin] Администрирование HDFS 71](#_Toc96429525)

[2.17.6 [dfsrouter] Запуск маршрутизатора 80](#_Toc96429526)

[2.17.7 [dfsrouteradmin] Управление маршрутизатором 80](#_Toc96429527)

[2.17.8 [diskbalancer] Управление балансировщиком 81](#_Toc96429528)

[2.17.9 [ec] Запуск интерфейс командной строки ErasureCoding. 82](#_Toc96429529)

[2.17.10 [haadmin] Управление NameNode 82](#_Toc96429530)

[2.17.11 [journalnode] Запуск JournalNode 83](#_Toc96429531)

[2.17.12 [mover] Запуск утилиты переноса данных 83](#_Toc96429532)

[2.17.13 [namenode] Управление NameNode 84](#_Toc96429533)

[2.17.14 [nfs3] Запуск шлюза NFS3 86](#_Toc96429534)

[2.17.15 [portmap] Запуск карты портов RPC 86](#_Toc96429535)

[2.17.16 [secondarynamenode] Запуск SecondaryNameNode 86](#_Toc96429536)

[2.17.17 [storagepolicies] Управление политиками хранения 87](#_Toc96429537)

[2.17.18 [zkfc] Запуск процесса Zookeeper Failover Controller 87](#_Toc96429538)

[3 АДМИНИСТРИРОВАНИЕ HIVE 88](#_Toc96429539)

[3.1 Tez 88](#_Toc96429540)

[3.1.1 Tez UI 88](#_Toc96429541)

[3.2 Настройка памяти для Hive/Tez 88](#_Toc96429542)

[4 АДМИНИСТРИРОВАНИЕ YARN 90](#_Toc96429543)

[4.1 Hadoop: Capacity Scheduler 90](#_Toc96429544)

[4.1.1 Функции 91](#_Toc96429545)

[4.1.2 Конфигурация 93](#_Toc96429546)

[4.1.3 Изменение конфигурации очереди 109](#_Toc96429547)

[4.1.4 Обновление контейнера 111](#_Toc96429548)

[4.2 Hadoop: Fair Scheduler 112](#_Toc96429549)

[4.2.1 Иерархия очередей и политика 114](#_Toc96429550)

[4.2.2 Размещение приложений в очередях 114](#_Toc96429551)

[4.2.3 Установка 114](#_Toc96429552)

[4.2.4 Конфигурация 115](#_Toc96429553)

[4.2.5 Администрирование 126](#_Toc96429554)

[4.3 YARN Timeline Service v.2 127](#_Toc96429555)

[4.3.1 Архитектура 128](#_Toc96429556)

[4.3.2 Конфигурация. Basic 129](#_Toc96429557)

[4.3.3 Конфигурация. Advanced 130](#_Toc96429558)

[4.3.4 Конфигурация. Security 131](#_Toc96429559)

[4.3.5 Включение поддержки CORS 132](#_Toc96429560)

[4.3.6 Включение Timeline Service v.2 132](#_Toc96429561)

[4.3.7 Основные конфигурации Timeline Service v.2 134](#_Toc96429562)

[4.3.8 Запуск Timeline Service v.2 135](#_Toc96429563)

[4.3.9 Включение MapReduce 136](#_Toc96429564)

[4.3.10 Обновление с alpha1 до alpha2 136](#_Toc96429565)

[4.3.11 Публикация определённых данных приложения 136](#_Toc96429566)

[4.3.12 REST API 139](#_Toc96429567)

[4.4 Hadoop: YARN Federation 174](#_Toc96429568)

[4.4.1 Архитектура 174](#_Toc96429569)

[4.4.2 Запуск приложений через субкластеры 178](#_Toc96429570)

[4.4.3 Конфигурация 180](#_Toc96429571)

[4.4.4 Запуск тестового задания 185](#_Toc96429572)

ОПИСАНИЕ RT DATALAKE

**RT Datalake** представляет собой масштабируемую платформу с открытым исходным кодом, предназначенную для хранения, обработки и анализа больших объёмов данных.

Платформа спроектирована так, чтобы обеспечить быструю, лёгкую и незатратную загрузку данных различного формата из большого количества источников.

Платформа представляет собой комплекс сервисов, которые обеспечивают не только хранение и обработку данных, но и управление, безопасность и операции.

Платформа включает в себя Hadoop, который состоит из **MapReduce**, Hadoop Distributed File System (**HDFS**) и Yet Another Resource Negotiator (**YARN**), а также **Hive**, **Knox**.

Все компоненты интегрированы друг с другом и протестированы на совместимость.

АДМИНИСТРИРОВАНИЕ HDFS

HDFS Federation

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

Background

**HDFS** имеет два основных слоя:

1. Namespace:

Состоит из каталогов, файлов и блоков;

Поддерживает все операции с файловой системой, связанные с пространством имён, такие как создание, удаление, изменение и просмотр файлов и каталогов;

1. Block Storage Service:

Управление блоком (выполняется в Namenode):

* Обеспечивает членство в кластере Datanode путём обработки регистраций и передачи heartbeats-сообщений;
* Обрабатывает отчёты 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 путём хранения блоков в локальной файловой системе и предоставления доступа на чтение/запись.

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**.

Несколько Namenodes/Namespaces

Для горизонтального масштабирования сервиса имён 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.

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

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

Пространство имён 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:

1. Namespace Scalability — Federation добавляет горизонтальное масштабирование пространства имён. Большие развёртывания или развёртывания, использующие множество небольших файлов, выигрывают от масштабирования пространства имён, позволяя добавлять больше Namenodes в кластер.
2. Performance — пропускная способность файловой системы не ограничивается одним Namenode. Добавление дополнительных Namenodes в кластер увеличивает пропускную способность чтения/записи файловой системы.
3. 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 имён.

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 разделённым 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.

1. 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:

1. Добавить dfs.nameservices в конфигурацию.
2. Обновить конфигурацию с помощью суффикса NameServiceID, чтобы использовать Federation.
3. Добавить новую конфигурацию, связанную с Namenode, в файл конфигурации.
4. Распространить файл конфигурации на все узлы в кластере.
5. Запустить новый Namenode и Secondary/Backup.
6. 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> |

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 в 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;

blockpool — политика балансирует хранилище на уровне пула блоков, который в свою очередь балансируется на уровне Datanode.

|  |
| --- |
| **Внимание.**Balancer балансирует только данные и не балансирует пространство имён. |

Decommissioning

Вывод из эксплуатации следующий — 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> |

1. 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 ёмкость 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 мёртвых данных для каждого Namenode. Также предоставляется ссылка для доступа к веб-интерфейсу каждого Namenode;

Статус декомиссии Datanodes.

HDFS Erasure Coding

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 тёплых 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 объём ресурсов, что и первая реплика.

Поэтому естественным улучшением является использование Erasure Coding (EC) вместо репликации, что обеспечивает тот же уровень отказоустойчивости при гораздо меньшем объёме 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.

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 путём вычисления декодирования на основе сохранившихся данных и чётности 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, 3х-реплицированный файл с 6 блоками будет занимать 6 \* 3 = 18 блоков дискового пространства. Но при развёртывании EC (6 данных, 3 чётности) он будет занимать только 9 блоков дискового пространства.

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 счёт 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 объединёнными пространствами имён Namespaces.

В типичных кластерах **HDFS** небольшие файлы могут занимать более 3/4 общего объёма 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.

1. **Расширения NameNode** — чередующиеся файлы **HDFS**, логически состоящие из групп блоков, каждая из которых содержит определённое количество внутренних блоков. Чтобы уменьшить потребление памяти NameNode от этих дополнительных блоков, введён новый иерархический протокол именования блоков. Идентификатор группы блоков может быть выведен из идентификатора любого из её внутренних блоков. Это позволяет управлять на уровне группы блоков, а не на уровне одного блока.
2. **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.
3. **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 передаётся 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 чётности декодируются из входных данных. Все недостающие данные и блоки чётности декодируются вместе;

Передача сгенерированных блоков данных на целевые узлы — после завершения декодирования восстановленные блоки передаются на целевые DataNodes.

1. **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 чётности 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. Её 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 ЕС требует перезаписи его данных; поэтому рекомендуется копировать файл (например, через distcp), а не переименовывать его.

Пользователям позволяется определять свои собственные политики EC с помощью XML-файла, который должен состоять из следующих трёх частей:

layoutversion — указывает версию формата XML-файла политики ЕС;

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 ЕС с именем user\_ec\_policies.xml.template находится в каталоге Hadoop conf.

1. **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.

Развёртывание

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 ЕС политики RS (6,3) это означает минимум 9 узлов DataNodes.

Файлы erasure coding распределяются по стойкам с целью обеспечения её 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 чётности EC. Формула для расчёта получается: (блоки данных + блоки чётности) / блоки чётности с округлением в большую сторону. Для политики ЕС 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 чётности, **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 ЕС 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 передаётся 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 имён 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 определён 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 ЕС пути устанавливается со значением по умолчанию в конфигурации;

-replicate — применение схемы по умолчанию REPLICATION для каталога, принятие каталогом репликации 3х;

-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 (включённые, отключённые и удалённые) политики erasure coding, зарегистрированные в **HDFS**. Только включённые политики подходят для использования с командой setPolicy;

[-addPolicies -policyFile <file>] — добавление списка пользовательских политик erasure coding. Пример политики приведён 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.

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.

Квоты

**HDFS** позволяет администратору устанавливать квоты на количество используемых имён и объём пространства для отдельных каталогов. Квоты имён и квоты пространства работают независимо, но администрирование и реализация этих двух типов квот тесно параллельны.

Квоты имён

Квота имён — это жёсткое ограничение на количество имён файлов и каталогов в root-дереве директории со следующими правилами:

1. Создание файлов и каталогов завершается ошибкой в случае превышения квоты.
2. Квоты придерживаются переименованных каталогов, но операция переименования не может быть выполнена в случае, если она приводит к нарушению квоты.
3. Попытка установить квоту будет успешной, даже если каталог нарушает эту новую квоту.
4. Новая созданная директория не имеет связанных квот.
5. 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), выдаётся предупреждение для каждого из таких нарушений. Установка или удаление квоты создает запись в журнале.

Квоты пространств

Квота пространств — это жёсткое ограничение на количество байт, используемых файлами в root-дереве директории со следующими правилами:

1. Аллокация блоков не выполняется, если квота не позволяет записать полный блок.
2. Каждая реплика блока подсчитывается по квоте.
3. Квоты придерживаются переименованных каталогов, но операция переименования не может быть выполнена в случае, если она приводит к нарушению квоты.
4. Новая созданная директория не имеет связанных квот.
5. Самая большая квота — Long.Max\_Value. Нулевая квота по-прежнему позволяет создавать файлы, но в них не могут быть добавлены блоки.
6. Каталоги не используют пространство файловой системы хоста и не учитывают квоту пространства.
7. Пространство файловой системы хоста, используемое для сохранения метаданных файла, не учитывается в квоте.
8. 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), выдаётся предупреждение для каждого из таких нарушений. Установка или удаление квоты создает запись в журнале.

Квоты типа хранилища

Квота типа хранилища — это жёсткое 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 имён для эффективного управления используемого хранилища кластера. Например:

1. 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.
2. Для каталогов без настроенной политики хранения администратор может не настраивать квоту типа хранения. Квота может быть настроена, даже если определённый 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.
3. Квота типа хранения DISK ограничена за исключением случаев, когда DISK не является доминирующим носителем данных (например, кластер с преимущественным типом хранения ARCHIVE).

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 имён для каждого каталога. Это не является ошибкой, если каталог не имеет квоты имён;

hdfs dfsadmin -setSpaceQuota <N> <directory>...<directory> — установка квоты пространств в значение N байт для каждого каталога. Это жёсткое 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 нулём, 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 жёсткое 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 нулём, ни положительным целым числом. Квота для конкретного типа хранилища задаётся, 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.

Команда отчёта

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

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

Где:

С помощью опции -q сообщается установленное для каждого каталога значение квоты имён, оставшаяся доступная квота имён, 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 её типов хранилищ.

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. Распространённые случаи использования моментальных снимков — резервное копирование данных, защита от ошибок пользователя и аварийное восстановление.

Реализация снапшотов **HDFS** очень рациональна:

1. Создание снапшота происходит мгновенно: стоимость O(1) без учёта времени поиска inode.
2. Дополнительная память используется только при внесении изменений относительно снапшота: использование памяти равно O(M), где M — количество изменённых файлов/каталогов.
3. 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 нём находятся снапшоты.

Вложенные каталоги в snapshottable в настоящее время не допускаются. Другими словами, директория не может быть установлена как snapshottable, если хотя бы один её 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 нём, 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 |

|  |
| --- |
| **Внимание.**В примере используется опция preserve для сохранения меток времени, владельца, прав доступа, списков ACL и XAttrs. |

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 путём в актуальной версии системы.

Управление снапшотами

Операции администратора

Описанные далее операции возможны только при наличии привилегий суперпользователя.

1. Разрешение создания снапшотов. При успешном завершении операции каталог становится snapshottable.

Команда:

|  |
| --- |
| hdfs dfsadmin -allowSnapshot <path> |

Аргумент:

path — путь к директории snapshottable.

1. 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.

1. 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.

1. 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.

1. 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.

1. Получение списка всех каталогов snapshottable, где у текущего пользователя есть разрешение на создание снапшотов.

Команда:

|  |
| --- |
| hdfs lsSnapshottableDir |

1. 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 удалён;

M — файл/каталог изменён;

R — файл/каталог переименован.

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

Отчёт 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 отчёт о различиях следующий:

|  |
| --- |
| R. /foo -> /foo2M. /foo/bar |

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

ACL на HDFS

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

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.

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:

-b — удаление всех записей, но с сохранением записей ACL. Записи для пользователей и групп сохраняются для совместимости с разрешениями;

-k — удаление ACL по умолчанию;

-Р — 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, разделённые запятыми;

<path> — путь к файлу или директории для изменения.

Например:

|  |
| --- |
| hdfs dfs -setfacl -m user:hadoop:rw- /filehdfs dfs -setfacl -x user:hadoop /filehdfs dfs -setfacl -b /filehdfs dfs -setfacl -k /dirhdfs dfs -setfacl --set user::rw-,user:hadoop:rw-,group::r--,other::r-- /filehdfs dfs -setfacl -R -m user:hadoop:r-x /dirhdfs 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:

-R — список ACL всех рекурсивных файлов и каталогов;

<path> — путь к файлу или директории списка

Например:

|  |
| --- |
| hdfs dfs -getfacl /filehdfs dfs -getfacl -R /dir |

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

Архивные хранилища

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

Для реализации архивного хранилища необходимо:

1. Выключить DataNode.
2. Назначить тип хранения ARCHIVE.
3. Установить политики хранения HOT, WARM или COLD в файлах и каталогах **HDFS**.
4. 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.

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.

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.

|  |
| --- |
| **Внимание.**В настоящее время политики хранения нельзя редактировать. |

Настройка архивного хранилища

Для настройки архивного хранилища необходимо выполнить следующие действия:

1. Выключить DataNode.

Закрыть DataNode с помощью соответствующих команд.

1. 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> |

1. 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 |

1. Запуск DataNode.

Запустить DataNode с помощью соответствующих команд.

1. 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, разделённые пробелами;

-f <local file> — локальный файл, содержащий список файлов/каталогов **HDFS** для миграции.

|  |
| --- |
| **Внимание.**Если оба параметра -p и -f опущены, путь по умолчанию является корневым каталогом. |

Пример:

|  |
| --- |
| hdfs mover /cold1/testfile |

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** даёт много существенных преимуществ:

1. 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**.
2. 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.
3. 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.
4. 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:

1. 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 отчётности менее полезно, так как подобные данные, вероятно, могут быть прочитаны только один раз.
2. 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.

В архитектуре NameNode отвечает за координацию всех кэш-файлов с неактивными данными в кластере. NameNode периодически получает кэш-отчет от каждого DataNode. Кэш-отчёт 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 имён 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.

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

|  |
| --- |
| **Внимание.**В настоящее время кэширование может применяться только к каталогам и файлам. |

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.

Настройка централизованного кэширования

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

|  |
| --- |
| **Внимание.**При использовании централизованного управления кэшем **HDFS** обязательно должен быть включён JNI. |

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

В настоящее время требуется только одно свойство:

dfs.datanode.max.locked.memory — свойство определяет максимальный объём памяти в байтах, который будет использовать DataNode для кэширования. Также необходимо увеличить размер заблокированного объёма 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 объёма 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 отчёта 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, распределённый 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> |

Ограничения ОС

Если выдаётся 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 объём 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 выдаётся либо более высокое значение, чем настроенное, либо строка “unlimited”, что указывает на отсутствие ограничения.

|  |
| --- |
| **Внимание.**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 |

|  |
| --- |
| **Внимание.**Приведённая информация не применяется к развёртыванию в Windows. Windows не имеет прямого эквивалента ulimit -l. |

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 создаётся Cache Pools, а затем в него добавляется Cache Directives.

Команды Cache Pools

Команды Cache Pools:

1. Добавление нового Cache Pool.

Команда:

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

Функции:

<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 байт, 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.

1. Изменение метаданных существующего Cache Pool.

Команда:

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

Функции:

<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 байт, 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.

1. Удаление Cache Pool. Также удаляет пути, связанные с ним.

Команда:

|  |
| --- |
| hdfs cacheadmin -removePool <name> |

Функции:

<name> — имя удаляемого Cache Pool.

1. Отображение информации об одном или нескольких Cache Pool, например, имя, владелец, группа, разрешения и прочее.

Команда:

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

Функции:

-stats — отображение дополнительной статистики по Cache Pool;

<name> — если параметр задан, то выдается только упомянутый Cache Pool.

1. Отображение подробной информации о команде.

Команда:

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

Функции:

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

Команды Cache Directives

Команды Cache Directives:

1. Добавление нового Cache Directive.

Команда:

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

Функции:

<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.

1. Удаление Cache Directive.

Команда:

|  |
| --- |
| hdfs cacheadmin -removeDirective <id> |

Функции:

<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.

1. Удаление всех Cache Directives по указанному пути.

Команда:

|  |
| --- |
| hdfs cacheadmin -removeDirectives <path> |

Функции:

<path> — путь Cache Directives для удаления. Необходимо разрешение Write Cache Pool, к которому относятся директивы. Можно использовать команду -listDirectives для отображения списка Cache Directives.

1. Возврат списка Cache Directives.

Команда:

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

Функции:

<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.

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.

|  |
| --- |
| **Внимание.**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) развёрнута 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 развёртывания версии 2.6.0 (или более поздней) необходимо обновить конфигурацию любых внешних приложений, чтобы включить SASL. Если для SASL включён 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 перенесён путём 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**, подключённый для SASL, может подключаться к обоим.

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> |

|  |
| --- |
| **Внимание.**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. |

1. 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.

1. Запустить DataNode.

Примеры ACL

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 трёх 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 — Иннокентий, 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 keshasales22K Nov 18 10:55 sales-data |

Получается, только Иннокентий 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 Иннокентию, Диане и Эдуарду 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 учётную запись пользователя, например, salesmgr, и разрешение Иннокентию, Диане и Эдуарду использовать учётную запись salesmgr с помощью sudo или аналогичных механизмов. Недостатком обходного пути является то, что он создаёт сложность для конечных пользователей, требуя от них применения разных учётных записей для разных действий.

Предположим далее, что помимо сотрудников по продажам все руководители компании должны иметь возможность читать данные о продажах. Это ещё 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.

Пример 1. Доступ к группе

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

1. Установить ACL.

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

1. Запустить getfacl, чтобы проверить результаты.

|  |
| --- |
| > hdfs dfs -getfacl /sales-data# file: /sales-data# owner: kesha# group: salesuser::rw-group::r--group:execs:r--mask::r--other::--- |

1. 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-dataFound 1 items-rw-r-----+ 3 kesha sales 0 2014-03-04 16:31 /sales-data |

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

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:

1. Установить default ACL в родительский каталог.

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

1. Создать подкаталоги.

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

1. Убедиться, что **HDFS** автоматически применил default ACL в подкаталоги.

|  |
| --- |
| > hdfs dfs -getfacl -R /monthly-sales-data# file: /monthly-sales-data# owner: kesha# group: salesuser::rwxgroup::r-xother::---default:user::rwxdefault:group::r-xdefault:group:execs:r-xdefault:mask::r-xdefault:other::---# file: /monthly-sales-data/FEB# owner: kesha# group: salesuser::rwxgroup::r-xgroup:execs:r-xmask::r-xother::---default:user::rwxdefault:group::r-xdefault:group:execs:r-xdefault:mask::r-xdefault:other::---# file: /monthly-sales-data/JAN# owner: kesha# group: salesuser::rwxgroup::r-xgroup:execs:r-xmask::r-xother::---default:user::rwxdefault:group::r-xdefault:group:execs:r-xdefault:mask::r-xdefault: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.

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:

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

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

1. Запустить getfacl для проверки результатов.

|  |
| --- |
| > hdfs dfs -getfacl /monthly-sales-data# file: /monthly-sales-data# owner: kesha# group: salesuser::rwxuser:diana:---group::r-xmask::r-xother::---default:user::rwxdefault:group::r-xdefault:group:execs:r-xdefault:mask::r-xdefault:other::--- |

Новая запись ACL добавляется к существующим разрешениям, определенным в Permission Bits. Иннокентий 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:

1. Если пользователь является владельцем файла, применяются разрешения “owner”.
2. Если у пользователя есть запись в списке ACL, применяются соответствующие права.
3. 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).
4. 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.

HAR. Архивы 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 имён. В результате место на диске не используется полностью из-за ограничения пространства имён.

Архивы **Hadoop** (HAR) используются для устранения ограничений пространства имён, 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 **Hadoop** по умолчанию равен 256 МБ. Файлы и блоки являются объектами имеё в **HDFS**, что означает, что они занимают пространство имён (место в NameNode). Таким образом, ёмкость пространства имён системы ограничена физической памятью NameNode.

Когда в системе хранится много мелких файлов, они занимают большую часть пространства имён. Как следствие, дисковое пространство не используется полностью из-за ограничения пространства имён. 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 имён, но занимали только 30% дискового пространства кластера.

Архивы HAR могут использоваться для устранения ограничений пространства имён, 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 счёт сокращения использования пространства имён и уменьшения нагрузки на работу в NameNode, оптимизирует память в NameNode и распределяет управление пространством имён по нескольким NameNodes. Кроме того, HAR обеспечивает параллельный доступ к исходным файлам с помощью заданий **MapReduce**.

Компоненты HAR

Формат модели данных

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

|  |
| --- |
| foo.har/\_masterindex //stores hashes and offsetsfoo.har/\_index //stores file statusesfoo.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 путём рекурсивного перемещения исходных каталогов, а затем список разбивается на карту входящих задач. Каждая задача создаёт файл (около 2 ГБ, настраивается) из подмножества исходных файлов и выводит метаданные. В итоге, reduce task собирает метаданные и генерирует индексные файлы.

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 приведённом примере создаётся 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.

|  |
| --- |
| **Внимание.**Архивирование не удаляет исходные файлы. При необходимости удаления входных файлов после создания архива (в целях сокращения пространства имён), исходные файлы должны удаляться вручную. |

Хотя команда архивации может быть запущена из файловой системы хоста, файл архива создаётся в **HDFS** из существующих каталогов. Если ссылаться на каталог в файловой системе хоста, а не на **HDFS**, выдаётся 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/zoohdfs dfs -mkdir /user/hadoophdfs dfs -mkdir /user/hadoop/dir1hdfs dfs -mkdir /user/hadoop/dir2 |

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/dir1har:///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/hadoophar:///user/zoo/foo.har/hadoop/dir1har:///user/zoo/foo.har/hadoop/dir2 |

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

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.

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 |

1. 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.

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 определённого ключа можно использовать параметр qry:

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

Прямой доступ к удалённому агенту JMX

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

Например, следующие параметры JVM в hadoop-env.sh используются для включения удалённого агента JMX для NameNode. Он работает на порту 8004 с отключённым 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](http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html). Также можно использовать инструмент [jmxquery](https://code.google.com/p/jmxquery/) для извлечения информации через JMX.

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

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

|  |
| --- |
| **Внимание.**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 удалённому агенту JMX.

Настройка Rack Awareness

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

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

Создание скрипта Rack Topology

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

1. Создать скрипт топологии и файл данных. Скрипт топологии должен быть исполняемым.
2. 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 prefixRACK\_PREFIX=default# To test, supply a hostname as script input:if [ $# -gt 0 ]; thenCTL\_FILE=${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 0fiwhile [ $# -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 " fidoneelse 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 01192.0.2.1 02192.0.2.2 03 |

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

Добавление свойства Script Topology в core-site.xml

Добавление свойства Script Topology в core-site.xml:

1. Остановить **HDFS**.
2. 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> |

Перезапуск HDFS и MapReduce

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

Контроль работы Rack Awareness

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

1. 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> |

1. Команда **Hadoop** fsck должна возвращать на подобии следующего (в случае двух стоек):

|  |
| --- |
| Status: HEALTHYTotal size: 123456789 BTotal dirs: 0Total files: 1Total 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: 3Average block replication: 3.0Corrupt blocks: 0Missing replicas: 0 (0.0 %)Number of data-nodes: 40Number of racks: 2FSCK ended at Mon Jan 13 17:10:51 UTC 2014 in 1 milliseconds |

1. Команда **Hadoop** dfsadmin -report возвращает отчёт, содержащий имя стойки рядом с каждой машиной. Отчёт должен выглядеть примерно следующим образом (частично):

|  |
| --- |
| [bsmith@hadoop01 ~]$ sudo -u hdfs hadoop dfsadmin -reportConfigured 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: 181Blocks with corrupt replicas: 0Missing blocks: 0-------------------------------------------------Datanodes available: 5 (5 total, 0 dead)Name: 192.0.2.0:50010 (h2d1.phd.local)Hostname: h2d1.phd.localRack: /default/rack\_02Decommission Status : NormalConfigured 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 |

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.

Также для настройки локального чтения данных на **HDFS** необходимо в файл hdfs-site.xml добавить свойства, приведённые 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. Рекомендуется установить значение false;

dfs.client.use.legacy.blockreader.local, значение false — установка значения false указывает, что используется новая версия локального чтения. 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> |

Настройка WebHDFS

Для настройки WebHDFS необходимо выполнить следующие шаги:

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

|  |
| --- |
| <property> <name>dfs.webhdfs.enabled</name> <value>true</value></property> |

1. [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 развёртывается 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 развёртывается NameNode;
* Realm\_Name — название сферы Kerberos.
1. Перезапустить сервисы NameNode и DataNode.

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

**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 определён 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.

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.

Обратная связь

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

Обратная совместимость

Реализация ACL обратно совместима с существующими Permission Bits. Изменения, применяемые посредством разрешённых 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.

Накладные расходы

Добавление ACL не приводит к негативному влиянию на потребление системных ресурсов при развёртывании. 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.

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 путём определения дополнительных групп или пользователей. Также такие списки требуют большей памяти и хранилища, и для каждой проверки разрешений требуется больше времени.

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.

Снапшоты

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

Инструментарий

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

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: |

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- |

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 определённого 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=CNgroup::rwxgroup:sales\_CN:r-xcountry=GBgroup::rwxgroup:sales\_GB:r-xcountry=USgroup::rwxgroup:sales\_US:r-x |

|  |
| --- |
| **Внимание.**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.

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::rwxdefault:user:kesha:rw-default:user:diana:r--default:user:edik: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.

ACL/Permissions Only

Списки управления доступом **HDFS** поддерживают развёртывания, 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::--- |

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 определённого 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 Иннокентия к dir3, file1, file2 и file3:

|  |
| --- |
| user:kesha:--- |

Удаление разрешений на dir2 означает, что Иннокентий 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, Иннокентий не сможет получить к нему доступ.

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.

Shell-команды

Для работы с **HDFS** через командную строку необходимо использовать нативный shell-клиент для **HDFS**.

Для использования shell-команд, с помощью которых можно взаимодействовать с **HDFS**, необходимо в терминале запустить скрипт bin/hdfs.

При запуске скрипта bin/hdfs без аргументов будет выведен перечень всех возможных команд.

Чтобы запустить конкретную команду, необходимо воспользоваться синтаксисом и перечнем команд, приведёнными ниже.

Общий синтаксис команд:

|  |
| --- |
| hadoop [SHELL\_OPTIONS] COMMAND [GENERIC\_OPTIONS] [COMMAND\_OPTIONS] |

Описание параметров и классов команд представлено ниже.

Таблица 16 — Параметры и классы

| COMMAND\_OPTIONS | Описание |
| --- | --- |
| SHELL\_OPTIONS | Общий набор shell-параметров. |
| GENERIC\_OPTIONS | Общий набор параметров, поддерживаемый несколькими командами. |
| COMMAND COMMAND\_OPTIONS | В следующих пунктах раздела описаны различные пользовательские команды с их параметрами. |

[balancer] Балансировка кластера

Команды balancer запускают утилиту балансировки кластера.

Администратор может просто нажать Ctrl-C, чтобы остановить процесс перебалансировки.

Обратите внимание, что политика блочного пула более строгая, чем политика узла данных.

Помимо ниже параметров команды введена функция закрепления, чтобы предотвратить перемещение определённых реплик балансировщиком/движителем. Эта функция закрепления отключена по умолчанию, и её можно включить с помощью свойства конфигурации dfs.datanode.block-pinning.enabled. Когда эта функция включена, она влияет только на блоки, которые записываются в избранные узлы, указанные в вызове create(). Эта функция полезна, когда мы хотим сохранить локальность данных для таких приложений, как HBase regionserver.

[policy] Балансировка кластера

Команда policy позволяет произвести балансировку кластера.

Если установлены параметры:

datanode (по умолчанию) — кластер сбалансирован, если сбалансирован каждый DataNode;

blockpool — кластер сбалансирован, если сбалансирован каждый пул блоков в каждом DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-policy <policy>] |

[threshold] Установка процента ёмкости диска

Команда threshold устанавливает процент ёмкости диска. Команда перезаписывает порог по умолчанию.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-threshold <threshold>] |

[exclude] Исключение DataNode из балансировки

Команда exclude исключает указанные DataNode из балансировки балансировщиком.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-exclude [-f <hosts-file> | <comma-separated list of hosts>]] |

[include] Включение DataNode в балансировку

Команда include включает только указанные DataNode для балансировки балансировщиком.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-include [-f <hosts-file> | <comma-separated list of hosts>]] |

[source] Выбор DataNode в качестве исходного узла

Команда source позволяет выбрать только указанные DataNode в качестве исходных узлов.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-source [-f <hosts-file> | <comma-separated list of hosts>]] |

[blockpools] Формирование списка пулов блоков для работы балансировщика

Команда blockpools позволяет сформировать список пулов блоков, с которыми будет работать балансировщик.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-blockpools <comma-separated list of blockpool ids>] |

[idleiterations] Установка количества итераций простоя

Команда idleiterations устанавливает максимальное количество итераций простоя перед выходом. Команда перезаписывает количество итераций, установленное по умолчанию (5).

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-idleiterations <idleiterations>] |

[runDuringUpgrade] Разрешение/запрет на запуск балансировщика во время обновления HDFS

Команда runDuringUpgrade разрешает/запрещает запускать балансировщик во время текущего обновления **HDFS**. Действие является нежелательным, поскольку команда не влияет на используемое пространство на чрезмерно загруженных машинах.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [-runDuringUpgrade] |

[help] Вызов справки

Команда help вызывает справку с информацией об использовании инструмента и последующий выход.

Синтаксис команды:

|  |
| --- |
| hadoop balancer [--h | --help] |

[cacheadmin] Администрирование кэша

Команда cacheadmin позволяет работать с кэшированием, а именно: созданием, изменением и перечислением Cache Pools и Cache Directives.

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 следует создать Cache Pools, а затем добавить в него Cache Directives.

Команды и их параметры COMMAND\_OPTIONS представлены в разделах ниже.

[addDirective] Добавление нового Cache Directive

Команда addDirective позволяет добавить новый Cache Directive.

Синтаксис команды:

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

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<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;

<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 [smhd]. Значение never означает, что директива никогда не истекает. Если параметр не установлен, директива никогда не истекает.

[modifyDirective] Изменение метаданных существующего Cache Directive

Команда modifyDirective позволяет вносить изменения в метаданные существующего Cache Directive.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-modifyDirective -id <id> [-path <path>] [-force] [-replication <replication>] [-pool <pool-name>] [-ttl <time-to-live>]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<id> — идентификатор Cache Directive;

<path> — путь к каталогу кэша или файлу;

-force — пропуск проверки ограничений ресурсов Cache Pool;

<replication> — восьмеричные разрешения в формате UNIX, назначенные Cache Pool;

<pool-name> — Cache Pool, к которому добавлен Cache Directive;

<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 [smhd]. Значение never означает, что директива никогда не истекает. Если параметр не установлен, директива никогда не истекает.

[listDirectives] Возврат списка Cache Directives

Команда listDirectives позволяет выводить список Cache Directive.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-listDirectives [-stats] [-path <path>] [-pool <pool>] [-id <id>]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<path> — список Cache Directives данного пути. Если в <path>, 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.

[removeDirective] Удаление Cache Directive

Команда removeDirective позволяет удалить Cache Directive.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-removeDirective <id>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<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] Удаление всех Cache Directives по указанному пути

Команда removeDirectives позволяет удалить все Cache Directive по указанному пути.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-removeDirectives -path <path>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<path> — путь Cache Directives для удаления. Необходимо разрешение Write Cache Pool, к которому относятся директивы. Можно использовать команду -listDirectives для отображения списка Cache Directives.

[addPool] Добавление нового Cache Pool

Команда addPool позволяет добавить новый Cache Directive.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-addPool <name> [-owner <owner>] [-group <group>] [-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<name> — имя нового Cache Pool;

<owner> — имя пользователя владельца Cache Pool. По умолчанию используется текущий пользователь;

<group> — гггруппа, 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] Изменение метаданных существующего Cache Pool

Команда modifyPool позволяет изменить метаданные существующего Cache Pool.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-modifyPool <name> [-owner <owner>] [-group <group>] [-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<name> — имя требующего изменения Cache Pool;

<owner> — имя пользователя владельца Cache Pool;

<group> — группа, которой назначен Cache Pool;

<mode> — восьмеричные разрешения в формате 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] Удаление Cache Pool

Команда removePool позволяет удалить Cache Pool. Также удаляет пути, связанные с ним.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-removePool <name>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<name> — имя удаляемого Cache Pool.

[listPools] Отображение информации о Cache Pool

Команда listPools позволяет отобразить информацию об одном или нескольких Cache Pool, например, имя, владельца, группу, разрешения и прочее.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-listPools [-stats] [<name>]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-stats — отображение дополнительной статистики по Cache Pool;

<name> — если параметр задан, то выдается только упомянутый Cache Pool.

[help] Отображение подробной информации о команде

Команда help позволяет отобразить подробную о команде.

Синтаксис команды:

|  |
| --- |
| hadoop cacheadmin [-help <command-name>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<command-name> — отображение подробной информации по указанной команде. Если команда не указана, отображается подробная справка по всем командам.

[crypto] Управление шифрованием

Команда crypto позволяет управлять шифрованием.

Команды и их параметры COMMAND\_OPTIONS представлены в разделах ниже.

[createZone] Создание новой зоны шифрования

Команда createZone позволяет создать новую зону шифрования.

Синтаксис команды:

|  |
| --- |
| hadoop crypto -createZone -keyName <keyName> -path <path> |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

path — путь к создаваемой зоне шифрования. Это должен быть пустой каталог. По этому пути создается каталог для мусора;

keyName — имя ключа для использования в зоне шифрования. Имена клавиш в верхнем регистре не поддерживаются.

[listZones] Отображение информации о зонах шифрования

Команда listZones позволяет перечислить все зоны шифрования. Требуются разрешения суперпользователя.

Синтаксис команды:

|  |
| --- |
| hadoop crypto -listZones |

[provisionTrash] Подготовка каталога корзины для зоны шифрования

Команда provisionTrash позволяет подготовить каталог корзины для зоны шифрования.

Синтаксис команды:

|  |
| --- |
| hadoop crypto -provisionTrash -path <path> |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

path — путь к корню зоны шифрования.

[help] Отображение подробной информации о команде

Команда help позволяет отобразить подробную информацию о команде.

Синтаксис команды:

|  |
| --- |
| hadoop crypto -help <command-name> |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

<command-name> — отображение подробной информации по указанной команде. Если команда не указана, отображается подробная справка по всем командам.

[datanode] Запуск DataNode

Команда datanode позволяет запустить DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop datanode [-regular | -rollback | -rollingupgrade rollback] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-regular — нормальный запуск DataNode (по умолчанию);

-rollback — откатить DataNode до предыдущей версии. Это следует использовать после остановки узла данных и распространения старой версии hadoop.

-rollingupgrade rollback — откатить операцию последовательного обновления.

[dfsadmin] Администрирование HDFS

Команды dfsadmin предназначены для администрирования **HDFS**.

Ниже в разделах представлены возможные команды для администрирования **HDFS**.

[report] Отображение информации и статистики HDFS

Команда report позволяет отобразить основную информацию и статистику файловой системы **HDFS**.

Использование команды dfs может отличаться от использования команды du, поскольку она измеряет необработанное пространство, используемое репликациями, контрольными суммами, снапшотами и т.д. для всех DataNode.

Необязательные параметры команды могут использоваться для фильтрации списка отображаемых узлов данных.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-report [-live] [-dead] [-decommissioning] [-enteringmaintenance] [-inmaintenance]] |

[safemode] Ручное управление режимом Safemode

Команда safemode предназначена для ручного управления режимом Safemode (безопасным режимом).

Режим Safemode — это состояние узла NameNode, в котором он:

не принимает изменения в пространстве имён (только для чтения);

не реплицирует и не удаляет блоки.

Вход в режим Safemode производится автоматически при запуске NameNode. Выхода из режима производится также автоматически, когда настроенный минимальный процент блоков удовлетворяет минимальному условию репликации. Если NameNode обнаруживает какую-либо аномалию, он будет оставаться в режиме SafeMode до тех пор, пока проблема не будет решена. Если эта аномалия является следствием преднамеренного действия, администратор может использовать команду -safemode forceExit для выхода из режима Safemode.

Случаи, когда может потребоваться использование forceExit:

1. Метаданные NameNode несовместимы.

Если Namenode обнаруживает, что метаданные были изменены вне диапазона и могут вызвать потерю данных, NameNode перейдёт в состояние SafemodeforceExit. В этот момент пользователь может либо перезапустить NameNode с правильными файлами метаданных, либо использовать forceExit (если потеря данных допустима).

Откат вызывает замену метаданных и редко может вызвать состояние принудительного выхода из режима Safemode на NameNode.

В этом случае вы можете продолжить, введя -safemode forceExit.

В режим Safemode можно войти вручную, но тогда из него также необходимо будет выйти вручную.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-safemode enter | leave | get | wait | forceExit] |

[saveNamespace] Сохранение текущего пространства имён

Команда saveNamespace позволяет сохранить текущее пространство имён в каталогах хранилища и сбросить журнал изменений.

Для выполнения команды требуется вход в режим Safemode. Если задан параметр beforeShutdown, NameNode выполняет контрольную точку тогда и только тогда, когда ни одна контрольная точка не была сделана в течение временного окна (настраиваемое количество периодов контрольной точки). Обычно данная команда используется перед завершением работы NameNode, чтобы предотвратить возможное повреждение fsimage или журналов изменений.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-saveNamespace [-beforeShutdown]] |

[rollEdits] Свёртывание журнала изменений

Команда rollEdits сворачивает журнал редактирования на активном NameNode.

С течением времени количество файлов журнала редактирования растёт, а также NameNode сохраняет старые версии файла fsimage.

Это будет использовать дисковое пространство на NameNode и может вызвать проблемы с диском в более длительной работе кластера. Кроме того, если вторичная NameNode не настроена или работает некорректно, эти файлы редактирования будут создаваться в большом количестве, каждый файл будет содержать примерно 1 миллион транзакций. Из-за этого время запуска NameNode увеличится, и NameNode может даже не запуститься, если памяти будет недостаточно для выполнения операции создания контрольной точки.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-rollEdits] |

[restoreFailedStorage] Включение/выключение попыток автоматического восстановления неудачных репликаций хранилища

Команда restoreFailedStorage включает/выключает автоматические попытки восстановления неудачных репликаций хранилища.

Если сбойное хранилище снова станет доступным, система попытается восстановить журнал изменений и/или fsimage во время контрольной точки.

Параметр check вернёт текущую настройку.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-restoreFailedStorage true |false |check] |

[refreshNodes] Обновление узлов

Команда refreshNodes повторно читает хосты и исключает файлы для обновления перечня DataNodes, которым разрешено подключаться к NameNode, и тех, которые должны быть выведены или повторно введены в эксплуатацию.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshNodes] |

[setQuota] Установка квоты имени каталога

Команда setQuota позволяет установить квоту имени, равной N для каждого каталога.

Наилучший вариант для каждого каталога, с сообщением об ошибках, если N не является положительным длинным целым числом, каталог не существует, или это файл, или каталог немедленно превысит новую квоту.Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-setQuota <quota> <dirname>...<dirname>] |

[clrQuota] Удаление квоты имени каталога

Команда clrQuota позволяет удалить квоту имён для каждого каталога. Наилучший вариант для каждого каталога с сообщением об ошибках, если каталог не существует или является файлом. Если для каталога нет квоты, это не ошибка.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-clrQuota <dirname>...<dirname>] |

[setSpaceQuota] Установка квоты типа хранилища/пространства каталога

Команда setSpaceQuota позволяет:

установить квоту пространства в N байт для каждого каталога. Это жёсткое ограничение на общий размер всех файлов в дереве каталогов. Квота пространства также учитывает репликацию, т.е. 1 Гб данных при репликации занимает 3 Гб квоты. N также можно указать с двоичным префиксом для удобства, например, 50 g для 50 гигабайт и 2 t для 2 терабайт и т.д. Максимальные усиления для каждого каталога, с сообщением об ошибках, если N не равно нулю и не является положительным целым числом, каталог не существует, или это файл, или каталог немедленно превысит новую квоту;

установить квоту типа хранилища равной N байт типа хранилища, указанного для каждого каталога. Это жёсткое ограничение на общее использование типа хранилища для всех файлов в дереве каталогов. Использование квоты типа хранилища отражает предполагаемое использование на основе политики хранения. Например, 1 Гб данных с репликацией и политикой хранения ALL\_SSD занимает 3 Гб квоты SSD. N также можно указать с двоичным префиксом для удобства, например, 50 g для 50 гигабайт и 2 t для 2 терабайт и т.д. Наилучший вариант для каждого каталога, с сообщением об ошибках, если N не равно нулю и не является положительным целым числом, каталог не существует, или это файл, или каталог немедленно превысит новую квоту. Квота для конкретного типа хранилища устанавливается, когда указана опция -storageType. Доступные типы хранилища: RAM\_DISK, DISK, SSD, ARCHIVE.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>] |

[clrSpaceQuota] Удаление квоты пространства каталога/типа хранилища

Команда clrSpaceQuota позволяет:

удалить любую квоту пространства для каждого каталога. Наилучший вариант для каждого каталога с сообщением об ошибках, если каталог не существует или является файлом. Если для каталога нет квоты, это не ошибка;

удалить квоту типа хранилища, указанную для каждого каталога. Наилучший вариант для каждого каталога с сообщением об ошибках, если каталог не существует или является файлом. Это не является ошибкой, если в каталоге нет квоты на тип хранилища для указанного типа хранилища. Квота для конкретного типа хранилища очищается, если указан параметр -storageType. Доступные типы хранилища: RAM\_DISK, DISK, SSD, ARCHIVE.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>] |

[finalizeUpgrade] Завершение обновления

Команда finalizeUpgrade позволяет завершить обновление HDFS. DataNodes удаляют свою предыдущую резервную копию кластера, которая была сделана во время предыдущего обновления после того, как NameNode произведёт те же действия. После этого считается, что процесс обновления завершён.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-finalizeUpgrade] |

[rollingUpgrade] Последовательное обновление

Команда rollingUpgrade позволяет выполнять действие последовательного обновления.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-rollingUpgrade [<query> |<prepare> |<finalize>]] |

[upgrade] Текущий статус обновления

Команда upgrade позволяет запросить текущий статус обновления, используя параметр query, или завершить текущее обновление **HDFS**, используя параметр finalize.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-upgrade [query | finalize] |

[refreshServiceAcl] Перезагрузка файла политики авторизации

Команда refreshServiceAcl позволяет перезагрузить файл политики авторизации на уровне сервиса.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshServiceAcl] |

[refreshUserToGroupsMappings] Обновление сопоставления групп и пользователей

Команда refreshUserToGroupsMappings позволяет обновить сопоставления пользователей и групп.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshUserToGroupsMappings] |

[refreshSuperUserGroupsConfiguration] Обновление сопоставления proxy-групп суперпользователя

Команда refreshSuperUserGroupsConfiguration позволяет обновить сопоставления proxy-групп суперпользователя.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshSuperUserGroupsConfiguration] |

[refreshCallQueue] Обновление очереди вызовов

Команда refreshCallQueue позволяет перезагрузить очередь вызовов из config.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshCallQueue] |

[refresh] Обновление ресурса

Команда refresh запускает обновление во время выполнения ресурса, указанного в <key> на <host:ipc\_port>. Все остальные параметры отправляются на хост.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refresh <host:ipc\_port> <key> [arg1..argn]] |

[reconfig] Заупск реконфигурации

Команда reconfig запускает реконфигурацию или выводит статус текущей реконфигурации, или выводит список реконфигурируемых свойств. Второй параметр указывает тип узла.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-reconfig <namenode|datanode> <host:ipc\_port> <start |status |properties>] |

[printTopology] Отображение топологии

Команда printTopology позволяет вывести дерево стоек и их узлов, как передаётся в NameNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-printTopology] |

[refreshNamenodes] Перезагрузка файлов конфигурации DataNode

Команда refreshNamenodes перезагружает файлы конфигурации для данного узла NameNode, прекращает обслуживание удалённых пулов блоков и начинает обслуживание новых пулов блоков.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-refreshNamenodes datanodehost:port] |

[getVolumeReport] Получение отчёта об объёме DataNode

Команда getVolumeReport позволяет получить отчёт об объёме для данного узла NameNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-getVolumeReport datanodehost:port] |

[deleteBlockPool] Удаление блока пула

Команда deleteBlockPool при принудительном вводе каталог пула блоков для данного идентификатора пула блоков на указанном DataNode удаляет пул вместе с его содержимым, в противном случае каталог удаляется только в том случае, если он пуст. Команда завершится ошибкой, если DataNode все ещё обслуживает пул блоков. Используйте команду refreshNamenodes, чтобы завершить работу службы пула блоков на DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-deleteBlockPool datanode-host:port blockpoolId [force]] |

[setBalancerBandwidth] Изменение пропускной способности сети

Команда setBalancerBandwidth изменяет пропускную способность сети, используемую каждым DataNode во время балансировки блоков **HDFS**. <bandwidth> — это максимальное количество байтов в секунду, которое будет использоваться каждым DataNode. Это значение переопределяет параметр dfs.datanode.balance.bandwidthPerSec. Новое значение не сохраняется в DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-setBalancerBandwidth <bandwidth in bytes per second>] |

[getBalancerBandwidth] Отображение пропускной способности сети

Команда getBalancerBandwidth позволяет вывести пропускную способность сети (в байтах в секунду) для данного DataNode. Это максимальная пропускная способность сети, используемая DataNode во время балансировки блоков **HDFS**.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-getBalancerBandwidth <datanode\_host:ipc\_port>] |

[fetchImage] Загрузка fsimage

Команда fetchImage загружает самый последний fsimage из NameNode и сохраняет его в указанном локальном каталоге.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-fetchImage <local directory>] |

[allowSnapshot] Разрешение на создание снапшотов

Команда allowSnapshot устанавливает разрешение на создание снапшотов каталога. Если операция завершится успешно, каталог станет снапшотом.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-allowSnapshot <snapshotDir>] |

[disallowSnapshot] Запрет на создание снапшотов

Команда disallowSnapshot устанавливает запрет на создание снапшотов каталога. Перед тем, как установить запрет на создание снапшотов, необходимо удалить все снапшоты каталога.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-disallowSnapshot <snapshotDir>] |

[shutdownDatanode] Завершение работы DataNode

Команда shutdownDatanode позволяет отправить запрос на завершение работы для данного DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-shutdownDatanode <datanode\_host:ipc\_port> [upgrade]] |

[evictWriters] Завершение клиентов DataNode, записывающих блок

Команда evictWriters позволяет заставить DataNode завершить все клиентские соединения, которые записывают блок. Это полезно, если вывод из эксплуатации зависает из-за медленной записи.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-evictWriters <datanode\_host:ipc\_port>] |

[getDatanodeInfo] Получение информации о DataNode

Команда getDatanodeInfo позволяет получить информацию о данном DataNode.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-getDatanodeInfo <datanode\_host:ipc\_port>] |

[metasave] Сохранение метаданных NameNode

Команда metasave позволяет сохранить структуры метаданных NameNode в filename в каталоге, указанном свойством hadoop.log.dir. Filename перезаписывается, если он уже имеется. Filename будет содержать по одной строке для каждого из следующих параметров:

сообщения heartbeat DataNodes с NameNode;

блоки, ожидающие репликации;

блоки, которые в настоящее время реплицируются;

блоки, ожидающие удаления.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-metasave filename] |

[triggerBlockReport] Отображение отчёта о блоках DataNode

Команда triggerBlockReport запускает отчёт о блоках для данного DataNode. Если не указан параметр <incremental>, то запустится отчёт о полном блоке.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-triggerBlockReport [-incremental] <datanode\_host:ipc\_port>] |

[listOpenFiles] Отображение перечня открытых файлов

Команда listOpenFiles выводит перечень всех открытых файлов, управляемых в настоящий момент узлом NameNode, а также имя клиента и клиентский компьютер, к которому они обращаются. Список открытых файлов будет отфильтрован по заданному типу и пути. Добавляя параметр <blockingDecommission>, будут перечислены только открытые файлы, которые блокируют вывод DataNode из эксплуатации.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-listOpenFiles [-blockingDecommission] [-path <path>]] |

[help] Вызов справки

Команда help отображает справку для данной команды или всех команд, если ни одна не указана.

Синтаксис команды:

|  |
| --- |
| hadoop dfsadmin [-help [cmd]] |

[dfsrouter] Запуск маршрутизатора

Команда dfsrouter запускает маршрутизатор.

Синтаксис команды:

|  |
| --- |
| hadoop dfsrouter |

[dfsrouteradmin] Управление маршрутизатором

Команда dfsrouteradmin предназначена для управления маршрутизатором.

Синтаксис команды:

|  |
| --- |
| hadoop dfsrouteradmin [-add <source> <nameservice1, nameservice2, ...> <destination> [-readonly] [-order HASH|LOCAL|RANDOM|HASH\_ALL] -owner <owner> -group <group> -mode <mode>] [-update <source> <nameservice1, nameservice2, ...> <destination> [-readonly] [-order HASH|LOCAL|RANDOM|HASH\_ALL] -owner <owner> -group <group> -mode <mode>] [-rm <source>] [-ls <path>] [-setQuota <path> -nsQuota <nsQuota> -ssQuota <quota in bytes or quota size string>] [-clrQuota <path>] [-safemode enter | leave | get] [-nameservice disable | enable <nameservice>] [-getDisabledNameservices] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-add source nameservices destination — добавить запись в таблицу монтирования или обновить, если она существует;

-update source nameservices destination — обновить запись в таблице монтирования или создать ее, если она не существует;

-rm source — удалить точку монтирования по указанному пути;

-ls path — вывести список точек монтирования по указанному пути;

-setQuota path -nsQuota nsQuota -ssQuota ssQuota — установить квоту для указанного пути;

-clrQuota path — очистить квоту для данной точки монтирования;

-safemode enter leave get — ручная установка маршрутизатора, входящего в безопасный режим или выходящего из него. Параметр get будет использоваться для проверки, находится ли маршрутизатор в состоянии безопасного режима;

-nameservice disable enable nameservice — отключить/включить службу имён из федерации. Если отключено, запросы не будут поступать в эту службу имён;

-getDisabledNameservices — получить службы имён, отключённые в федерации.

[diskbalancer] Управление балансировщиком

Команда diskbalancer предназначена для управления балансировщиком.

Синтаксис команды:

|  |
| --- |
| hadoop diskbalancer [-plan <datanode> -fs <namenodeURI>] [-execute <planfile>] [-query <datanode>] [-cancel <planfile>] [-cancel <planID> -node <datanode>] [-report -node <file://> | [<DataNodeID|IP|Hostname>,...]] [-report -node -top <topnum>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-plan — создаёт план балансировщика;

-execute — выполняет заданный план на узле данных;

-query — получает текущий статус балансировщика дисков с узла данных;

-cancel — отменяет текущий план;

-report — сообщает информацию об объёме от узлов данных.

[ec] Запуск интерфейс командной строки ErasureCoding.

Команда ec запускает интерфейс командной строки ErasureCoding.

Синтаксис команды:

|  |
| --- |
| hadoop ec [generic options] [-setPolicy -policy <policyName> -path <path>] [-getPolicy -path <path>] [-unsetPolicy -path <path>] [-listPolicies] [-addPolicies -policyFile <file>] [-listCodecs] [-enablePolicy -policy <policyName>] [-disablePolicy -policy <policyName>] [-help [cmd ...]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-setPolicy — установить указанную политику ErasureCoding в каталог;

-getPolicy — получить информацию о политике ErasureCoding по указанному пути;

-unsetPolicy — отменить политику ErasureCoding, установленную предыдущим вызовом «setPolicy» в каталоге;

-listPolicies — список всех поддерживаемых политик ErasureCoding;

-addPolicies — добавить список политик кодирования стирания;

-listCodecs — получить список поддерживаемых кодеков стирания кодеков в системе;

-enablePolicy — включить политику ErasureCoding в системе;

-disablePolicy — отключить политику ErasureCoding в системе.

[haadmin] Управление NameNode

Команда haadmin предназначена для управления состояниями NameNode.

Синтаксис команды:

|  |
| --- |
| hadoop haadmin -transitionToActive <serviceId> [--forceactive]hadoop haadmin -transitionToStandby <serviceId>hadoop haadmin -transitionToObserver <serviceId>hadoop haadmin -failover [--forcefence] [--forceactive] <serviceId> <serviceId>hadoop haadmin -getServiceState <serviceId>hadoop haadmin -getAllServiceStatehadoop haadmin -checkHealth <serviceId>hadoop haadmin -help <command> |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-checkHealth — проверить состояние данного NameNode;

-failover — инициирует переключение между двумя NameNodes;

-getServiceState — определяет, является ли данный NameNode активным или резервным;

-getAllServiceState — возвращает состояние всех NameNodes;

-transitionToActive — переводит состояние заданного NameNode в Active;

-transitionToStandby — переводит состояние данного NameNode в режим ожидания;

-transitionToObserver — переводит состояние данного NameNode в Observer;

-help [cmd] — отображает справку для данной команды или всех команд, если ни одна не указана.

[journalnode] Запуск JournalNode

Команда journalnode предназначена для запуска узла журналов.

Синтаксис команды:

|  |
| --- |
| hadoop journalnode |

[mover] Запуск утилиты переноса данных

Команда mover запускает утилиту переноса данных.

Обратите внимание, что если опущены параметры -p и -f, по умолчанию используется корневой каталог.

Также введена функция закрепления, чтобы предотвратить перемещение определённых реплик балансировщиком/утилитой. Эта функция закрепления отключена по умолчанию, и её можно включить с помощью свойства конфигурации dfs.datanode.block-pinning.enabled. Когда эта функция включена, эта функция влияет только на блоки, которые записываются в избранные узлы, указанные в вызове create (). Эта функция полезна, когда мы хотим сохранить локальность данных для таких приложений, как HBase regionserver.

Синтаксис команды:

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

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-f <local file> — укажите локальный файл, содержащий список файлов/каталогов HDFS для переноса;

-p <files/dirs> — укажите разделённый пробелами список файлов/каталогов HDFS для переноса.

[namenode] Управление NameNode

Команды namenode позволяют управлять NameNode.

Ниже в разделах представлены возможные команды для администрирования **HDFS**.

[backup] Запуск backup-узла

Команда backup запускает backup-узел.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-backup] |

[checkpoint] Запуск checkpoint-узла

Команда checkpoint запускает checkpoint-узел, т.е. узел контрольной точки.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-checkpoint] |

[format] Форматирование NameNode

Команда format форматирует указанный NameNode. Он запускает NameNode, форматирует его, а затем закрывает. Выдаёт исключение NameNodeFormatException, если dir уже существует и для кластера отключено переформатирование.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-format [-clusterid cid ] [-force] [-nonInteractive] ] |

[upgrade] Запуск NameNode с обновлением версии

Команда upgrade запускает NameNode с опцией обновления после распространения новой версии **Hadoop**.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-upgrade [-clusterid cid] [-renameReserved<k-v pairs>] ] |

[upgradeOnly] Обновление NameNode

Команда upgradeOnly обновляет указанный NameNode, а затем выключает его.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-upgradeOnly [-clusterid cid] [-renameReserved<k-v pairs>] ] |

[rollback] Откат NameNode

Команда rollback откатывает NameNode к предыдущей версии. Команду следует использовать после остановки кластера и разворачивания старой версии **Hadoop**.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-rollback] |

[rollingUpgrade] Управление непрерывным обновлением NameNode

Команда rollingUpgrade обновляет и закрывает указанный NameNode.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-rollingUpgrade <rollback |started> ] |

[importCheckpoint] Загрузка и сохранение изображения из каталога контрольных точек

Команда importCheckpoint загружает изображение из каталога контрольных точек и сохраняет его в текущем. Каталог dir контрольной точки считывается из свойства dfs.namenode.checkpoint.dir.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-importCheckpoint] |

[initializeSharedEdits] Форматирование общего каталога изменений и копирование его в журнал

Команда initializeSharedEdits форматирует новый общий каталог изменений и копирует в достаточное количество сегментов журнала, чтобы можно было запустить резервный NameNode.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-initializeSharedEdits] |

[bootstrapStandby] Загрузка резервных каталогов NameNode

Команда bootstrapStandby позволяет загрузить резервные каталоги хранилища NameNode путём копирования последнего снимка пространства имён из активного NameNode. Команда используется при первой настройке кластера высокой доступности.

Параметры -force или -nonInteractive имеют то же значение, что и описанный в команде -format. Параметр skipSharedEditsCheck пропускает проверку правок, которая гарантирует, что имеется достаточно правок в общем каталоге для запуска с последней контрольной точки в активном узле имён.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-bootstrapStandby [-force] [-nonInteractive] [-skipSharedEditsCheck] ] |

[recover] Восстановление метаданных

Команда recover позволяет восстановить потерянные метаданные в повреждённой файловой системе.

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-recover [-force] ] |

[metadataVersion] Отображение версии метаданных

Команда metadataVersion позволяет вывести версии метаданных программного обеспечения и образа (при условии наличия настроенных каталогов).

Синтаксис команды:

|  |
| --- |
| hadoop namenode [-metadataVersion ] |

[nfs3] Запуск шлюза NFS3

Команда nfs3 запускает шлюз NFS3.

Синтаксис команды:

|  |
| --- |
| hadoop nfs3 |

[portmap] Запуск карты портов RPC

Команда portmap запускает карту портов RPC.

Синтаксис команды:

|  |
| --- |
| hadoop portmap |

[secondarynamenode] Запуск SecondaryNameNode

Команда secondarynamenode запускает SecondaryNameNode.

Синтаксис команды:

|  |
| --- |
| hadoop secondarynamenode [-checkpoint [force]] | [-format] | [-geteditsize] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-checkpoint [force] — контрольные точки SecondaryNameNode, если размер EditLog>=fs.checkpoint.size. Если используется параметр force, контрольная точка не зависит от размера EditLog;

-format — форматирование локального хранилища во время запуска;

-Targetitsize — отображение количества транзакций без отметок на NameNode.

[storagepolicies] Управление политиками хранения

Команда storagepolicies предназначена для управления политиками хранения.

Синтаксис команды:

|  |
| --- |
| hadoop storagepolicies [-listPolicies] [-setStoragePolicy -path <path> -policy <policy>] [-getStoragePolicy -path <path>] [-unsetStoragePolicy -path <path>] [-satisfyStoragePolicy -path <path>] [-isSatisfierRunning] [-help <command-name>] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-listPolicies — перечисляет все политики хранения;

-setStoragePolicy -path <path> -policy <policy> — устанавливает политику хранения для файла или каталога;

-getStoragePolicy -path <path> — получить политику хранения файла или каталога;

-unsetStoragePolicy -path <path> — отменить политику хранения для файла или каталога. После команды unset будет применяться политика хранения ближайшего предка, и, если нет политики для какого-либо предка, будет применяться политика хранения по умолчанию;

-satisfyStoragePolicy -path <path> — запланировать перемещение блоков на основе текущей политики хранения файла/каталога;

-isSatisfierRunning — статус;

-help <command-name> — вызов справки.

[zkfc] Запуск процесса Zookeeper Failover Controller

Команда zkfc запускает процесс Zookeeper Failover Controller.

Синтаксис команды:

|  |
| --- |
| hadoop zkfc [-formatZK [-force] [-nonInteractive]] |

В команде могут быть использованы следующие параметры команды COMMAND\_OPTION:

-formatZK — отформатировать экземпляр Zookeeper;

-force — форматирует znode, если znode существует;

-nonInteractive — форматирование znode прерывается, если znode существует, если не указана опция -force.

-h — показать справку.

АДМИНИСТРИРОВАНИЕ HIVE

Tez

**Tez** позволяет создать комплексный ациклический граф задач для обработки данных. В настоящее время он реализован для **YARN**.

Включает основные возможности:

1. Гибкая модель исполнения.
2. Упрощение развёртывания.
3. Повышение производительности по сравнению с **MapReduce**.
4. Оптимальное управление ресурсами.
5. Планирование реконфигурации во время выполнения.

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.

|  |
| --- |
| **Внимание.**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. |

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.

АДМИНИСТРИРОВАНИЕ YARN

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.

Функции

CapacityScheduler поддерживает следующие функции:

1. 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.

1. Гарантии производительности (Capacity Guarantees).

Очереди распределяются по части пропускной способности сети в том смысле, что в их распоряжении находится определённая 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 жёсткими.

1. 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.

1. 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.

1. Мультитенантность (Multi-tenancy).

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

1. 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 её дочерних очередей. Текущие приложения продолжают выполняться и, таким образом, очередь может быть аккуратно дренажирована. Администраторы также могут запускать остановленные очереди.

1. Планирование на основе ресурсов (Resource-based Scheduling).

Поддержка ресурсоёмких 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.

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

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

1. 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.

1. 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.

1. 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.

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 предопределённую 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, разделённых 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-applications — 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 жёсткое 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 переопределён для каждой очереди путём 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-resource-percent — 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 путём 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 её 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 её 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 её 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 определённую 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 отключённое и является жёстким 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 определённый 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, задаётся в секундах. Любое меньшее или равное нулю значение считается как отключённое. Если пользователь отправляет приложение с незаданным значением срока, то оно задаётся автоматически. Это конфигурация на определённый 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.

|  |
| --- |
| **Внимание.**Приоритет приложения не изменяется при перемещении приложения в другую очередь. |

Преимущественное право в Capacity Scheduler

CapacityScheduler поддерживает возможность преимущественного права (preemption) контейнера от очередей, чьё 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.ProportionalCapacityPreemptionPolicy, 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 мёртвую 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 мёртвую 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 путём 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 путём 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 приведённых 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.

1. 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. Также 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> |

1. 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.

1. 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> |

1. 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

1. 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.

1. 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 ждёт, 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** развёртывается 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.

1. 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 путём запуска кластера **YARN** через веб-интерфейс:

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

Изменение конфигурации очереди

Изменение конфигурации очереди через файл осуществляется путём 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:

1. Остановка очереди — перед удалением leaf-очереди она не должна иметь запущенных/ожидающих приложений и должна быть в статусе STOPPED путём изменения yarn.scheduler.capacity.<queue-path>.state. Перед удалением родительской очереди все её дочерние очереди не должны иметь запущенных/ожидающих приложений и должны быть в статусе STOPPED. Родительская очередь также должна быть STOPPED;
2. Удаление очереди — удалить конфигурации очереди из файла и запустить обновление.

Изменение конфигурации очереди через API осуществляется путём 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.

|  |
| --- |
| **Внимание.**Функция изменения конфигурации очереди через API находится в альфа-фазе и может быть изменена. |

Обновление контейнера

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:

1. Resource Update — когда Application Master может запросить Resource Manager обновить ресурсный размер контейнера. Например: изменить контейнер 2GB, 2 vcore на контейнер 4GB, 2 vcore.
2. 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 приведённым 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 обновлённого контейнера, содержащий обновлённый 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 обновлённого 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.

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 объём 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, включённого 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 определённых 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 объёма 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.

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 имён её 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 путём 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 ещё не поддерживаются в оригинальном (MR1) Fair Scheduler. Среди них — использование индивидуальных политик, управляющих приоритетом “boosting” над определёнными приложениями.

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 приведён формат файла распределения для настройки этих политик.

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> |

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

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 определённым 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, поделённая 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 включённом параметре около половины нераспределённых 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 задаёт никаких ограничений;

yarn.scheduler.fair.locality.threshold.node — число возможностей планирования для приложений, которые запрашивают контейнеры на определённых 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 определённых 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.

1. **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 неудовлетворённой, если её использование памяти ниже минимального совместно используемого объёма памяти. В соответствии с принципом dominant resource очередь считается неудовлетворённой, если её 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 её 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 её 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 отдаётся 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;

aclAdministerApps — список пользователей и/или групп, которые могут управлять очередью. В настоящее время единственным административным действием является уничтожение приложения;

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.

1. **Элементы пользователя**. Представляют собой настройки, управляющие поведением отдельных пользователей. Они могут содержать одно свойство:

maxRunningApps — ограничение количества запущенных приложений для конкретного пользователя.

1. **Элемент** userMaxAppsDefault. Устанавливает лимит запуска приложения по умолчанию для всех пользователей, у которых не указано иное ограничение.
2. **Элемент** defaultFairSharePreemptionTimeout. Задаёт тайм-аут преимущественного права preemption для root-очереди. Переопределяется элементом fairSharePreemptionTimeout в root-очереди. По умолчанию значение Long.MAX\_VALUE.
3. **Элемент** defaultMinSharePreemptionTimeout. Устанавливает минимальное время ожидания преимущественного права preemption для root-очереди. Переопределяется элементом minSharePreemptionTimeout в root-очереди. По умолчанию значение Long.MAX\_VALUE.
4. **Элемент** defaultFairSharePreemptionThreshold. Задаёт порог преимущественного права для root-очереди. Переопределяется элементом fairSharePreemptionThreshold в root-очереди. По умолчанию значение 0.5f.
5. **Элемент** queueMaxAppsDefault. Устанавливает ограничение по умолчанию для запущенного приложения для очередей. Переопределяется элементом maxRunningApps в каждой очереди.
6. **Элемент** queueMaxResourcesDefault. Задаёт максимальный лимит ресурсов по умолчанию для очереди. Переопределяется элементом maxResources в каждой очереди.
7. **Элемент** queueMaxAMShareDefault. Устанавливает ограничение ресурса Application Master по умолчанию для очереди. Переопределяется элементом maxAMShare в каждой очереди.
8. **Элемент** defaultQueueSchedulingPolicy. Устанавливает политику планирования по умолчанию для очередей. Переопределяется элементом schedulingPolicy в каждой очереди, если указан. По умолчанию fair.
9. **Элемент** reservation-agent. Задаёт 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.
10. **Элемент** reservation-policy. Задаёт 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.
11. **Элемент** reservation-planner. Задаёт 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.
12. **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 создаёт 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 её родительской (приведено далее в примере распределённого 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 распределённого 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> |

|  |
| --- |
| **Внимание.**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 учётом пробела). Действия в очереди разрешены, если пользователь/группа является членом Queue ACL самой очереди или любой из её 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.

|  |
| --- |
| **Внимание.**Пробел является разделителем. Для того, чтобы указать только группы ACL, значение должно начинаться с символа пробела. |

|  |
| --- |
| **Внимание.**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 определённых 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 учётом 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.

|  |
| --- |
| **Внимание.**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.

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 изменён.

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 ещё 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 её 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 неё завершается неудачей.

Дамп состояния Fair Scheduler

Fair Scheduler может периодически сбрасывать своё состояние. По умолчанию данная функция отключена, но администратор может включить её, 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 объём данных, поэтому для того, чтобы вывести состояние в отдельный файл, необходимо раскомментировать раздел Fair scheduler state dump в log4j.properties.

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:

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 распределённую архитектуру записи и масштабируемое backend-хранилище.

YARN Timeline Service v.2 отделяет сбор (запись) данных от обслуживания (чтения) данных. Он использует распределённые 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.

1. **Улучшения юзабилити**.

В большинстве случаев пользователи интересуются информацией на уровне “потоков” (flows) или логических групп приложений **YARN**. Гораздо более распространённым 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.

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 объём записей.

Timeline-читатели — это отделённые от timeline-коллекторов демоны, предназначенные для обслуживания запросов через REST API.

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.

Отключённый 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.

Конфигурация. Basic

Конфигурация. Basic:

yarn.timeline-service.enabled — указывает клиентам, включён ли сервис Timeline. При включённом 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.

Конфигурация. Advanced

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, введённые 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 завершённого 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.

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 — разделённый запятыми список пользователей и после пробела разделённый запятыми список групп. Функция позволяет введённому списку пользователей и групп читать данные и отклонять остальных пользователей и группы. По умолчанию установлено значение none. Если авторизация включена, то данный параметр обязателен.

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.

Включение Timeline Service v.2

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

1. Настройка кластера HBase.
2. Включение сопроцессора.
3. 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 развёртывания. 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 развёртывания “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/coprocessorhadoop 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.hdfs.location. 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..

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> |

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 |

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> |

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.

Публикация определённых 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.2TimelineV2Client 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) |

Ещё один адрес должен быть извлечён из распределённого 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 задаётся 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 tagsSet<String> tags = new HashSet<>();tags.add(TimelineUtils.generateFlowNameTag("distributed grep"));tags.add(Timelineutils.generateFlowVersionTag("3df8b0d6100530080d2e0decf9e528e57c42a90a"));tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));appContext.setApplicationTags(tags); |

|  |
| --- |
| **Внимание.**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. |

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

С помощью Query Flows API можно получить список активных потоков, запущенных за последнее время. Если используется конечная точка REST без имени кластера, берётся 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/orGET /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, разделённые 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 (ОК) — 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/orGET /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>…) — разделённый 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 разделённый 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 (ОК) — 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 определённый 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, берётся 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}orGET /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>…) — разделённый 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 разделённый 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 (ОК) — 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 определённого потока. Если используется конечная точка REST без имени кластера, берётся 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}/appsorGET /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 оp 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 оp 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 её 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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 (ОК) — 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 определённого flow run. Если используется конечная точка REST без имени кластера, берётся 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}/appsorGET /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 оp 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 оp 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 её 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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 (ОК) — 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}orGET /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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 (ОК) — 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}orGET /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 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 оp 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 оp 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 её 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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 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": [ ], "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 (ОК) — 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, берётся 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}orGET /ws/v2/timeline/users/{userid}/entities/{entitytype} |

**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 оp 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 оp 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 её 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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 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": [ ], "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 (ОК) — 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 определённую 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 путём указания типа сущности как YARN\_APPLICATION\_ATTEMPT, а application attempt ID в виде идентификатора сущности.

**HTTP**:

|  |
| --- |
| GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}/{entity id}orGET /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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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;

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 (ОК) — 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, берётся 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}orGET /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>…) — разделённый 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 разделённый 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>…) — разделённый 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 разделённый 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 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": [ ], "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 (ОК) — 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, берётся 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-typesorGET /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 (ОК) — 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.

Hadoop: YARN Federation

Архитектура

Известно, что **YARN** масштабируется до тысяч узлов. Масштабируемость **YARN** определяется Resource Manager, и она пропорциональна количеству узлов, активных приложений и контейнеров, а также 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 путём 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.

YARN Sub-cluster

Субкластер (sub-cluster) — это кластер **YARN** размером до нескольких тысяч узлов. Точный размер субкластера определяется с учётом простоты развёртывания/обслуживания, согласования с сетевыми зонами и 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 — её 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 ёмкости, 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 (путём 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.

Роль AMRMProxy:

1. 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).

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 включённого 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.

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:

1. 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.

1. 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 определённых случаях запрашивал ресурсы у других субкластеров.

1. 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.

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.

Последовательность для 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.

1. Затем Application Master запрашивает контейнеры, используя информацию о местонахождении, предоставляемую **HDFS**.
2. 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.

1. 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.
2. 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.
3. AMRMProxy пересылает ответ распределения обратно в Application Master.
4. Application Master запускает контейнер на целевом NodeManager (в субкластере 2), используя стандартные протоколы **YARN**.

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> |

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](https://mvnrepository.com/artifact/mysql/mysql-connector-java) и добавить её в CLASSPATH. Затем схема БД создаётся путём 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.

|  |
| --- |
| **Внимание.**FederationStateStoreUser.sql определяет для БД пользователя/пароль по умолчанию, для которого настоятельно рекомендуется установить собственный надёжный пароль. |

Для SQL-сервера процесс аналогичен, но драйвер jdbc уже включён. Скрипты SQL-сервера находятся в каталоге sbin/FederationStateStore/SQLServer/.

Опциональные:

yarn.federation.failover.enabled — следует ли повторить попытку, учитывая отказоустойчивость Resource Manager в каждом субкластере. Пример значения:

|  |
| --- |
| true |

yarn.federation.blacklist-subclusters — список чёрных 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 путём сериализации policymanager, который был сконфигурирован программно, или путём 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 разнесённым, 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 ещё 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 своём 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 — разделённый 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 — разделённый 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, включён ли AMRMProxy. Пример значения:

|  |
| --- |
| true |

yarn.nodemanager.amrmproxy.interceptor-class.pipeline — разделённый 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, включён 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 |

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 передаётся на маршрутизатор, который использует сгенерированную политику из GPG, чтобы выбрать домашний Resource Manager для задания.

Выходные данные приведённого примера задания должны быть примерно такими:

|  |
| --- |
| 2017-07-13 16:29:25,055 INFO mapreduce.Job: Job job\_1499988226739\_0001 running in uber mode : false2017-07-13 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 secondsEstimated 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 приведённого задания такие же, как и при запуске без Federation. Чтобы получить все преимущества Federation, рекомендуется использовать большее количество mappers, чем того требует кластер. Для приведённого примера это число составляет 16.