Платформа работы с данными
Sber Data Platform

Руководство пользователя для работы с компонентами SDP Hadoop

Оглавление

1 Компонент Ambari ... 4
1.1 Что такое Apache Ambari ... 4
1.2 Архитектура ... 5
1.2.1 Агенты ... 5
1.2.2 Сервер Ambari ... 6
1.3 Метрики Ambari ... 7
1.4 API Ambari ... 8
1.4.1 Initialization vars ... 8
1.4.2 STATE CONTROL ... 8
1.4.3 SERVICE CHECK ... 10
1.4.4 CONFIGURATION MANAGEMENT ... 13

2 Компонент HBase ... 16
2.1 Как хранится информация в NoSQL СУБД: модель данных ... 16
2.2 Работа HBase с массивами Big Data: принцип действия ... 17
2.3 Кластерная архитектура ... 19
2.4 Hbase Shell ... 20
2.5 Java API ... 23

3 Компонент HDFS ... 27
3.1 Особенности HDFS ... 27
3.2 Архитектура HDFS ... 27
3.3 Цели HDFS ... 28
3.4 Основные операции HDFS ... 29

4 Компонент Hive ... 33
4.1 Особенности Hive ... 33
4.2 Архитектура Hive ... 33
4.3 Работа Hive ... 34
4.3.1 Форматы хранения данных ... 35
4.3.2 Основные сущности в Hive: ... 37

5 Компонент HUE ... 43
5.1 Что такое HUE? ... 43
5.2 Hue interface concept ... 43
5.3 Hue Server ... 43
5.4 Hue DB ... 44
5.5 Основной функционал HUE ... 44
5.6 Интеграции HUE с компонентами Hadoop: ... 44

6 Компонент MapReduce ... 45
6.1 Почему MapReduce? ... 45
6.2 Принцип работы MapReduce ... 45
6.3 MapReduce - API JAVA ... 47
6.3.1 JobContext Interface ... 47
6.3.2 Job Class ... 47
6.3.3 Методы ... 48
6.3.4 Mapper Class ... 49
6.3.5 Reducer Class ... 49

7 Компонент Oozie ... 50
7.1 Как работает Oozie? ... 50
7.2 Упаковка и развертывание workflow приложения Oozie ... 51
7.3 Command Line Tools ... 52

8 Компонент Solr ... 54
8.1 Основные возможности ... 54
8.2 Архитектура ... 54
8.3 Интерфейсы ... 54
8.4 API коллекций ... 55
8.5 Asynchronous Calls ... 55
8.5.1 Примеры асинхронных запросов ... 56
8.5.2 REQUESTSTATUS: статус запроса асинхронного вызова ... 56
8.5.3 DELETESTATUS: удалить статус ... 56
8.6 Команды управления кластерами и узлами ... 57
8.6.1 CLUSTERSTATUS ... 57
8.6.2 CLUSTERPROP: свойства кластера ... 57
8.6.3 BALANCESHARDUNIQUE: сбалансируйте свойство между узлами ... 58

8.6.4 UTILIZENODE: использование нового узла ... 58
8.6.5 REPLACENODE: перемещение всех реплик в одном узле в другой ... 58
8.6.6 DELETENODE: удаление реплик в узле ... 59
8.6.7 ADDROLE: добавление роли ... 59
8.6.8 REMOVEROLE: удалить роль ... 60
8.6.9 OVERSEERSTATUS: статус и статистика Overseerstatus ... 60
8.6.10 MIGRATESTATEFORMAT: миграция состояния кластера ... 60

9 Компонент Spark ... 62
9.1 Что такое Spark? ... 62
9.2 Компоненты Spark ... 62
9.2.1 Ядро Spark ... 63
9.2.2 Скорость ... 63
9.2.3 Простота использования ... 63
9.2.4 Обработка больших наборов данных ... 63
9.2.5 Функциональность ... 64
9.2.6 SparkSQL ... 64
9.2.7 Spark Streaming ... 64
9.2.8 MLlib ... 65
9.2.9 GraphX ... 65
9.3 Spark Shell ... 65
9.3.1 Преобразования RDD ... 65
9.3.2 Действия RDD ... 67
9.3.3 Числовые операции RDD ... 68

10 Компонент Sqoop ... 70
10.1 Архитекрута Sqoop ... 70
10.2 Импорт данных ... 70
10.3 Экспорт данных ... 74
10.4 Sqoop Job ... 74

11 Компонент Tez ... 76
11.1 Сравнение производительности MapReduce и TEZ ... 76
11.2 Сравнение Tez и Spark ... 77
11.3 API для определения ориентированных ациклических графов (DAG) ... 77
11.4 Интерфейсы взаимодействия ... 77

12 Компонент YARN ... 78
12.1 Архитектура YARN ... 78
12.1.1 Resource Manager (менеджер ресурсов) ... 79
12.1.2 Node Manager (диспетчер узлов) ... 79
12.1.3 Application Master (мастер приложений) ... 79
12.1.4 Container (контейнер) ... 80
12.2 Подача заявки в Yarn ... 80
12.3 Рабочий процесс приложения в Hadoop YARN ... 81

13 Компонент Zookeeper ... 82
13.1 Свойства Zookeeper: ... 82
13.2 Для чего нужен Zookeeper? ... 83
13.3 Поддерживаемые операции ... 83
13.3.1 CLI ZooKeeper ... 83
13.3.2 API ZooKeeper ... 85

1 Компонент Ambari

1.1 Что такое Apache Ambari

Ambari позволяет системным администраторам управлять и контролировать кластер Hadoop, а также интегрировать Hadoop с существующей корпоративной инфраструктурой. Основные возможности:

- Обеспечение работоспособности
  - Пошаговая установка сервисов Hadoop для любого количества хостов
  - Поддерживает конфигурации сервисов Hadoop для кластера
- Управление кластером Hadoop
  - Централизованное управление запуском, остановкой и конфигурацией сервисов Hadoop для всего кластера
- Контроль кластера Hadoop
  - Доступ к панели контроля состояния кластера Hadoop
  - Запускает систему измерений Ambari для записи измерений
  - Запускает фреймворк сигналов Ambari для системных оповещений (ядро вышло из строя, недостаточно свободного дискового пространства и др.).
- Интеграция Hadoop
  - Интеграция возможностей Hadoop для ваших приложений при помощи REST API

Ambari в SDP Hadoop в настоящее время позволяет управлять следующими компонентами:

- HDFS
- YARN
- MapReduce
- Hive
- Tez
- HBase
- ZooKeeper
- Oozie
- Spark2
- Kerberos

1.2 Архитектура

Высокоуровневая архитектура Ambari представлена на рисунке 1. Ambari состоит из двух основных компонентов: сервера (Ambari Server) и агентов (Agents).

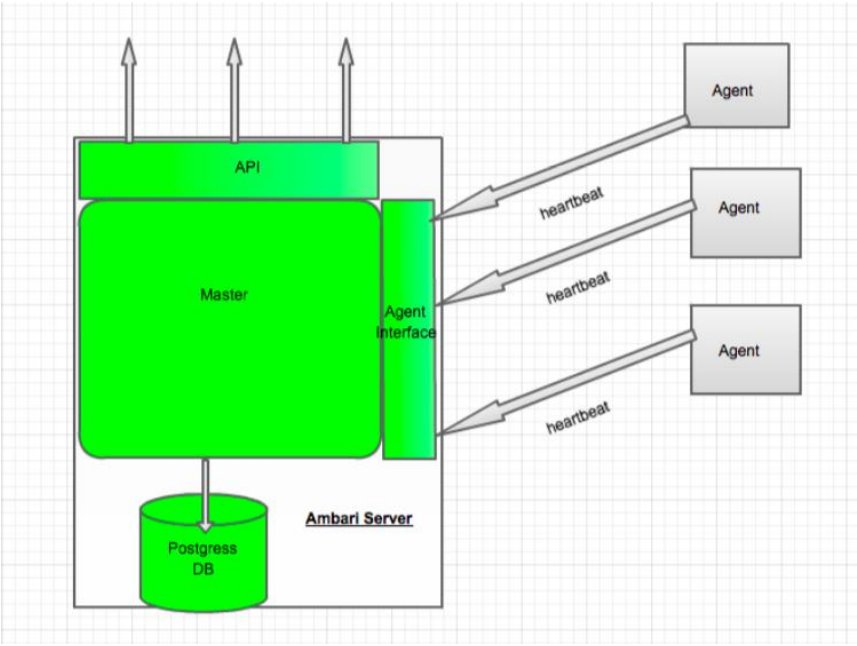


Рисунок 1. Верхнеуровневая архитектура Ambari

1.2.1 Агенты

Агенты каждые несколько секунд отправляют запрос (heartbeat) мастеру, в ответе запроса они получают команды от мастера. Такие запросы - это единственный способ мастера отправить команду агентам.

Полученная команда помещается в очередь, называемую очередь заданий (action queue). Задачи из нее получает исполнитель задач (action executioner). Исполнитель задач подбирает правильный инструмент для выполнения (Puppet, Python, и т.д.) в зависимости от типа команды и задачи. Таким образом, задачи, полученные в ответе на запрос, будут обработаны агентом асинхронно. Исполнитель задач помещает сообщения о процессе выполнения задачи в очередь сообщений (message queue). Затем агент все сообщения, лежащие в очереди, отправит мастеру в следующем запросе.

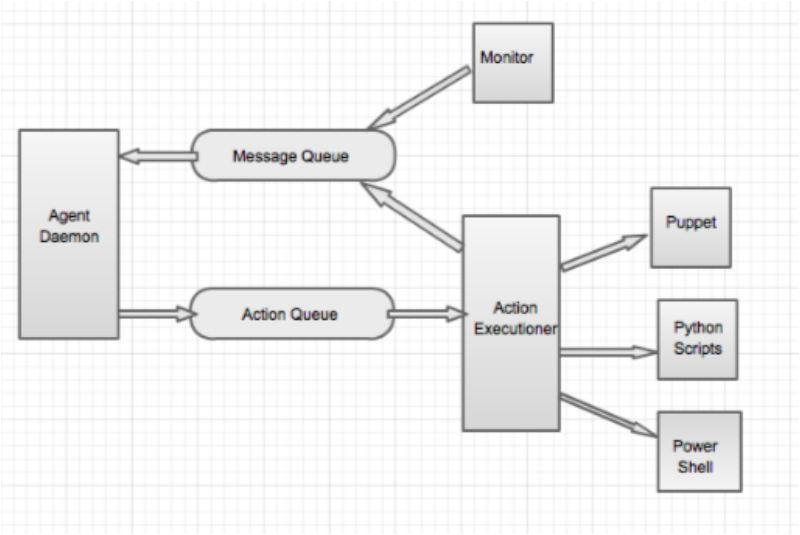


Рисунок 2. Архитектура Ambari Agent

1.2.2 Сервер Ambari

Запросы на сервер попадают с помощью API. Сначала для каждого запроса генерируется уникальный request id, затем в Координаторе (Coordinator) для пришедшего в запросе API вызывается обработчик (handler).

Обработчик API реализует шаги, необходимые для выполнения запрошенного API. Например, в случае добавления нового сервиса к существующему кластеру такими шагами будут: установить все компоненты сервиса вместе с необходимыми зависимостями (prerequisites), запустить их в специальном порядке и перенастроить сервер Nagios, чтобы добавить новый сервис к мониторингу.

Координатор может обращаться к Трекеру Зависимостей (Dependency Tracker), чтобы найти полный набор компонентов и их необходимые состояния. Затем Координатор передает список компонентов и их состояния в Планировщик Этапов (Stage Planner). Планировщик Этапов возвращает поэтапную последовательность операций, которые необходимо произвести на каждой ноде, на которой переданные компоненты будут установлены/запущены/изменены. Планировщик Этапов также генерирует манифест (список задач для каждой отдельной ноды для каждого отдельного этапа), используя Генератор Манифеста (Manifest Generator).

Затем Координатор передает упорядоченный список этапов и request id в Менеджер Задач (Action Manager).

Менеджер Задач обновляет в FSM состояние каждой ноды и ее компонентов, что будет обозначать, что операция в состоянии выполнения. Важно, что FSM обновляется для каждой затронутой ноды. На этой стадии FSM может обнаружить недействительное программное событие и бросить ошибку, что прервет выполнение операции и все задачи будут помечены как упавшие с ошибкой.

Дальше Менеджер Задач для каждой операции создает идентификатор action id и добавляет его в план. Менеджер Задач выберет первый этап из плана и все действия этого этапа добавит в очередь каждой затронутой ноды. Когда с первым этапом будет закончено, Менеджер Задач перейдет к следующему и так до конца. Для запланированных задач запускается таймер.

Обработчик Запросов (Heartbeat Handler) получит запрос на выполнение задачи и уведомит об этом Менеджер Задач. Затем Менеджер Задач отправит в FSM событие, чтобы обновить его состояние. В случае таймаута, задача будет запланирована на выполнение заново или помечена как завершившаяся с ошибкой. Когда все ноды, затронутые для задачи, достигнут состояния выполнения (придет ответ или наступит время финального таймаута), задача будет считаться выполненной. И когда все задачи этапы будут завершены, этап будет считаться выполненным.

Завершение задачи также записывается в базе данных. Затем Менеджер Задач переходит к следующему этапу, и все повторяется.

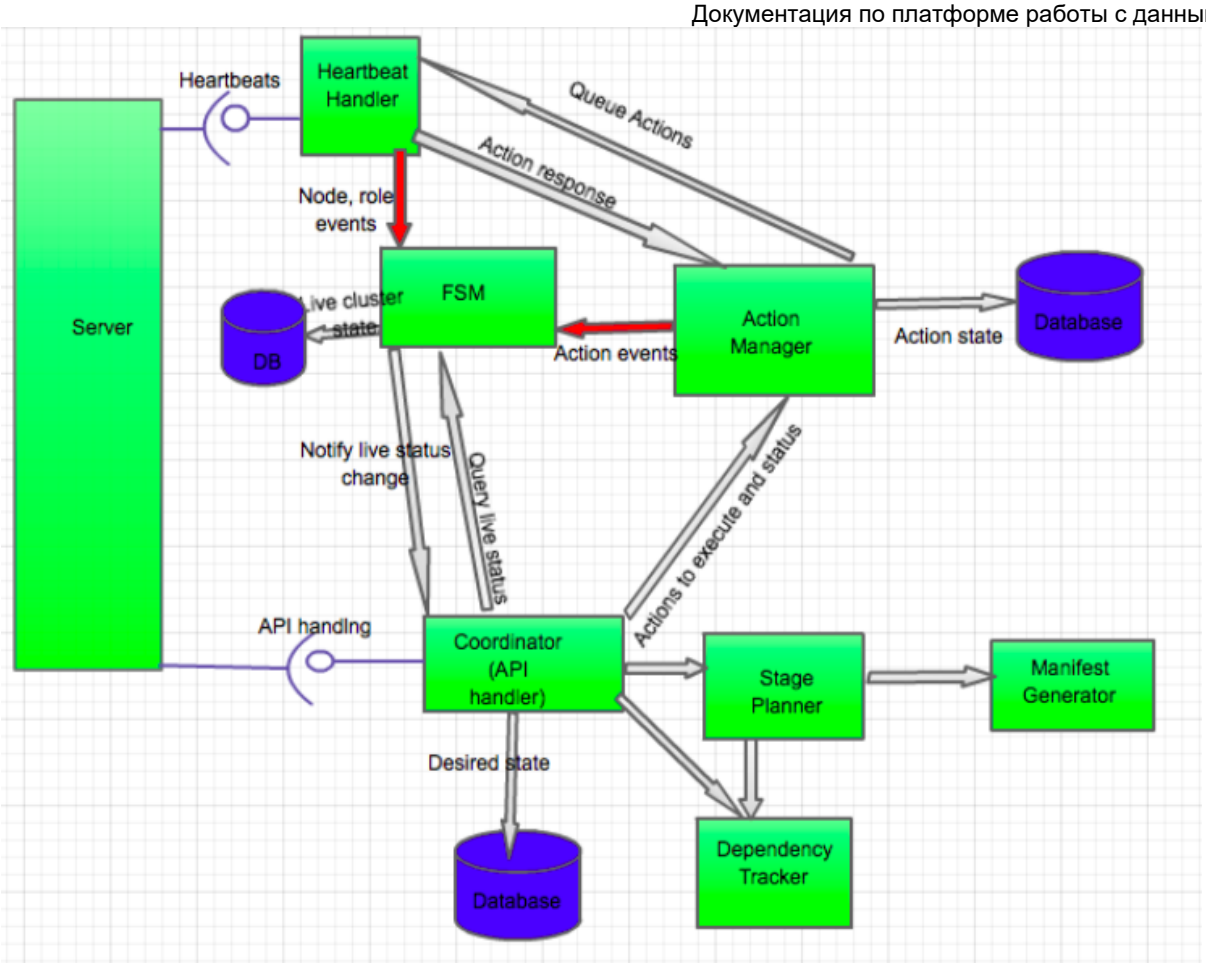


Рисунок 3. Архитектура Ambari Server

1.3 Метрики Ambari

Ambari Metrics System ("AMS") - это система сбора, агрегирования и обслуживания метрик Hadoop и системы в кластерах, управляемых Ambari.

Терминология

Срок	Определение
Ambari Metrics System ("AMS")	Встроенная система сбора метрик для **Кенаф**.
Metrics Collector	Автономный сервер, который собирает метрики, агрегирует метрики, обслуживает метрики из приемников служб Hadoop и **Metrics Monitor**.
Metrics Monitor	Устанавливается на каждом узле кластера для сбора метрик системного уровня и пересылки в **Metrics Collector**.
Metrics Hadoop Sinks	Подключается к различным приемникам компонентов Hadoop для отправки метрик Hadoop в **Metrics Collector**.

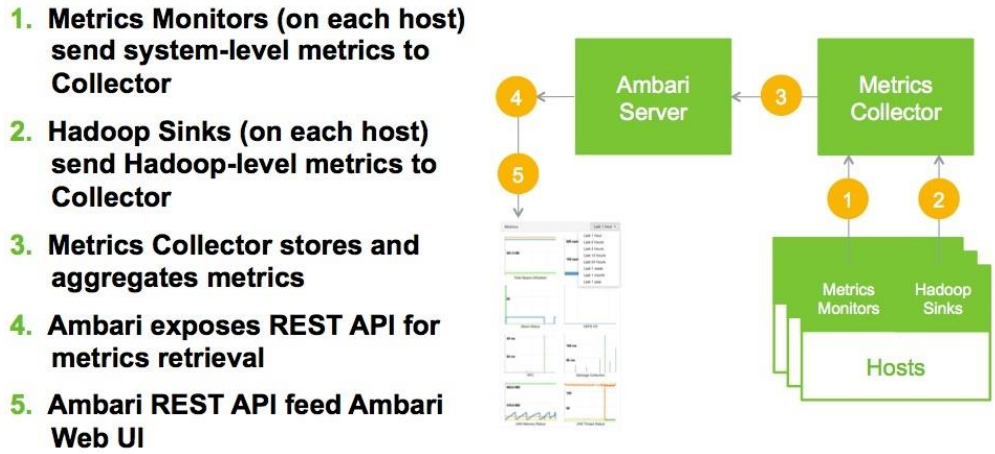


Рисунок 4. Архитектура системы метрик Ambari

Metrics Collector - это демон, который получает данные от зарегистрированных издателей (мониторов и приемников). Сам Collector построен с использованием таких технологий Hadoop, как HBase Phoenix и ATS. Сборщик может хранить данные в локальной файловой системе (называемой «встроенным режимом») или использовать внешнюю HDFS (именуемую «распределенным режимом»).

1.4 API Ambari

1.4.1 Initialization vars

AMBARI_USERNAME=infra

AMBARI_PASSWORD=

AMBARI_HOSTNAME=host1.dev.ru

CLUSTER_NAME=SDP_405cf916

SERVICE=ZOOKEEPER

COMPONENT=ZOOKEEPER_SERVER

HOST_NAME=host2.dev.ru

1.4.2 STATE CONTROL

1.4.2.1 START ALL SERVICES

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"_PARSE_.START.ALL_SERVICES","operation_level":{"level":"CLUSTER","clust
er_name":"$CLUSTER_NAME"}},"Body":{"ServiceInfo":{"state":"STARTED"}}}' http://$AMBARI_HOSTNAME:
8080/api/v1/clusters/$CLUSTER_NAME/services
```

1.4.2.2 STOP ALL SERVICES

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"_PARSE_.STOP.ALL_SERVICES","operation_level":{"level":"CLUSTER","cluster
_name":"$CLUSTER_NAME"}},"Body":{"ServiceInfo":{"state":"INSTALLED"}}}' http://$AMBARI_HOSTNAME:8
080/api/v1/clusters/$CLUSTER_NAME/services
```

1.4.2.3 START SERVICE

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo": {"context" :"Start ZOOKEEPER via REST"}, "Body": {"ServiceInfo": {"state":
"STARTED"}}}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/services/$SERVICE
```

1.4.2.4 STOP SERVICE

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo": {"context" :"Stop ZOOKEEPER via REST"}, "Body": {"ServiceInfo": {"state":
"INSTALLED"}}}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/services/$SERVICE
```

1.4.2.5 GET STATE OF COMPONENT ON HOST

```
curl -k -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By:ambari" -i -X GET
http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/hosts/$HOST_NAME/host_component
s/$COMPONENT
```

1.4.2.6 START COMPONENT ON HOST

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"Start ZOOKEEPER_SERVER;"},
"Body":{"HostRoles":{"state":"STARTED"}}}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_N
AME/hosts/$HOST_NAME/host_components/$COMPONENT
```

1.4.2.7 STOP COMPONENT ON HOST

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"Stop ZOOKEEPER_SERVER"},
"Body":{"HostRoles":{"state":"INSTALLED"}}}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_
NAME/hosts/$HOST_NAME/host_components/$COMPONENT
```

1.4.2.8 START SPECIFIED COMPONENTS ON HOST

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"Start Specified Host
Components","operation_level":{"level":"HOST","cluster_name":"sdp","host_names":"bigtop1.vagrant.local"},"q
uery":"HostRoles/component_name.in(APP_TIMELINE_SERVER,DATANODE,HISTORYSERVER,METRICS
_COLLECTOR,METRICS_GRAFANA,METRICS_MONITOR,NAMENODE,NFS_GATEWAY,NODEMANAGE
R,RANGER_ADMIN,RANGER_TAGSYNC,RANGER_USERSYNC,RESOURCEMANAGER,SECONDARY_N
AMENODE,ZOOKEEPER_SERVER)"},"Body":{"HostRoles":{"state":"STARTED"}}}' http://$AMBARI_HOSTNA
ME:8080/api/v1/clusters/$CLUSTER_NAME/hosts/$HOST_NAME/host_components
```

1.4.2.9 STOP SPECIFIED COMPONENTS ON HOST

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X PUT -
d '{"RequestInfo":{"context":"Stop Specified Host
Components","operation_level":{"level":"HOST","cluster_name":"sdp","host_names":"bigtop1.vagrant.local"},"q
uery":"HostRoles/component_name.in(APP_TIMELINE_SERVER,DATANODE,HISTORYSERVER,METRICS
_COLLECTOR,METRICS_GRAFANA,METRICS_MONITOR,NAMENODE,NFS_GATEWAY,NODEMANAGE
R,RANGER_ADMIN,RANGER_TAGSYNC,RANGER_USERSYNC,RESOURCEMANAGER,SECONDARY_N
```

AMENODE,ZOOKEEPER_SERVER)"},"Body":{"HostRoles":{"state":"INSTALLED"}}}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/hosts/$HOST_NAME/host_components

1.4.2.10 RESTART ALL REQUIRED SERVICES

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X POST -d '{"RequestInfo":{"command":"RESTART","context":"Restart all required services","operation_level":"host_component"},"Requests/resource_filters":[{"hosts_predicate":"HostRoles/stale_configs=true&HostRoles/cluster_name=sdp"}]}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/requests

1.4.3 SERVICE CHECK

1.4.3.1 GET STATE OF SERVICE

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -i -H 'X-Requested-By: ambari' -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/services/$SERVICE

1.4.3.2 GET STATE OF COMPONENT

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -i -H 'X-Requested-By: ambari' -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/services/$SERVICE/components/$COMPONENT

1.4.3.3 GET STATE OF COMPONENT ON HOST

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -i -H 'X-Requested-By: ambari' -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/hosts/$HOST_NAME/host_components/$COMPONENT

1.4.3.4 GET STATE OF ALL COMPONENTS ON HOST

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -i -H 'X-Requested-By: ambari' -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/hosts/$HOST_NAME/host_components?fields=HostRoles/state

1.4.3.5 GET ALERTS

curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -i -H 'X-Requested-By: ambari' -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/alert_definitions

1.4.3.6 START SINGLE SERVICE CHECK (e.g. HDFS Service Check):

```
payload()
{
  cat <<EOF
{
   "RequestInfo":{
      "context":"HDFS Service Check",
      "command":"HDFS_SERVICE_CHECK"
```

```
    },
    "Requests/resource_filters":[
      {
        "service_name":"HDFS"
      }
    ]
  }
  EOF
  }

  curl -ivk -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -H "Content-Type:
  application/json" -X POST -
  d "$(payload)" http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/requests
```

1.4.3.7 GET SERVICE CHECK RESULT:

```
  curl -ivk -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -H "Content-Type:
  application/json" -X GET http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/requests/$ID
```

1.4.3.8 START BULK SERVICE CHECKS

(e.g. HDFS, Yarn, MapReduce2 Service Checks)

```
  payload()
  {
    cat <<EOF
  [
    {
      "RequestSchedule":{
        "batch":[
          {
            "requests":[
              {
                "order_id":1,
                "type":"POST",
                "uri":"/api/v1/clusters/SDP_405cf916/requests",
                "RequestBodyInfo":{
                  "RequestInfo":{
                    "context":"HDFS Service Check (batch 1 of 3)",
                    "command":"HDFS_SERVICE_CHECK"
```

```
      },
      "Requests/resource_filters":[
        {
          "service_name":"HDFS"
        }
      ]
    }
  },
  {
    "order_id":2,
    "type":"POST",
    "uri":"/api/v1/clusters/SDP_405cf916/requests",
    "RequestBodyInfo":{
      "RequestInfo":{
        "context":"Zookeeper Service Check (batch 2 of 3)",
        "command":"ZOOKEEPER_QUORUM_SERVICE_CHECK"
      },
      "Requests/resource_filters":[
        {
          "service_name":"ZOOKEEPER"
        }
      ]
    }
  },
  {
    "order_id":3,
    "type":"POST",
    "uri":"/api/v1/clusters/SDP_405cf916/requests",
    "RequestBodyInfo":{
      "RequestInfo":{
        "context":"Oozie Service Check (batch 3 of 3)",
        "command":"OOZIE_SERVICE_CHECK"
      },
      "Requests/resource_filters":[
        {
```

```
                "service_name":"OOZIE"
              }
            ]
          }
        }
      ]
    },
    {
      "batch_settings":{
        "batch_separation_in_seconds":1,
        "task_failure_tolerance":1
      }
    }
  ]
}
}
]
EOF
}
```

```
curl -ivk -H "X-Requested-By: ambari" -u $AMBARI_USERNAME:$AMBARI_PASSWORD -X POST -
d "$(payload)" http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/request_schedules
```

1.4.4 CONFIGURATION MANAGEMENT

1.4.4.1 GET ALL CONFIGURATIONS

```
curl -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -i -X GET
http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/configurations |
?type=global&tag=version1
```

1.4.4.2 GET ALL SERVICE CONFIGURATIONS

```
curl -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -i -X GET
http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/configurations/service_config_versions
| ?is_current=true
```

Description of fields:

configurations - contains list of configs related to this service config version

createtime - timestamp indicating when service config version was created

group_id and group_name - id and name of config group for which service config version was created. -
1 indicates default one (theres no group from server point of view)

hosts - also related to config groups - list of hosts in _cg

is_current - indicates if service config version is currently active (its configs are used for service)

service_config_version - number of service config version (per-service numeration)

service_config_version_note - user comment left on service config version creation

service_name - name of service which service config version is related to

1.4.4.3 CHANGE SERVICE CONFIGURATION

```
payload()
{
  cat <<EOF
{
  "Clusters": {
    "desired_config": {
      "type": "zoo.cfg",
      "tag": "version100501",
      "properties": {
        "autopurge.purgeInterval" : "24",
        "autopurge.snapRetainCount" : "30",
        "clientPort" : "2181",
        "dataDir" : "/hadoop/zookeeper",
        "initLimit" : "10",
        "quorum.auth.enableSasl" : "false",
        "quorum.cnxn.threads.size" : "20",
        "syncLimit" : "4",
        "tickTime" : "3000"
      }
    }
  }
}
EOF
}

curl -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -i -X PUT -
d "$(payload)" http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME
```

1.4.4.4 RESTART ALL REQUIRED SERVICES

```
curl -i -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -X POST -
d '{"RequestInfo":{"command":"RESTART","context":"Restart all required
```

```
services","operation_level":"host_component"},"Requests/resource_filters":[{"hosts_predicate":"HostRoles/stale_configs=true&HostRoles/cluster_name=sdp"}]}' http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME/requests
```

1.4.4.5 ROLLBACK TO ANY service config version

```
payload()
{
  cat <<EOF
{
  "Clusters": {
    "desired_service_config_versions": {
      "service_name" : "ZOOKEEPER",
      "service_config_version" : 3,
      "service_config_version_note" : "Manual rollback to service config version 1 (ZOOKEEPER)"
    }
  }
}
EOF
}

curl -u $AMBARI_USERNAME:$AMBARI_PASSWORD -H "X-Requested-By: ambari" -i -X PUT -d "$(payload)" http://$AMBARI_HOSTNAME:8080/api/v1/clusters/$CLUSTER_NAME
```

2 Компонент HBase

HBase – это не реляционная, распределенная база данных с открытым исходным кодом, написанная на языке Java. СУБД относится к категории «семейство столбцов» (wide-column store) и представляет собой колоночно-ориентированное, мультиверсионное хранилище типа «ключ-значение» (key-value). Она работает поверх распределенной файловой системы HDFS и обеспечивает возможности BigTable для Hadoop, реализуя отказоустойчивый способ хранения больших объёмов разреженных данных.

2.1 Как хранится информация в NoSQL СУБД: модель данных

В HBase данные хранятся в таблицах, в которых есть строки и столбцы. Это совпадение терминологии с реляционными базами данных (RDBMS), но это не правильная аналогия. Вместо этого может быть полезно думать о таблице HBase как о многомерной карте.

- **Table.** Таблица HBase состоит из нескольких строк.
- **Row.** Строка в HBase состоит из ключа строки и одного или нескольких столбцов со значениями, связанными с ними. Строки сортируются в алфавитном порядке по ключу строки по мере их сохранения. По этой причине дизайн ключа строки очень важен. Цель состоит в том, чтобы хранить данные таким образом, чтобы связанные строки находились рядом друг с другом. Распространенным шаблоном ключей строк является домен веб-сайта. Если ваши ключи строк являются доменами, вам, вероятно, следует хранить их в обратном порядке (org._____.www,). Таким образом, все домены находятся рядом друг с другом в таблице, а не распределены по первой букве субдомена.
- **Column.** Столбец в HBase состоит из семейства столбцов и спецификатора столбца, разделенных символом (двоеточие).
- **Column Family.** Семейства столбцов физически объединяют набор столбцов и их значения, часто по соображениям производительности. Каждое семейство столбцов имеет набор свойств хранения, например, должны ли его значения кэшироваться в памяти, как сжимаются его данные или кодируются ключи строк, и другие. Каждая строка в таблице имеет одинаковые семейства столбцов, хотя данная строка может ничего не хранить в данном семействе столбцов.
- **Column Qualifier.** К семейству столбцов добавляется спецификатор столбца, чтобы обеспечить индекс для данного фрагмента данных. Для данного семейства content столбцов может использоваться квалификатор столбца content:html, а для другого - content:pdf. Хотя семейства столбцов фиксируются при создании таблицы, квалификаторы столбцов являются изменяемыми и могут сильно различаться между строками.
- **Cell.** Ячейка представляет собой комбинацию строки, семейства столбцов и спецификатора столбца и содержит значение и метку времени, которая представляет версию значения.
- **Timestamp.** Отметка времени записывается рядом с каждым значением и является идентификатором для данной версии значения. По умолчанию отметка времени представляет время на RegionServer, когда данные были записаны, но вы можете указать другое значение отметки времени при помещении данных в ячейку.

<table, RowKey, Column Family, Column, timestamp> -> Value

Данные в HBase организованы в таблицы, проиндексированные первичным ключом (**RowKey**).

Для каждого первичного ключа может храниться неограниченный набор **атрибутов (колонок)**.

Колонки организованны в **группы колонок** (**Column Family**). Обычно в одну группу объединяют колонки с одинаковым шаблоном использования и хранения. Список и названия **групп колонок** фиксирован и имеет четкую схему. На уровне группы колонок задаются такие параметры как time to live (TTL) и максимальное количество хранимых **версий**.

Для каждого **атрибута** может храниться несколько различных **версий**. Разные версии имеют разный штамп времени (**timestamp, ts**).

Записи физически хранятся в порядке, отсортированном по первичному ключу. При этом информация из разных колонок хранится отдельно, благодаря чему можно считывать данные только из нужного семейства колонок, таким образом, ускоряя операцию чтения.

Атрибуты, принадлежащие одной **группе колонок** и соответствующие одному **ключу,** физически хранятся как отсортированный список. Любой **атрибут** может отсутствовать или присутствовать для каждого ключа. Отсутствие атрибута не влечет никаких накладных расходов на хранение пустых значений.

Если разница между штампом времени (**timestamp**) для определенной **версии** и текущим временем больше TTL, такая запись помечается **к удалению**. Аналогично, если количество **версий** для определённого **атрибута** превысило максимальное количество версий.

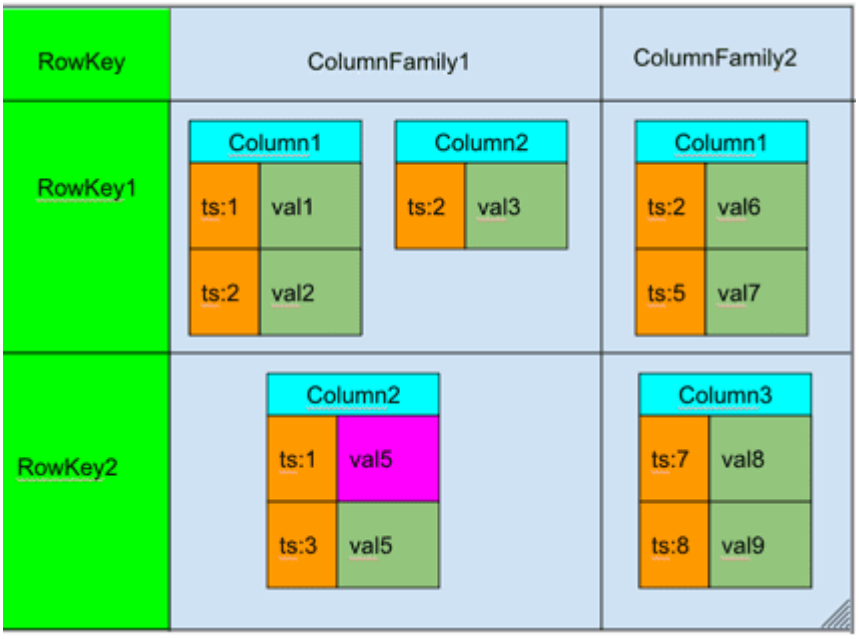


Рисунок 1. Модель данных HBase

2.2 Работа HBase с массивами Big Data: принцип действия

HBase обеспечивает произвольный доступ в реальном времени к данным в Hadoop в сочетании с удобством пакетной обработки. Благодаря этому можно работать с очень большими таблицами, что позволяет эффективно хранить многостраничные или разреженные данных.

Поскольку HBase является распределенной СУБД, она может работать на десятках и сотнях физических серверов, обеспечивая бесперебойную работу даже при выходе из строя некоторых узлов. Распределение данных по разным физическим машинам кластера обеспечивается механизмом **регионирования** – автоматической горизонтальной группировки табличных строк.

Регион — это диапазон записей, соответствующих определенному диапазону подряд идущих первичных ключей. Каждый регион содержит следующие параметры:

- **Persistent Storage**— основное хранилище данных в Hbase. Данные физически хранятся на HDFS, в специальном формате HFile, отсортированные по значению первичного ключа (RowKey). Одной паре (регион, column family) соответствует как минимум один HFIle.
- **MemStore**— буфер на запись, специально выделенная область памяти для накопления данных перед записью, поскольку обновлять каждую запись в отсортированном HFile довольно дорого. Как только MemStore наполнится до некоторого критического значения, создается очередной HFile, куда будут записаны новые данные.
- **BlockCache**— кэш на чтение, позволяющий существенно экономить время на часто читаемых данных.
- **Write Ahead Log (WAL)** – специальный файл для логгирования всех операций с данными, чтобы их можно было восстановить в случае сбоя.

Изначально таблица состоит из одного региона, который разбивается на новые по мере роста (после превышения конкретно заданного порогового размера). Когда таблица становится слишком большой для одного сервера, она обслуживается кластером, на каждом узле которого размещается подмножество регионов таблицы. Таким образом, регионирование обеспечивает распределение нагрузки на таблицу. Совокупность отсортированных регионов, доступных по сети, образует общее содержимое распределенной таблицы.

Поскольку данные по одному региону могут храниться в нескольких HFile, для ускорения работы HBase периодически объединяет их, выполняя одну из 2-х операций под названием **compaction:**

- **Minor Compaction**, который запускается автоматически и выполняется в фоновом режиме. Имеет низкий приоритет по сравнению с другими операциями.
- **Major Compaction**, запускаемый вручную или в случае наступления определенных условий (триггеров), например, срабатывание по таймеру. Имеет высокий приоритет и может существенно замедлить работу кластера. Эту операцию рекомендуется выполнять при невысокой нагрузке на кластер. Во время выполнения Major Compaction также происходит физическое удаление данных, ранее помеченных соответствующей меткой **tombstone**.

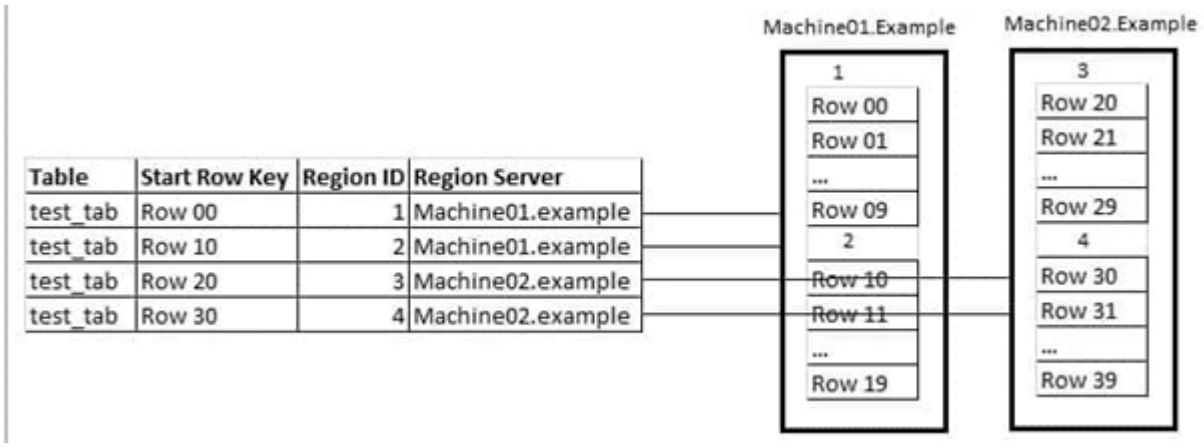


Рисунок 2. Разбиение HBase на регионы

Помимо Compaction-операций, HBase позволяет также работать с данными, считывая и записывая новую информацию. Однако, в отличие от реляционных СУБД, HBase поддерживает только 4 основные действия по обработке данных, которые отличаются от классических SQL-запросов. Операции применяются через экземпляры таблиц:

Операция	Описание	Синтаксис
Добавить новую или обновить	Временной штамп (timestamp) этой	put '<table name>','row1','<colfamily:colname>','<value>'

Операция	Описание	Синтаксис
существующую запись.	записи может быть задан вручную или установлен автоматически как текущее время.	
Получить данные по определенному первичному ключу (RowKey).	Можно указать Column Family, откуда будет считана информация и количество версий, которые требуется прочитать.	get '<table name>','row1'
Выборка всех данных таблицы, удовлетворяющих условиям	В качестве условий можно задать фильтры, Column Family, откуда будет производиться чтение, максимальное количество версий для каждой записи, временной интервал, фильтр по префиксу идентификатора row итд	scan '<table name>' {FILTER => 'name filter'}
Удалить данные из таблицы	Физического удаления при этом не произойдет, оно будет отложено до следующего Major Compaction.	delete '<table name>', '<row>', '<column name>', '<time stamp>'

2.3 Кластерная архитектура

Для распределенной работы с данными в HBase имеются следующие основные архитектурные компоненты:

- **Region Server**, который обслуживает один или несколько регионов. Region Server включает memstore для кэширования часто используемых строк в памяти.

- **Master Server** – главный узел в кластере, управляющий распределением регионов по региональным серверам, включая ведение их реестра, управление запусками регулярных задач и других организационных действий.

  Вычислительная модель MapReduce может использоваться для загрузки большого объема данных, при этом на этапе Reduce выполняется загрузка данных в таблицу – локальная запись данных на соответствующем Region Server

  За координацию работ между сервисами HBase отвечает служба ZooKeeper, предназначенная для управления конфигурациями и синхронизации сервисов. Серверы ZooKeeper и HMaster предоставляют клиентам информацию о топологии кластера. Клиенты подключаются к ним и загружают список регионов, содержащихся в региональных серверах, и диапазонах ключей, размещенных в регионах.

  Высокая доступность данных и отказоустойчивость HBase обеспечиваются благодаря следующем особенностям этой распределенной СУБД класса NoSQL:

- развертывание на нескольких экземплярах HMaster и ZooKeeper;
- распределение данных по многим узлам гарантирует, что сбой одного из них не приведет к потере доступности данных;
- формат HFile, хранящий данные непосредственно в HDFS, можно читать и записывать с помощью многих инструментов стека Big Data (например, Hive, MapReduce, Tez), что позволяет практически в режиме реального времени анализировать данные, не копируя их в другие хранилища.

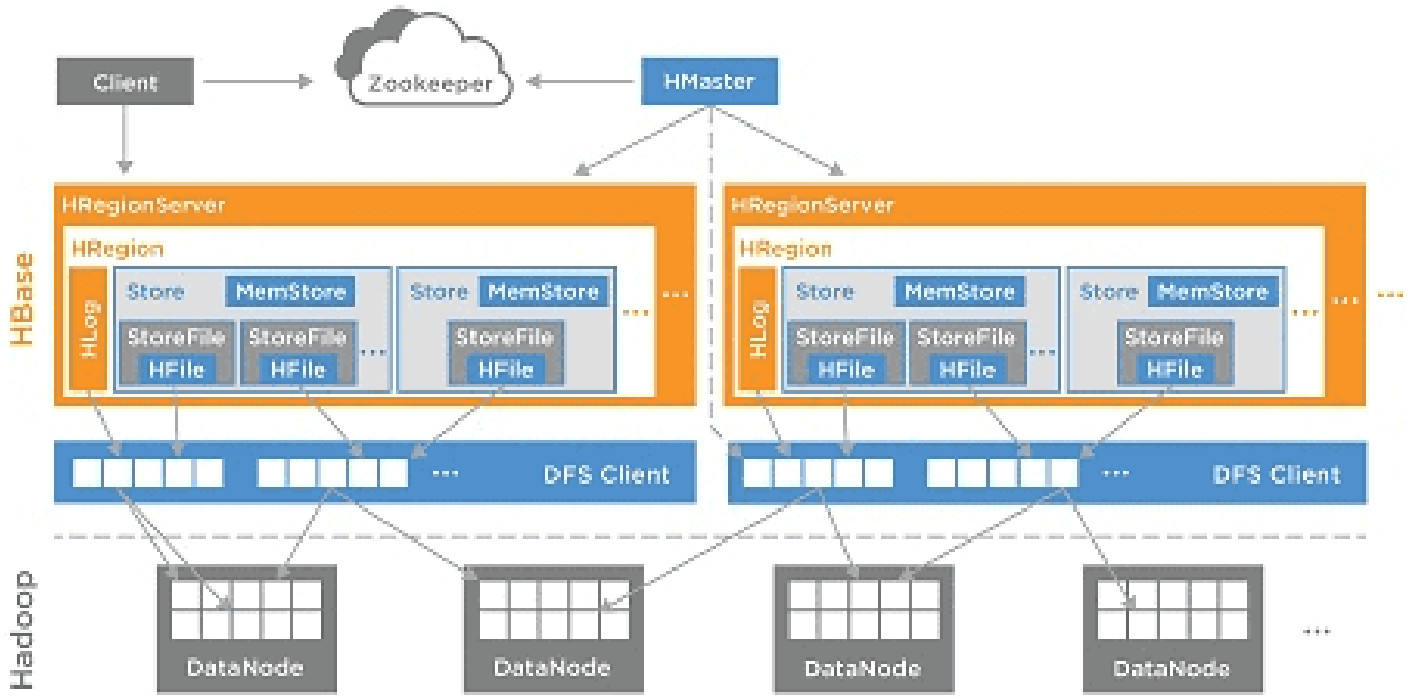


Рисунок 3. Кластерная архитектура HBase

2.4 Hbase Shell

Операция	Описание	Синтаксис
Создать таблицу	Создание таблицы в HBase	create '<table name>','<column family>'

Операция	Описание	Синтаксис
Вывод таблиц	Вывод списка всех таблиц в HBase	list
Выключить таблицу	Чтобы удалить таблицу или изменить ее настройки, сначала необходимо отключить таблицу. После выключения таблицы всё ещё можно увидеть её в списке всех таблиц с помощью команды **list** .	disable '<table name>'
Проверка выключения таблицы	Команда используется для определения отключена ли таблица.	is_disabled 'table name'
Выключить все таблицы	Эта команда используется для отключения всех таблиц, соответствующих данному регулярному выражению.	disable_all 'r.*'
Включить таблицу	Включить таблицу	enable '<table name>'
Проверка включения таблицы	Команда используется для определения	is_enabled '<table name>'

Операция	Описание	Синтаксис
	включена ли таблица.	
Информация о таблице	Эта команда возвращает описание таблицы	describe 'table name'
Изменить свойств, атрибутов таблицы	Команда, используемая для внесения изменений в существующую таблицу. С помощью этой команды вы можете изменить свойства или атрибуты таблицы, такие как установить версию, установить и удалить операторы области видимости таблицы, а также удалить семейство столбцов из существующей таблицы.	alter <tablename>, NAME=>'<column familyname>', VERSIONS=><new version n
Проверка наличия таблицы в БД	Проверить существование таблицы, используя команду **exists**	exists 'table name'
Удалить таблицу	Прежде чем удалить таблицу необходимо выключить её (**disable**)	drop 'table name'
Удалить таблицы, соответствую	Эта команда используется для	drop_all ‘t.*’

Операция	Описание	Синтаксис
щие регулярному выражению	удаления таблиц, соответствующих «регулярному выражению», данному в команде. Примечание: прежде чем удалить таблицу необходимо выключить её (disable)	

2.5 Java API

Операции	Описание	Синтаксис

Создать таблицу	Создать таблицу в HBase с помощью **CreateTable()** метод **HBaseAdmin** класса. Этот класс принадлежит пакету **org.apache.hadoop.hbase.client** .	// Instantiating configuration class
		Configuration con = HBaseConfiguration.create();
		
		// Instantiating HbaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(con);
		
		// Instantiating table descriptor class
		HTableDescriptor tableDescriptor = new
		HTableDescriptor(TableName.valueOf("table name"));
		
		// Adding column families to table descriptor
		tableDescriptor.addFamily(new HColumnDescriptor("nam
		
		// Execute the table through admin
		admin.createTable(tableDescriptor);

Вывод списка таблиц	Существует метод **listTables ()** в классе **HBaseAdmin,** который получает список всех таблиц в HBase. Этот метод возвращает массив объектов **HTableDescriptor** .	// Instantiating a configuration class Configuration conf = HBaseConfiguration.create();

Операции	Описание	Синтаксис

		// Instantiating HBaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// Getting all the list of tables using HBaseAdmin obje
		HTableDescriptor[] tableDescriptor = admin.listTables
		
		// printing all the table names.
		for (int i=0; i<tableDescriptor.length;i++){
		System.out.println(tableDescriptor[i].getNameAsStr
		}

Выключить таблицу	Чтобы проверить, отключена ли таблица, **используется** метод **isTableDisabled(),** а для отключения таблицы - **метод disableTable()** . Эти методы принадлежат классу **HBaseAdmin** .	// Instantiating configuration class
		Configuration conf = HBaseConfiguration.create();
		
		// Instantiating HBaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// Verifying weather the table is disabled
		Boolean bool = admin.isTableDisabled("table name")
		
		// Disabling the table using HBaseAdmin object
		if(!bool){
		admin.disableTable("table name");

Проверка включения таблицы	Чтобы проверить, включена ли таблица, **используется** метод **isTableEnabled()** ; и для включения таблицы **используется** метод **enableTable()** . Эти методы принадлежат классу **HBaseAdmin** .	// Instantiating configuration class
		Configuration conf = HBaseConfiguration.create();
		
		// Instantiating HBaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// Verifying whether the table is disabled
		Boolean bool = admin.isTableEnabled("table name");
		
		// Enabling the table using HBaseAdmin object

Операции	Описание	Синтаксис
		if(!bool){ admin.enableTable("table name");

Изменить данные таблицы	Можно добавить **Column Family** таблицы с помощью метода **addColumn()** из **HBAseAdmin** класса. Метод **addColumn()** требует имя таблицы и объект класса **HColumnDescriptor** . Поэтому **создайте** экземпляр класса **HColumnDescriptor** . В свою очередь, конструктор **HColumnDescriptor** требует добавления имени семейства столбцов.	// Instantiating configuration class.
		Configuration conf = HBaseConfiguration.create();
		
		// Instantiating HBaseAdmin class.
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// Instantiating columnDescriptor class
		HColumnDescriptor columnDescriptor = new HColum
		
		// Adding column family
		admin.addColumn("table name", columnDescriptor);

Проверка наличия таблицы в БД	Можно проверить существование таблицы в HBase с использованием **tableExists()** метода в **HBaseAdmin** класса.	// Instantiating configuration class
		Configuration conf = HBaseConfiguration.create();
		
		// Instantiating HBaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// Verifying the existance of the table
		boolean bool = admin.tableExists("table name");

Удалить таблицу	Можно удалить таблицу с помощью **метода deleteTable()** в классе **HBaseAdmin** .	// Instantiating configuration class
		Configuration conf = HBaseConfiguration.create();
		
		// Instantiating HBaseAdmin class
		HBaseAdmin admin = new HBaseAdmin(conf);
		
		// disabling table named emp
		admin.disableTable("table name");
		
		// Deleting emp

Операции	Описание	Синтаксис
		admin.deleteTable("table name");

3 Компонент HDFS

HDFS содержит очень большой объем данных и обеспечивает более легкий доступ. Чтобы хранить такие большие данные, файлы хранятся на нескольких машинах. Эти файлы хранятся в резервном режиме, чтобы спасти систему от возможных потерь данных в случае сбоя. HDFS также делает приложения доступными для параллельной обработки.

3.1 Особенности HDFS

- Подходит для распределенного хранения и обработки данных.
- Hadoop предоставляет командный интерфейс для взаимодействия с HDFS.
- Встроенные серверы namenode и datanode помогают пользователям легко проверять состояние кластера.
- Потоковый доступ к данным файловой системы.
- HDFS предоставляет права доступа к файлам и аутентификацию.

3.2 Архитектура HDFS

HDFS подобна другим традиционным файловым системам: файлы хранятся в виде блоков, существует маппинг между блоками и именами файлов, поддерживается древовидная структура, поддерживается модель доступа к файлам основанная на правах и т. п.

Простой Hadoop-кластер состоит из нод трех типов: *NameNode, Secondary NameNode, Datanode*.

Namenode — мозг системы. Как правило, одна нода на кластер (больше в случае Namenode Federation, но мы этот случай оставляем за бортом). Хранит в себе все метаданные системы — непосредственно маппинг между файлами и блоками.

Secondary NameNode — 1 нода на кластер. Secondary NameNode не является репликой NameNode. Состояние файловой системы хранится непосредственно в файле fsimage и в лог файле edits, содержащим последние изменения файловой системы (похоже на лог транзакций в мире РСУБД). Secondary Namenode можно рассматривать как помощник Namenode, который берет на себя часть рабочей нагрузки Namenode.

Namenode сохраняет информацию о метаданных файловой системы в двух файлах fsimage и edit logs.

Из этих двух файлов **fsimage** консультируется, чтобы получить информацию метаданных при запуске Namenode. После запуска Namenode все изменения пространства имен записываются в **edit logs.**

Secondary NameNode может взять на себя ответственность за слияние fsimage и файлов журнала редактирования периодически, таким образом, размер журнала редактирования сохраняется в пределах лимита, и Namenode имеет объединенный файл fsimage.

В реальном кластере NameNode и Secondary NameNode — отдельные сервера, требовательные к памяти и к жесткому диску.

DataNode — Таких нод в кластере очень много. Они хранят непосредственно блоки файлов. Нода регулярно отправляет NameNode свой статус (показывает, что еще жива) и ежечасно — репорт, информацию обо всех хранимых на этой ноде блоках. Это необходимо для поддержания нужного уровня репликации.

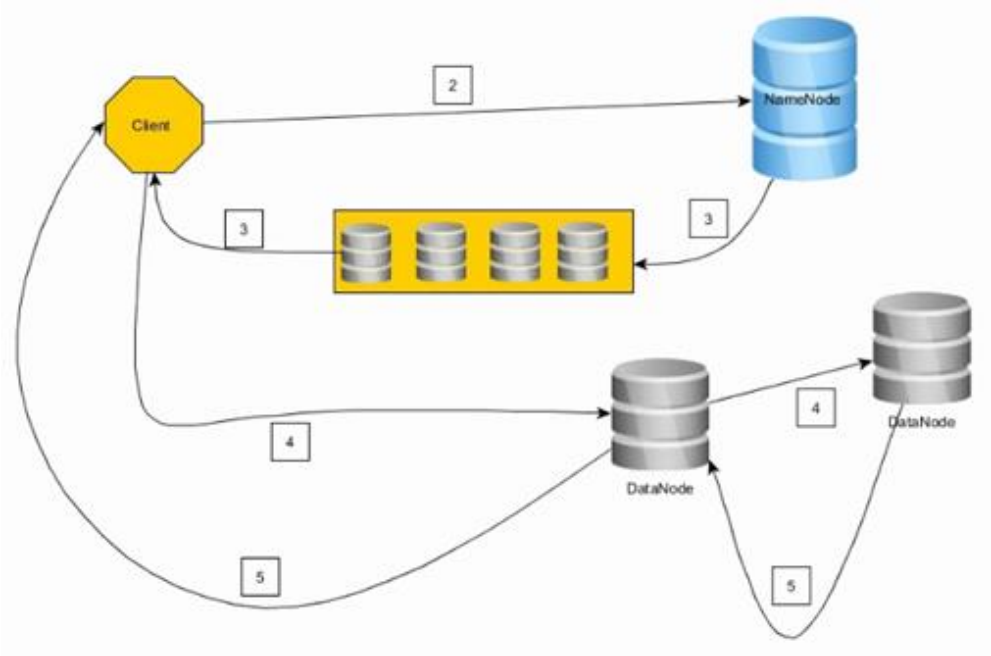


Рисунок 1. Архитектура HDFS

1. Клиент разрезает файл на цепочки блокового размера.
2. Клиент соединяется с NameNode и запрашивает операцию записи, присылая количество блоков и требуемый уровень репликации
3. NameNode отвечает цепочкой из DataNode.
4. Клиент соединяется с первой нодой из цепочки (если не получилось с первой, со второй и т. д. не получилось совсем — откат). Клиент делает запись первого блока на первую ноду, первая нода — на вторую и т. д.
5. По завершении записи в обратном порядке (4 -> 3, 3 -> 2, 2 -> 1, 1 -> клиенту) присылаются сообщения об успешной записи.
6. Как только клиент получит подтверждение успешной записи блока, он оповещает NameNode о записи блока, затем получает цепочку DataNode для записи второго блока и т.д.

Клиент продолжает записывать блоки, если сумеет записать успешно блок хотя бы на одну ноду, т. е. репликация будет работать по хорошо известному принципу «eventual», в дальнейшем NameNode обязуется компенсировать и таки достичь желаемого уровня репликации.

3.3 Цели HDFS

Обнаружение и восстановление неисправностей - поскольку HDFS включает в себя большое количество commodity оборудования, отказ компонентов происходит часто. Поэтому HDFS должны иметь механизмы быстрого и автоматического обнаружения и восстановления неисправностей.

Огромные наборы данных - HDFS должны иметь сотни узлов на кластер для управления приложениями, имеющими огромные наборы данных.

Аппаратное обеспечение при данных - запрошенная задача может быть выполнена эффективно, когда вычисление происходит рядом с данными. Особенно там, где задействованы огромные наборы данных, это уменьшает сетевой трафик и увеличивает пропускную способность.

3.4 Основные операции HDFS

Операция	Описание	Синтаксис
Вывод списка содержимого каталога HDFS	Список содержимого каталога, указанного по пути, с указанием имен, разрешений, владельца, размера и даты изменения для каждой записи. Где **<path>** путь к каталогу	hdfs dfs -ls <path>
Вывод рекурсивно всех записей во всех подкаталогах HDFS	Ведет себя как **-ls**, но рекурсивно отображает записи во всех подкаталогах. **<path>** путь к каталогу, где необходимо создать директорию	hdfs dfs -lsr <path>
Создать директорию HDFS	Создает каталог с именем path в HDFS. **<path>** путь к каталогу, где необходимо создать директорию	hdfs dfs -mkdir <path>
Копирование файла/каталога из локальной системы в HDFS	**<local path>** - путь к файлу в локальной системе **<hdfs path>** - путь к директории в HDFS	hdfs dfs -put <local path> <hdfs path>
Скопировать файлы в локальную файловую систему из HDFS	Файлы, которые не проходят проверку CRC, могут быть скопированы с помощью параметра **-ignore crc**. Файлы и CRC могут быть скопированы с помощью опции **-crc**. **<local path>** - путь к файлу в локальной системе **<hdfs path>** - путь к директории в HDFS	hdfs dfs -get [-ignore] [-crc] <hdfs path> <local path>

Операция	Описание	Синтаксис
Использование диска (в байтах) HDFS	Показывает использование диска в байтах для всех файлов, соответствующих пути; имена файлов сообщаются с полным префиксом протокола HDFS. **<path>** путь к каталогу	hdfs dfs -du <path>
Перемещение файла/каталога в переделах HDFS	Перемещает файл или каталог, указанный **<src>**, в **<dest>**, в пределах HDFS.	hdfs dfs -mv <src> <dest>
Копировать файлы/каталоги в переделах HDFS	Копирует файл или каталог, идентифицированный **<src>**, в **<dest>** в пределах HDFS.	hdfs dfs -cp <src> <dest>
Удалить файл/пустой каталог HDFS	Удаляет файл или пустой каталог, определенный путем. **<path> -** путь к файлу/каталогу	hdfs dfs -rm <path>
Удалить файл/каталог рекурсивно HDFS	Удаляет файл или каталог, идентифицированный по пути. Рекурсивно удаляет все дочерние записи (например, файлы или подкаталоги пути).	hdfs dfs -rmr <path>
Отобразить содержимое файла HDFS	Отображает содержимое файла filename на stdout. **<filename> -** имя файла	hdfs dfs -cat <path>/<filename>
Создать файл HDFS	Создает файл по пути, содержащий текущее время в виде метки времени. Сбой происходит, если файл уже существует в пути, если только он не имеет размер 0.	hdfs dfs -touchz <path>

Операция	Описание	Синтаксис
	<path> - путь к каталогу	
Обновление времени доступа и изменения файла в HDFS	**<path> -** путь к каталогу/фалу в hdfs **<name>** - имя файла/каталога **<timestamp> -** время и дата, на которую необходимо заменить. Формат: YYMMDD:HHMMSS	hdfs dfs -touch -t <timestamp> <path>/<name>
Вывод информации о каталоге/файле HDFS	Выводит информацию о пути. Формат-это строка, которая принимает размер файла в блоках (%b), имя файла (%n), размер блока (%o), репликацию (%r) и дату модификации (%y, %Y). **<path> -** путь к каталогу/файлу	15hdfs dfs -stat [format] <path>
Вывод проверки каталогов/файлов HDFS	**<path> -** путь к каталогу/файлу [parametr]: -d Check whether the path given by the user is a directory or not, return 0 if it is a directory. -e Check whether the path given by the user exists or not, return 0 if the path exists. -f Check whether the path given by the user is a file or not, return 0 if it is a file. -s Check if the path is not empty, return 0 if a path is not empty. -r return 0 if the path exists and read permission is granted -w return 0 if the path exists and write permission is granted -z Checks whether the file	hdfs dfs -test [parametr] <path>

Операция	Описание	Синтаксис
	size is 0 byte or not, return 0 if the file is of 0 bytes.	
Вывод текста, содержащегося в файле HDFS	**<path> -** путь к файлу	hdfs dfs -text <path>
Добавление содержимого одного или нескольких локальных файлов, указанных в localsrc, в предоставленный конечный файл в HDFS	<filenames> - имя файлов на локальной системе <hdfsPath> - путь до директории в hdfs	hdfs dfs -appendToFile <filenames> <hdfsPath>
Подсчитывание количества файлов, каталогов и байтов по путям, соответствующими указанному шаблону файлов в HDFS	**<path> -** путь к каталогу/файлу [parametr]: -q – shows quotas(quota is the hard limit on the number of names and amount of space used for individual directories) -u – it limits output to show quotas and usage only -h – shows sizes in a human-readable format -v – shows header line	hdfs dfs -count [parametr] <path>
Поиск файла/каталога в HDFS	**<path> -** путь по поиску **<name>** - имя файла/каталога	hdfs dfs -find <path> -name <name> -print
Слияние файлов/каталогов в HDFS	**<hdfsPath>**- путь к каталогу/фалу в hdfs **<localPath>** - путь на локальной системе	hdfs dfs -getmerge <hdfsPath> <localPath>
Отображение последних 1 КБ данных файла на консоли в HDFS	**<path> -** путь к каталогу/фалу в hdfs **<name>** - имя файла/каталога	hdfs dfs -tail <path>/<name>

4 Компонент Hive

Hive - это инструмент инфраструктуры хранилища данных для обработки структурированных данных в Hadoop.
Он находится на вершине Hadoop для обобщения больших данных и облегчает запросы и анализ.

4.1 Особенности Hive

- Хранит схему в базе данных и обрабатывает данные HDFS.
- Предназначен для OLAP.
- Предоставляет язык типов SQL для запросов, называемых HiveQL или HQL.
- Знаком, быстр, масштабируем и расширяем.

4.2 Архитектура Hive

На следующей диаграмме компонентов показана архитектура Hive:

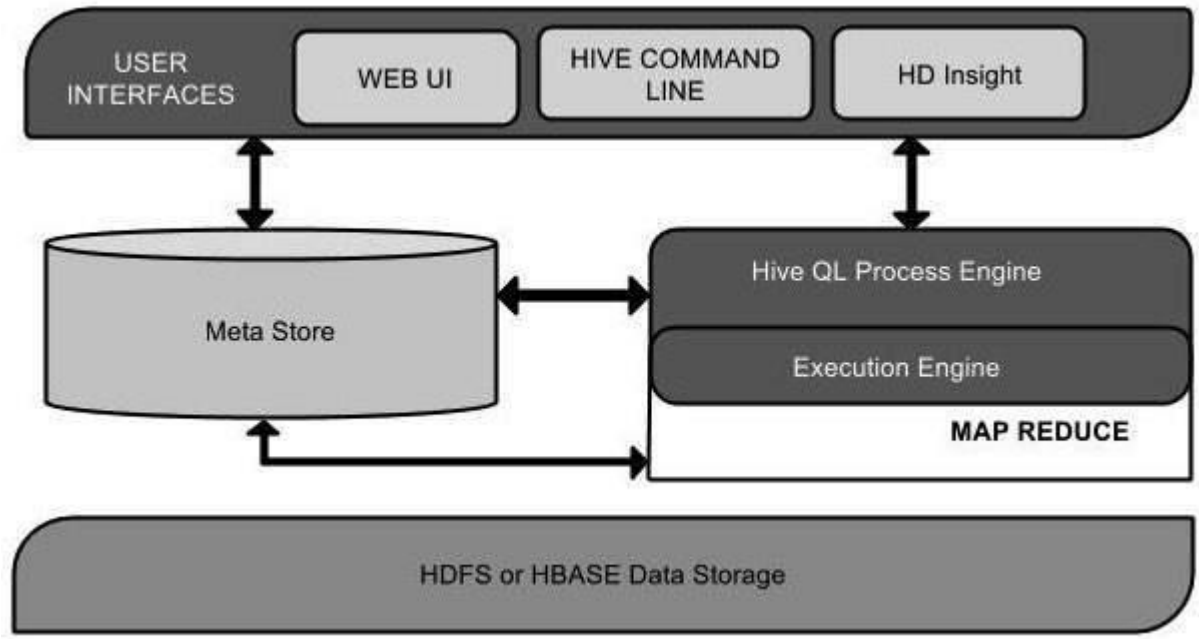


Рисунок 1. Архитектура Hive

В таблице ниже описаны модули компонентов на диаграмме:

Модуль	Операция
User Interface	Hive - это программное обеспечение инфраструктуры хранилища данных, которое может создавать взаимодействие между пользователем и HDFS. Пользовательские интерфейсы, которые поддерживает Hive, - это *Hive Web UI*, *Hive command line* и *Hive HD Insigh*t (в Windows server).
Meta Store	Для работы hive также необходимо хранилище метаданных. Дело в том что SQL предполагает работу с такими объектами как база данных, таблица, колонки, строчки, ячейки и тд. Поскольку сами данные, которые использует hive хранятся просто в виде файлов на hdfs — необходимо где-то хранить соответствие между объектами hive и реальными файлами. В качестве metastorage используется обычная реляционная СУБД, например PostgreSQL

Модуль	Операция
HiveQL Process Engine	HiveQL похож на SQL для запроса информации о схеме в Metastore. Это одна из замен традиционного подхода к MapReduce в программах. Вместо написания программы MapReduce на Java мы можем написать запрос для задания MapReduce и обработать его.
Execution Engine	Объединяющей частью Hiveql process Engine и MapReduce является Hive Execution Engine. Механизм выполнения обрабатывает запрос и генерирует результаты так же, как и результаты MapReduce
HDFS или HBASE	Распределенная файловая система Hadoop или HBASE - это методы хранения данных для хранения данных в файловой системе.

4.3 Работа Hive

На следующей диаграмме показан рабочий процесс между Hive и Hadoop.

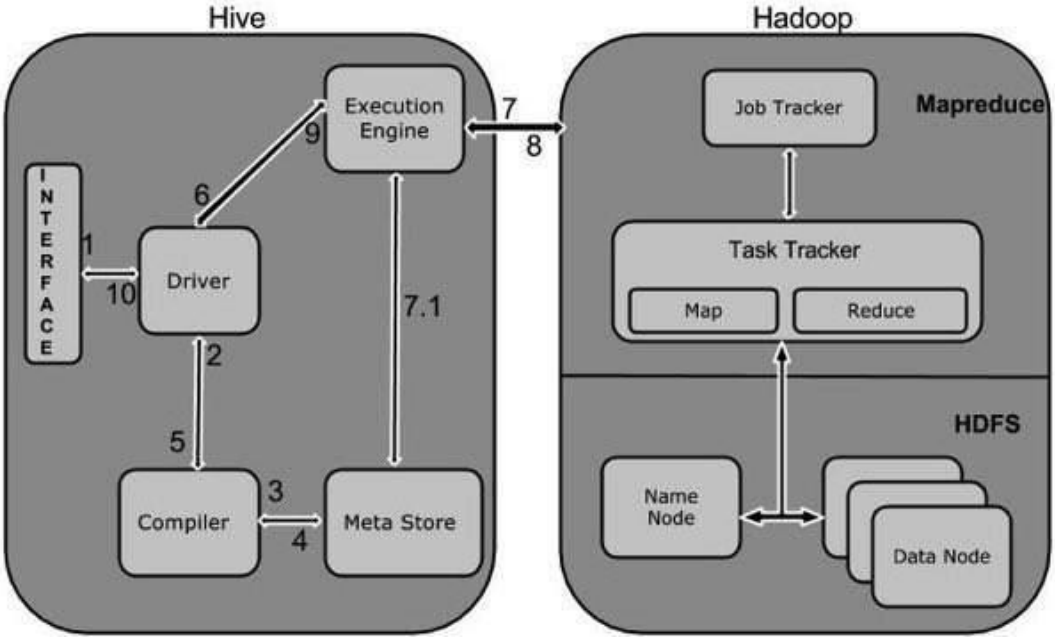


Рисунок 2. Работа Hive

Следующая таблица описывает, как Hive взаимодействует с Hadoop framework:

Шаг	Операция
1	**Выполнение запроса** Интерфейс Hive, такой как командная строка или веб-интерфейс, отправляет запрос драйверу (любой драйвер базы данных, такой как JDBC, ODBC и т. д.) во исполнение.
2	**Получить план** Драйвер пользуется помощью компилятора запросов, который анализирует запрос, чтобы проверить синтаксис и план запроса или требование запроса.

Шаг	Операция
3	**Получить метаданные** Компилятор отправляет запрос метаданных в Metastore (любую базу данных).
4	**Отправка метаданных** Metastore отправляет метаданные в качестве ответа компилятору.
5	**Отправить план** Компилятор проверяет требование и повторно отправляет план драйверу. До этого момента синтаксический анализ и компиляция запроса завершены.
6	**Выполнить план** Драйвер отправляет план выполнения в механизм выполнения.
7	**Выполнить задание** Внутренне задание процесса выполнения является заданием MapReduce. Механизм выполнения отправляет задание в JobTracker, который находится в узле Name, и назначает это задание TaskTracker, который находится в узле Data. Здесь запрос выполняет задание MapReduce.
7.1	**Метаданные Ops** Между тем в процессе выполнения механизм выполнения может выполнять операции метаданных с Metastore.
8	**Получение результата** Механизм выполнения получает результаты от узлов данных.
9	**Отправить результаты** Механизм выполнения отправляет эти результирующие значения драйверу.
10	**Отправить результаты** Драйвер отправляет результаты в интерфейсы Hive.

4.3.1 Форматы хранения данных

4.3.1.1 Текстовый файл

```
CREATE TABLE country (
    name STRING,
    states ARRAY<STRING>,
    cities_and_size MAP<STRING, INT>,
    parties STRUCT<name STRING, votes FLOAT, members INT>
)
ROW FORMAT DELIMITED
```

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

LINES TERMINATED BY '\n'

STORED AS TextFile;

Формат текстового файла используется по умолчанию для Hive. Он использует четыре разделителя для разделения выходного или входного файла на строки, столбцы и сложные типы данных. Это может быть явно выражено с помощью выражения

STORED AS TextFile.

Определение разделителей следует за определением схемы таблицы и должно иметь префикс:

ROW FORMAT DELIMITED

Выражение

LINES TERMINATED BY '\n'

описывает разделение строк по умолчанию на символ новой строки. К сожалению, этот разделитель не может быть установлен на что-либо еще, кроме новой строки. Чаще всего разделитель полей изменяется на запятую

FIELDS TERMINATED BY ','

или tab

FIELDS TERMINATED BY '\ t'

По умолчанию 0x01 или ^ A, также известный как Ctrl-A. По умолчанию для элементов коллекции - 0x02 и 0x03 для ключей карты. Важно понимать, что Hive не поддерживает экранированные символы. Например, лист, сохраненный в виде файла CSV из Excel, может быть отформатирован в двойных кавычках, чтобы экранировать запятые в полях.

4.3.1.2 Sequence файл

CREATE TABLE country (name STRING, population INT)

STORED AS SequenceFile;

Формат SequenceFile является значением по умолчанию в Hadoop и хранит данные в двоичных парах ключ-значение. Он оптимизирован для быстрого ввода-вывода и размера блока. Большинство инструментов в экосистеме Hadoop могут читать формат SequenceFile.

Hive сохранил свои данные строки в части значения пар. Этот формат более эффективен, чем простой текст, и его можно сжимать с доступными форматами в Hadoop либо по значению (строка), либо по блоку (блок данных). Последнее, как правило, более эффективно, так как сжатие с большей вероятностью находит приводимые шаблоны в большом блоке данных, чем в строке.

Недостаток этого формата заключается в том, что Hive должен читать и анализировать каждую строку для каждого запроса (при условии отсутствия разбиения), чтобы выполнить

условия запроса для него. Это может привести к ненужным операциям чтения, если только несколько строк или несколько столбцов имеют отношение к запросу.

4.3.1.3 RC файл

CREATE TABLE country (name STRING, population INT)

STORED AS RCFile;

Формат файла столбцов строк (RCFile) объединяет принципы хранения, ориентированные на строки и столбцы. В каждом файле хранится одна или несколько групп строк, которые представляют собой количество строк. Сами группы строк хранятся в столбчатой структуре.

Например, файл может хранить строки 1–1 000 в первой группе и строки 1 001–2 000 в следующей и обе группы в одном RCFile. Сама группа строк хранит все столбцы вместе. Первая группа затем сохранит первый столбец строки с 1 по 1000, а затем следующий столбец и так далее.

Преимущество группировки столбцов заключается в более эффективном сжатии, поскольку сходные данные находятся рядом друг с другом. Что более важно, условия запроса могут быть переданы для чтения только соответствующих частей таблицы. Это особенно полезно для широких таблиц и запросов, которые применяются только к нескольким столбцам. В этих случаях Hive может пропускать большие части данных для экономии времени ввода-вывода и вычислительного времени.

4.3.1.4 ORC файл

Формат файла ORC стал общедоступным с Hive версии 0.11, хотя некоторые его функции еще не полностью реализованы. ORC следует рассматривать как альтернативу RCFile. ORC выходит за рамки RCFile и представляет оптимизированное для столбцов хранилище (например, кодирование переменной длины для целых чисел), большие размеры блоков (лучший дисковый ввод-вывод и меньшее количество файлов с меньшей загрузкой namenode), базовую статистику по столбцам в файле и простые файловые индексы для пропуска целых групп строк если они не соответствуют запросу.

4.3.2 Основные сущности в Hive:

- База данных
- Таблица
- Партиция (partition)
- Бакет (bucket)

4.3.2.1 База данных

База данных представляет аналог базы данных в реляционных СУБД. База данных представляет собой пространство имён, содержащее таблицы.

Database и Schema в данном контексте это одно и тоже. Необязательная добавка IF NOT EXISTS как не сложно догадаться создает базу данных только в том случае если она еще не существует.

4.3.2.2 Таблица

Таблица в hive представляет из себя аналог таблицы в классической реляционной БД. Основное отличие — что данные hive’овских таблиц хранятся прост в виде обычных файлов на hdfs. Это могут быть обычные текстовые csv-файлы, бинарные sequence-файлы, более сложные колоночные parquet-файлы и другие форматы. Но в любом случае данные, над которыми настроена hive-таблица очень легко прочитать и не из hive.

Таблицы в hive бывают двух видов:

Классическая таблица, данные в которую добавляются при помощи hive. ,

4.3.2.3 Партиция (partition)

Так как hive представляет из себя движок для трансляции SQL-запросов в mapreduce-задачи, то обычно даже простейшие запросы к таблице приводят к полному сканированию данных в этой таблицы. Для того чтобы избежать полного сканирования данных по некоторым из колонок таблицы можно произвести партиционирование этой таблицы. Это означает, что данные относящиеся к разным значениям будут физически храниться в разных папках на HDFS.

Партиционирование очень удобно, например, для разделения логов по датам, так как правило любые запросы за статистикой содержат ограничение по датам. Это позволяет существенно сократить время запроса.

4.3.2.4 Основные операции в Hive

Консоль beeline	**JDBC**
Создания базы данных. База данных в Hive - это пространство имен или набор таблиц.	

CREATE DATABASE [IF NOT EXISTS] db_name;	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("CREATE DATABASE db_name");
	con.close();
	<jdbc url for hive> - Данную ссылку можно найти в Ambari вашего кластера на вкладке Hive.

Создание таблицы.	

CREATE [TEMPORARY][EXTERNAL] TABLE [IF NOT EXISTS][db_name.] table_name [(col_name data_type [COMMENT col_comment],...)][COMMENT table_comment] [ROW FORMAT row_format][STORED AS file_format];	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("CREATE TABLE IF NOT EXISTS" + " table_name (int_field int, string_field String");
	con.close();

Изменение таблицы	
Оператор ALTER TABLE принимает любой из следующих синтаксисов,	// Зарегистрировать драйвер и создать экземпляр драйвера

основанных на том, какие атрибуты мы хотим изменить в таблице:
ALTER TABLE table_name RENAME TO new_name;
ALTER TABLE table_name ADD COLUMNS (col_spec[, col_spec ...]);
ALTER TABLE table_name DROP [COLUMN]column_name;
ALTER TABLE table_name CHANGE column_name new_name new_type;
ALTER TABLE table_name REPLACE COLUMNS (col_spec[, col_spec ...]);
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("ALTER TABLE table_name RENAME TO table_new_name;");
	con.close();
	Со всеми остальными атрибутам через jdbc аналогично.

Добавление данных в таблицу	

INSERT INTO TABLE table_name VALUES [fields ...];	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("INSERT INTO TABLE table_name VALUES (1, "Name 1"), (2, "Name 2");");
	con.close();

Удаление БД со всеми таблицами	

DROP DATABASE [IF EXISTS] testdb [RESTRICT|CASCADE];	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("DROP DATABASE testdb");
	con.close();

Удаление таблицы Когда вы удаляете таблицу из Метастора Hive, она удаляет данные таблицы / столбца и их метаданные. Это может быть обычная таблица (хранящаяся в Metastore) или внешняя таблица (хранящаяся в локальной файловой системе); Hive обрабатывает обе таблицы одинаково, независимо от их типов, но при удалении внешних таблиц файлы не удаляются, их нужно удалять отдельными командами, например, командами HDFS	
DROP TABLE [IF EXISTS] table_name;	// Зарегистрировать драйвер и создать экземпляр драйвера

	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("DROP TABLE IF EXISTS table_name");
	con.close();

Выборка значений	

SELECT [ALL | DISTINCT] select_expr, select_expr, ... FROM table_reference [WHERE where_condition] [GROUP BY col_list] [HAVING having_condition] [CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]] [LIMIT number];	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	// Инструкция выполнения
	Resultset res = stmt.executeQuery("SELECT * FROM table_name");
	System.out.println("Result:");
	while (res.next()) {
	System.out.println(res.getString(1)+""+res.getInt(2));
	}
	con.close();

Загрузка данных При вставке данных в Hive лучше использовать LOAD DATA для хранения массовых записей. Существует два способа загрузки данных: один-из локальной файловой системы, а второй-из файловой системы Hadoop.	

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE table_name
[PARTITION (partcol1=val1, partcol2=val2 ...)]
	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt'" + "OVERWRITE INTO TABLE table_name;");
	con.close();

Партиции Hive организует таблицы в партиции. Это способ разделения таблицы на связанные части на основе значений секционированных столбцов, таких как дата, город и отдел. Используя партицию, легко запросить часть данных. Таблицы или партиции подразделяются на buckets, чтобы обеспечить дополнительную структуру данных, которые могут быть использованы для более эффективного запроса. Бакетинг работает на основе значения хэш-функции некоторого столбца таблицы.	

Добавление раздела
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec [LOCATION 'location1'] partition_spec [LOCATION 'location2']...;
partition_spec:: (p_column = p_col_value, p_column = p_col_value, ...);
Переименование раздела
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_spec;
Удаление раздела
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION partition_spec,...;
	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	stmt.executeQuery("ALTER TABLE table_name PARTITION partition_spec" + "RENAME TO PARTITION partition_spec;");
	con.close();

Представление	

Создать представление
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...)]
[COMMENT table_comment] AS SELECT ...
Удалить представление
DROP VIEW view_name
	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);
	// Соединение
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	Statement stmt = con.createStatement();
	Resultset res = stmt.executeQuery("CREATE VIEW temp_view AS"
	+"SELECT * FROM table_name"
	+"WHERE size>200;");
	System.out.println(res);
	con.close();

Индекс (указатель на определенный столбец таблицы)	

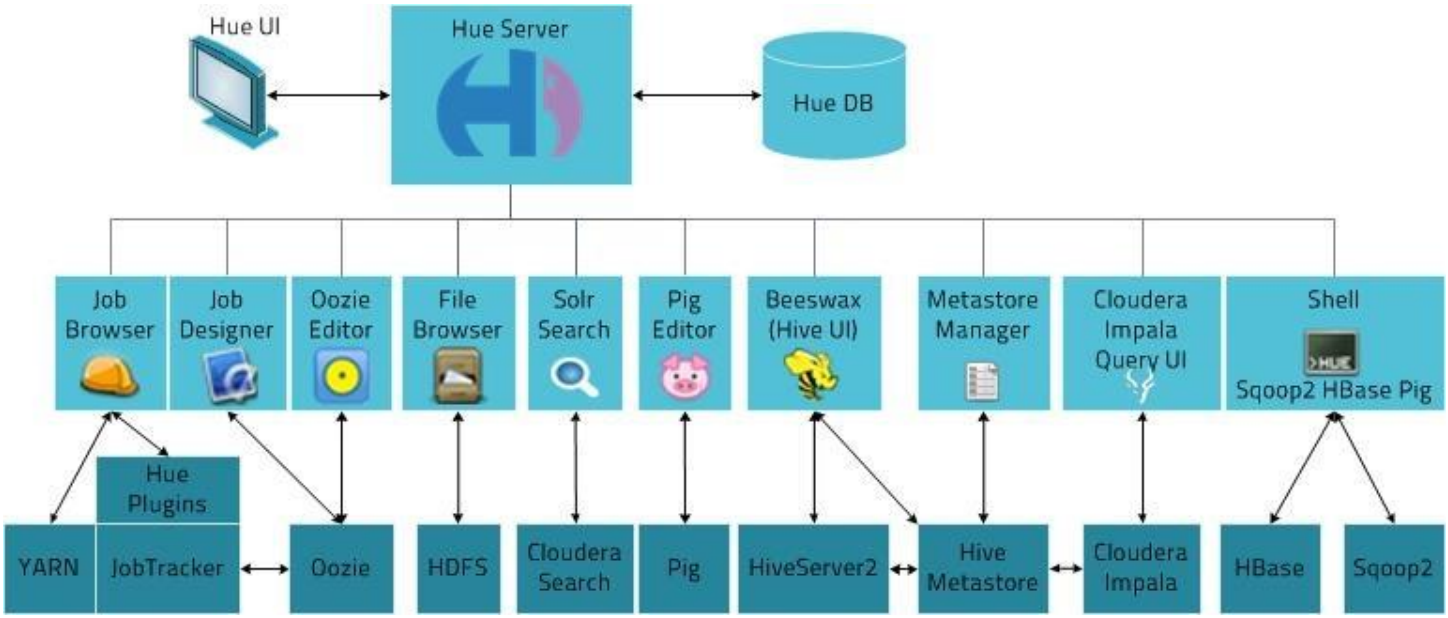
Создать индексы
CREATE INDEX index_name ON TABLE base_table_name
	// Зарегистрировать драйвер и создать экземпляр драйвера
	Class.forName(driverName);

(col_name, ...) AS 'index.handler.class.name' [WITH DEFERRED REBUILD]
 	
[IDXPROPERTIES (property_name=property_value, ...)] [IN TABLE index_table_name] [PARTITIONED BY (col_name, ...)] [[ROW FORMAT ...] STORED AS ... | STORED BY ...] [LOCATION hdfs_path] [TBLPROPERTIES (...)]
 	
Удалить индексы
DROP INDEX <index_name> ON <table_name>
	// Соединение
	
	Connection con = DriverManager.getConnection("<jdbc url for hive>", "", "");
	
	Statement stmt = con.createStatement();
	
	stmt.executeQuery("CREATE INDEX index_name ON TABLE table_name(name)"
	+"AS 'org.apache.hadoop.hive.ql.index"
	+".compact.CompactIndexHandler';");
	
	con.close();

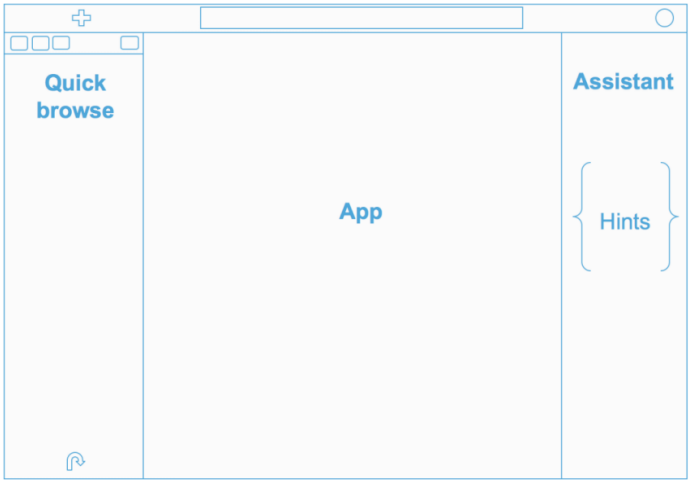
5 Компонент HUE

5.1 Что такое HUE?

Hadoop User Experience (HUE) - это веб-приложение с открытым исходным кодом, используемое для графического использования основных приложений Hadoop Ecosystem. Hue предоставляет интерфейсы для взаимодействия с такими компонентами, как: HDFS, MapReduce, Hive и Spark. Работать с Hue можно непосредственно из браузера, что повышает производительность и удобство работы с платформой Hadoop. Другими словами, HUE, обеспечивает консолидированное «представление поверх Hadoop», значительно облегчая доступ к кластеру, его компонентам и самим данным, избавляя от необходимости использовать командную строку.



5.2 Hue interface concept



5.3 Hue Server

Hue Server - веб-приложение, созданное на основе веб-фреймворка Python Django. Работает "из коробки" на веб сервере CherryPy, но так же может использовать любой другой, поддерживающий стандарт WSGI.

Django, работающий на сервере / веб-сервере WSGI, управляет отправкой URL-адресов, выполняет логический код приложения и объединяет представления из их

шаблонов. Django использует базу данных для управления данными сеанса, и приложения Hue могут использовать ее также для своих «моделей». (Например, приложение JobDesigner сохраняет проекты заданий в базе данных.).

5.4 Hue DB

Hue поддерживает возможность использовать любую из следующих СУБД: SQLITE3, POSTGRESQL. Целевой вариант использования - POSTGRESQL.

5.5 Основной функционал HUE

- Доступ к HDFS системе
- Job Designer
- Hadoop Shell Access
- Spark Notebooks
- Импорт данных из Hadoop
- SQL редактор для Hive, PostgreSQL
- Oozie workflows and Editor
- Dashboard
- User Administration

5.6 Интеграции HUE с компонентами Hadoop:

Компонент	Интерфейс взаимодействия	Описание
HDFS	WebHDFS	Hue использует WebHDFS Rest API для взаимодействия с HDFS. При указании NameNode будет использоваться именно она. Для работы в режиме HA - вместо NameNode надо указывать HttpFs. Для этого надо убедиться, что в HDFS включен dfs.webhdfs.enable в hdfs-site.xml
Hive	HiveServer2 Thrift server	Для работы с Hive - используется HiveServer2 Thrift server
Spark2	Livy REST API	Livy - это служба, которая обеспечивает простое взаимодействие со Spark2 через интерфейс REST.
Yarn	Resource Manager REST API History server API	Как и для HDFS - Hue поддерживает один или два сервиса resourcemanager (два для HA). Нужно указать адреса сервисов YARN в двух секциях [[[default]] и [[[ha]]]
Oozie	Oozie REST	Для взаимодействия с Oozie требуется сам Oozie
HBASE	HBase Thrift	Приложение Hue обменивается данными через прокси-сервер, называемый HBase Thrift Server, который затем пересылает запросы в HBase.

6 Компонент MapReduce

6.1 Почему MapReduce?

Традиционные корпоративные системы обычно имеют централизованный сервер для хранения и обработки данных. На следующем рисунке показан схематический вид традиционной корпоративной системы. Традиционная модель, безусловно, не подходит для обработки огромных объемов масштабируемых данных и не может быть приспособлена стандартными серверами баз данных. Кроме того, централизованная система создает слишком много узких мест при одновременной обработке нескольких файлов.

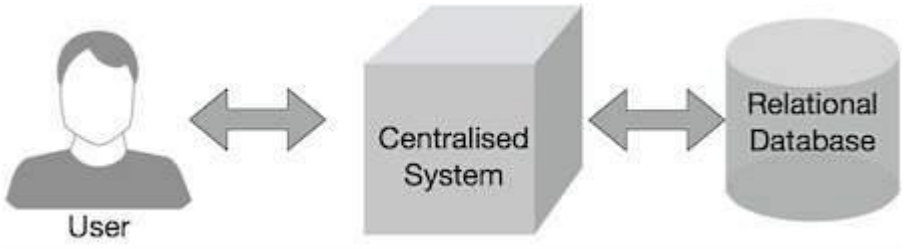


Рисунок 1. Традиционной вид корпоративной системы

Google решил эту проблему узкого места с помощью алгоритма под названием MapReduce. MapReduce разбивает задачу на мелкие части и распределяет их по множеству компьютеров. Позже результаты собираются в одном месте и интегрируются для формирования результирующего набора данных.

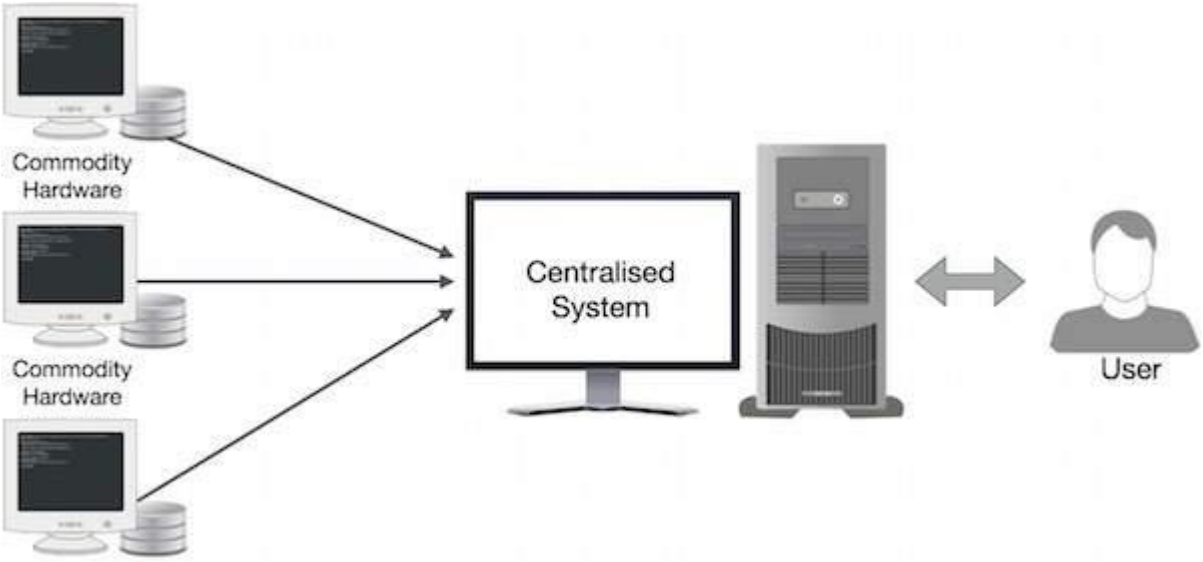


Рисунок 2. Вид корпоративной системы с использованием технологии MapReduce

6.2 Принцип работы MapReduce

Алгоритм MapReduce содержит две важные задачи, а именно Map и Reduce.

- Задача Map берет набор данных и преобразует его в другой набор данных, где отдельные элементы разбиваются на кортежи (пары ключ-значение).
- Задача Reduce принимает выходные данные карты в качестве входных данных и объединяет эти кортежи данных (пары ключ-значение) в меньший набор кортежей.

Задача Reduce всегда выполняется после Map.

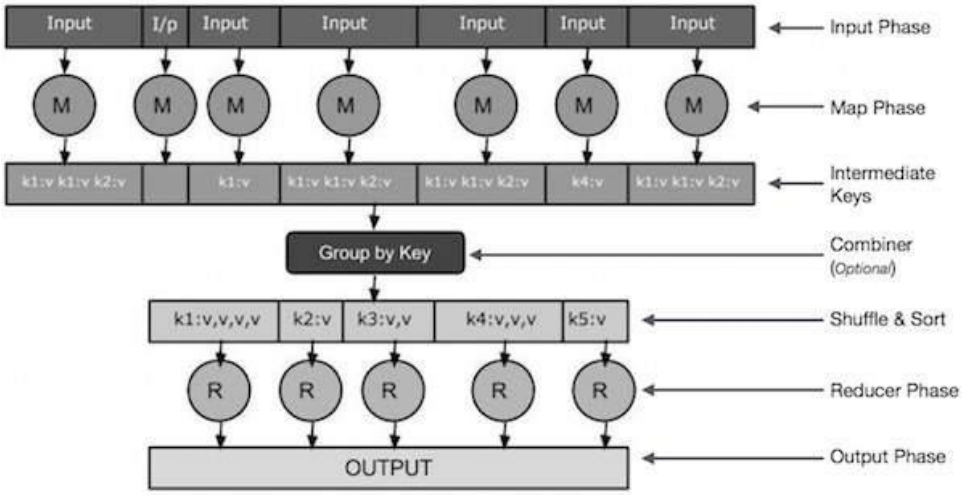


Рисунок 3. Работа MapReduce

- **Input Phase (фаза ввода)** - здесь у нас есть "считыватель" записей, который переводит каждую запись во входной файл и отправляет проанализированные данные в *mapper* в виде пар ключ-значение.
- **Map** - это определяемая пользователем функция, которая берет серию пар ключ-значение и обрабатывает каждую из них, чтобы создать ноль или больше пар ключ-значение.
- **Intermediate Keys (промежуточные ключи)** - это пары ключ-значение, генерируемые mapper'ом, которые называются промежуточными ключами.
- **Combiner (сумматор)** - это тип локального *Reducer'a*, который группирует аналогичные данные из фазы карты в идентифицируемые наборы. Он принимает промежуточные ключи от mapper'a в качестве входных данных и применяет пользовательский код для агрегирования значений в небольшом объеме одного mapper'a. Он не является частью основного алгоритма MapReduce, но необязателен.
- **Shuffle and Sort (перемешивание и сортировка)** - задача *Reduce* начинается с шага перемешивания и сортировки. Он загружает сгруппированные пары ключ-значение на локальную машину, где работает *Reducer*. Отдельные пары ключ-значение сортируются по ключу в более крупный список данных. Список данных группирует эквивалентные ключи вместе, так что их значения могут быть легко повторены в задаче *Reduce*.
- **Reduce** - *Reducer* принимает сгруппированные парные данные ключ-значение в качестве входных данных и запускает функцию reduce на каждом из них. Здесь данные могут быть агрегированы, отфильтрованы и объединены несколькими способами, и это требует широкого спектра обработки. После завершения выполнения он дает ноль или более пар ключ-значение для последнего шага.
- **Output Phase (выходная фаза)** − в выходной фазе у нас есть выходной "formatter" , который переводит конечные пары ключ-значение из функции редуктора и записывает их в файл с помощью записывающего устройства.

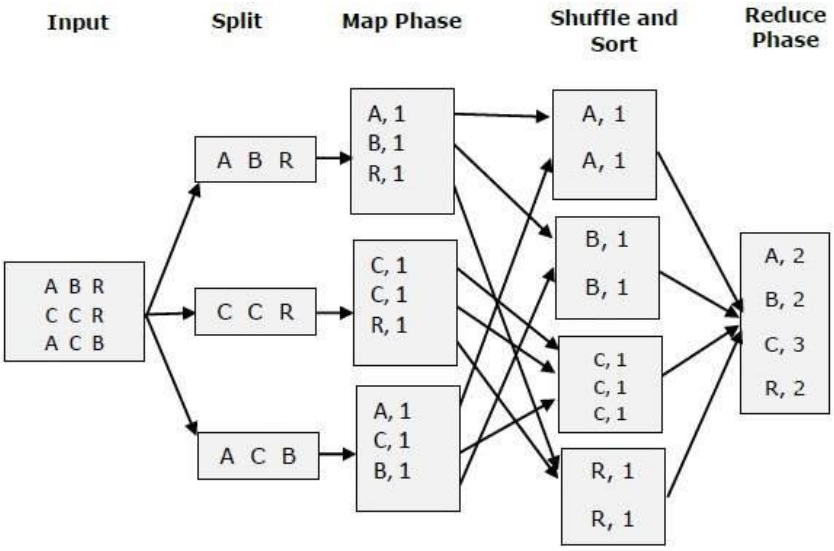


Рисунок 4. Диаграмма работы Map & Reduce

6.3 MapReduce - API JAVA

- JobContext Interface
- Job Class
- Mapper Class
- Reducer Class

6.3.1 JobContext Interface

Интерфейс JobContext - это супер-интерфейс для всех классов, который определяет различные службы в MapReduce. Это дает доступ только к чтению, которое предоставляется службам во время их выполнения.

Ниже приведены подчиненные интерфейсы интерфейса JobContext.

No.	Subinterface Description
1.	MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> Определяет контекст, который передается Mapper.
2.	ReduceContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> Определяет контекст, который передается Reducer.

Класс Job - это основной класс, реализующий интерфейс JobContext.

6.3.2 Job Class

Класс Job - самый важный класс в API MapReduce. Он позволяет пользователю настраивать службу, отправлять его, контролировать его выполнение и запрашивать состояние. Установленные методы работают только до тех пор, пока службе не будет отправлено, после чего они выдадут исключение *IllegalStateException*.

Обычно пользователь создает приложение, описывает различные аспекты службы, а затем отправляет её и отслеживает выполнение.

Вот пример того, как отправить службу-

```
// Create a new Job

Job job = new Job(new Configuration());

job.setJarByClass(MyJob.class);


// Specify various job-specific parameters

job.setJobName("myjob");

job.setInputPath(new Path("in"));

job.setOutputPath(new Path("out"));


job.setMapperClass(MyJob.MyMapper.class);

job.setReducerClass(MyJob.MyReducer.class);


// Submit the job, then poll for progress until the job is complete
```

```
job.waitForCompletion(true);
```

6.3.2.1 Конструкторы

No	Краткое описание
1	Job()
2	Job(Configuration conf)
3	Job(Configuration conf, String jobName)

6.3.3 Методы

Основные методы класса:

No	Описание
1	getJobName() Указанное пользователем имя службы.
2	getJobState() Возвращает текущее состояние службы.
3	isComplete() Проверяет, завершена служба или нет.
4	setInputFormatClass() Устанавливает InputFormat для службы.
5	setJobName(String name) Устанавливает указанное пользователем имя службы.
6	setOutputFormatClass() Устанавливает выходной формат для службы.
7	setMapperClass(Class) Устанавливает Mapper для службы.
8	setReducerClass(Class) Устанавливает Reducer для службы.
9	setPartitionerClass(Class) Устанавливает разделитель для службы.

No	Описание
10	setCombinerClass(Class) Устанавливает сумматор для job'ы.

6.3.4 Mapper Class

Класс Mapper определяет службу Map. Сопоставляет входные пары "ключ-значение" с набором промежуточных пар "ключ-значение". Карты - это отдельные job'ы, которые преобразуют входные записи в промежуточные записи. Преобразованные промежуточные записи не обязательно должны быть того же типа, что и входные записи. Данная входная пара может отображаться на ноль или много выходных пар.

6.3.4.1 Методы

map **-** основной метод класса.

```
map(KEYIN key,  VALUEIN value,  org.apache.hadoop.mapreduce.Mapper.Context context)
```

Этот метод вызывается один раз для каждой пары "ключ-значение" в разбиении ввода.

6.3.5 Reducer Class

Класс Reducer определяет задание Reduce в MapReduce. Он сокращает набор промежуточных значений, имеющих общий ключ, до меньшего набора значений. Реализации Reducer могут получить доступ к Configuration для задания через метод JobContext.getConfiguration ().

Reducer имеет три основных этапа - перемешивание, сортировку и сокращение:

- Shuffle - Reducer копирует отсортированный вывод из каждого Mapper, используя HTTP по сети.
- Sort - фреймворк сортирует входные данные Reducer по ключам (поскольку разные Mappers могут выводить один и тот же ключ). Фазы перетасовки и сортировки происходят одновременно, то есть во время выборки выходных данных они объединяются.
- Reduce - на этом этапе метод reduce (Object, Iterable, Context) вызывается для каждого <ключа (коллекции значений)> в отсортированных входных данных.

6.3.5.1 Методы

reduce **-** основной метод класса.

```
reduce(KEYIN key,  Iterable< VALUEIN> >  values,  org.apache.hadoop.mapreduce.Reducer.Context context)
```

Этот метод вызывается один раз для каждого ключа в коллекции пар ключ-значение.

7 Компонент Oozie

Oozie — это проект с открытым исходным кодом на основе технологии Java™, упрощающий процесс создания потоков работ и координацию заданий. Oozie предоставляет принципиальную возможность объединения нескольких последовательно выполняемых заданий в одну логическую единицу работы. Одно из преимуществ инфраструктуры Oozie состоит в том, что она полностью интегрирована со стеком Hadoop и поддерживает задания Hadoop для MapReduce, Pig, Hive и Sqoop. Кроме того, ее можно применять для планирования заданий с учетом специфики системы, например, Java-программ. Oozie позволяет Hadoop-администраторам создавать сложные преобразования данных, способные объединять обработку разнообразных индивидуальных задач и даже подпотоков работ. Такая возможность улучшает контролируемость сложных заданий и облегчает повторение этих заданий с заранее заданными интервалами.

На практике существуют различные типы заданий Oozie:

- Задания типа *Oozie Workflow*— представляются в виде ориентированных ациклических графов, задающих последовательность выполняемых действий.
- Задания типа *Oozie Coordinator*— представляют задания потока работ Oozie, инициируемые по времени и по доступности данных.
- Задание типа *Oozie Bundle*— облегчает упаковку нескольких заданий типа Oozie Workflow; упрощает управление жизненным циклом этих заданий.

7.1 Как работает Oozie?

Поток работ Oozie представляет собой коллекцию действий, организованных в виде ориентированного ациклического графа (directed acyclic graph, DAG). Этот граф может содержать два типа узлов: узлы управления (control node) и узлы действия (action node). *Узлы управления*, которые используются для определения временной последовательности заданий, предоставляют правила для начала и окончания потока работ, а также управляют последовательностью выполнения потока работ с помощью точек принятия возможных решений, в качестве которых используются узлы ветвления (fork node) и узлы соединения (join node). *Узлы действия* используются для инициирования выполнения (включения) задач. В частности, узел действия может представлять собой MapReduce-задание, Pig-приложение, задачу файловой системы или Java-приложение (действия shell и ssh применять не рекомендуется).

Инфраструктура Oozie использует собственную реализацию стека Hadoop, которая поддерживает все типы заданий Hadoop и интегрирована с обычным стеком Hadoop. В частности, Oozie отвечает за инициирование действий потока работ, а фактическое выполнение задач осуществляется с помощью Hadoop MapReduce. Это позволяет Oozie использовать существующие Hadoop-механизмы для выравнивания нагрузки, для переключения при отказе и т.д. Oozie обнаруживает завершение задач посредством таких механизмов, как обратный вызов (callback) и опрос (polling). Когда инфраструктура Oozie запускает какую-либо задачу, она предоставляет ей уникальный HTTP URL для обратного вызова, а после завершения выполнения этой задачи отсылает по этому URL соответствующее уведомление. Если задача не может выполнить обратный вызов по этому URL, инфраструктура Oozie опрашивает эту задачу на предмет ее завершения. На рис.1 показан пример потока работ Oozie, включающий шесть узлов действия (Pig-скрипт, MapReduce-задания (MR), Java-код, HDFS-задача) и пять узлов управления (Start, Decision, Fork, Join, End). Кроме того, потоки работ Oozie могут быть снабжены параметрами ("параметризированы"). При отправке задания потока работ необходимо задать соответствующие значения для этих параметров. При использовании надлежащих параметров несколько идентичных заданий потоков работ могут выполняться одновременно.

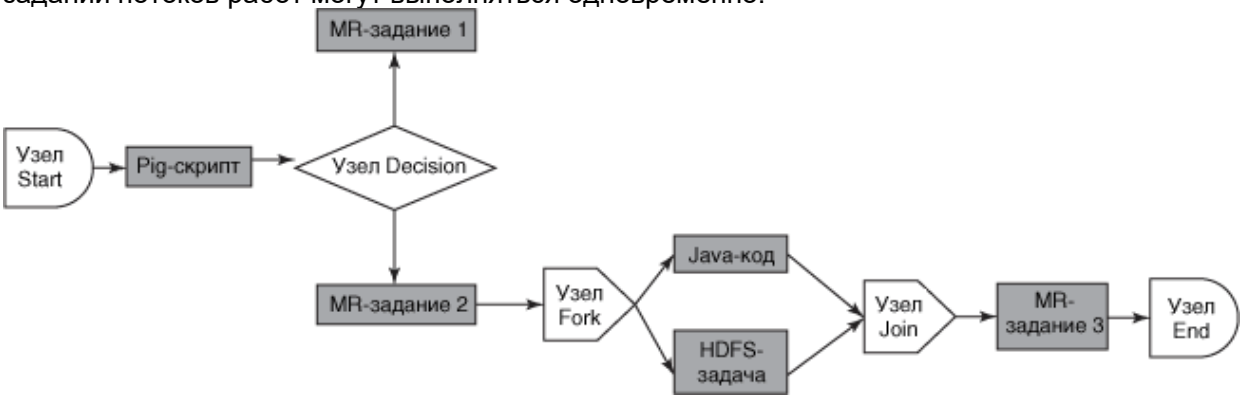


Рисунок 1. Пример потока работ Oozie

На практике иногда необходимо выполнять задания типа Oozie Workflow с регулярными временными интервалами, но в координации с другими условиями, такими как доступность определенных данных или завершение каких-либо других событий или задач. В этих ситуациях задания типа Oozie Coordinator позволяют пользователю моделировать триггеры выполнения потока работ в форме предикатов данных, времени или событий. При этом задания потока работ будут запускаться после удовлетворения соответствующих предикатов. Задание Oozie Coordinator также может управлять несколькими потоками работ, которые зависят от результатов последующих потоков работ. Выходные результаты последующих потоков работ становятся входными данными для очередного потока работ. Эта цепочка носит название *data application pipeline* (конвейер применения данных).

Язык для определения потока работ Oozie базируется на технологии XML и носит название *Hadoop Process Definition Language*. В комплект поставки Oozie входит программа командной строки для отправки заданий. Эта программа взаимодействует с Oozie-сервером посредством REST. Чтобы отправить или выполнить задание с использованием клиента Oozie, предоставьте инфраструктуре Oozie полный маршрут к своему файлу workflow.xml в HDFS в качестве параметра для клиента Oozie. В Oozie отсутствует понятие глобальных свойств. Все свойства, в том числе *jobtracker* и *namenode*, необходимо отправлять в составе каждого выполняемого задания. Для сохранения состояния в Oozie используется реляционная СУБД.

7.2 Упаковка и развертывание workflow приложения Oozie

Workflow приложение состоит из определения рабочего процесса и всех связанных ресурсов, таких как файлы MapReduce, сценарии Pig и т. д. Приложения должны следовать простой структуре каталогов и развертываться в HDFS, чтобы Oozie мог получить к ним доступ.

Необходимо сохранить рабочий процесс.xml (файл определения рабочего процесса) в каталоге верхнего уровня (родительский каталог с именем рабочего процесса). Каталог Lib содержит Jar-файлы, содержащие классы MapReduce. Приложение рабочего процесса, соответствующее этому макету, может быть построено с помощью любого инструмента сборки, например, Ant или Maven.

Шаги для запуска задания рабочего процесса Oozie

В этом разделе мы рассмотрим, как выполнить задание рабочего процесса. Для этого мы будем использовать инструмент командной строки Oozie (клиентская программа, которая взаимодействует с сервером Oozie).

1. Экспортируйте переменную окружения OOZIE_URL, которая сообщает команде oozie, какой сервер Oozie использовать (здесь мы используем один, работающий локально):

```
export OOZIE_URL="http://localhost:11000/oozie"
```

2. Запустите задание рабочего процесса с помощью

```
oozie job -config ch05/src/main/resources/max-temp-workflow.properties -run
```

Параметр -config относится к локальному файл свойств, содержащий определения параметров в XML-файле рабочего процесса, а также oozie.wf.application.path, который сообщает Oozie местоположение приложения рабочего процесса в HDFS. Пример содержимого файла свойств:

```
nameNode=hdfs://localhost:8020

jobTracker=localhost:8021

oozie.wf.application.path=${nameNode}/user/${user.name}/<name of workflow>
```

3. Получите статус Workflow

Статус задания Workflow можно увидеть с помощью подкоманды "job" с параметром "- info "и указанием идентификатора задания после "- info".

```
oozie job -info <job id>
```

Вывод показывает состояние, которое является одним из запущенных, убитых или успешных.

4. Результаты успешного выполнения рабочего процесса можно увидеть с помощью команды Hadoop

```
hadoop fs -cat <location of result>
```

7.3 Command Line Tools

Oozie предоставляет утилиту командной строки Oozie для выполнения заданий и задач администратора.

```
oozie version : show client version
```

Ниже приведены некоторые из других операций задач Oozie.

```
oozie jobs <OPTIONS> : jobs status
        -auth <arg>       select authentication type [SIMPLE|KERBEROS]
        -bulk <arg>       key-value pairs to filter bulk jobs response. e.g.
                bundle=<B>\;coordinators=<C>\;actionstatus=<S>\;startcreatedtime=
                <SC>\;endcreatedtime=<EC>\;startscheduledtime=<SS>\;endscheduledt
                ime=<ES>\; bundle, coordinators and actionstatus can be multiple
                comma separated valuesbundle and coordinators can be id(s) or
                appName(s) of those jobs. Specifying bundle is mandatory, other
                params are optional
        -doas <arg>       doAs user, impersonates as the specified user
        -filter <arg>
                user=<U>\;name=<N>\;group=<G>\;status=<S>\;frequency=<F>\;unit=<M
                >\;startcreatedtime=<SC>\;endcreatedtime=<EC> \;sortBy=<SB>(valid
                unit values are 'months', 'days', 'hours' or 'minutes'.
                startcreatedtime, endcreatedtime: time of format
                yyyy-MM-dd'T'HH:mm'Z'. valid values for sortBy are 'createdTime'
                or 'lastModifiedTime'.)
        -jobtype <arg>    job type ('Supported in Oozie-2.0 or later versions ONLY -
                'coordinator' or 'bundle' or 'wf'(default))
        -kill             bulk kill operation
        -len <arg>        number of jobs (default '100')
        -localtime        use local time (same as passing your time zone to -timezone).
                Overrides -timezone option
        -offset <arg>     jobs offset (default '1')
        -oozie <arg>      Oozie URL
        -resume           bulk resume operation
        -suspend          bulk suspend operation
        -timezone <arg>   use time zone with the specified ID (default GMT).
```

See 'oozie info -timezones' for a list

-verbose verbose mode

Для проверки состояния задания можно использовать следующие команды.

-auth <arg> select authentication type [SIMPLE|KERBEROS]

-doas <arg> doAs user, impersonates as the specified user.

-filter <arg> user = <U>; name = <N>; group = <G>; status = <S>; ...

-jobtype <arg> job type ('Supported in Oozie-2.0 or later versions ONLY -

coordinator' or 'wf' (default))

-len <arg> number of jobs (default '100')

-localtime use local time (default GMT)

-offset <arg> jobs offset (default '1')

-oozie <arg> Oozie URL

-verbose verbose mode

Список джобов: oozie jobs --oozie http://oozie-url:11000/oozie

Состояния джоба: oozie job -oozie http://oozie-url:11000/oozie-info <wfid>

Логи джоба: oozie job -oozie http://oozie-url:11000/oozie-log <wfid>

8 Компонент Solr

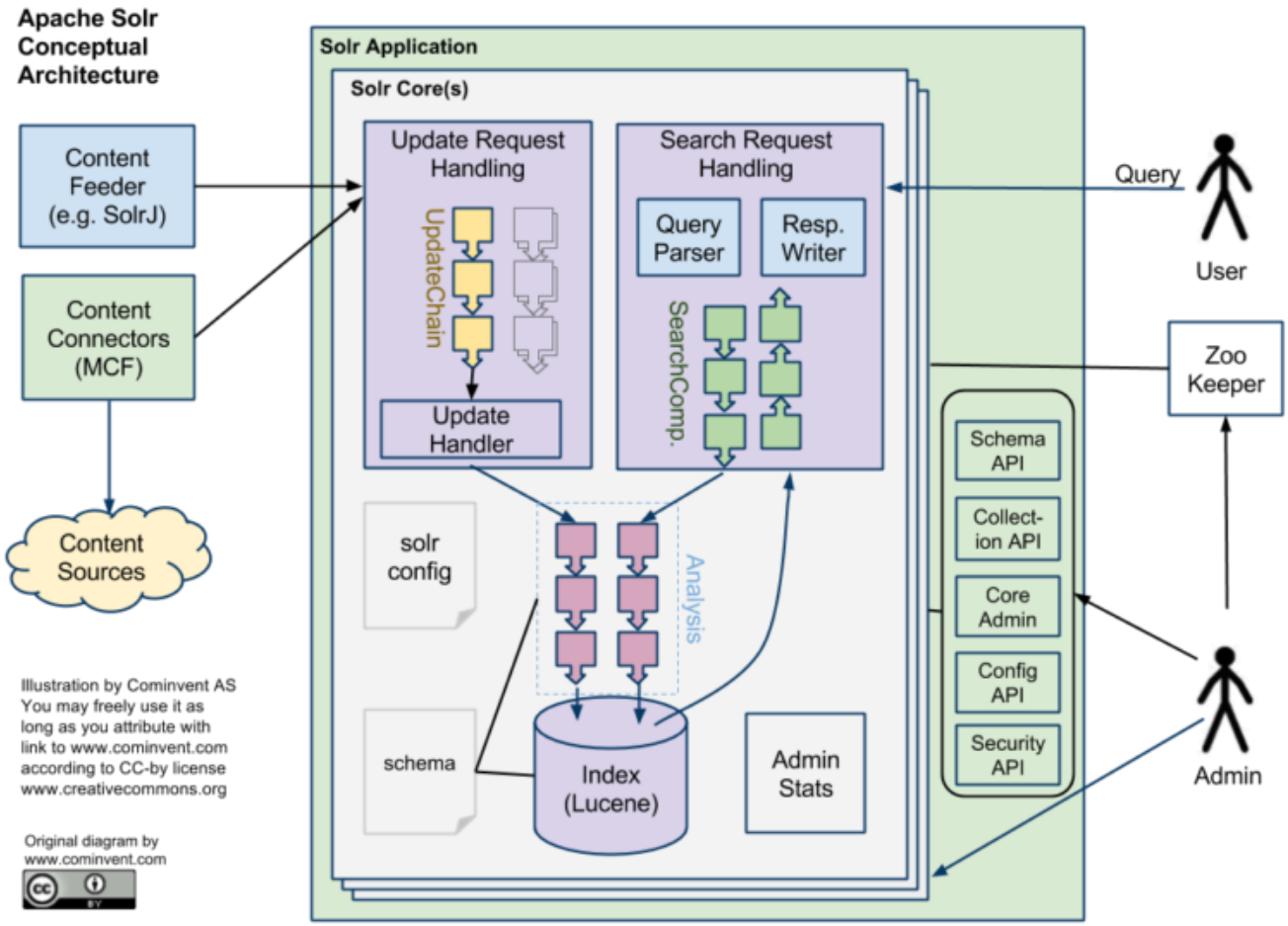
Solr - платформа полнотекстового поиска с открытым исходным кодом, основанная на проекте Apache Lucence.

8.1 Основные возможности

- Полнотекстовый поиск
- Подсветка результатов
- Фасетный поиск
- Динамическая кластеризация
- Интеграция с базами данных
- Обработка документов со сложным форматом (например, Word, PDF)

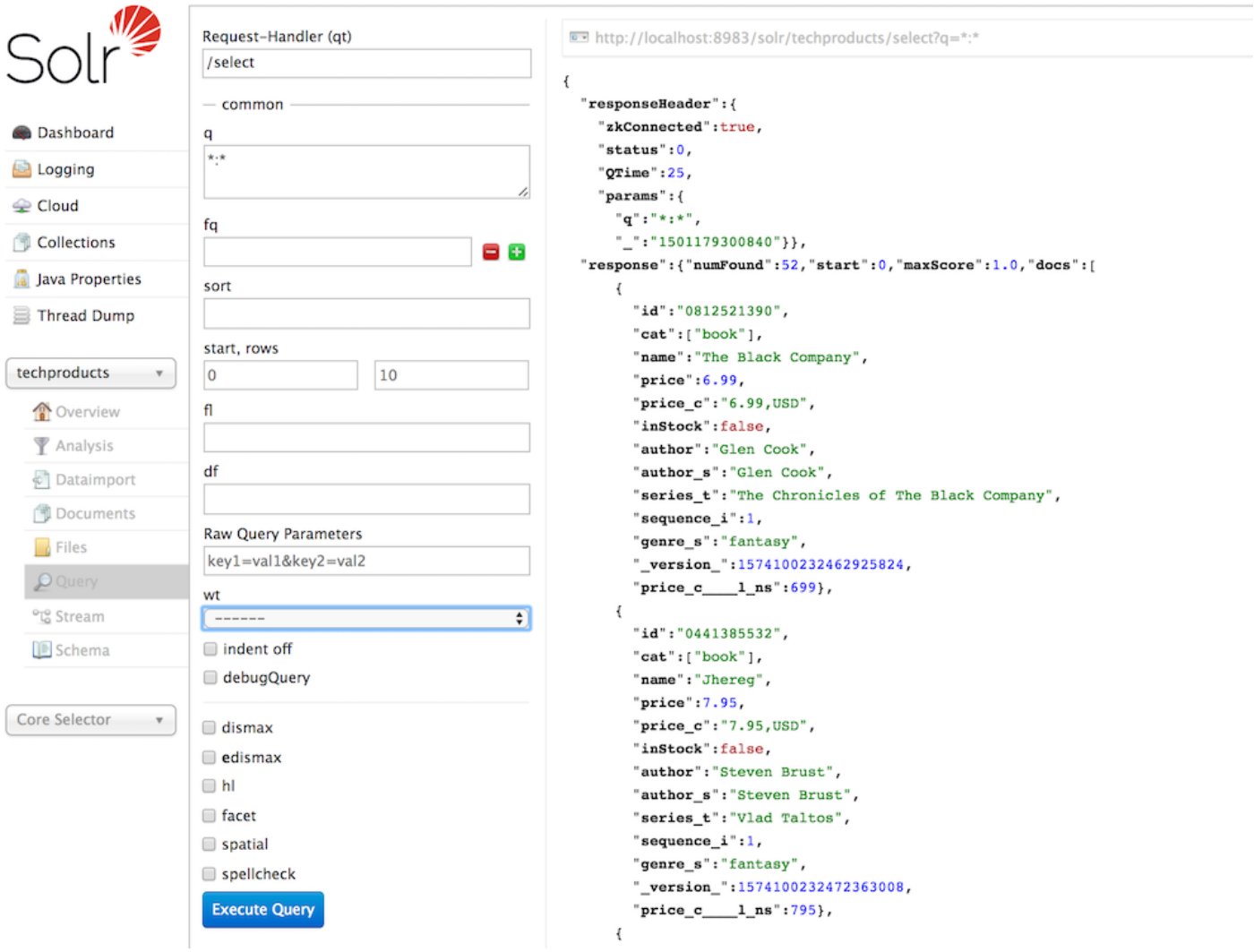
Так как в Solr есть возможность распределенного поиска и репликации, Solr хорошо масштабируем. Solr является вторым по популярности поисковым движком

8.2 Архитектура



8.3 Интерфейсы

Есть web-интерфейс в котором доступны функции управления шардированием, администрирование и конструктор запросов.



8.4 API коллекций

Кластер SolrCloud включает в себя ряд компонентов. API Collections предоставляет возможность управлять кластером, включая коллекции, сегменты, реплики, резервные копии, выбор лидера и другие операции.

Поскольку этот API содержит большое количество команд и опций, их можно сгруппировать команды в следующие подразделы:

- **Управление кластером и узлом**: определение свойств для всего кластера; проверка состояния кластера; удаление реплик из узла; использование вновь добавленного узла; добавление или удаление ролей для узла.
- **Управление коллекциями**: создание, перечисление, перезагрузка и удаление коллекций; установка свойств коллекций; перенос документов в другую коллекцию; перебалансировка лидеров; резервное копирование и восстановление коллекций.
- **Псевдонимы коллекций**: создание, перечисление или удаление псевдонимов коллекций; установка свойств псевдонимов.
- **Управление осколком**: создание и удаление осколка; разделение осколка на два или более дополнительных осколка; принудительное использование лидера осколка.
- **Управление репликами**: добавление или удаление реплики; установка свойств реплики; перемещение реплики на другой узел.

8.5 Asynchronous Calls

Поскольку некоторые вызовы API-интерфейса сбора могут быть длительными задачами (например, SPLITSHARD), при желании вы можете выполнить вызовы асинхронно. Указание async = <request-id>

позволяет выполнять асинхронный вызов, статус которого можно запросить с помощью вызова REQUESTSTATUS в любое время.

На данный момент REQUESTSTATUS не очищает автоматически структуры данных отслеживания, что означает, что статус выполненных или неудачных задач остается в ZooKeeper, если не очищен вручную. DELETESTATUS можно использовать для очистки сохраненных статусов. Однако существует ограничение в 10 000 на количество ответов на асинхронный вызов, хранящихся в кластере.

8.5.1 Примеры асинхронных запросов

Ввод

http://localhost:8983/solr/admin/collections?action=SPLITSHARD&collection=collection1&shard=shard1&async=1000&wt=xml

Выход

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">99</int>
  </lst>
  <str name="requestid">1000</str>
</response>
```

8.5.2 REQUESTSTATUS: статус запроса асинхронного вызова

Запросить статус и ответ уже отправленного вызова API асинхронной коллекции (ниже). Этот вызов также используется для очистки сохраненных статусов.

/admin/collections?action=REQUESTSTATUS&requestid=request-id

8.5.2.1 Параметры REQUESTSTATUS

requestid

Пользовательский идентификатор запроса для запроса. Это можно использовать для отслеживания состояния отправленной асинхронной задачи. Этот параметр обязателен.

8.5.3 DELETESTATUS: удалить статус

Удаляет сохраненный ответ уже завершенного или неудачного ответа Асинхронный API сбора данных вызов.

/admin/collections?action=DELETESTATUS&requestid=request-id

8.5.3.1 Параметры DELETESTATUS

requestid

Идентификатор запроса асинхронного вызова, сохраненный ответ которого должен быть очищен.

flush

Установлен в true чтобы очистить все сохраненные завершенные и неудачные ответы на асинхронные запросы.

8.6 Команды управления кластерами и узлами

Извлеките состояние кластера, включая коллекции, сегменты, реплики, имя конфигурации, а также псевдонимы коллекций и свойства кластера.

8.6.1 CLUSTERSTATUS

/admin/collections?action=CLUSTERSTATUS

8.6.1.1 Параметры CLUSTERSTATUS

collection

Коллекция или псевдоним, для которого запрашивается информация. Если этот параметр опущен, то будет возвращена информация обо всех коллекциях в кластере. Если указан псевдоним, то будет возвращена информация о коллекциях в этом псевдониме.

shard

Сегмент, для которого запрашивается информация. Несколько имен сегментов можно указать в виде списка, разделенного запятыми.

route

Это можно использовать, если вам нужны сведения о сегменте, к которому принадлежит конкретный документ, и вы не знаете, под какой сегмент он подпадает.

8.6.1.2 Ответ CLUSTERSTATUS

Ответ будет включать в себя состояние запроса и состояние кластера.

8.6.2 CLUSTERPROP: свойства кластера

Добавьте, отредактируйте или удалите свойство всего кластера.

/admin/collections?action=CLUSTERPROP&name=propertyName&val=propertyValue

8.6.2.1 Параметры CLUSTERPROP

name

Название свойства. Поддерживаемые имена: autoAddReplicas, legacyCloud, location, maxCoresPerNode, urlScheme и defaultShardPreferences. Можно задать и другие свойства (например, если они вам нужны для пользовательских плагинов), но они должны начинаться с префикса ext.. Неизвестные свойства, которые не начинаются с ext. будут отклоняться.

val

Значение свойства. Если значение пусто или равно нулю, то свойство не задано.

8.6.2.2 Ответ CLUSTERPROP

Ответ будет содержать статус запроса и свойства, которые были обновлены или удалены. Если состояние отличается от "0", то сообщение об ошибке объяснит, почему запрос не удался.

8.6.3 BALANCESHARDUNIQUE: сбалансируйте свойство между узлами

/admin/collections?action=BALANCESHARDUNIQUE&collection=collectionName&property=propertyName

Гарантирует, что определенное свойство равномерно распределено между физическими узлами, составляющими коллекцию. Если свойство уже существует на реплике, прилагаются все усилия, чтобы оставить его там. Если свойство является **нет** для любой реплики на осколке выбирается одна и добавляется свойство.

8.6.3.1 Параметры BALANCESHARDUNIQUE

collection

Имя коллекции, в которой необходимо сбалансировать имущество. Этот параметр является обязательным.

property

Свойство балансировать. Константа property. добавляется к этому свойству, если оно не указано явно. Этот параметр является обязательным.

onlyactivenodes

По умолчанию используется значение true. Обычно экземпляр свойства создается только на активных узлах. Если этот параметр указан как false, то неактивные узлы также включаются для распределения.

shardUnique

Что-то вроде предохранительного клапана. Существует одно заранее определенное свойство (preferredLeader) что по умолчанию это значение равно true. Для всех остальных сбалансированных свойств этот параметр должен быть установлен в значение true или будет возвращено сообщение об ошибке.

8.6.3.2 Ответ BALANCESHARDUNIQUE

Ответ будет включать в себя статус запроса. Если состояние отличается от "0", то сообщение об ошибке объяснит, почему запрос не удался.

8.6.4 UTILIZENODE: использование нового узла

Эта команда может быть использована для перемещения некоторых реплик с существующих узлов на новый узел или менее загруженный узел, чтобы уменьшить нагрузку на существующий узел.

При этом используются ваши политики и предпочтения автоматического масштабирования, чтобы определить, какая реплика должна быть перемещена. Сначала он пытается исправить любые нарушения политики, а затем пытается переместить некоторую нагрузку с наиболее загруженных узлов в соответствии с предпочтениями.

/admin/collections?action=UTILIZENODE&node=nodeName

8.6.4.1 Параметры UTILIZENODE

node

Имя узла, который необходимо использовать. Этот параметр является обязательным.

8.6.5 REPLACENODE: перемещение всех реплик в одном узле в другой

Эта команда воссоздает реплики из одного узла (источника) в другой узел(ы) (цель). После копирования каждой реплики реплики в исходном узле удаляются.

Для исходных реплик, которые также являются лидерами сегментов, операция будет ждать количество секунд, установленное с помощью timeout параметр, чтобы убедиться, что есть активная реплика,

которая может стать лидером (либо существующая реплика становится лидером, либо новая реплика завершает восстановление и становится лидером).

API использует платформу автоматического масштабирования для поиска узлов, которые могут удовлетворять требованиям диска для новых реплик, но только при настройке политики автоматического масштабирования. Дополнительные сведения см. В разделе Политика и предпочтения автоматического масштабирования.

```
/admin/collections?action=REPLACENODE&sourceNode=source-node&targetNode=target-node
```

8.6.5.1 Параметры REPLACENODE

sourceNode

Исходный узел, с которого необходимо скопировать реплики. Этот параметр является обязательным.

targetNode

Целевой узел, на который будут скопированы реплики. Если этот параметр не указан, Solr автоматически идентифицирует узлы на основе политик или количества ядер в каждом узле.

parallel

Если этот флаг установлен в true, все реплики создаются в отдельных потоках. Имейте в виду, что это может привести к очень высокому сетевому и дисковому вводу-выводу, если реплики имеют очень большие индексы. Значение по умолчанию равно false.

async

Запросите идентификатор для отслеживания этого действия, которое будет обрабатывается асинхронно.timeout время в секундах для ожидания создания новых реплик и полного восстановления реплик лидеров. Значение по умолчанию равно 300 или 5 минут.

8.6.6 DELETENODE: удаление реплик в узле

Удаляет все реплики всех коллекций в этом узле. Обратите внимание, что сам узел останется в качестве живого узла после этой операции.

```
/admin/collections?action=DELETENODE&node=nodeName
```

8.6.6.1 Параметры DELETENODE

node

Узел, который нужно удалить. Этот параметр является обязательным.

async

Запросите идентификатор для отслеживания этого действия, которое будет обрабатывается асинхронно.

8.6.7 ADDROLE: добавление роли

```
/admin/collections?action=ADDROLE&role=roleName&node=nodeName
```

Назначает роль данному узлу в кластере. Единственная поддерживаемая роль-это overseer.

Используйте эту команду, чтобы выделить определенный узел в качестве надзирателя. Вызовите его несколько раз, чтобы добавить больше узлов. Это полезно в больших кластерах, где надзиратель, скорее всего, будет перегружен. Если он доступен, то один из узлов списка, которым назначена роль "надзирателя", станет надзирателем. Система назначит эту роль любому другому узлу, если ни один из назначенных узлов не запущен и не работает.

8.6.7.1 Параметры ADDROLE

role

Название роли. Единственная поддерживаемая роль на данный момент это overseer. Этот параметр является обязательным.

node

Имя узла, которому будет назначена роль. Можно назначить роль еще до запуска этого узла. Этот параметр запускается.

8.6.7.2 Ответ ADDROLE

Ответ будет содержать статус запроса и свойства, которые были обновлены или удалены. Если состояние отличается от "0", то сообщение об ошибке объяснит, почему запрос не удался.

8.6.8 REMOVEROLE: удалить роль

Удалите назначенную роль. Этот API используется для отмены ролей, назначенных с помощью операции ADDROLE

```
/admin/collections?action=REMOVEROLE&role=roleName&node=nodeName
```

8.6.8.1 Параметры REMOVEROLE

role

Название роли. Единственная поддерживаемая роль на данный момент это overseer. Этот параметр является обязательным.

node

Имя узла, с которого должна быть удалена роль.

8.6.8.2 Ответ REMOVEROLE

Ответ будет содержать статус запроса и свойства, которые были обновлены или удалены. Если состояние отличается от "0", то сообщение об ошибке объяснит, почему запрос не удался.

8.6.9 OVERSEERSTATUS: статус и статистика Overseerstatus

Возвращает текущее состояние супервайзера, статистику производительности различных API-интерфейсов супервайзера и последние 10 сбоев для каждого типа операции.

```
/admin/collections?action=OVERSEERSTATUS
```

8.6.10 MIGRATESTATEFORMAT: миграция состояния кластера

API утилиты экспертного уровня для перемещения коллекции из общего доступа clusterstate.json Узел ZooKeeper (созданный с помощью stateFormat=1, по умолчанию во всех выпусках Solr до версии 5.0) для каждой коллекции state.json хранится в ZooKeeper (создается с помощью stateFormat=2, текущий дефолт) плавно без каких-либо простоев приложения.

```
/admin/collections?action=MIGRATESTATEFORMAT&collection=<collection_name>
```

8.6.10.1 Параметры MIGRATESTATEFORMAT

collection

Имя коллекции, из которой будет выполнен перенос clusterstate.json к своим собственным state.json Узел Zookeeper. Этот параметр является обязательным.

async

Запросите идентификатор для отслеживания этого действия, которое будет обрабатывается асинхронно.

9 Компонент Spark

9.1 Что такое Spark?

Spark – это инструмент для «молниеносных кластерных вычислений». Spark предоставляет быструю и универсальную платформу для обработки данных. Hadoop Spark ускоряет работу программ в памяти более чем в 100 раз, а на диске – более чем в 10 раз.

При изучении **Spark** стоит отметить еще один немаловажный аспект: здесь предоставляется готовая интерактивная оболочка (**REPL**). При помощи **REPL** можно протестировать результат выполнения каждой строки кода без необходимости сначала программировать и выполнять все задание целиком.

Spark имеет следующие особенности:

- В настоящее время предоставляется API для Scala, Java и Python, также готовится поддержка других языков (например, R)
- Хорошо интегрируется с экосистемой Hadoop и источниками данных (HDFS, Amazon S3, Hive, HBase, Cassandra, etc.)
- Может работать на кластерах под управлением Hadoop YARN, а также работать в автономном режиме

9.2 Компоненты Spark

Ядро Spark дополняется набором мощных высокоуровневых библиотек, которые бесшовно стыкуются с ним в рамках того же приложения. В настоящее время к таким библиотекам относятся SparkSQL, Spark Streaming, MLlib (для машинного обучения) и GraphX.

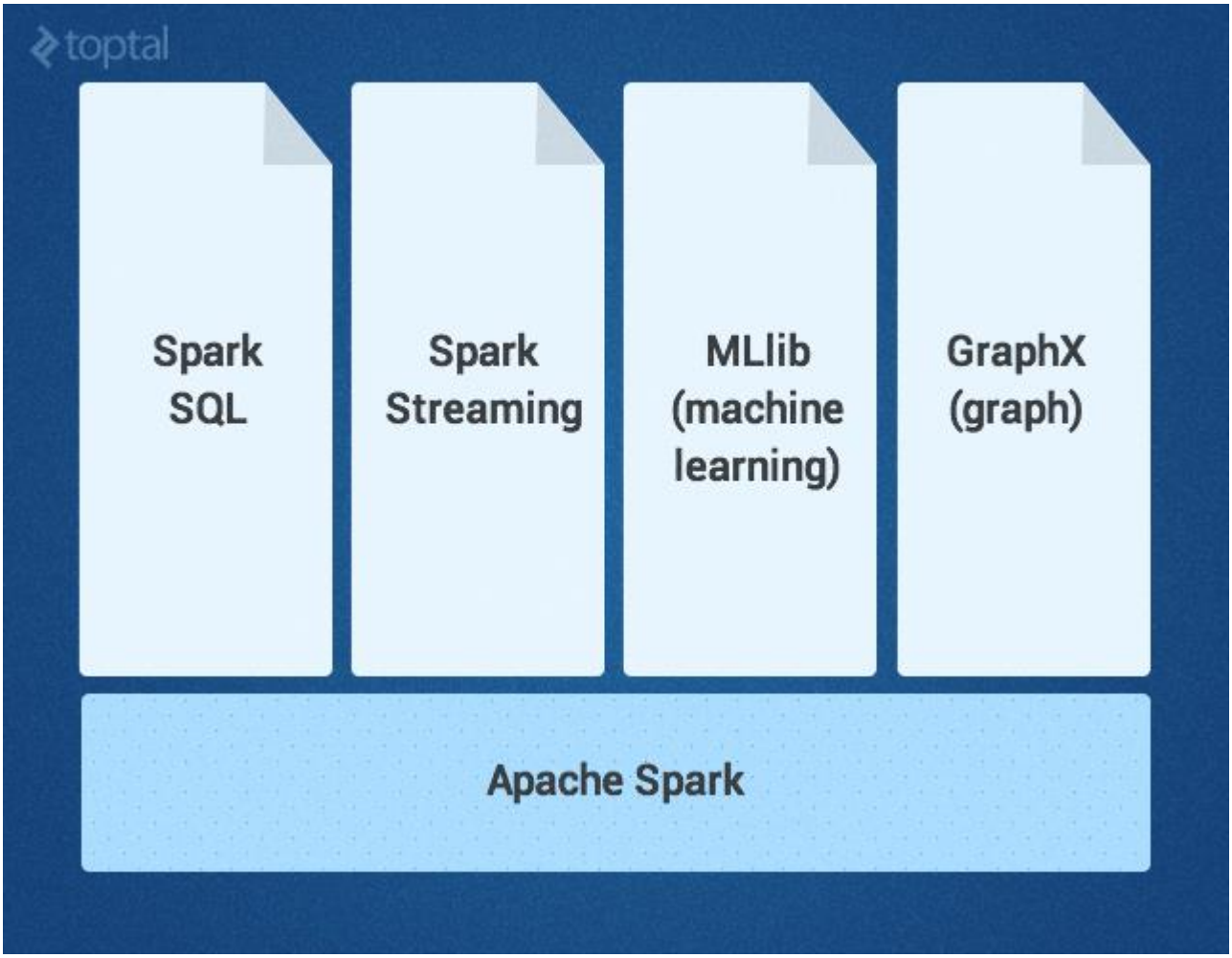


Рисунок 1. Компоненты Spark

9.2.1 Ядро Spark

Ядро Spark – это базовый движок для крупномасштабной параллельной и распределенной обработки данных. Ядро отвечает за:

- управление памятью и восстановление после отказов
- планирование, распределение и отслеживание заданий кластере
- взаимодействие с системами хранения данных

В Spark вводится концепция RDD (устойчивый распределенный набор данных) – неизменяемая отказоустойчивая распределенная коллекция объектов, которые можно обрабатывать параллельно. В RDD могут содержаться объекты любых типов; RDD создается путем загрузки внешнего набора данных или распределения коллекции из основной программы (driver program). В RDD поддерживаются операции двух типов:

- **Трансформации** – это операции (например, отображение, фильтрация, объединение и т.д.), совершаемые над RDD; результатом трансформации становится новый RDD, содержащий ее результат.
- **Действия** – это операции (например, редукция, подсчет и т.д.), возвращающие значение, получаемое в результате некоторых вычислений в RDD.

Трансформации в Spark осуществляются в «ленивом» режиме — то есть, результат не вычисляется сразу после трансформации. Вместо этого они просто «запоминают» операцию, которую следует произвести, и набор данных (напр., файл), над которым нужно совершить операцию. Вычисление трансформаций происходит только тогда, когда вызывается действие, и его результат возвращается основной программе. Благодаря такому дизайну повышается эффективность Spark. Например, если большой файл был преобразован различными способами и передан первому действию, то Spark обработает и вернет результат лишь для первой строки, а не станет прорабатывать таким образом весь файл.

По умолчанию каждый трансформированный RDD может перевычисляться всякий раз, когда вы выполняете над ним новое действие. Однако RDD также можно долговременно хранить в памяти, используя для этого метод хранения или кэширования; в таком случае Spark будет держать нужные элементы на кластере, и вы сможете запрашивать их гораздо быстрее.

9.2.2 Скорость

- **Spark—**это вычислительный инструмент, работающий со «скоростью света». Благодаря уменьшению количества чтения-записи на диск и хранения промежуточных данных в памяти, Spark запускает приложения в ***100 раз быстрее в памяти и в 10 раз быстрее на диске, чем Hadoop***.
- **Hadoop MapReduce**—MapReduce *читает и записывает на диск, а это снижает скорость* обработки и эффективность в целом.

9.2.3 Простота использования

- **Spark**—многие библиотеки Spark облегчают выполнение большого количества основных высокоуровневых операций при помощи *RDD (Resilient Distributed Dataset/*эластичный распределённый набор данных*)*.
- **Hadoop**—в MapReduce разработчикам нужно написать вручную каждую операцию, что только усложняет процесс при масштабировании сложных проектов.

9.2.4 Обработка больших наборов данных

- **Spark**—так как, Spark оптимизирован относительно скорости и вычислительной эффективности при помощи хранения основного объёма данных в памяти, а не на диске, он может показывать более низкую производительность относительно Hadoop MapReduce в случаях, когда размеры данных становятся такими огромными, что недостаточность RAM становится проблемой.
- **Hadoop** —Hadoop MapReduce позволяет обрабатывать огромные наборы данных параллельно. Он разбивает большую цепочку на небольшие отрезки, чтобы обрабатывать каждый отдельно на разных узлах данных. Если итоговому датасету необходимо больше, чем имеется в доступе RAM, Hadoop MapReduce может сработать лучше, чем Spark.Поэтому Hadoop стоит выбрать в том случае, когда скорость обработки не критична и решению задач можно отвести ночное время**,** чтобы утром результаты были готовы.

9.2.5 Функциональность

Spark—*неизменный победитель в этой категории.Ниже я даю список основных задач по анализу больших данных, в которых Spark опережает Hadoop по производительности:*

- **Итеративная обработка.** Если по условию задачи нужно обрабатывать данные снова и снова, Spark разгромит Hadoop MapReduce. Spark RDD активирует многие операции в памяти, в то время как Hadoop MapReduce должен записать промежуточные результаты на диск.
- **Обработка в почти что реальном времени.** Если бизнесу нужны немедленные инсайты, тогда стоит использовать Spark и его обработку прямо в памяти.
- **Обработка графов.** Вычислительная модель Spark хороша для итеративных вычислений, которые часто нужны при обработке графов. И в Spark есть GraphX—API для расчёта графов.

9.2.6 SparkSQL

SparkSQL – это компонент Spark, поддерживающий запрашивание данных либо при помощи SQL, либо посредством Hive Query Language. Библиотека возникла как порт Hive для работы поверх Spark (вместо MapReduce), а сейчас уже интегрирована со стеком Spark. Она не только обеспечивает поддержку различных источников данных, но и позволяет переплетать SQL-запросы с трансформациями кода; получается очень мощный инструмент.

9.2.7 Spark Streaming

Spark Streaming поддерживает обработку потоковых данных в реальном времени; такими данными могут быть файлы логов рабочего веб-сервера (напр. Apache Flume и HDFS/S3), информация из соцсетей, например, Twitter, а также различные очереди сообщений вроде *Kafka*. «Под капотом» Spark Streaming получает входные потоки данных и разбивает данные на пакеты. Далее они обрабатываются движком Spark, после чего генерируется конечный поток данных (также в пакетной форме) как показано ниже.

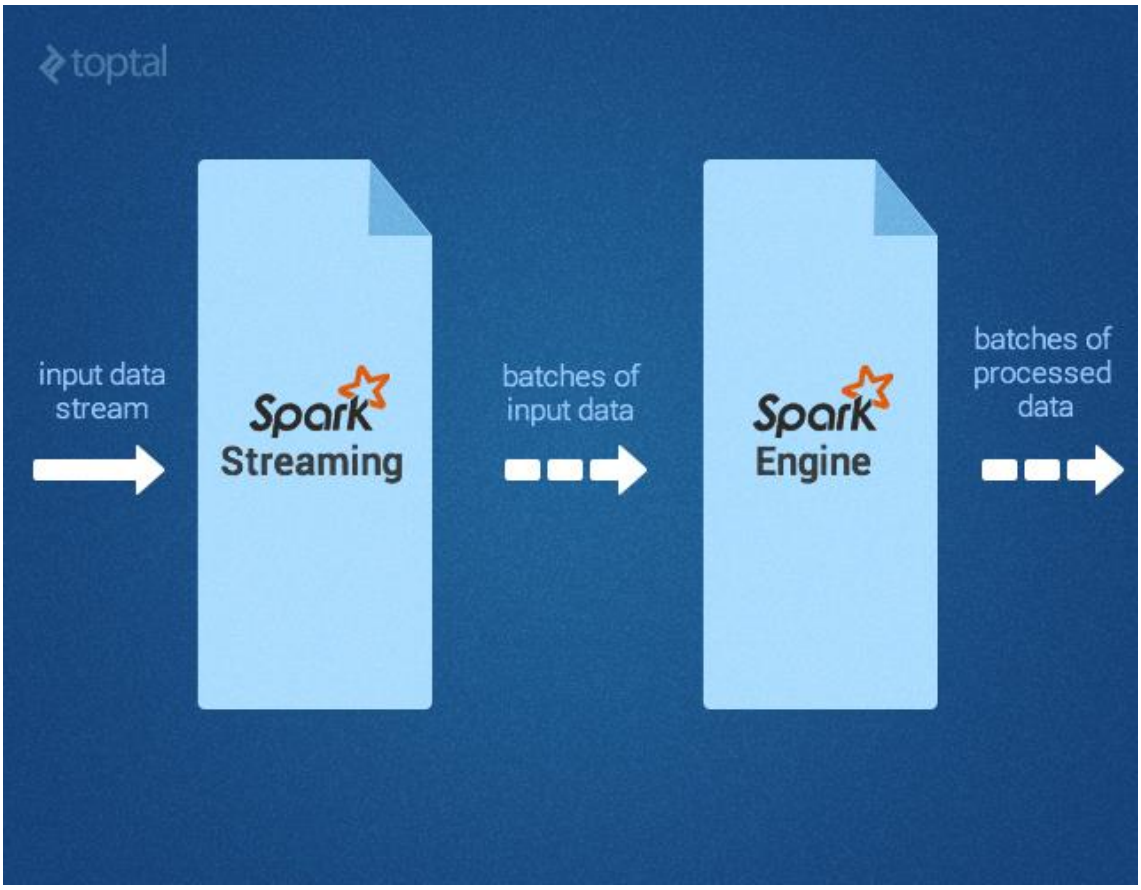


Рисунок 2. Потоки в Spark

API Spark Streaming точно соответствует API Spark Core, поэтому программисты без труда могут одновременно работать и с пакетными, и с потоковыми данными.

9.2.8 MLlib

MLlib – это библиотека для машинного обучения, предоставляющая различные алгоритмы, разработанные для горизонтального масштабирования на кластере в целях классификации, регрессии, кластеризации, совместной фильтрации и т.д. Некоторые из этих алгоритмов работают и с потоковыми данными — например, линейная регрессия с использованием обычного метода наименьших квадратов или кластеризация по методу k-средних.

9.2.9 GraphX

GraphX – это библиотека для манипуляций над графами и выполнения с ними параллельных операций. Библиотека предоставляет универсальный инструмент для ETL, исследовательского анализа и итерационных вычислений на основе графов. Кроме встроенных операций для манипуляций над графами здесь также предоставляется библиотека обычных алгоритмов для работы с графами, например, PageRank.

9.3 Spark Shell

9.3.1 Преобразования RDD

RDD transformations возвращает указатель на новый RDD и позволяет создавать зависимости между RDDs. Каждый RDD в цепочке зависимостей (строка зависимостей) имеет функцию для вычисления своих данных и имеет указатель (зависимость) на свой родительский RDD. Spark ленив, поэтому ничего не будет выполнено, если вы не вызовете какое-то преобразование или действие, которое вызовет создание и выполнение задания.

Преобразование RDD - это не набор данных, а шаг в программе (возможно, единственный шаг), указывающий Spark, как получить данные и что с ними делать. Ниже приведен список преобразований RDD.

Преобразования и смысл
map(func) Возвращает новый распределенный набор данных, сформированный путем передачи каждого элемента источника через функцию **func** .
filter(func) Возвращает новый набор данных, сформированный путем выбора тех элементов источника, для которых **func** возвращает true.
flatMap(func) Подобно map, но каждый элемент ввода может быть сопоставлен с 0 или более элементами вывода (поэтому *func* должен возвращать Seq, а не один элемент).
mapPartitions(func) Подобно map, но запускается отдельно для каждого раздела (блока) RDD, поэтому **func** должен иметь тип Iterator <T> ⇒ Iterator <U> при запуске в RDD типа T.
mapPartitionsWithIndex(func) Подобно map Partitions, но также предоставляет **func** целочисленным значением, представляющим индекс раздела, поэтому **func** должен иметь тип (Int, Iterator <T>) ⇒ Iterator <U> при работе с RDD типа T.
sample(withReplacement, fraction, seed)

Преобразования и смысл
Произведите **выборку** части данных с заменой или без нее, используя заданное начальное число генератора случайных чисел.
union(otherDataset) Возвращает новый набор данных, содержащий объединение элементов исходного набора данных и аргумента.
intersection(otherDataset) Возвращает новый RDD, который содержит пересечение элементов в исходном наборе данных и аргументе.
distinct([numTasks]) Возвращает новый набор данных, содержащий отдельные элементы исходного набора данных.
groupByKey([numTasks]) При вызове набора данных из пар (K, V) возвращает набор данных из пар (K, Iterable <V>). **Примечание.** Если вы группируете для выполнения агрегирования (например, суммы или среднего) по каждому ключу, использование reduceByKey или aggregateByKey даст гораздо лучшую производительность.
reduceByKey(func, [numTasks]) При вызове для набора данных из пар (K, V) возвращает набор данных из пар (K, V), где значения для каждого ключа агрегируются с использованием заданной функции сокращения *func* , которая должна иметь тип (V, V) ⇒ V Как и в groupByKey, количество задач сокращения настраивается с помощью необязательного второго аргумента.
aggregateByKey(zeroValue)(seqOp, combOp, [numTasks]) При вызове набора данных из пар (K, V) возвращает набор данных из пар (K, U), где значения для каждого ключа агрегированы с использованием заданных функций объединения и нейтрального «нулевого» значения. Позволяет использовать тип агрегированного значения, отличный от типа входного значения, избегая при этом ненужного выделения. Как и в groupByKey, количество задач уменьшения настраивается с помощью необязательного второго аргумента.
sortByKey([ascending], [numTasks]) При вызове набора данных из пар (K, V), где K реализует Ordered, возвращает набор данных из пар (K, V), отсортированных по ключам в порядке возрастания или убывания, как указано в логическом аргументе ascending.
join(otherDataset, [numTasks]) При вызове наборов данных типа (K, V) и (K, W) возвращает набор данных из пар (K, (V, W)) со всеми парами элементов для каждого ключа. Внешние соединения поддерживаются через leftOuterJoin, rightOuterJoin и fullOuterJoin.
cogroup(otherDataset, [numTasks])

Преобразования и смысл
При вызове наборов данных типа (K, V) и (K, W) возвращает набор данных из кортежей (K, (Iterable <V>, Iterable <W>)). Эта операция также называется группой C.
cartesian(otherDataset) При вызове наборов данных типов T и U возвращает набор данных из пар (T, U) (все пары элементов).
pipe(command, [envVars]) Передайте каждый раздел RDD через команду оболочки, например сценарий Perl или bash. Элементы RDD записываются в стандартный ввод процесса, а строки, выводимые на его стандартный вывод, возвращаются как RDD строк.
coalesce(numPartitions) Уменьшите количество разделов в RDD до numPartitions. Полезно для более эффективного выполнения операций после фильтрации большого набора данных.
repartition(numPartitions) Перемешивайте данные в RDD случайным образом, чтобы создать больше или меньше разделов и сбалансировать их между собой. Это всегда перемешивает все данные по сети.
repartitionAndSortWithinPartitions(partitioner) Переразбейте RDD в соответствии с заданным разделителем и в каждом результирующем разделе отсортируйте записи по их ключам. Это более эффективно, чем вызов повторного разбиения, а затем сортировка внутри каждого раздела, поскольку он может подтолкнуть сортировку к механизму перемешивания.

9.3.2 Действия RDD

Действие и значение
reduce(func) Агрегируйте элементы набора данных с помощью функции **func** (которая принимает два аргумента и возвращает один). Функция должна быть коммутативной и ассоциативной, чтобы ее можно было правильно вычислить параллельно.
collect() Возвращает все элементы набора данных в виде массива в программу драйвера. Обычно это полезно после фильтра или другой операции, которая возвращает достаточно небольшое подмножество данных.
count() Возвращает количество элементов в наборе данных.
first() Возвращает первый элемент набора данных

Действие и значение
take(n) Возвращает массив с первыми **n** элементами набора данных.
takeSample (withReplacement,num, [seed]) Возвращает массив с случайной выборкой num элементами набора данных, с или без замены, при необходимости предварительно указав семя генератора случайных чисел.
takeOrdered(n, [ordering]) Возвращает первые **n** элементов RDD, используя их естественный порядок или настраиваемый компаратор.
saveAsTextFile(path) Записывает элементы набора данных в виде текстового файла (или набора текстовых файлов) в заданном каталоге в локальной файловой системе, HDFS или любой другой файловой системе, поддерживаемой Hadoop. Spark вызывает toString для каждого элемента, чтобы преобразовать его в строку текста в файле.
saveAsSequenceFile(path) (Java and Scala) Записывает элементы набора данных в виде файла последовательности Hadoop по заданному пути в локальной файловой системе, HDFS или любой другой файловой системе, поддерживаемой Hadoop. Это доступно в RDD пар ключ-значение, реализующих интерфейс Hadoop Writable. В Scala он также доступен для типов, которые неявно конвертируются в Writable (Spark включает преобразования для базовых типов, таких как Int, Double, String и т. Д.).
saveAsObjectFile(path) (Java and Scala) Записывает элементы набора данных в простом формате с использованием сериализации Java, которые затем можно загрузить с помощью SparkContext.objectFile().
countByKey() Доступно только для RDD типа (K, V). Возвращает хэш-карту пар (K, Int) со счетчиком каждого ключа.

9.3.3 Числовые операции RDD

Spark позволяет выполнять различные операции с числовыми данными, используя один из предопределенных методов API. Числовые операции Spark реализованы с помощью алгоритма потоковой передачи, который позволяет строить модель по одному элементу за раз. Эти операции вычисляются и возвращаются как объект **StatusCounter** путем вызова метода **status()**. Ниже приводится список числовых методов, доступных в **StatusCounter** .

Methods & Meaning
count() Количество элементов в RDD.
Mean() Среднее значение элементов в RDD.

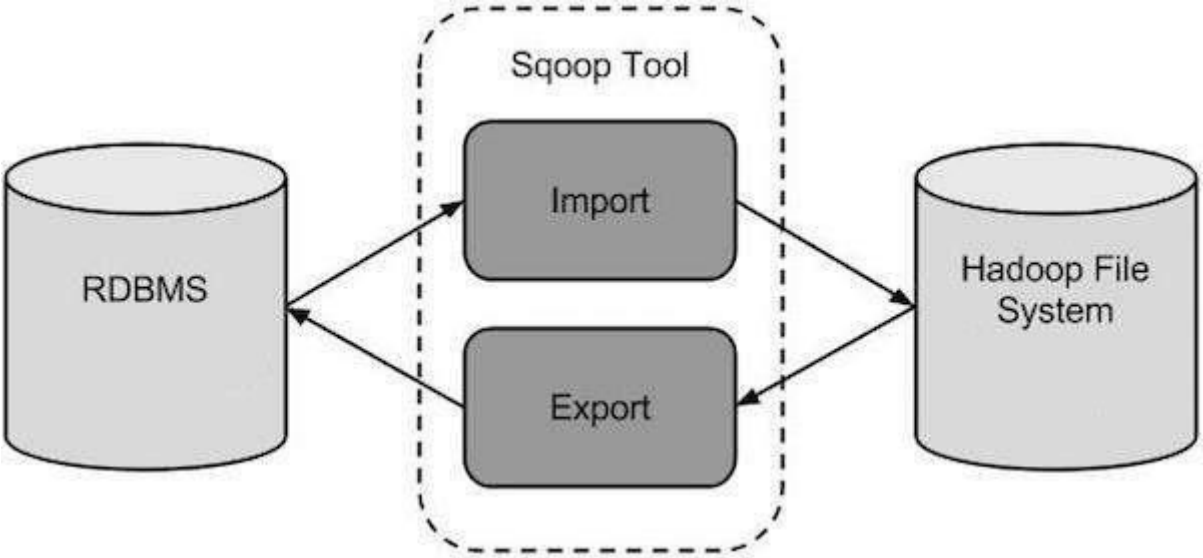
Methods & Meaning
Sum() Общая стоимость элементов в RDD.
Max() Максимальное значение среди всех элементов в RDD.
Min() Минимальное значение среди всех элементов в RDD.
Variance() Дисперсия элементов.
Stdev() Стандартное отклонение.

10 Компонент Sqoop

Sqoop - это инструмент, предназначенный для передачи данных между Hadoop и реляционными базами данных или мэйнфреймами. Вы можете использовать Sqoop для импорта данных из системы управления реляционными базами данных (RDBMS), например, PostgreSQL, или мэйнфрейма в распределенную файловую систему Hadoop (HDFS), преобразовать данные в Hadoop MapReduce, а затем экспортировать данные обратно в СУБД.

Sqoop автоматизирует большую часть этого процесса, полагаясь на базу данных для описания схемы импортируемых данных. Sqoop использует MapReduce для импорта и экспорта данных, что обеспечивает параллельную работу, а также отказоустойчивость.

10.1 Архитекрута Sqoop



Ключевой функционал Sqoop:

- Последовательный импорт данных из мэйнфрейма.
- Импорт напрямую в ORC File. Улучшенное сжатие и облегченное индексирование повышает производительность запросов.
- Импорт данных. Перемещает определенные данные из внешних хранилищ и EDW в Hadoop, чтобы оптимизировать экономическую эффективность комбинированного хранения и обработки данных;
- Параллельная передача данных. Для более быстрой работы и оптимального использования системы.
- Sqoop использует структуру YARN для импорта и экспорта данных, что обеспечивает отказоустойчивость поверх параллелизма.
- Эффективный анализ данных. Повышает эффективность анализа данных путем объединения структурированных данных с неструктурированными данными при чтении;

10.2 Импорт данных

Импорт осуществляется в два этапа. На первом этапе Sqoop просматривает базу данных, чтобы собрать необходимые метаданные для импортируемых данных. Второй этап представляет собой работу Hadoop, которую Sqoop загружает в кластер. Это та работа, которая передает данные, используя метаданные, полученные на предыдущем шаге.

Импортируемые данные сохраняются в директории на HDFS на основе таблицы, которая была импортирована. Как и в случае с большинством аспектов работы Sqoop, пользователь может указать любой альтернативный каталог, в котором файлы должны быть заполнены.

По умолчанию эти файлы содержат поля, разделённые запятыми, с новыми строками, отделяющими различные записи. Можно легко переопределить формат, в котором копируются данные, явно указав разделитель полей и символ, разбивающий записи.

Тип импорта	Синтаксис	Аргумент
Sqoop-import	$ sqoop import (generic args) (import args) $ sqoop-import (generic args) (import args)	Специфические общие аргументы Hadoop должны предшествовать любым аргументам импорта, а аргументы импорта могут иметь любой порядок.
Importing a Table into HDFS	$ sqoop import \ --connect \ --table \ --username \ --password \ --target-dir	--connect JDBC url по которому подключаться к базе данных --table Имя исходной таблицы для импорта --username Имя пользователя для подключения к базе данных --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password --target-dir Импорт данных в указанный каталог
Importing Selected Data from Table	$ sqoop import --connect \ --table \ --username \ --password \ --columns \ --where	--columns Выбирает подмножество столбцов --where Извлекает данные, удовлетворяющие условию --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password
Importing Data from Query	$ sqoop import \ --connect \ --table \ --username \ -m 1 --password \ --query	--query Выполняет предоставленный SQL-запрос и импортирует результаты --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password
Importing Data into Hive	$ sqoop import --connect \ --table \ --username \	Указанные ниже аргументы Hive используются с командой импорта sqoop для прямой

Тип импорта	Синтаксис	Аргумент
	--password \ -m 1 --hive-import \ --create-hive-table \ --hive-table	загрузки данных в Hive: --hive-import Импортировать таблицы в Hive --create-hive-table Создает таблицу Hive и завершается ошибкой, если эта таблица уже существует --hive-table Устанавливает имя таблицы Hive для импорта --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password
Importing Data into HBase	$ sqoop import --connect \ --table \ --username \ --password \ -m 1 --hbase-create-table \ --hbase-table \ --column-family	Указанные ниже аргументы HBase используются с командой импорта sqoop для прямой загрузки данных в HBase: --column-family Устанавливает семейство столбцов для импорта --hbase-create-table Если указано, создает отсутствующие таблицы HBase и завершается ошибкой, если уже существует. --hbase-row-key Определяет, какой столбец использовать в качестве ключа строки --hbase-table Импорт в таблицу Hbase --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password

Тип импорта	Синтаксис	Аргумент

Sqoop-Import-all-Tables	$ sqoop import-all-tables (generic args) (import args)
	$ sqoop-import-all-tables (generic args) (import args)
	$ sqoop-import-all-tables ---connect --username --password
		Параметры, таких как --table, --split-by, --columns и --where, являются недопустимыми.

Incremental Append import	$ sqoop import --connect \ --table \ --username \ --password \ -m 1 --incremental append \ --check-column \ --last-value	--incremental append Режим Append должен использоваться, когда новые строки постоянно добавляются с увеличивающимися значениями. --check-column отслеживаемый столбец, который постоянно увеличивается --last-value Sqoop импортирует строки, значение которых больше значения last-value --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password
Incremental Lastmodified import	$ sqoop import --connect \ --table \ --username \ --password \ -m 1 --incremental lastmodified \ --check-column \ --last-value	--incremental lastmodified Режим Lastmodified должен использоваться, когда записи таблицы могут быть обновлены, и каждое такое обновление будет устанавливать текущее значение отметки времени в столбец, который был изменен последним. --check-column отслеживаемый столбец, который

Тип импорта	Синтаксис	Аргумент
		постоянно обновляется --last-value. отметка времени --password Пароль подключающегося пользователя. Для ввода пароля используйте аргумент -P вместо --password

10.3 Экспорт данных

Экспорт осуществляется в два этапа. Первый шаг состоит в просмотре базы данных на наличие метаданных, использующихся на втором этапе передачи данных. Sqoop разделяет входной массив данных на части и затем использует индивидуальные задачи, чтобы загрузить части в базу данных. Каждая задача выполняет передачу в течение многих операций, чтобы обеспечить оптимальную пропускную способность и минимальное использование ресурсов.

Некоторые соединения поддерживают перемещение таблиц, что позволяет изолировать таблицы для работающих систем от возможных повреждений в случае сбоев работы по какой-либо причине.

Тип	Синтаксис	Аргумент

Sqoop-Export	$ sqoop export (generic args) (export args)
	$ sqoop-export (generic args) (export args)
	$ sqoop-export ---connect --username --password --export-dir
		Команда экспорта Sqoop экспортирует набор файлов из каталога HDFS обратно в таблицы СУБД. Целевая таблица уже должна существовать в базе данных.

10.4 Sqoop Job

Команда Sqoop job позволяет нам создавать работу. Job запоминает параметры, используемые для создания задания, поэтому их можно вызывать в любое время с теми же аргументами.

Тип	Синтаксис	Аргумент

Sqoop create Job	Общий синтаксис
	$ sqoop job (generic args) (job args) [-- [subtool name] (subtool args)]
	$ sqoop-job (generic args) (job args) [-- [subtool name] (subtool args)]
		--create Определяет новое задание с указанным идентификатором задания (именем). Фактическая команда импорта sqoop должна быть разделена знаком «-»
		--delete Удаляет сохраненное задание
		--exec Выполняет сохраненное задание

	Пример вызываемой команды: $ sqoop job --create -- import --connect --table	--show Показать конфигурацию сохранения задания --list Список всех сохраненных заданий
Sqoop Delete Job	$ sqoop job --detele	--delete Удаляет сохраненное задачу
Sqoop Show Job	$ sqoop job --show	--show Показать конфигурацию сохраненной задачи

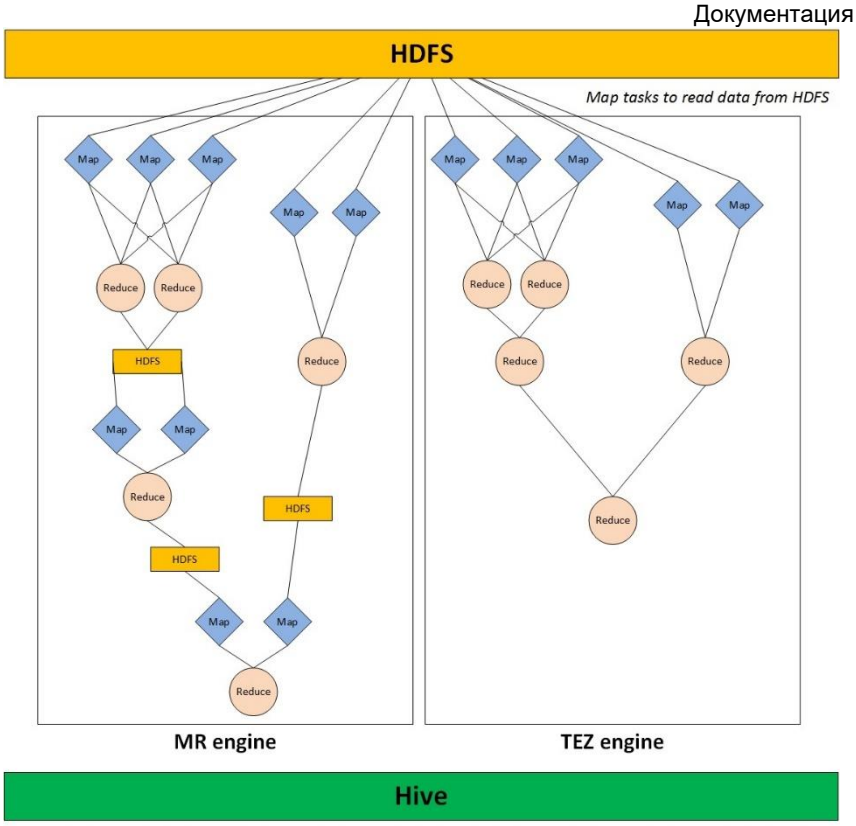
11 Компонент Tez

Tez - это платформа на основе YARN, которая обеспечивает высокопроизводительную обработку данных в Hadoop. Основная особенность Tez заключается в создании DAG (directed acyclic graph — направленного ациклического графа) и часто используется как альтернатива Hadoop MapReduce.

Суть DAG сводится к тому, что вместо пары Mapper-Reducer мы строим нецикличный направленный граф, каждая вершина которого является Mapper.Class'ом или Reduser.Class'ом, а ребра означают потоки данных / порядок выполнения. Кроме DAG, Tez предоставил еще несколько бонусов: ускоренный запуск job'ов (можно посылать DAG-job'ы через уже запущенный Tez-Engine), возможность удерживать ресурсы в памяти ноды между шагами, самостоятельно запускать распараллеливание и т. д. Естественно, вместе с Tez вышла и соответствующая надстройка над Hive.

11.1Сравнение производительности MapReduce и TEZ

Типичное задача для MapReduce состоит из следующих шагов:	**Типичное задача для Tez состоит из следующих шагов:**
1. Чтение данных из файла → первое обращение к диску; 2. Запуск Mappers; 3. Запись вывода Map -> второе обращение к диску; 4. Выполните перемешивание и сортировку -> чтение вывода Map, третье обращение к диску; 5. Запись в случайном порядке и сортировка -> запись отсортированных данных для редукторов -> четвертое обращение к диску; 6. Запуск редукторов, которые читают отсортированные данные -> пятое обращение к диску и вывод; 7. Запись вывода редукторов -> шестое обращение к диску;	1. Выполняет запрос (не нуждается в считывании данных с диска). 2. После того, как вы будете готовы выполнить некоторые вычисления (аналогичные действиям в Spark), получите данные с диска, выполните все шаги и произведите вывод.



Просматривается эффективность Tez, которую дает отказ от многократного обращения к диску. Промежуточные результаты Tez хранятся в памяти (не записываются на диски). Вдобавок ко всему есть векторизация (обработка пакета строк вместо одной строки за раз). Все это увеличивает эффективность времени запроса.

11.2 Сравнение Tez и Spark

Tez и Spark частично предлагают схожие функции. Они оптимизированы для работы с Hive и YARN и поддерживают ациклические ориентированные графы. Обе платформы могут обрабатывать различные типы данных и хранить результаты в памяти. Тем не менее, есть важные отличительные особенности.

Использование Tez особенно полезно, когда используется Hive. Tez оптимизирован для этой среды. Spark больше подходит для реализации распределенных алгоритмов машинного обучения, например, для создания заданий преобразования или для создания собственных языков обработки данных.

11.3 API для определения ориентированных ациклических графов (DAG)

API от Tez позволяет описывать обработку данных в форме DAG. Он состоит из трех основных компонентов:

- Ациклически направленный граф (DAG) - определяет актуальную задачу. DAG предназначен для каждого задания по обработке данных
- Вершина (Vertex) - определяет логику, ресурсы и среду для выполнения логики. Объект вершины создается для каждого шага обработки задания.
- Edge - определяет связь между производящей и потребляющей вершиной.

11.4 Интерфейсы взаимодействия

Интерфейс взаимодействия в Tez реализуется при использовании Hive селектов

12 Компонент YARN

Hadoop *YARN* связывает блок хранения Hadoop, то есть HDFS (распределенная файловая система Hadoop), с различными инструментами обработки. *Основная идея YARN* - облегчить MapReduce, взяв на себя ответственность за управление ресурсами и планирование работ. *YARN* начал предоставлять Hadoop возможность запускать задания, не связанные с MapReduce, в рамках Hadoop.

YARN позволяет использовать различные методы обработки данных, такие как обработка графиков, интерактивная обработка, потоковая обработка, а также пакетная обработка для запуска и обработки данных, хранящихся в HDFS. Таким образом, *YARN* открывает Hadoop для других типов распределенных приложений, помимо MapReduce. *YARN* выполняет операции в соответствии с требованиями пользователей, используя различные инструменты, такие как Spark для обработки в реальном времени, Hive для SQL, HBase для NoSQL и другие.

Помимо управления ресурсами, *YARN* также выполняет планирование работ. *YARN* выполняет все ваши действия по обработке, распределяя ресурсы и планируя задачи. Архитектура Hadoop *YARN* состоит из следующих основных компонентов:

- ResourceManager: запускается на главном демоне и управляет распределением ресурсов в кластере.
- Диспетчер узлов: они работают на подчиненных демонах и отвечают за выполнение задачи на каждом узле данных.
- Мастер приложений: управляет жизненным циклом пользовательских заданий и потребностями отдельных приложений в ресурсах. Он работает вместе с Node Manager и следит за выполнением задач.
- Контейнер: пакет ресурсов, включая ОЗУ, ЦП, сеть, жесткий диск и другое (на одном узле).

12.1 Архитектура YARN

На изображении ниже представлена архитектура YARN.

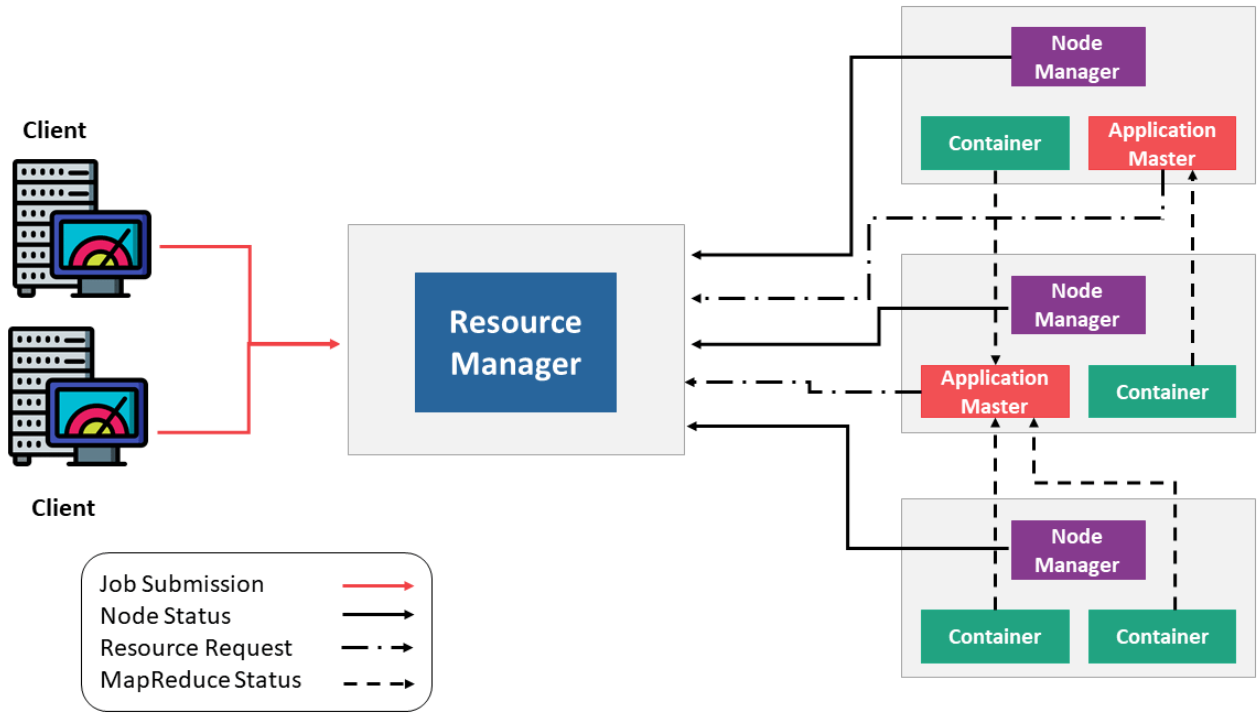


Рисунок 1. Архитектура YARN

12.1.1 Resource Manager (менеджер ресурсов)

Это основной элемент в распределении ресурсов.

- После получения запросов на обработку он передает части запросов соответствующим менеджерам узлов, где и происходит фактическая обработка.
- Он является арбитром ресурсов кластера и решает распределение доступных ресурсов для конкурирующих приложений.
- Оптимизирует использование кластера, например, постоянное использование всех ресурсов с учетом различных ограничений, таких как гарантии емкости, справедливость и SLA.
- Он состоит из двух основных компонентов: а) Планировщик; б) Диспетчер приложений.

12.1.1.1 **Планировщик**

Планировщик отвечает за распределение ресурсов между различными запущенными приложениями с учетом ограничений емкости, очередей и т. д.
В ResourceManager он называется чистым планировщиком, что означает, что он не выполняет никакого мониторинга или отслеживания состояния приложений.
В случае сбоя приложения или аппаратного сбоя Планировщик не гарантирует перезапуск невыполненных задач.
Выполняет планирование на основе требований приложений к ресурсам.
Он имеет подключаемый модуль политики, который отвечает за разделение ресурсов кластера между различными приложениями.
Существует два таких подключаемых модуля: Capacity Scheduler и Fair Scheduler, которые в настоящее время используются в качестве планировщиков в ResourceManager.

12.1.1.2 Менеджер приложений

Он отвечает за прием представленных вакансий.
Согласовывает первый контейнер с диспетчером ресурсов для выполнения конкретного приложения Application Master.
Управляет запуском мастеров приложений в кластере и предоставляет сервис для перезапуска контейнера мастера приложений в случае сбоя.

12.1.2 Node Manager (диспетчер узлов)

- Он заботится об отдельных узлах в кластере Hadoop и управляет пользовательскими заданиями и рабочим процессом на данном узле.
- Он регистрируется в диспетчере ресурсов и отправляет контрольные сообщения о состоянии работоспособности узла.
- Его основная цель - управлять контейнерами приложений, назначенными ему менеджером ресурсов.
- Он постоянно обновляется с помощью диспетчера ресурсов.
- Application Master запрашивает назначенный контейнер у Node Manager, отправляя ему контекст запуска контейнера (CLC), который включает в себя все, что необходимо приложению для запуска. Диспетчер узлов создает запрошенный процесс контейнера и запускает его.
- Контролирует использование ресурсов (память, ЦП) отдельных контейнеров.
- Выполняет управление журналом.
- Он также убивает контейнер в соответствии с указаниями диспетчера ресурсов.

12.1.3 Application Master (мастер приложений)

- Заявка - это отдельная работа, отправленная в фреймворк. Каждое такое приложение имеет связанный с ним уникальный мастер приложения, который представляет собой конкретную сущность платформы.
- Это процесс, который координирует выполнение приложения в кластере, а также управляет ошибками.
- Его задача - согласовывать ресурсы с диспетчером ресурсов и работать с диспетчером узлов для выполнения и мониторинга задач компонента.
- Он отвечает за согласование подходящих контейнеров ресурсов из ResourceManager, отслеживание их статуса и мониторинг прогресса.
- После запуска он периодически отправляет контрольные сигналы диспетчеру ресурсов, чтобы подтвердить свое работоспособность и обновить запись о своих потребностях в ресурсах.

12.1.4 Container (контейнер)

- Это набор физических ресурсов, таких как ОЗУ, ядра ЦП и диски на одном узле.
- Контейнеры YARN управляются контекстом запуска контейнера, который является жизненным циклом контейнера (CLC). Эта запись содержит карту переменных среды, зависимостей, хранящихся в удаленно доступном хранилище, токены безопасности, полезную нагрузку для служб Node Manager и команду, необходимую для создания процесса.
- Он предоставляет приложению права на использование определенного количества ресурсов (памяти, ЦП и т. Д.) На определенном хосте.

12.2 Подача заявки в Yarn

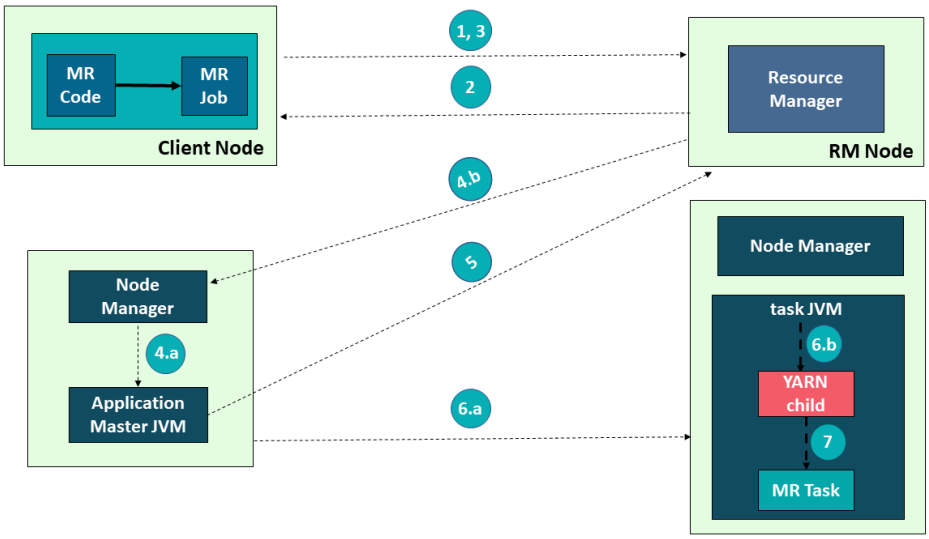


Рисунок 2. Подача заявки в Yarn

1. Отправить заявку
2. Получить идентификатор приложения
3. Контекст подачи заявки
4. а) Запуск запуска контейнера; б) Запустить Мастер приложений
5. Распределение ресурсов
6. а) Контейнер; б) Запуск
7. Выполнить

12.3 Рабочий процесс приложения в Hadoop YARN

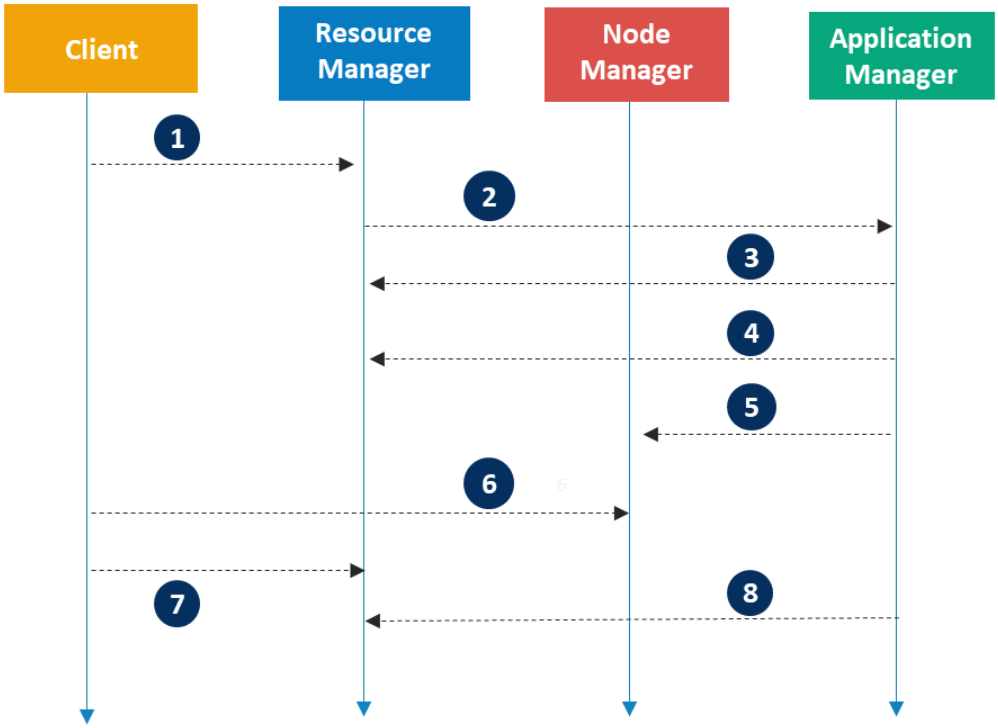


Рисунок 3. Рабочий процесс приложения

1. Клиент подает заявку
2. Resource Manager выделяет контейнер для запуска Application Manager
3. Application Manager регистрируется в Resource Manager
4. Диспетчер приложений запрашивает контейнеры у диспетчера ресурсов
5. Application Manager уведомляет Node Manager о запуске контейнеров
6. Код приложения выполняется в контейнере
7. Клиент связывается с диспетчером ресурсов / диспетчером приложений для отслеживания статуса приложения.
8. Application Manager отменяет регистрацию в Resource Manager

13 Компонент Zookeeper

ZooKeeper - это централизованная служба для поддержки информации о конфигурации, именования, обеспечения распределенной синхронизации и предоставления групповых служб. Все эти виды услуг используются в той или иной форме распределенными приложениями. Управление и координация службы, особенно в распределенной среде, является сложным процессом, ZooKeeper решает эту проблему благодаря своей простой архитектуре и API. Не беспокоясь о распределенной природе приложения, ZooKeeper позволяет разработчикам сосредоточиться на основной логике приложения. Для получения информации каждый клиентский компьютер связывается с одним из серверов. ZooKeeper следит за синхронизацией, а также за координацией всего кластера.

13.1Свойства Zookeeper:

- пространство ключей образует дерево (иерархию, подобную файловой системе);
- значения могут содержаться в любом узле иерархии, а не только в листьях (как если бы файлы одновременно были бы и каталогами), узел иерархии называется znode;
- между клиентом и сервером двунаправленная связь, следовательно, клиент может подписываться как изменение конкретного значения или части иерархии;
- возможно создать временную пару ключ/значение, которая существует, пока клиент, её создавший, подключен к кластеру;
- все данные должны помещаться в память;
- устойчивость к смерти некритического количества узлов кластера.

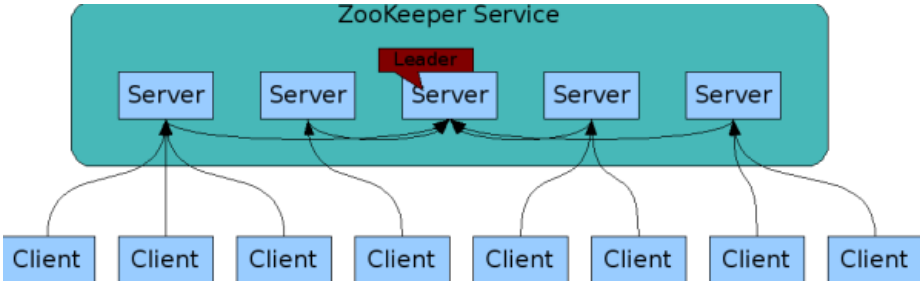


Рисунок 1. Архитектура сервиса Zookeeper

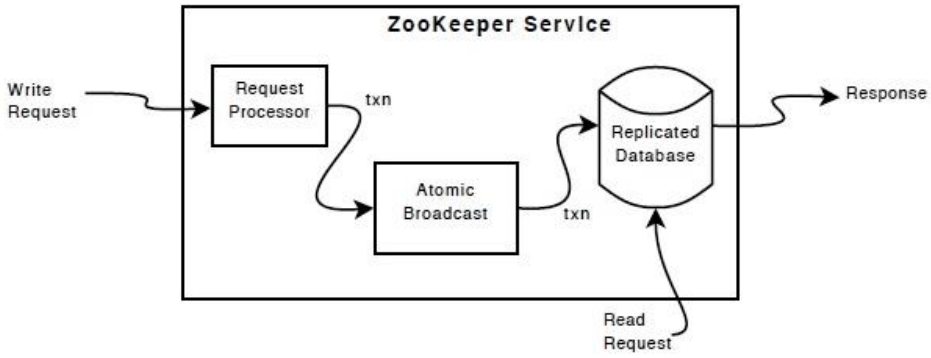


Рисунок 2. Компоненты Zookeper

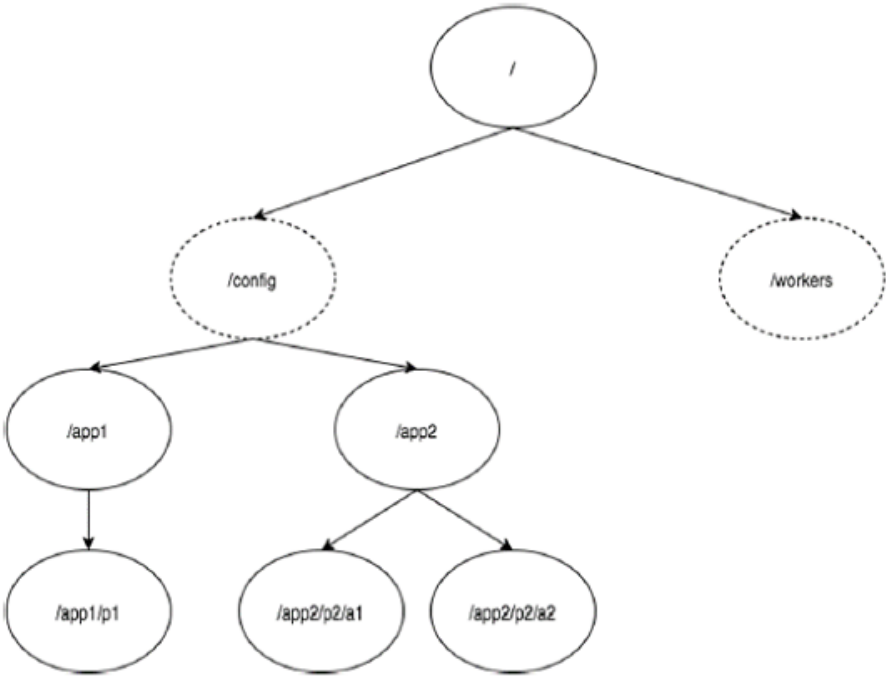


Рисунок 3 - Иерархическое пространство имен

13.2 Для чего нужен Zookeeper?

ZooKeeper - сервис, используемый кластером (группой узлов) для согласования друг с другом и поддержания общих данных с надежными методами синхронизации. ZooKeeper сам по себе является распределенным приложением, предоставляющим услуги для написания распределенных приложений. Основные услуги, предоставляемые ZooKeeper:

- **Служба имен** - идентификация узлов в кластере по имени. Очень похоже на DNS, но только для узлов.
- **Управление конфигурацией** - последняя и актуальная информация о системе новому подключаемому узлу.
- **Управление кластером** - добавление/удаление узла в\из кластера и статус узла в реальном времени.
- **Выбор лидера** - выбор узла в качестве лидера в целях согласования.
- **Служба блокировки и синхронизации** - блокировка данных в момент их изменения. Механизм помогает пользователю в автоматическом восстановлении в случае ошибки при подключении других распределенных приложений, таких как HBase.
- **Высоконадёжная регистрация данных** - доступ к данным даже в случае, когда один или несколько узлов упали.

13.3 Поддерживаемые операции

13.3.1 CLI ZooKeeper

Интерфейс командной строки ZooKeeper (CLI) используется для взаимодействия с ZooKeeper в целях разработки. Это полезно для отладки и работы с различными опциями.

Чтобы выполнить операции CLI ZooKeeper, сначала включите сервер ZooKeeper (*"/usr/hdp/current/zookeeper-server/bin/zkServer.sh start"*), а затем клиент ZooKeeper ("*/usr/hdp/current/zookeeper-server/bin/zkCli.sh* как только клиент запустится, вы сможете выполнить следующую основные операции:

Операция	Описание	Синтаксис
create	Создает новый znode (пару ключ/значение) и возвращает ключ. Кажется странным, что возвращается ключ, если он указывается как аргумент, но дело в том, что ZooKeeper в качестве ключа можно указать префикс и сказать, что znode последовательный, тогда к префиксу добавится выровненное число, и результат будет использоваться в качестве ключа. Гарантируется, что создавая последовательные znode с одним и тем же префиксом, ключи будут образовывать возрастающую (в лексико-графическом смысле) последовательность. Аргумент *flag* указывает, будет ли созданный znode эфемерным, постоянным или последовательным. Эфемерные Znode(флаг: e) будет автоматически удален по истечении сеанса или при отключении клиента. Последовательные znodes гарантируйте, что путь znode будет уникальным. Если флаги не указаны, то znode считается *постоянным*.	create [-flag] /path /data
create "children"	Создание дочерних элементов аналогично созданию новых znodes. Единственная разница заключается в том, что путь дочернего znode будет иметь и родительский путь.	create /parent/path/subnode/path /data
get	Он возвращает связанные данные znode и метаданные указанного znode. Вы получите информацию, например, когда данные были изменены в последний раз, где они были изменены, и информацию о данных. Этот CLI также используется для назначения часов, чтобы показать уведомление о данных.	get /path
set data	Установить данные указанного znode.	set /path /data
ls	Эта команда используется для перечисления и	ls /path

Операция	Описание	Синтаксис
	отображения *дочерних* элементов znode.	
check status	Статус описывает метаданные указанного znode. Он содержит такие сведения, как метка времени, номер версии, ACL, длина данных и дочерний znode.	stat /path
remove a znode	Удаляет указанный znode и рекурсивно все его дочерние элементы. Это произойдет только в том случае, если такой znode доступен.	rmr /path
watch	Наблюдатель показывает уведомление при изменении указанных данных znode или дочерних данных znode. Можно установить *watch* только в команда get. Вывод аналогичен обычной команде get, но она будет ждать изменений znode в фоновом режиме.	get /path watch 1

13.3.2 API ZooKeeper

ZooKeeper имеет официальную привязку API для Java и C. сообщество ZooKeeper предоставляет неофициальный API для большинства языков (.NET, python и т. д.). Используя API ZooKeeper, приложение может подключаться, взаимодействовать, манипулировать данными, координировать и, наконец, отключаться от ZooKeeper.

ZooKeeper API имеет богатый набор функций, чтобы получить всю функциональность ZooKeeper простым и безопасным способом. ZooKeeper API предоставляет как синхронные, так и асинхронные методы.

Центральная часть API **ZooKeeper - это класс** ZooKeeper. Он предоставляет опции для подключения ZooKeeper в своем конструкторе и имеет следующие методы:

Операция	Описание	Синтаксис
Подключение к Zookeeper	Класс ZooKeeper обеспечивает функциональность соединения через свой конструктор.	Подпись конструктора выглядит следующим образом − ZooKeeper(String connectionString, int sessionTimeout, Watcher watcher) Где, **connectionString** – строка подключения к сеансу. **sessionTimeout** - тайм-аут сеанса в миллисекундах.

Операция	Описание	Синтаксис
		watcher - объект, реализующий интерфейс "наблюдатель". ZooKeeper возвращает состояние соединения через объект watcher.
Создание Znode	Метод **Create** позволяет создать новый znode в ZooKeeper. После компиляции и выполнения приложения в ZooKeeper будет создан znode с указанными данными. Можно проверить это с помощью *get команды* CLI ZooKeeper **zk Cli.sh**.	Сигнатура метода **Create** выглядит следующим образом - create (String path, byte [] data, List<ACL> acl, CreateMode creat Где, **path**− Путь Znode. Например, /myapp1, / myapp2, /myapp1/mydata1, myapp2/mydata1/myanothersubdata **data**− данные для хранения в указанном пути znode **acl** - список управления доступом создаваемого узла. ZooKeeper API предоставляет статический интерфейс ZooDefs. Ids чтобы получить некоторые из основных списков acl. Идентификационная карточка. OPEN_ACL_UNSAFE возвращает список acl для открытых znodes. **createMode** - тип узла, либо эфемерный, последовательный, либо оба. Это Ан **перечисление**.
Проверка существования Znode	Метод **Exists** позволяет проверить существование znode. Он возвращает метаданные znode, если указанный znode существует.	Сигнатура метода **Exists** выглядит следующим образом - exists(String path, boolean watcher) Где, **path** - Путь Znode **watcher** - логическое значение для указания того, следует ли смотреть указанный znode или нет
Метод getData	Метод **getData** позволяет получить данные, вложенные в указанный	Сигнатура метода **getData** выглядит следующим образом - getData(String path, Watcher watcher, Stat stat) Где, **path** - Путь Znode. **watcher** − Функция обратного вызова типа **Watcher**. ZooKeeper будет уведомлять через обратный вызов Watcher, когда

Операция	Описание	Синтаксис
	znode и его статус. Приложение будет ждать дальнейшего уведомления от ZooKeeper. Измените данные указанного znode с помощью команды *set* в ZooKeeper CLI **zkCli.sh**.	данные указанного znode изменятся. Это одноразовое уведомление. **stat** - Возвращает метаданные znode.
Метод setData	Метод **setData** для изменения данных, вложенных в указанный znode. Можно проверить с помощью *get команды в* CLI ZooKeeper, **zkCli.sh**.	Сигнатура метода **setData** выглядит следующим образом - setData (String path, byte[] data, int version) Где, **path** - Путь Znode **data** - данные для хранения в указанном пути znode. **version** - Текущая версия znode. ZooKeeper обновляет номер версии znode всякий раз, когда данные изменяются.
Метод getChildren	Метод **getChildren** позволяет получить все подузлы конкретного znode.	Сигнатура метода **getChildren** выглядит следующим образом - getChildren(String path, Watcher watcher) Где, **path** - Путь Znode. **watcher** − Функция обратного вызова типа "Watcher". ZooKeeper будет уведомлять, когда указанный znode будет удален или ребенок под znode будет создан / удален. Это одноразовое уведомление.
Удалить Znode	Метод **delete** для удаления указанного znode.	Сигнатура метода **Delete** выглядит следующим образом - delete (String path, int version) Где, **path** - Путь Znode. **version** - Текущая версия znode.

Операция	Описание	Синтаксис

Отключения от сервера zookeeper	Закрыть соединение	Метод **Close** выглядит следующим образом -
		public void close() throws InterruptedException {
		zoo.close();
		}