Дата последней сборки документа:
24 мая 2021 г. в 22:15

Заметки по большим данным и облачным технологиям

Подвойский А.О.

Здесь приводятся заметки по некоторым вопросам, касающимся больших данных, облачных технологий, машинного обучения, анализа данных, программирования на языках `Python`, `R` и прочим сопряженным вопросам так или иначе, затрагивающим работу с данными.

Содержание

1. Основные термины и определения

Витрина данных (Data Mart) – срез хранилища данных, представляющий собой массив тематической, узконаправленной информации, ориентированный, например, на пользователей одной рабочей группы или департамента.

2. Установка Hadoop на MacOS X

Установить `hadoop` на MacOS X можно с помощью менеджера пакетов `brew`

```
brew install hadoop
```

После установки Hadoop остается внести несколько изменений в конфигурационные файлы и настроить переменные окружения.

В файле, расположенном

по пути `/usr/local/Cellar/hadoop/3.3.0/libexec/etc/hadoop/hadoop-env.sh` изменить путь до JAVA_HOME. Узнать домашнюю директорию java можно так

```
/usr/libexec/java_home # /Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home
```

/usr/local/Cellar/hadoop/3.3.0/libexec/etc/hadoop/hadoop-env.sh

```
# The java implementation to use...
# variable ...
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home
```

Далее в файле `core-site.xml` нужно внести следующие изменения

core-site.xml

```
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/Cellar/hadoop/hdfs/tmp</value>
    <description>A base for other temporary directories</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:8020</value>
  </property>
</configuration>
```

Теперь внесем изменения в файл `mapred-site.xml`

mapred-site.xml

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>
</configuration>
```

И, наконец, внесем изменения в файл

hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Проверим включен ли ssh

```
ssh localhost
```

Если эта команда вызывает ошибку, то следует настроить ssh следующим образом (нужно сообщить системе о ключах, которые мы собираемся использовать)

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

Последним шагом требуется отформатировать HDFS

```
cd /usr/local/opt/hadoop
hdfs namenode -format
```

Вывод последней команды должен содержать следующую строку

```
...
2021-04-02 14:58:30,137 INFO common.Storage: Storage directory /usr/local/Cellar/hadoop/hdfs/tmp
    /dfs/name has been successfully formatted.
```

Теперь можно задать псевдоними для команд запуска и остановки hadoop сервисов

~/.zshrc

```
alias hstart="/usr/local/Cellar/hadoop/3.3.0/sbin/start-all.sh"
alias hstop="/usr/local/Cellar/hadoop/3.3.0/sbin/stop-all.sh"

source ~/.zshrc
```

Теперь можно запустить Hadoop

```
hstart
```

и проверить запущенные сервисы

```
jps
# ----
47728 ResourceManager
47538 SecondaryNameNode
47826 NodeManager
47908 Jps
47302 NameNode
47401 DataNode
```

Получить доступ к Hadoop можно через web-интерфейс

- http://localhost:9870: менеджер ресурсов,
- http://localhost:8088: трекер заданий,
- http://localhost:8042: информация по узлам.

3. Как и зачем разворачивать приложение на Apache Spark в Kubernetes

Оригинальная статья: https://habr.com/ru/company/mailru/blog/549052/

Для частого запуска Spark-приложений, особенно в промышленной эксплуатации, необходимо максимально упростить процесс запуска задач, а также уметь гибко настраивать их конфигурации. В этом может помочь Kubernetes: он позволяет решать задачи изоляции рабочих сред, гибкого управления ресурсами и масштабирования.

Почему стоит запускать Spark именно в Kubernetes Основные преимущества, которые дает запуск Spark внутри Kubernetes:

- *Изоляция сред.* В традиционном развертывании в Hadoop-кластере есть проблема версионности Spark. Если необходимо перейти на новую версию Spark, то это добавляет проблем командам администрирования. Администраторам нужно организовать бесшовный апргрейд кластера, а дата-инженерам нужно проверить все свои пайплайны и убедиться, что они будут правильно работать в новой версии. Используя Spark в Kubernetes, вы решаете эту проблему. Потому что каждый член команды может создать себе *отдельное окружение*, которое будет работать в независимом контейнере, упаковать в него Spark-приложение со всем кодом и любыми зависимостями. Можно использовать любую версию Spark, любые зависимости и никому не мешать.
- *Управление ресурсами.* Kubernetes позволяет накладывать ограничение ресурсов на разные приложения и разные типы пайплайнов, например, используя Namespace.
- *Гибкое масштабирование.* Kubernetes в облаке умеет задействовать огромное количество ресурсов на то время, когда они реально используются. Допустим, ваше приложение обычно использует 10 ядер процессора, но иногда для сложной обработки ему нужно 500 или 1000 ядер. В этом случае вам нужно подключить функцию автомасштабирования кластера. Тогда, если ваше приложение запросит 500 ядер, облако их выделит. А когда приложение перестанет генерировать такую нагрузку, ненужные ресурсы автоматически вернутся в облако.
- *Повышение эффективности использования ресурсов.* Если у вас уже есть рабочий кластер Kubernetes, то его можно использовать, чтобы поднять Spark или любое другое приложение. При этом не придется создавать новый кластер.

Способы запуска Spark в Kubernetes Spark можно запускать в Kubernetes, начиная с версии 2.3.

Spark можно запускать в Kubernetes двумя способами:

- *Spark-submit*: это Spark-native подход. Вы используете spark-submit, задаете, как обычно, все параметры, а в качестве менеджера ресурсов указываете Kubernetes. В этом случае в момент spark-submit внутри Kubernetes создается *под*, на котором сначала разместиться Driver. Далее этот Driver будет напрямую взаимодействовать с API Kubernetes и создавать Executor по указанным параметрам. При этом Kubernetes *не будет знать*, что внутри него работает именно Spark, для него это будет просто еще одно приложение.
- *Kubernetes Operator for Spark*: это Kubernetes-native путь. В этом случае Kubernetes понимает, что внутри него работает именно Spark. При это вы получаете более удобный доступ к логам, текущему состоянию Job и статусу приложения. Mail Solution Cloud рекомендует использовать именно этот подход.

Производительность и особенность работы Spark в Kubernetes Для управления ресурсами и планирования приложения в Spark часто используют Yarn. Долгое время Spark в Kubernetes существенно отставал по скорости и эффективности от Spark в Yarn. Но на текущий момент производительность практически выровнялась. Yarn остается быстрее в среднем на 4-5%.

ВАЖНО: здесь нужно отметить, что для тестирования использовались локальные SSD-диски. Производительность Spark *в облачном Kubernetes* будет *ниже* из-за того, что мы используем S3 и *доступ к данным осуществляется по сети*. S3-хранилище позволяет разделить storage и compute слои, а также неограничено масштабируется под любой объем данных. Но доступ к ним будет медленее из-за сетевой задержки, тогда как в *классическом Hadoop-кластере* приложения размещаются *рядом с данными*, поэтому там задержки на передачу данных минимальны.

ВАЖНО: Другой тонкий момент заключается в том, что Spark в процессе работы активно использует диски для сохранения промежуточного состояния – spill-файлов. Тип используемого диска существенно влияет на производительность. В любом случае при прочих равных *Spark в Kubernetes* будет работать *медленнее*, чем в классическом *Hadoop-кластере*. Однако, если Hadoop-кластер перегружен, то Kubernetes может обогнать его. Облако позволяет получить огромное количество ресурсов и быстро обработать данные. А загруженный Hadoop-кластер будет долго обрабатывать данные, несмотря на то, что задержка на передачу данных минимальна.

Для увеличения производительности в качестве диска для spill-файлов можно подключить оперативную память. При этом нужно понимать, что если spill-файлы будут слишком большими, то Spark может упасть. Поэтому нужно знать, как работает ваше приложение, какие оно обрабатывает данные и какие совершает операции.

Еще один момент: Kubernetes потребляет часть ресурсов ноды для своих служебных целей. Поэтому, если создать ноду, например, с 4 ядрами и 16 Гб оперативной памяти, то Executor не сможет использовать все эти ресурсы. Best practice – выделять для Executor 75-85% от объема ресурсов, либо смотреть по конкретной ситуации.

Еще стоит упомянуть о Dynamic Allocation. В Hadoop он работает за счет того, что там есть External Shuffle Service. Эти промежуточные файлы в *Hadoop* сохраняются *не на самих Executor*. А в *Kubernetes* они на сохраняются *на Executor*, и мы *не можем уничтожить* те Executor, которые содержат эти shuffle-файлы. То есть в Kubernetes можно активировать Dynamic Allocation, но он будет не такой не такой эффективный, как в Hadoop.

Инструкции по установке Spark в Kubernetes можно найти здесь https://github.com/stockblog/webinar_spark_k8s.

4. Компоненты экосистемы Hadoop

Avro – система сериализации для выполнения эффективных межъязыковых вызовов RPC и долгосрочного хранения данных.

MapReduce – модель распределенной обработки данных и исполнительная среда, работающая на больших кластерах типовых машин.

HDFS – распределенная файловая система, работающая на больших кластерах стандартных машин.

Hive – распределенное хранилище данных. В принципе Hive можно называть платформой пакетного обработки или СУБД. Hive управляет данными, хранимыми в HDFS, и предоставля-

ет язык запросов на базе SQL (которые преобразуются ядром времени выполнения в задания MapReduce) для работы с этими данными.

HBase – распределенная нереляционная столбцово-ориентированная *база данных*, построенная на основе HDFS. HBase использует HDFS для организации хранения данных и поддерживает как пакетные вычисления с использованием MapReduce, так и точечные запросы (произвольное чтение данных).

Sqoop – инструмент эффективной массовой пересылки данных между структурированными хранилищами (такими, как реляционные базы данных) и HDFS.

Oozie – сервис запуска и планирования заданий Hadoop (включая задания MapReduce, Pig, Hive и Sqoop jobs).

5. Apache Drill

6. Tarantool

В силу архитектурных особенностей Тарантул позволяет обрабатывать большие объемых данных, поэтому эта СУБД широко применяется в различных BigData-проектах.

Благодаря высокой скорости обработки данных, типовыми сценариями применения Tarantool в BigData-считаются следующие:

- ускорение распределенных вычислений, в т.ч. на Apache Hadoop и Spark, а также выполнение аналитических SQL-запросов с большими данными в MPP-СУБД, таких как Arenadata DB на базе Greenplum,
- гибридная транзакционно-аналитическая обработка больших данных,
- оперативное кэширование для систем потоковой передачи и шин данных.

Кроме всего прочего Tarantool может использоваться в системах интернета вещей, в т.ч. промышленного интернета вещей. Tarantool IIoT поддерживает основные протоколы работы с датчиками (MQTT и MRAA), которые генерируют большие объемы данных для обработки в реальном времени. Еще эта версия Tarantool позволяет создавать сценарии для описания процессов получения показателей с промышленных устройств, их обработки, сохранения и передачи.

7. Apache HBase

HBase – распределенная нереляционная (столбцово-ориентирования) база данных формата «ключ-значение».

7.1. Установка и запуск

Подробности, связанные с установкой различных режимах (автономном, распределенном и т.д.) можно узнать на странице https://hbase.apache.org/book.html.

Скачать tar-архив можно здесь https://www.apache.org/dyn/closer.lua/hbase/2.4.0/hbase-2.4.0-bin.tar.gz

```
curl -O https://apache-mirror.rbc.ru/pub/apache/hbase/2.4.0/hbase-2.4.0-bin.tar.gz
```

Теперь следует распоковать архив

```
tar -xvzf hbase-2.4.0...
```

перейти в директорию `hbase-2.4.0` и задать путь до java в файле `hbase-env.sh`, раскоментировав нужную строку

conf/hbase-env.sh

```
export JAVA_HOME=/usr/local/Cellar/openjdk/15.0.1
```

В конфигурационном файле командаой оболочки удобно задать переменные окружения для Java и HBase

~/.zshrc

```
# for HBase
export JAVA_HOME="/usr/local/Cellar/openjdk/15.0.1"
export PATH="${PATH}:/Users/leor.finkelberg/hbase/hbase-2.4.0/bin"
```

Диреткорию размещения java на MacOS X следует искать с помощью менеджера пакетов `brew`

```
brew list java # /usr/local/Cellar/openjdk/15.0.1/bin/java
```

ВАЖНО: обновить java, можно скачав соответствующую версию с ресурса https://www.oracle.com/java/technologies/javase-jdk15-downloads.html.

Запустить HBase можно с помощью сценария командной оболочки из `bin/`

```
start-hbase.sh
```

Подключиться к запущенному экземпляру можно так

```
hbase shell
```

Для того чтобы убедиться, что процесс HMaster запущен можно воспользоваться утилитой `jps`.

Бывает удобно следить за работой приложения с помощью Web-интерфейса, доступного на http://localhost:16010.

Закончить сессию можно с помощью команды `quit`. Затем нужно остановить HBase

```
stop-hbase.sh
```

8. Apache Cassandra

9. Сравнение ClickHouse, Druid и Pinot

Полезная ссылка: статья «Сравнение открытых OLAP-систем BigData: ClickHouse, Druid и Pinot» https://habr.com/ru/company/oleg-bunin/blog/351308/.

Выжимка из этой статьи. Druid или Pinot имеет смысл применять, если

- в организации есть эксперты по Java,
- есть большой кластер,
- много таблиц,
- работать приходится с несколькими несвязанными наборами данных,
- таблицы и наборы данных периодически появляются в кластере и удаляются из него,
- таблицы значительно растут и сжимаются,
- запросы разнородные,
- запросы часто затрагивают данные, расположенные во всем кластере,

- кластер развернут в облаке (в случае ClickHouse *облако не используется*; кластер должен быть развернут на специфической конфигурации физических серверов),
- кластеры Hadoop или Spark уже существуют и могут быть использованы.

ClickHouse, Druid и Pinot – три открытых *хранилища данных*, которые позволяют выполнять *аналитические запросы* на больших объемах данных с интерактивными задержками.

Рассматриваемые системы выполняют запросы быстрее, чем системы BigData из семейства класса SQL-on-Hadoop: Hive, Impala, Presto и Spark, даже когда последние доступ к данным, хранящимся в колоночном формате – к примеру, Parquet или Kudu. Это происходит потому, что в ClickHouse, Druid и Pinot:

- имеется свой собственный формат для хранения данных с индексами, и они тесно интегрированы с движками обработки запросов,
- Данные распределены относительно «статично» между узлами, и при распределеном выполнении запроса это можно использовать. Обратная сторона медали при этом в том, что ClickHouse, Druid и Pinot **не поддерживают запросы, которые требуют перемещения большого количества данных между узлами** – к примеру, join между двумя большими таблицами.

ClickHouse, Druid и Pinot **не поддерживают точечные обновления и удаления**, в противоположность колоночным системам вроде Kudu, InfluxDB и Vertica. Это дает ClickHouse, Druid и Pinot возможность производить более эффективное колоночное сжатие и более агрессивные индексы, что означает **большую эффективность использования ресурсов** и быстрое выполнение запросов.

Все три системы поддерживают потоковое поглощение данных из Kafka. Druid и Pinot поддерживают потоковую передачу данных, стриминг в лямбда-стиле и пакетное поглощение одних и тех же данных.

Архитектуры Druid и Pinot почти что идентичны друг другу, в то время как ClickHouse стоит слегка в стороне. В Druid и Pinot, все данные в каждой «таблице» (как бы они не назывались в терминологии этих систем) разбиваются на указанное количество частей. По временной оси, данные обычно разделены с заданным интервалом. Затем эти части данных запечатываются индивидуально в самостоятельные автономные сущности, называемые сегментами. Каждый сегмент включает в себя метаданные таблицы, сжатые столбчатые данные и индексы.

Сегменты хранятся в файловой системе хранилища «глубокого хранения» (например, HDFS) и могут быть загружены на узлы обработки запросов, но последние не отвечают за устойчивость сегментов, поэтому узлы обработки запросов могут быть заменены относительно свободно. **Сегменты не привязаны жестко к конкретным узлам** и могут быть загружены на те или иные узлы. Специальный выделенный сервер (которые называется координатором в Druid и контролером в Pinot) отвечает за присвоение сегментов узлам, и перемещению сегментов между узлами, если потребуется.

ВАЖНО: все три системы имеют *статическое распределение данных между узлами*, поскольку загрузки сегментов и их перемещения в Druid – и видимо в Pinot – являются дорогими операциями и потому не выполняются для каждой отдельной очереди, а происходят обычно раз в несколько минут/часов/дней.

Метаданные сегментов хранятся в ZooKeeper – напрямую в случае Druid, и при помощи фреймворка Helix в Pinot.В Druid метаданные также хранятся в базе SQL.

ClickHouse больше напоминает «традиционные» базы данных вроде PostgreSQL. ClickHouse можно установить на один узел. При малых масштабах (менее 1 Тб памяти, менее 100 ядер CPU),

ClickHouse выглядит гораздо более интересным вариантом, чем Druid или Pinot – в силу того, что ClickHouse проще и имеет меньше движущихся частей и сервисов.

Druid и Pinot больше напоминают другие системы Big Data из экосистемы Hadoop. Они сохраняют свои «самоуправляемые» свойства даже на очень больших масштабах (более 500 узлов), в то время как ClickHouse потребует для этого достаточно много работы профессиональных SRE. Кроме того, Druid и *Pinot* занимают выигрышную позицию в плане оптимизации инфраструктурной стоимости больших кластеров, и *лучше подходят для облачных окружений*, чем ClickHouse.

10. Apache Pinot

Apache Pinot https://pinot.apache.org/ – распределенное масштабируемое *OLAP-хранилище данных* с низкой задержкой, работающее практически в реальном времени. Или другими словами, Pinot – *хранилище*, которое позволяет выполнять *аналитические запросы* на больших данных с *интерактивными задержками*. Оно может принимать данные из пакетных источников (HDFS, Amazon S3, Azure ADLS, Google Cloud Storage), а также из потоковых источников данных, таких как Apache Kafka.

Этот инструмент выполнения OLAP-запросов с малой задержкой особенно полезен там, где требуется быстрая аналитика и агрегирование неизменяемых данных, в т.ч. в реальном времени. Еще Pinot можно использовать для выполнения типичных аналитических операций с OLAP-кубами, включая детализацию и свертку крупномасштабных многомерных данных. Для визуализации данных к Pinot можно подключать различные BI-инструменты: Superset, Tableau, PowerBI, а также запускать ML-алгоритмы для обнаружения аномалий в хранящихся данных.

Pinot предоставляет разные возможности для разных BigData-специалистов:

- для аналитиков и дата-инжинеров это масштабируемая платформа данных для бизнес-аналитики, которая объединяет технологии BigData с традиционной ролью хранилища данных, облегчая их анализ и генерацию отчености,
- разработчики приложений могут рассматривать Pinot как неизменяемое совокупное хранилище, которое получает события из источников потоковых данных, таких как Kafka, и делает их доступными для SQL-запросов. Это устраняет недостатки микросервисной архитектуры, когда каждое приложение должно предоставлять собственное хранилище данных вместо совместного использования одного OLTP-хранилища для чтения и записи.

Еще из наиболее важных на практике функций *Apache Pinot* стоит отметить следующие:

- колоночный формат хранения данных с различными схемами сжатия,
- разные варианты индексирования (отсортированный, растровый или инвертированный индекс),
- оптимизация плана запроса на основе метаданных запроса и сегмента,
- *SQL-подобный язык*, который поддерживает *выбор*, *агрегацию*, *фильтрацию*, *группировку*, сортировку и отдельные запросы к данным,
- поддержка многозначных полей.

Apache Pinot активно используется для оперативной аналитики больших данных в реальном времени.

11. Apache Kylin

12. Apache Impala

Apache Impala – это массово-параллельный механизм интерактивного выполнения SQL-запросов к данным, хранящимся в HDFS, Kudu, HBase, Microsoft Azure Data Lake Storage (ADLS) или Amazon Simple Storage (S3). Также Impala называют MPP-движком или *распределенной СУБД*.

Impala использует тот же синтаксис SQL (HiveQL), драйвер ODBC и пользовательский интерфейс как и Apache Hive.

Чтобы избежать задержки, Impala обходит MapReduce для прямого доступа к данным с помощью специализированного механизма распределенных запросов. В результате производительность на порядок выше чем у Hive (платформа Hive построена на базе MapReduce).

Для данных IoT и связанных с ними сценариях, Impala вместе со streaming решениями, такими как NiFi, Kafka или Spark Streaming, и соответствующими хранилищами данных, такими как Kudu, может обеспечить непрерывную конвейерную обработку со времени задержки менее чем 10 секунд. Благодаря встроенным функциям чтения/записи на S3, ADLS, HDFS, Hive, HBase и многим другим, Impala является превосходным SQL-движком для использования при запуске кластера до 1000 узлов, и более 100 триллионов строк в таблицах или датасетах размером в 50 BP и более.

ВАЖНО: Если сравнивать Hive и Impala в смысле контекста использования, то Hive более адаптирована для промышленной эксплуатации в условиях высоких нагрузок, когда допустима некоторая временная задержка (latency). Поэтому Hive в большей степени востребована у инженеров больших данных в рамках построения сложных ETL-конвейеров. В тоже время быстрая и безопасная, но не слишком надежная Impala больше подходит для менее масштабных проектов и пользуется популярностью у аналитиков данных. То есть эти BigData-инструменты не конкурируют, а дополняют друг друга.

13. Приемы работы с `hadoop fs`

ВАЖНО: при работе с локальной файловой системой, HDFS, WebHDFS, S3 FS и т.д. следует пользоваться `hadoop fs`, а при работе с HDFS – `hdfs dfs`[1].

Вывести наполнение директории

```
hdfs dfs -ls / # наполнение корня директории
hdfs dfs -ls -d /hadoop
hdfs dfs -ls -h /data
hdfs dfs -ls -R /hadoop # вывести рекурсивно список всех файлов в поддиректориях hadoop
hdfs dfs -ls /hadoop/dat* # вывести список всех файлов по шаблону
```

Вывести информацию об используемом дисковом пространстве

```
hdfs dfs -df hdfs:/
```

Вывести информацию о здоровье файловой системы Hadoop

```
hdfs fsck /
```

Создать директорию на HDFS

[1] `hdfs dfs` используется вместо `hadoop dfs`, считающейся устаревшей

```
hdfs dfs -mkdir /hadoop
```

Скопировать файл, расположенный в локальной файловой системе, на HDFS

```
hdfs dfs -copyFromLocal ~/python_scripts /hadoop
```

Скопировать файл, расположенный на HDFS, в локальную файловую систему

```
hdfs dfs -copyToLocal /hadoop/work/DBSCAN_test.py ~/GARBAGE
```

Задать значение фактора реплицирования (по умолчанию равен 3)

```
hdfs dfs -setrep -w 2 /usr/sample
```

Скопировать директорию с одного узла кластера на другой

```
hdfs dfs -distcp hdfs://namenodeA/apache_hadoop hdfs://namenodeB/hadoop
```

Вывести информацию по статистике файлов/директорий; здесь `%b` – блоки

```
hdfs dfs -stat "%F %u:%g %b %y %n" /hadoop/work/DBSCAN_test.py
# regular file leor.finkelberg:supergroup 597 2021-04-02 22:01:50 DBSCAN_test.py
```

14. Apache Hive

Apache Hive представляет собой *систему управления базами данных* (СУБД), построенную поверх Hadoop. В официальной документации платформа описана как *хранилище данных* (DWH). Apache Hive позволяет работать с данными, хранящимися в HDFS или HBase, с помощью языка HiveQL, очень близкому к стандартному SQL.

ВАЖНО: Apache Hive не является реляционной базой данных, не поддерживает онлайн-транзакции (OLTP), в том числе обновления отдельных строк таблицы.

Apache Hive более удобна в использовании по сравнению с Hadoop и Pig, поскольку не требует изучения специального доменно-специфичного языка (Pig Lating) или Java. А это значит, что платформа Hive является универсальной, то есть может быть задействована как

- система *пакетного* анализа,
- *нереляционное хранилище данных*,
- часть процесса ETL.

Hive и другие фреймворки, построенные *на базе MapReduce*, лучше всего подходят для *длительных пакетных заданий*, например, для пакетной обработки заданий типа ETL.

Live Long And Process, также известная как LLAP, является механизмом выполнения под управлением Hive, который поддерживает длительные процессы, используя один и те же ресурсы для кэширования и обработки.

При работе с Hive можно выделить следующие объекты

- Базы данных,
- Таблицы,
- Партиции,
- Бакеты.

База данных в Hive представляет собой то же, что и база данных в реляционных СУБД. База данных в Hive – это пространство имен, содержащее таблицы. Команда создания базы данных выглядит следующим образом

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Здесь DATABASE и SCHEMA одно и то же.

Пример создания базы данных

```
CREATE DATABASE userdb;
```

Для переключения на соответствующую базу данных используется команда USE

```
USE userdb;
```

В целом таблицы в Hive представляют собой то же, что и таблицы в классических реляционных базах данных, но есть и отличия. Основное отличие таблиц Hive состоит в том, что они хранятся в виде *обычных файлов* на HDFS. Это могут быть обычные текстовые csv-файлы, бинарные sequence-файлы, более сложные колоночные parquet-файлы и т.д.

Пример создания таблицы

```
CREATE TABLE IF NOT EXISTS employee (
  eid INT,
  name STRING,
  salary STRING,
  destination STRING
)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Здесь создается таблица, данные которой будут храниться в виде обычных csv-файлов. Столбцы разделены символом табуляции. После этого можно загрузить данные в таблицу.

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

Для создания партиционированной таблицы необходимо указать по каким столбцам будет выполняться партиционирование

```
CREATE TABLE IF NOT EXISTS employee_partitioned (
  eid INT,
  name STRING,
  salary STRING,
  destination STRING
)
COMMENT 'Employee details'
PARTITIONED BY (birth_year INT, birth_month STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

При заливке данных в такую таблицу необходимо явно указать, в какой партиции требуется разместить данные

```
LOAD DATA PATH '/user/root/sample.txt' OVERWRITE
INTO TABLE employee_partitioned
PARTITION (birth_year=1998, birth_month='May');
```

14.1. Форматы хранения данных

Форматы хранения данных:

- AVRO: формат на основе JSON, включающий поддержку RPC и сериализацию,
- Parquet: колоночный формат хранения,
- ORC: быстрый колоночный формат хранения,
- RCFile: формат размещения данных для реляционных таблиц,
- SequenceFile: бинарный формат данных с записью определенных типов данных.

Текстовый файл Пример создания таблицы в текстовом формате. Формат текстового файла используется по умолчанию для Hive.

```
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;
```

Файл последовательностей (SequenceFile) Формат SequenceFile является значением по умолчанию в Hadoop и хранит данные в *парах ключ-значение*. Большинство инструментов в экосистеме Hadoop могут читать формат SequenceFile

```
CREATE TABLE country (
  name STRING,
  population INT
)
STORED AS SequenceFile;
```

Файл столбцов строк (RCFile) Грубо говоря, RCFile это стратегия хранения данных в формате «строки столбцов». RCFile сочетает в себе достоинства как строко-ориентированных, так столбцово-ориентированных стратегий. Чтобы сериализовать таблицу, RCFile разбивает ее сначала по горизонтали, а затем по вертикали, вместо того чтобы разбить ее только по горизонтали, как это делается в обычных реляционных СУБД. Горизонтальное разбиение разбивает таблицу, как следует из названия стратегии, по горизонтали на несколько групп строк на основе заданного пользователем значения, определяющим размер группы строк.

```
CREATE TABLE country (
  name STRING,
  population INT
)
```

```
STORED AS RCFile;
```

Пример. Пусть есть таблица вида

```
c1 | c2 | c3 | c4
---+----+----+---
11 | 12 | 13 | 14
21 | 22 | 23 | 24
31 | 32 | 33 | 34
41 | 42 | 43 | 44
51 | 52 | 53 | 54
```

Как обсуждалось выше на первом этапе таблица разбивается по горизонтали на группы строк

```
c1 | c2 | c3 | c4
---+----+----+---
11 | 12 | 13 | 14
----------------
21 | 22 | 23 | 24
----------------
31 | 32 | 33 | 34
================ -- граница групп строк
41 | 42 | 43 | 44
----------------
51 | 52 | 53 | 54
```

Затем в каждой группе строк RCFile разбивает данные по вертикали

```
1-ая группа         2-ая группа
c1 | 11 | 21 | 31 || 41 | 51
c2 | 12 | 22 | 32 || 42 | 52
c3 | 13 | 23 | 33 || 43 | 53
c4 | 14 | 24 | 34 || 44 | 54
```

То есть таблица будет сериализована как

```
11, 21, 31; 41, 51;
12, 22, 32; 42, 52;
13, 23, 33; 43, 53;
14, 24, 34; 44, 54;
```

ORC файл ORC файл следует рассматривать как альтернативу RCFile.

15. Apache Ignite

Apache Ignite – распределенная база данных, позволяющая выполнять высоко-производительные вычисления в оперативной памяти.

16. Логическая витрина для доступа к большим данным

Пример. Рассмотрим некоторый промышленный комплекс, обладающий огромным количеством оборудования, обвешанного различными датчиками, регулярно сообщающими сведения о состоянии этого оборудования. Для простоты рассмотрим только два аггрегата (котел и резервуар), и три датчика (температуры котла и резервуара, а также давления в котле).

Эти датчики контролируются АСУ разных производителей и выдают информацию в разные хранилища: сведения о температуре и давлении в котле поступают в HBase, а данные о температуре в резервуаре пишутся в лог-файлы, расположенные в HDFS.

Данные о датчиках могут храниться, например, в `PostgreSQL`, а показания этих датчиков – в HDFS, HBase и т.п. Теперь пусть мы хотим предоставить аналитику возможность делать запросы. Заранее построить и запрограммировать сложные запросы не получится. Выполнение любого сложного, тяжелого запроса требует связывания данных из разных источников, в том числе из находящихся за пределами нашего модельного примера. Извне могут поступать, например, справочные сведения о рабочих диапазонах температуры и давления для разных видов оборудования, фасетные классификаторы, позволяющие определить, какое оборудование является маслонаполненным и др. Все подобные запросы аналитик формулирует в терминах концептуальной модели предметной области, то есть ровно в тех выражениях, в которых он думает о работе своего предприятия.

Витрина данных – предметно-ориентированная и, как правило, содержащая данные по одному из направлений деятельности компании база данных. Она отвечает тем же требованиям, что и хранилище данных, но в отличие от него, нейтрально к приложениям. В витрине информация храниться оптимизированно с точки зрения решения конкретных задач.

Витрины данных имеют следующие достоинства:

- пользователи ведят и работают только с теми данными, которые им действительно нужны,
- для витрин данных не требуется использовать мощные вычислительные средства.

К недостаткам витрин данных можно отнести сложность контроля целостности и противоречивости данных.

Список иллюстраций

Список литературы

1. *Сенько А.* Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. – СПб.: Питер, 2019. – 448 с.
2. *Уайт Т.* Hadoop: Подробное руководство. – СПб.: Питер, 2013. – 672 с.