海豚调度器：http://42.194.186.185:12345/dolphinscheduler

HBase：http://42.194.186.185:16010/master-status

脚本

```
#/bin/bash
#nohup不挂起，&后台运行 , 2>1& > path:不管是正常还是报错的信息都写入
#zookeeper启动服务
echo '正在启动zookeeper'
hosts=(node1 node2 node3)
for host in ${hosts[*]}
do
 ssh $host "source /etc/profile;/export/server/zookeeper/bin/zkServer.sh start"
done

#HBase启动服务--只需要启动一个
echo '正在启动HBase'
/export/server/hbase/bin/start-hbase.sh

#hive的metastore和hiveserver2的服务
echo '正在启动hive-metastore'
nohup /export/server/hive/bin/hive --service metastore 2>&1 > /tmp/hive-metastore.log &
echo '正在启动hive-hiveserver2'
nohup /export/server/hive/bin/hive --service hiveserver2 2>&1 > /tmp/hive-hiveserver2.log &
#nohup /export/server/hive/bin/hive --service metastore &
#nohup /export/server/hive/bin/hive --service hiveserver2 &
#启动MR和spark的历史服务
echo '正在启动MR-history'
/export/server/hadoop/sbin/mr-jobhistory-daemon.sh start historyserver
echo '正在启动spark-history'
/export/server/spark/sbin/start-history-server.sh
#启动spark连接客户端相关参数
echo '正在启动spark客户端连接'
/export/server/spark/sbin/start-thriftserver.sh \
--hiveconf hive.server2.thrift.port=10001 \
--hiveconf hive.server2.thrift.bind.host=node1 \
--master local[*]

#明细定义
start-thriftserver.sh \
--name sparksql-thrift-server \
--master yarn \
--deploy-mode client \
--driver-cores 4 \

--driver-memory 12g \
--hiveconf hive.server2.thrift.http.port=10001 \
--num-executors 8 \
--executor-memory 4g \
--conf spark.sql.shuffle.partitions=2
#jps -m
jps -m
#第二种方案(企业)
#/export/server/spark/sbin/start-thriftserver.sh \
#--hiveconf hive.server2.thrift.port=10001 \
#--hiveconf hive.server2.thrift.bind.host=node1 \
#--master yarn \
#--num-executors 50 \
#--executor-cores 4 \
#--executor-memory 12G \
echo '正在启动海豚调度器'
/export/server/dolphinscheduler/bin/start-all.sh
 echo '查看zookeeper运行状态'
 /export/server/zookeeper/bin/zkServer.sh status

  echo "正在启动kafka"
  nohup /export/server/kafka/bin/kafka-server-start.sh
/export/server/kafka/config/server.properties 2>&1 > /tmp/kafka.log &

  echo "启动Phoenix客户端"
  /export/server/phoenixbin/python2 sqlline.py node1:2181

```

```
kafka
#kafka创建主题
kafka-topics.sh --create --bootstrap-server node1:9092,node2:9092,node3:9092 --
partitions 3 --replication-factor 1 --topic ODS_BASE_LOG_1018
#kafka生产者
kafka-console-producer.sh --broker-list node1:9092,node2:9092,node3:9092 --topic test001
```

```
#kafka消费者
kafka-console-consumer.sh  --bootstrap-server node1:9092,node2:9092,node3:9092 --topic ODS_BASE_LOG_1018
#kafka查看主题
kafka-topics.sh --bootstrap-server node1:9092,node2:9092,node3:9092 --list
#kafka查看主题详情
kafka-topics.sh --bootstrap-server node1:9092 --describe --topic test001

flume
#启动flume
flume-ng agent -c conf -f /export/server/flume/conf/momo_taildir_source_kafka_sink.conf -n a1 -Dflume.root.logger=INFO,console

phoneix
#启动phoneix客户端
/export/server/phoenix/bin/python2 sqlline.py node1:2181

启动陌陌jar包
java -jar /export/data/momo_init/MoMo_DataGen.jar MoMo_Data.xlsx /export/data/momo_data/ 1000

```

下载

```
#dolphinscheduler调度器
wget https://archive.apache.org/dist/dolphinscheduler/1.3.5/apache-dolphinscheduler-incubating-1.3.5-dolphinscheduler-bin.tar.gz
#MySQL
wget https://dev.mysql.com/get/Downloads/MySQL-8.0/mysql-8.0.27-1.el8.x86_64.rpm-bundle.tar
#spark
wget https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
#anaconda3
wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh
#hadoop未编译
wget https://archive.apache.org/dist/hadoop/common/hadoop-3.3.0/hadoop-3.3.0-src.tar.gz
#zookeeper
wget https://archive.apache.org/dist/zookeeper/zookeeper-3.7.0/apache-zookeeper-3.7.0-bin.tar.gz
#hive
wget https://archive.apache.org/dist/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

```
#sqoop
wget https://archive.apache.org/dist/sqoop/1.4.7/sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
#sqoop-commns-lang.jar包
wget https://repo1.maven.org/maven2/commons-lang/commons-lang/2.6/commons-lang-2.6.jar
#sqoop-hive.jar包
wget https://repo1.maven.org/maven2/org/apache/hive/hive-common/3.1.2/hive-common-3.1.2.jar
#MySQL8.0jar包
wget https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.27/mysql-connector-java-8.0.27.jar
#MySQL5.0jar包
wget https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.49/mysql-connector-java-5.1.49.jar
#HBase
wget https://archive.apache.org/dist/hbase/2.1.0/hbase-2.1.0-bin.tar.gz
#Phoenix
wget https://archive.apache.org/dist/phoenix/apache-phoenix-5.0.0-HBase-2.0/bin/apache-phoenix-5.0.0-HBase-2.0-bin.tar.gz
#kafka
wget https://archive.apache.org/dist/kafka/2.4.1/kafka_2.12-2.4.1.tgz
#flume
wget https://archive.apache.org/dist/flume/1.9.0/apache-flume-1.9.0-bin.tar.gz
```

python安装库

```
pip install pyarrow,pyspark
```

python中创建使用pyspark

```
import string,time,os
from numpy import double
from pyspark import StorageLevel
from pyspark.sql import SparkSession, Row,functions
from pyspark.sql.functions import udf,pandas_udf
from pyspark.sql.types import *
import pandas as pd

# 这里选择本地pyspark环境执行spark代码
os.environ['JAVA_HOME']='/export/server/jdk'
os.environ['SPARK_HONE'] = '/export/server/spark'
PYSPARK_PYTHON = '/root/anaconda3/bin/python'
# 当存在多个版本时,不指定很可能会导致出错
```

```python
os.environ['PYSPARK_PYTHON'] = PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON'] = PYSPARK_PYTHON
if __name__ == '__main__':
    # 1.创建SparkSession上下文对象
    spark = SparkSession.builder.appName('spark_no_hive').master('local[*]')\
        .config('hive.metastore.uris','thrift://node1:9083')\
        .config('hive.metastore.warehouse.dir','/user/hive/warehouse')\
        .config('spark.sql.shuffle.partitions','4')\
        .enableHiveSupport()\
        .getOrCreate()
    spark.sparkContext.setLogLevel('WARN')
    #Arrow 是一种内存中的列式数据格式，用于spark中以jvm和python进程之间有效的传输数据
    #需要安装arrow pip install pyarrow
    #2.开启pyarrow，能加快计算速度，原理2个：1.基于内存减少了序列化和反序列化开销，2.基于向量计算vectorize
    spark.conf.set('spark.sql.execution.arrow.pyspark.enabled','true')
```

spark写出

```python
#3.将DataFrame数据保存为text格式
#保存为text格式，只能写出一列，不能写出多列
df.select(functions.concat_ws('_','user_id','movieID','rating','timestamp')).write.mode('overwrite').format('text').save('file://./out1/')
#4.将DataFrame数据保存为csv格式 -- header 不忽略字段
df.coalesce(1).write.mode('overwrite').format('csv').option('sep',',').option('header',True).save('file://./csv')
#5.将DataFrame数据保存为json格式
df.coalesce(1).write.mode('overwrite').format('json').save('file://./json')
#6.将DataFrame数据保存为parquet格式
df.coalesce(1).write.mode('overwrite').format('parquet').save('file://./parquet')
#parquet 语法2 --spark默认支持的格式就是parquet
df.coalesce(1).write.mode('overwrite').save('file://./parquet2')
#mysql连接
df.write.format('jdbc').mode('overwrite').option('url', 'jdbc:mysql://node1:3306/insurance') \
    .option('dbtable', 'policy_actuary') \
    .option('user', 'root').option('password', '123456').save()

#读取mysql
df=spark.read.format('jdbc') \
    .option('url', 'jdbc:mysql://42.194.186.185:3306/?serverTimezone=UTC&characterEncoding=utf8&useUnicode=true') \
    .option('dbtable', 'bigdata.tb_top10_movies') \
    .option('user', 'root').option('password', '123456') \
    .load()
spark=SparkSession.builder.appName("word").mater("local[*]").getOrcreate()
#2-加载text文件形成DataFrame
df=spark.read.format('text').load('path')'
#简化写法
df1=spark.read.text('path')

#3-加载csv文件形成DataFrame
df2=spark.read.format('csv')\
    .option('sep',';')\
    .option('header',True)\
    .option('encoding','utf-8')\
    .option('inferSchema',True)\
    .load(path)
#4-加载json文件形成DataFrame
df3=spark.read.format('json').load('path')

#5-加载parquet文件形成DataFrame
df4=spark.read.format('parquet')\
    .load(path)
 #6-添加schema加载文件形成DataFrame
df=spark.read.schema('id int,name string, score int')\
    .csv('path')
```

全局

```
#JAVA_HOME
JAVA_HOME=/export/server/jdk
CLASSPATH=.:$JAVA_HOME/lib
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME CLASSPATH PATH
#HADOOP_HOME
export HADOOP_HOME=/export/server/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
#hive和spark的免账号密码登录
alias beelinehive="/export/server/apache-hive/bin/beeline -u jdbc:hive2://node1:10000 -n root -p 123456"
alias beelinespark="/export/server/spark/bin/beeline -u jdbc:hive2://node1:10001 -n root -p 123456"
```

```
#SPARK_HOME
export SPARK_HOME=/export/server/spark
export PATH=$PATH:$SPARK_HOME/bin
#ANACONDA_HOME
export ANACONDA_HOME=/root/anaconda3
export PATH=$ANACONDA_HOME/bin:$PATH:/root
#HIVE
export HIVE_HOME=/export/server/hive
export PATH=$PATH:$HIVE_HOME/bin
#SQOOP_HOME
export SQOOP_HOME=/export/server/sqoop
export PATH=$PATH:$SQOOP_HOME/bin
#ZOOKEEPER_HOME
export ZOOKEEPER_HOME=/export/server/zookeeper/
export PATH=$PATH:$ZOOKEEPER_HOME/bin
#HBASE_HOME
export HBASE_HOME=/export/server/hbas
export PATH=$PATH:$HBASE_HOME/bin
#KAFKA_HOME
export KAFKA_HOME=/export/server/kafka
export PATH=:$PATH:${KAFKA_HOME}/bin
#phoenix_HOME
export PHONEIX_HOME=/export/server/phoenix
#FLUME
export FLUME_HOME=/export/server/flume
export PATH=$PATH:$FLUME_HOME/bin

```

```
【计算链条比较长，每一步骤如果有精度损失，那么叠加到最后，可能会导致结果不准确,则设置如下】
set spark.sql.decimalOperations.allowPrecisionLoss=false

【spark和vine导出打印表头，有数据库名】
set hive.cli.print.header=true;

【spark和vine导出打印表头，去数据库名】
set hive.cli.print.header=true;set hive.resultset.use.unique.column.names=false;

【spark设置shuffle并行度】
```

```
spark.sql('set spark.sql.shuffle.partitions=4')
set spark.sql.shuffle.partitions=4

【hdfs 查看总文件大小】
hdfs dfs -du -h path

【hdfs副本数】
dfs.replication

【修复MySQL元数据，和分区表的实际目录一致】
msck repair table 库.表;

【开启hive中的mapjoin】--大表的数据会分成多个task和整张小表去join
set hive.auto.convert.join=true [hive2默认为true]

【hive和spark的mapjoin小表大小】
set hive.auto.convert.join.noconditionaltask.size=10000000;【默认为10M单位字节】
set spark.sql.autoBroadcastJoinThreshold=51200000;

【hive的bucket mapjoin 分桶】
set hive.optimize.bucketmapjoin = true;【默认为true】

【sortmerge bucket mapjoin（大表join大表smb）】
set hive.auto.convert.join=true;
set hive.optimize.bucketmapjoin = true;  -- 开启 bucket map join
set hive.auto.convert.sortmerge.join=true; -- 开启 SBM join支持--归并
set hive.optimize.bucketmapjoin.sortedmerge = true;  -- 自动尝试开启 SMB join
--set hive.enforce.sorting=true;  -- 开启强制排序
--set hive.enforce.bucketing=true;-- 注释的只能在配置文件写死或者不执行
--set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;
set hive.auto.convert.join.noconditionaltask = true;
--下面可以不设置
set hive.auto.convert.join.noconditionaltask.size = 50000000;
set hive.auto.convert.sortmerge.join.bigtable.selection.policy
    = org.apache.hadoop.hive.ql.optimizer.TableSizeBasedBigTableSelectorForAutoSMJ;

【MR的task内存】
set mapreduce.map.java.opts=-Xmx6000m;
set mapreduce.map.memory.mb=6096;
set mapreduce.reduce.java.opts=-Xmx6000m;
set mapreduce.reduce.memory.mb=6096;

【MR缓冲区大小】
set mapreduce.task.io.sort.mb=100

【MR环形缓冲区溢出的阈值】
mapreduce.map.sort.spill.percent=0.8

【reduce拉取并行度】
set mapreduce.reduce.shuffle.parallelcopies=8
set mapreduce.reduce.shuffle.read.timeout=180000

【MR小文件处理】
#设置Hive中底层MapReduce读取数据的输入类：将所有文件合并为一个大文件作为输入
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
#如果hive的程序，只有maptask，将MapTask产生的所有小文件进行合并
set hive.merge.mapfiles=true;
#如果hive的程序，有Map和ReduceTask,将ReduceTask产生的所有小文件进行合并
set hive.merge.mapredfiles=true;
#每一个合并的文件的大小
set hive.merge.size.per.task=256000000;
#平均每个文件的大小，如果小于这个值就会进行合并
set hive.merge.smallfiles.avgsize=16000000;

【查看hive和spark的执行计划】
explain select...

【markdown代码折叠】
<details>
<summary><b>点击查看完整代码</b></summary>
<pre><code>
</code></pre>
</details>
【Linux关闭邮件提醒】
echo "unset MAILCHECK" >> /etc/profile
source /etc/profile

```

```
【Linux ifconfig没有ip】
[方案1推荐]
[root@localhost~]# ifup ens33
Error: Connection activation failed: No suitable device found for this
connection(device lo not availa ble because device is strictly unmanaged).

[root@localhost~]# chkconfig NetworkManager off
Note: Forwarding request to 'systemctI disable NetworkManager.service'.
Removed symlink /etc/systemd/system/multi-user.target.wants/NetworkManager.service.
Removed symlink /etc/systemd/system/dbus-org.freedesktop.NetworkManager.service.
Removed symlink /etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service.
Removed symlink /etc/systemd/system/network-online.target.wants/NetworkManager-wait-
online.service.

[root@localhost~]# chkconfig network on
[root@localhost~]# service NetworkManager stop
Redirecting to /bin/systemctl stop NetworkManager.service
[root@localhost~]# service network start
Starting network (via systemctl):[ ok]

[方案2]
ip add show
将link/ether后的MAC地址写到ens33中的HWADDR
vim /etc/sysconfig/network-scripts/ifcfg-ens33

【Linux搜索指定文件类型的指定内容】
grep -rin "9820" --include "*.xml" /export/server/
find / -name "*.log" | grep "error"

【SparkSQL报错Filesystem Closed】
修改hadoop配置文件core-site.xml:
<property>
        <name>fs.hdfs.impl.disable.cache</name>
        <value>true</value>
</property>

【启动hive的metastore、hiveserver2服务】
nohup hive --service metastore  2>&1 > /tmp/hive-metastore.log &
nohup hive --service hiveserver2 2>&1 > /tmp/hive-hiveserver2.log &

```

```

【hive元数据初始化和更新】
schematool -dbType mysql -initSchema
schematool -dbType mysql -upgradeSchema

【mysql的授权语句】
GRANT ALL PRIVILEGES ON hive.* TO 'root'@'%' IDENTIFIED BY '123456';
flush privileges;

【hive的jdbc连接地址】
jdbc:hive2://node1:10000


/export/server/zookeeper/bin/zkServer.sh stop
/export/server/zookeeper/bin/zkServer.sh start
/export/server/zookeeper/bin/zkServer.sh status

/export/server/dolphinscheduler/bin/stop-all.sh
/export/server/dolphinscheduler/bin/start-all.sh



【开启Hive的本地模式】
set hive.exec.mode.local.auto=true;(默认为false)
【hive开启负载均衡】平均分区，让每个分区都是同样的数据，如果多个去重则会报错，这个针对group by
hive.groupby.skewindata=true



【hadoop退出安全模式】
hdfs dfsadmin -safemode leave
hdfs dfsadmin -safemode forceExit

【重新启动zk和kafka】
zkServer.sh stop
kafka-server-stop.sh
zkServer.sh start
nohup /export/server/kafka/bin/kafka-server-start.sh
/export/server/kafka/config/server.properties &

```

```
【彻底删除kafka主题】
kafka-topics.sh --zookeeper node3:2181 --list
kafka-topics.sh --zookeeper node3:2181 --delete --topic spark_kafka

【zkCli.sh】
ls /config/topics
rmr /config/topics/spark_kafka
rmr /brokers/topics/spark_kafka
rmr /admin/delete_topics/spark_kafka

【创建主题】
kafka-topics.sh --zookeeper node3:2181 --create --topic spark_kafka --partitions 3 --replication-factor 1
kafka-topics.sh --zookeeper node3:2181 --list

【启动生产者和消费者】
kafka-console-producer.sh --broker-list node3:9092 --topic spark_kafka
kafka-console-consumer.sh --from-beginning --bootstrap-server node3:9092 --topic spark_kafka
kafka-console-consumer.sh --from-beginning --bootstrap-server node3:9092 --topic __consumer_offsets


【完整的删除再重建主题步骤start
关闭生产者和消费者

kafka-topics.sh --zookeeper node3:2181 --list
kafka-topics.sh --zookeeper node3:2181 --delete --topic stationTopic

kafka-topics.sh --zookeeper node3:2181 --list

zkCli.sh
ls /config/topics
rmr /config/topics/stationTopic
rmr /brokers/topics/stationTopic
rmr /admin/delete_topics/stationTopic

删除log文件目录

kafka-topics.sh --zookeeper node3:2181 --create --topic stationTopic --partitions 3 --replication-factor 1
```

```
kafka-topics.sh --zookeeper node3:2181 --list
再次重启kafka
kafka-server-stop.sh
nohup /export/server/kafka/bin/kafka-server-start.sh
/export/server/kafka/config/server.properties &

启动生产者
kafka-console-producer.sh --broker-list node3:9092 --topic stationTopic
启动消费者
kafka-console-consumer.sh --bootstrap-server node3:9092 --topic stationTopic --from-
beginning
完整的删除再重建主题步骤end】


【安装redis】
tar
yum install gcc
cd /redis3.2.8
make
make PREFIX=/usr/local/src/redis install
cp /redis3.2.8/redis.conf /redis/bin/redis.conf
启动redis
redis-server redis.conf
redis-cli -h 192.168.88.163

【启动spark-thriftserver】
/export/server/spark/sbin/start-thriftserver.sh \
  --hiveconf hive.server2.thrift.port=10001 \
  --hiveconf hive.server2.thrift.bind.host=node1 \
  --master local[*]

/export/server/spark/sbin/start-thriftserver.sh \
--hiveconf hive.server2.thrift.port=10001 \
--hiveconf hive.server2.thrift.bind.host=p1 \
--master local[*]


/export/server/spark/sbin/start-thriftserver.sh \
--hiveconf hive.server2.thrift.port=10001 \
--hiveconf hive.server2.thrift.bind.host=p1 \
--master yarn \
--deploy-mode client \
--driver-memory 1g \
--executor-memory 2g \
--executor-cores 4 \
--num-executors 25 \

!connect jdbc:hive2://192.168.88.166:10001

【kafka】
启动Zookeeper 服务
zookeeper-daemon.sh start

启动Kafka 服务
kafka-daemon.sh start


【免秘钥登录】
ssh-keygen -t rsa
ssh-copy-id node1
scp /root/.ssh/authorized_keys node2:/root/.ssh
scp /root/.ssh/authorized_keys node3:/root/.ssh

【sqoop】
sqoop import \
-D mapred.job.name=sqoop_import_dd_table  \
--connect "jdbc:mysql://192.168.88.163:3306/insurance"  \
--username root  \
--password "123456"  \
--table dd_table  \
--hive-import  \
--hive-database insurance  \
--hive-table dd_table  \
--hive-overwrite  \
-m 1  \
--fields-terminated-by ',' \
--delete-target-dir

【mysql】
```

```
mysqldump insurance --add-drop-trigger --result-file=/opt/insurance/insurance.sql --user=root --host=192.168.88.163 --port=3306
mysql -uusername -ppassword -h ip_address -P 3306 --default-character-set = utf8mb4 ${i} </ data / mysql_back / $ {i} .sql
mysql -uroot -p  --default-character-set=utf8mb4 insurance </opt/insurance/insurance.sql


【dolphinscheduler】
CREATE DATABASE dolphinscheduler DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci;
set global validate_password.policy=0;
set global validate_password.length=1;

GRANT ALL PRIVILEGES ON dolphinscheduler.* TO '{user}'@'%' IDENTIFIED BY '{password}';
flush privileges;

安装dolphinscheduler 要启动zookeeper
一定要配在ds的env中配置sqoop的环境变量
需要root给租户授权最大。
谁将脚本上传到Linux，谁就记得授执行权限。

安装dos2unix命令
 dos2unix /opt/insurance/sqoop/sqoop_import_mort_10_13.sh

修改工作流，要先下线


【structured Streaming】
--memory sink
CREATE TABLE db_spark.tb_word_count (
  id int NOT NULL AUTO_INCREMENT,
  word varchar(255) NOT NULL,
  count int NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY word (word)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

REPLACE INTO  tb_word_count (id, word, count) VALUES (NULL, ?, ?);


【spark yarn Pi】
```

```
/export/server/spark/bin/spark-submit \
--master yarn \
--class org.apache.spark.examples.SparkPi \
${SPARK_HOME}/examples/jars/spark-examples_2.11-2.4.5.jar \
10

【WordCount yarn】
/export/server/spark/bin/spark-submit \
--master yarn \
--driver-memory 512m \
--executor-memory 512m \
--executor-cores 1 \
--num-executors 2 \
--queue default \
--class cn.itcast.spark._2SparkWordCount \
/opt/spark-chapter01-1.0-SNAPSHOT.jar



【 Run application locally on 8 cores】
/export/server/spark/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
${SPARK_HOME}/examples/jars/spark-examples_2.11-2.4.5.jar \
  100

# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
${SPARK_HOME}/examples/jars/spark-examples_2.11-2.4.5.jar \
  1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000

# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \  # can be client for client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000

# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000

# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000

# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000

```

```
hive动态分区
------------------因为采用了动态分区插入技术 因此需要设置相关参数---------------
--分区
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions.pernode=10000;
set hive.exec.max.dynamic.partitions=100000;
set hive.exec.max.created.files=150000;
--hive压缩
set hive.exec.compress.intermediate=true;
set hive.exec.compress.output=true;
--写入时压缩生效
set hive.exec.orc.compression.strategy=COMPRESSION;
```

```
--开启分桶，如果不开启，是不会启动多个reduce分桶的
set hive.enforce.bucketing=true;
--2.x版本可以通过以下参数禁止使用load语句加载数据到表中
set hive.strict.checks.bucketing = true;
```

常见的文件类型：textfile/orc/parquet

```
hiveorc索引
stored as orc (’orc.create.index’=’true’)
查询数据时，开启row group index 过滤查询
hive.optimize.index.filter=true
```

```
hive优化
矢量化查询：开启了：Hive每个批次读取1024条，处理1024条
set hive.vectorized.execution.enabled = true;
set hive.vectorized.execution.reduce.enabled = true;

零拷贝：
功能：数据在操作系统中从内存中不用经过多次拷贝直接读取
```

```
不开启：必须在内存中经过多次交换才能读取到数据
开启了：数据可以直接从内存中读取
set hive.exec.orc.zerocopy=true;

关联优化器
功能：Hive在解析SQL语句，转换为MapReduce时候，可以将相关联的部分合并在一起执行
自动判断所有执行过程语法数是否有重合的过程，放在一起执行
set hive.optimize.correlation=true;

动态生成分区的线程数
说明：在执行动态分区过程中，可以运行多少个线程来生产分区数据，线程数量越多，执行效率越高，前提是有资源
hive.load.dynamic.partitions.thread  默认值为 15
```

hive建表

```
hive建表语句
create [external | temporary] table [dbname.]tbname(
        colName1 type1 comment '',
        colName2 type1 comment '',
        colName3 type1 comment '',
        ……
        colNameN type1 comment ''

) comment '表的注释'
partitioned by (col type)
clustered by (col) [sorted by col desc] into N buckets
row format delimited fields terminated by
stored as orc [tblpropertied ("orc.compress"="[ZLIB|SNAPPY]")]
location ‘指定路径’

jdbc:mysql://hadoop01:3306/yipin?useUnicode=true&characterEncoding=UTF-8&autoReconnect=true'
```

mapreduce内存溢出优化

```
MapReduce的内存配置
map阶段:mapreduce.map.memory.mb : 每一个map默认申请的内存大小，默认值为 0  表示自动获取；mapreduce.map.java.opts :  每一个map的jvm的内存大小；
注意:   mapreduce.map.java.opts一定要小于mapreduce.map.memory.mb；

```

```
reduce阶段:mapreduce.reduce.memory.mb : 每一个reduce默认申请的内存大小, 默认值为 0  表示自动获取; mapreduce.reduce.java.opts :  每一个reduce的jvm的内存大小;
注意:     mapreduce.reduce.java.opts 一定要小于mapreduce.reduce.memory.mb;

注意: MR中所有的内存配置, 都不能大于nodemanager的内存大小

jvm-java的格式为: -Xmx4096m
其他: 4096
```

压缩配置

```
yarn配置:  在CM中直接配置
mapreduce.map.output.compress  是否开启map端压缩配置     默认开启的
mapreduce.map.output.compress.codec  map端采用何种压缩方案
        建议配置为: org.apache.hadoop.io.compress.SnappyCodec

mapreduce.output.fileoutputformat.compress 是否开启reduce端压缩配置  默认不开启的 对最终结果有影响
mapreduce.output.fileoutputformat.compress.codec reduce端需要采用何种压缩操作
        建议配置为: org.apache.hadoop.io.compress.SnappyCodec

mapreduce.output.fileoutputformat.compress.type 采用压缩的方式
        建议: block 块压缩

hive配置:  此配置需要在会话中执行 MR开了压缩, 开启以下才会进行压缩
set hive.exec.compress.intermediate=true; 开启中间结果的压缩
set hive.exec.compress.output=true; 是否开启最终结果压缩

```