

修订记录

课程编码	适用产品	产品版本	课程版本ISSUE

开发/优化者	时间	审核人	开发类型（新开发/优化）	更新说明
王勒然 /WX1059060	2022.10.9		优化	
楼佳明 /WX1059060	2022.10.13		优化	
楼佳明 /WX1059060	2022.11.1		优化	

实验三：基于鲲鹏应用使能套件实现 Spark 算法优化



华为技术有限公司



目录

1 实验三：基于鲲鹏应用使能套件实现 Spark 算法优化.....	2
1.1 实验介绍	2
1.1.1 关于本实验	2
1.1.2 实验目的.....	2
1.1.3 实验环境说明.....	2
1.1.4 软件介绍.....	2
1.1.5 实验拓扑.....	3
1.2 准备环境	3
1.3 配置部署环境	10
1.4 部署 zookeeper-3.4.14.....	12
1.5 部署 Hadoop-3.1.4.....	15
1.6 部署 Spark	23
1.7 部署算法库.....	25
1.8 思考题	28

1

实验三：基于鲲鹏应用使能套件实现 Spark 算法优化

1.1 实验介绍

1.1.1 关于本实验

本实验指导用户基于华为云鲲鹏服务器部署 4 节点 Spark 集群，使能 Boostkit 机器学习&图分析算法库，验证其相比于原生算法的优化效果。

1.1.2 实验目的

- 掌握基于华为云鲲鹏服务器的 Spark 集群搭建；
- 掌握基于 Boostkit 使能算法优化的验证。

1.1.3 实验环境说明

本实验在华为云上完成，推荐配置如下：

表1-1 资源配置

ECS名称	规格
kp-test01	鲲鹏计算 鲲鹏内存优化型 km1.xlarge.8 4核 32GB;openEuler openEuler 20.03 64bit with ARM;通用型SSD 40GB;通用型SSD 50GB;全动态BGP 独享 按流量计费 100Mbit/s;

1.1.4 软件介绍

本实验设计的软件及链接如下表，请提前准备好相关软件。

表1-2 软件链接

软件名称	下载链接
zookeeper-3.4.14.tar.gz	https://repo.huaweicloud.com/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz

hadoop-3.1.4.tar.gz	https://repo.huaweicloud.com/apache/hadoop/common/hadoop-3.1.4/hadoop-3.1.4.tar.gz
spark-2.3.2-bin-hadoop2.7.tgz	https://repo.huaweicloud.com/apache/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
spark_algorithm.zip	https://chatlab.obs.cn-north-4.myhuaweicloud.com:443/spark_algorithm.zip?AccessKeyId=R5ONPGDIP4HMWBO8L06Z&Expires=1697936945&Signature=xxkr4lsmUcF4OUe4eCp6c1JhMkw%3D

1.1.5 实验拓扑

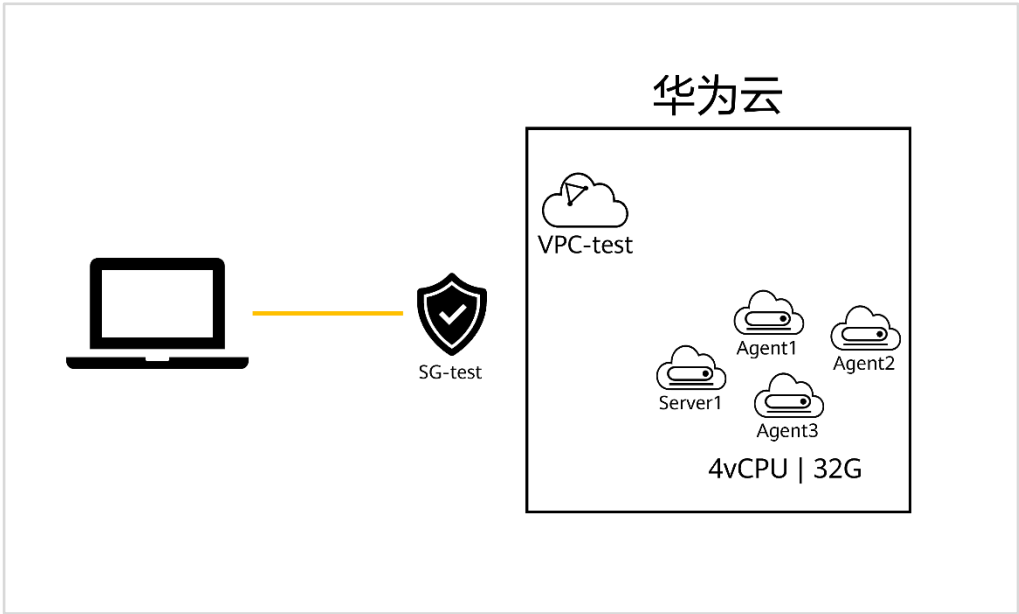
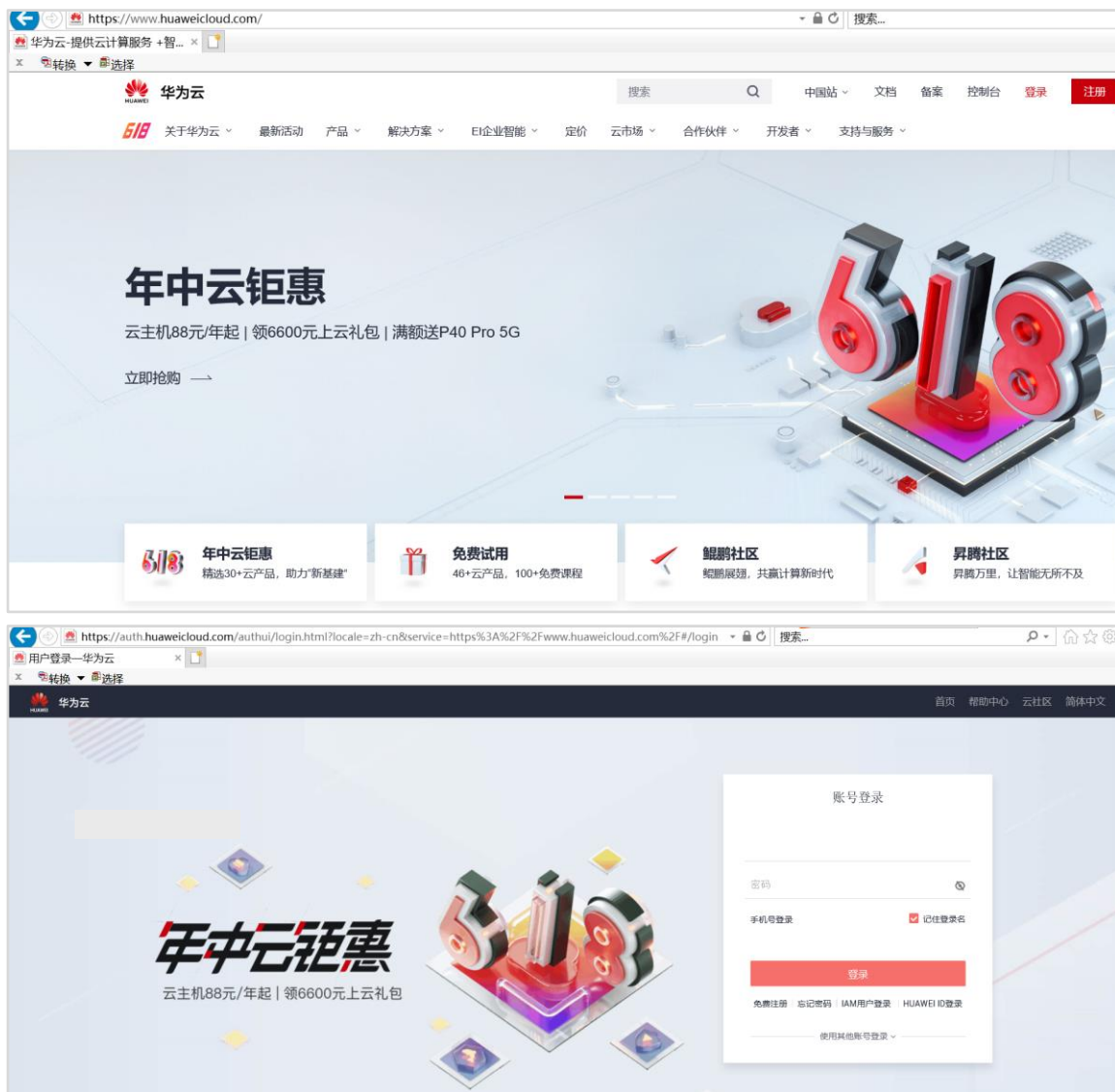


图1-1 实验环境

1.2 准备环境

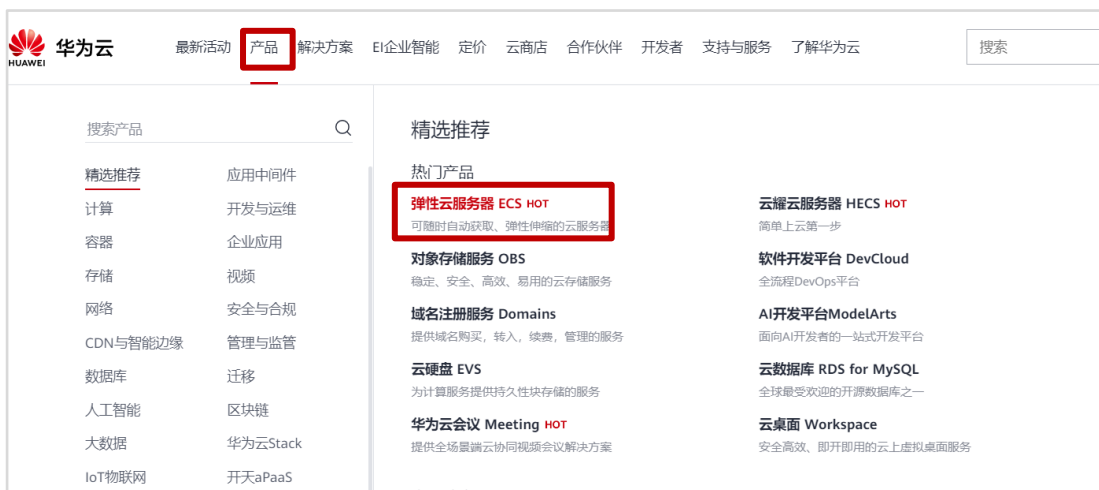
步骤 1 登录华为云

打开浏览器，输入华为云的域名：<https://www.huaweicloud.com>，点击右上角登录按钮，输入用户名与密码。



步骤 2 基础环境配置

- 1) 购买弹性云服务器 ECS：登录后，在产品下拉选项中点击弹性云服务器 ECS，在随后打开的页面中点击立即购买按钮。



本实验使用一套四节点的集群环境，由 1 个 Server 节点（管理节点）和 3 个 Agent 节点（计算节点）组成，所以总共需要购买 4 台 ECS。

2) ECS 基础配置（针对所有节点，包括 Server 和 Agent）。

CPU 架构：鲲鹏计算

CPU 和内存规格：鲲鹏内存优化型——km1.xlarge.8 (4vCPUs|32GB)



镜像：openEuler

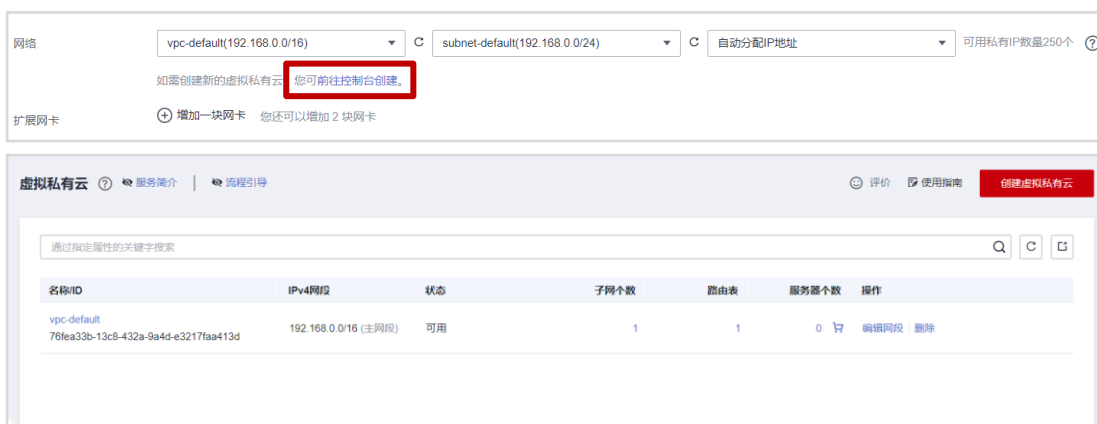
系统盘：通用型 SSD40GB - 1 块

数据盘：通用型 SSD50GB - 1 块



3) 网络配置：选择网络时，建议前往控制台创建一个 VPC（虚拟私有云），以免受到其他网络干扰。若有信任的 VPC，则集群的 4 个节点都选择该 VPC 网段。

- 点击前往控制台创建 - 创建虚拟私有云。



名称/ID	IPv4网段	状态	子网个数	路由表	服务器个数	操作
vpc-default 76fea33b-13c8-432a-9a4d-e3217faa413d	192.168.0.0/16 (主网段)	可用	1	1	0	编辑网段 删除

- 输入 VPC 基本信息：区域选择需要与 ECS 的区域保持一致，名称可以自己拟定，IPv4 网段可在建议中选择也可自己填写，只要与当前区域下其他 VPC 使用的网段不重叠即可。



该VPC网段 (192.168.0.0/16) 与当前区域下其他VPC网段重叠，如需使用VPC互通服务，建议您修改VPC网段。查看区域下已有vpc网段

- 默认子网：可用区需要与 ECS 的可用区保持一致，名称可以自己拟定，其他配置不需要修改。

默认子网

可用区

可用区1

名称

subnet-8b3b

子网IPv4网段

192 · 168 · 0 · 0 / 24

可用IP数: 251

子网创建完成后, 子网网段无法修改

子网IPv6网段

☐ 开启IPv6

关联路由表

默认

高级配置

网关

DNS服务器地址

NTP服务器地址

DHCP租约时间

标签

描述

VPC 创建完成后，集群中的所有节点都可以选择该 VPC 网段。

- 安全组的入方向规则必配置允许 TCP:22、TCP:8088、TCP:8020、TCP:50070 等端口，可以选择默认安全组，然后点击配置安全组规则，将这 3 个端口都开放。公网 IP 选择现在购买，线路选择动态 BGP。带宽可以选择按流量计费，带宽大小 100M。

安全组

sg-spark074cd27b-e686-4487-8112-53ec48581c...

新建安全组

安全组是网络防火墙功能，是一个逻辑上的分组，用于设置网络访问控制。请确保所选安全组已放通22端口（Linux SSH登录），3389端口（Windows远程登录）和 ICMP 协议（Ping）。配置安全组规则

隐藏安全组规则

入方向规则

出方向规则

安全组名称	优先级	策略	协议端口	类型	源地址	描述
sg-spark	1	允许	TCP: 50070	IPv4	0.0.0.0/0	--
	1	允许	TCP: 8020	IPv4	0.0.0.0/0	--
	1	允许	TCP: 8088	IPv4	0.0.0.0/0	--
	1	允许	全部	IPv6	sg-spark	允许安全组内的弹性云服务器彼此通信
	1	允许	TCP: 3389	IPv4	0.0.0.0/0	允许远程登录Windows弹性云服务器

弹性公网IP

☒ 现在购买 ☐ 使用已有 ☐ 暂不购买

线路

全动态BGP 静态BGP

不低于99.95%可用性保障

公网带宽

按带宽计费 流量较大或较稳定的场景

按流量计费 流量小或流量波动较大场景

加入共享带宽 多业务流量错峰分布场景

指定带宽上限，按实际使用的出公网流量计费，与使用时间无关。

带宽大小

5 10 20 50 100 自定义 100

带宽范围: 1-300 Mbit/s

释放行为

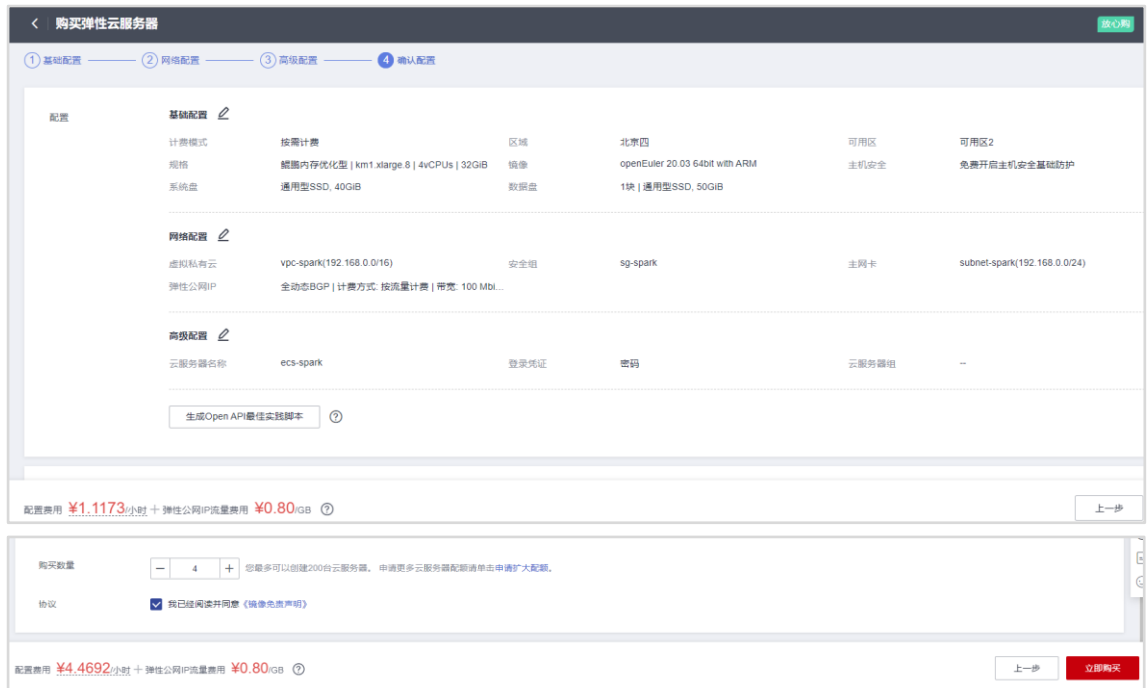
☒ 免费开启DDoS基础防护 ☐ 随实例释放

4) 高级配置。

云服务器名称可以自己拟定，密码自己拟定（需符合密码复杂度要求），云备份选择暂不购买，其余设置保持默认值即可。



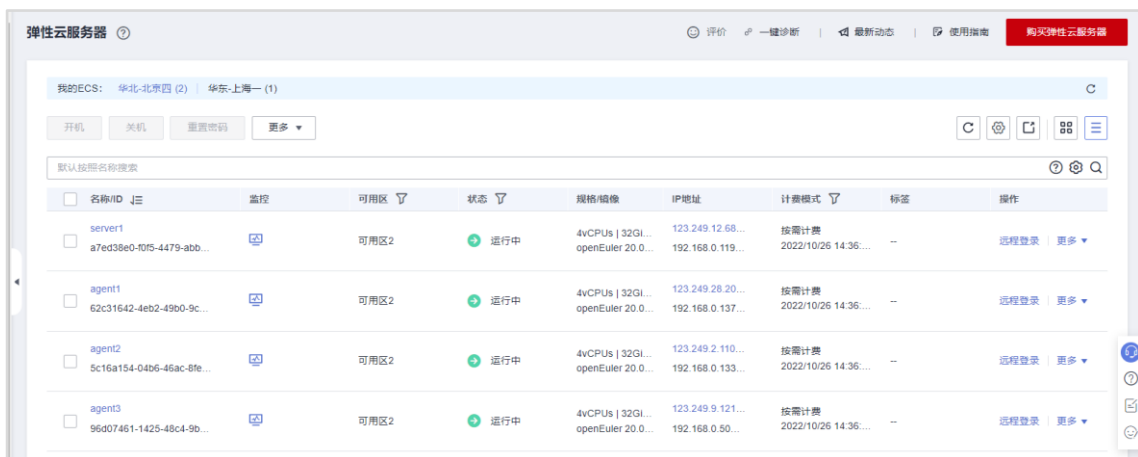
5) 确认 1 台服务器的配置无误后，数量选择 4，点击立即购买。



6) 回到弹性云服务器 ECS 管理列表，选择更多，修改名称，将 Server 节点的名称改为 server1，Agent 节点的名称分别改为 agent1、agent2 和 agent3。



名称/ID	规格/镜像	IP地址	计费模式	状态	操作
ecs-spark-0003 913bb0c1-1169-44...	4vCPUs 1... openEuler 2...	123.249.5... 192.168.0...	按需计费 2022/10/21 1...	运行中	远程登录 更多
ecs-spark-0001 71927b0d-26de-4c...	4vCPUs 1... openEuler 2...	123.249.1... 192.168.0...	按需计费 2022/10/21 1...	运行中	远程登录 更多
ecs-spark-0002 5a4d2e37-732c-45...	4vCPUs 1... openEuler 2...	120.46.21... 192.168.0...	按需计费 2022/10/21 1...	运行中	远程登录 更多
ecs-spark-0004 48008712-945a-43...	4vCPUs 1... openEuler 2...	123.249.1... 192.168.0...	按需计费 2022/10/21 1...	运行中	远程登录 更多



步骤 3 登录服务器

可使用 CloudShell 进行登录，也可使用其它方式进行登录。



步骤 4 挂盘（集群 4 个节点都需要操作）

依次登录四台 ECS，对每个节点做以下操作：

1) 对磁盘分区（仅首次登录时操作）

```
[root@server1~]# mkdir -p /data/data1
[root@server1~]# parted -s /dev/vdb mklabel gpt
[root@server1~]# parted -s /dev/vdb mkpart logical 0% 100%
```

2) 对磁盘格式化（仅首次登录时操作）

```
[root@server1~]# mkfs.xfs -f /dev/vdb
```

3) 数据盘挂载（每次重启 ECS 后都要操作）

```
[root@server1~]# mount -o defaults,noatime,nodiratime /dev/vdb /data/data1
```

1.3 配置部署环境

步骤 1 重命名主机

依次登录四台 ECS，将 ECS 的主机名分别修改为 server1、agent1、agent2、agent3，命令语法如下：

```
hostnamectl set-hostname 主机名 --static
```

如：在 Server 节点上执行：

```
hostnamectl set-hostname server1 --static
```

步骤 2 修改 hosts 文件

依次登录四台 ECS，修改 “/etc/hosts” 文件，命令语法如下：

```
vim /etc/hosts
```

在 hosts 文件中添加集群所有节点的 “地址-主机名” 映射关系。（IPaddress 为私有 IP）。

```
IPaddress1 server1
IPaddress2 agent1
IPaddress3 agent2
IPaddress4 agent3
```

配置结果如下图所示：

```

::1      localhost        localhost.localdomain  localhost6      localhost6.localdomain6
172.16.0.120  server1
172.16.0.166  agent1
172.16.0.48   agent2
172.16.0.53   agent3
127.0.0.1    localhost        localhost.localdomain  localhost4      localhost4.localdomain4
127.0.0.1    ecs-0005         ecs-0005
127.0.0.1    ecs-0003         ecs-0003

```

步骤 3 关闭防火墙

登录所有 ECS，执行以下命令关闭防火墙。

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

步骤 4 关闭 selinux 并配置 limits 参数

1) 登录所有 ECS，执行以下命令，更改 SELinux 为宽容模式。

```
setenforce 0
```

2) 检查 “/etc/selinux/config” 文件，是否永久禁止 SELinux 自动启动。

执行以下命令编辑配置文件

```
vim /etc/selinux/config
```

检查配置文件中 SELINUX=disabled 是否配置，若未配置则按如下所示配置：

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3) 编辑 “/etc/security/limits.conf” 文件。

执行以下命令，编辑 limits.conf 文件

```
vim /etc/security/limits.conf
```

添加或修改最后四行内容，如下所示：

```
* hard    nproc    65535
* soft    nproc    65535
* hard    nofile   65535
* soft    nofile   65535
```

修改结果如下图所示：

```
# - locks - max number of file locks the user can hold
# - sigpending - max number of pending signals
# - msgqueue - max memory used by POSIX message queues (bytes)
# - nice - max nice priority allowed to raise to values: [-20, 19]
# - rtprio - max realtime priority
#
#<domain>    <type>  <item>        <value>
#
#*             soft    core           0
#*             hard    rss            10000
#@student      hard    nproc          20
#@faculty      soft    nproc          20
#@faculty      hard    nproc          50
#ftp           hard    nproc          0
#@student      -       maxlogins       4
# End of file
* hard    nproc    65535
* soft    nproc    65535
* hard    nofile   65535
* soft    nofile   65535
```

步骤 5 配置 SSH 免密登录

登录所有节点，输入以下命令生成密钥，遇到提示时，直接按回车。

```
ssh-keygen -t rsa
```

在每台机器上配置对于自身节点和其他所有节点的 SSH 免密登录，遇到提示输入 “yes”，并输入相应节点的密码，命令语法如下：

```
ssh-copy-id root@主机名
```

例如：

```
ssh-copy-id root@server1
ssh-copy-id root@agent1
ssh-copy-id root@agent2
ssh-copy-id root@agent3
```

步骤 6 安装 JDK

由于 zookeeper、hadoop、spark 需要运行在 java 的环境中，所以需要提前安装好 java 的运行环境。

依次登录四台 ECS，执行以下命令，在 yum 中搜索 JDK。

```
yum search java|grep 1.8.0-openjdk
```

```
[root@server1 ~]# yum search java|grep 1.8.0-openjdk
Last metadata expiration check: 19:55:43 ago on Wed 23 Nov 2022 03:27:13 PM CST.
java-1.8.0-openjdk-debugsource.aarch64 : Debug sources for package java-1.8.0-openjdk
java-1.8.0-openjdk-debuginfo.aarch64 : Debug information for package java-1.8.0-openjdk
java-1.8.0-openjdk.aarch64 : OpenJDK Runtime Environment 8
java-1.8.0-openjdk.src : OpenJDK Runtime Environment 8
java-1.8.0-openjdk-src.aarch64 : OpenJDK Source Bundle 8
java-1.8.0-openjdk-demo.aarch64 : OpenJDK Demos 8
java-1.8.0-openjdk-devel.aarch64 : OpenJDK Development Environment 8
java-1.8.0-openjdk-javadoc.noarch : OpenJDK 8 API documentation
java-1.8.0-openjdk-headless.aarch64 : OpenJDK Headless Runtime Environment 8
java-1.8.0-openjdk-slowdebug.aarch64 : OpenJDK Runtime Environment 8 with full debug on
java-1.8.0-openjdk-javadoc-zip.noarch : OpenJDK 8 API documentation compressed in a single archive
java-1.8.0-openjdk-src-slowdebug.aarch64 : OpenJDK Source Bundle 8 for packages with debug on
java-1.8.0-openjdk-demo-slowdebug.aarch64 : OpenJDK Demos 8 with full debug on
java-1.8.0-openjdk-devel-slowdebug.aarch64 : OpenJDK Development Environment 8 with full debug on
java-1.8.0-openjdk-headless-slowdebug.aarch64 : OpenJDK Runtime Environment 8 with full debug on
```

注意：如果出现 NOT FOUND 报错，请稍后再尝试搜索命令。

执行以下命令，使用 yum 安装 openjdk 1.8。

```
yum install --nogpgcheck java-1.8.0-openjdk-devel.aarch64
```

执行以下命令，查看 java 版本。

```
java -version
```

```
[root@server1 ~]# java -version
openjdk version "1.8.0_262"
OpenJDK Runtime Environment Boole (build 1.8.0_262-b10)
OpenJDK 64-Bit Server VM Boole (build 25.262-b10, mixed mode)
```

1.4 部署 zookeeper-3.4.14

步骤 1 下载并解压 zookeeper-3.4.14

- 1) 登录 server1 节点，执行以下命令创建/opt/tools/installed 目录，下载 zookeeper 安装包。

```
mkdir -p /opt/tools/installed
cd /opt/tools/installed
wget https://repo.huaweicloud.com/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
```

- 2) 将 zookeeper-3.4.14.tar.gz 拷贝到 agent1 节点的 “/usr/local” 目录下，并解压。。

```
scp zookeeper-3.4.14.tar.gz root@agent1:/usr/local/
```

3) 登录 agent1 节点，解压 zookeeper。

```
cd /usr/local
tar -zxvf zookeeper-3.4.14.tar.gz
mv zookeeper-3.4.14 zookeeper
```

4) 在 agent1、agent2、agent3 节点上添加环境变量。

```
vim /etc/profile
```

在/etc/profile 文件末尾添加如下内容。

```
export ZOOKEEPER_HOME=/usr/local/zookeeper
export PATH=$ZOOKEEPER_HOME/bin:$PATH
```

5) 在 agent1、agent2、agent3 节点上，生效环境变量。

```
source /etc/profile
```

步骤 2 修改 zookeeper 配置文件

以下步骤均在 agent1 上执行。

1) 进入 ZooKeeper 所在目录。

```
cd /usr/local/zookeeper/conf
```

2) 拷贝配置文件。

```
cp zoo_sample.cfg zoo.cfg
```

3) 修改配置文件。

```
vim zoo.cfg
```

在 zoo.cfg 文件中修改数据目录

```
dataDir=/usr/local/zookeeper/tmp
```

并在文件的末尾添加以下代码：

```
server.1=agent1:2888:3888
server.2=agent2:2888:3888
server.3=agent3:2888:3888
```

注：server.1~server.3 是部署 ZooKeeper 的节点。

配置结果如下图所示：

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sake.
dataDir=/usr/local/zookeeper/tmp
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
server.1=agent1:2888:3888
server.2=agent2:2888:3888
server.3=agent3:2888:3888
```

4) 创建 tmp 目录作数据目录。

```
mkdir /usr/local/zookeeper/tmp
```

5) 在 tmp 目录中创建一个空文件，并向该文件写入 ID。

```
touch /usr/local/zookeeper/tmp/myid
echo 1 > /usr/local/zookeeper/tmp/myid
```

步骤 3 同步配置到其它节点

1) 在 agent1 上操作，将配置好的 ZooKeeper 从当前节点拷贝到其它节点。

```
scp -r /usr/local/zookeeper root@agent2:/usr/local
scp -r /usr/local/zookeeper root@agent3:/usr/local
```

2) 分别登录 agent1、agent2、agent3，创建软链接并修改 myid 内容。

- 在 agent1 上执行以下命令：

```
cd /usr/local
echo 1 > /usr/local/zookeeper/tmp/myid
```

- 在 agent2 上执行以下命令：

```
cd /usr/local
echo 2 > /usr/local/zookeeper/tmp/myid
```

- 在 agent3 上执行以下命令：

```
cd /usr/local
echo 3 > /usr/local/zookeeper/tmp/myid
```

步骤 4 运行验证

1) 分别在 agent1，agent2，agent3 上执行下列命令启动 ZooKeeper。


```
cd /usr/local/zookeeper/bin  
./zkServer.sh start
```

2) 当所有 agent 节点上全部启动 Zookeeper 后，在 3 台 agent 节点上可以使用以下命令查看 ZooKeeper 状态。

```
./zkServer.sh status
```

并且，在命令行输入“jps”命令后可以看到有 Jps 和 QuorumPeerMain 两个进程，如下图所示：

```
[i]# jps  
1456 WrapperSimpleApp  
2004 QuorumPeerMain  
2059 Jps
```

1.5 部署 Hadoop-3.1.4

步骤 1 获取 Hadoop-3.1.4 软件包

1) 在 server1 节点的“/usr/local”目录下，获取软件包。

输入以下命令获取软件包：

```
cd /usr/local/  
wget https://repo.huaweicloud.com/apache/hadoop/common/hadoop-3.1.4/hadoop-3.1.4.tar.gz
```

2) 解压 Hadoop 软件包。

输入以下命令解压 Hadoop 软件包：

```
tar -zxvf hadoop-3.1.4.tar.gz  
mv hadoop-3.1.4 hadoop
```

3) 配置 Hadoop 环境变量。

编辑配置文件：

```
vim /etc/profile
```

在/etc/profile 文件的最后添加以下内容，并保存退出。

```
export HADOOP_HOME=/usr/local/hadoop  
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

4) 生效环境变量：

```
source /etc/profile
```

步骤 2 修改 Hadoop 配置文件

Hadoop 所有的配置文件都在\$HADOOP_HOME/etc/hadoop 目录下，修改以下配置文件前，需要切换到“\$HADOOP_HOME/etc/hadoop”目录。

```
cd /usr/local/hadoop/etc/hadoop/
```

1) 修改 hadoop-env.sh。

在 server1 节点上执行以下命令，修改环境变量 JAVA_HOME 为绝对路径，并将用户指定为 root：

```
echo "export JAVA_HOME=/usr/lib/jvm/java" >> hadoop-env.sh
echo "export HDFS_NAMENODE_USER=root" >> hadoop-env.sh
echo "export HDFS_SECONDARYNAMENODE_USER=root" >> hadoop-env.sh
echo "export HDFS_DATANODE_USER=root" >> hadoop-env.sh
```

2) 修改 yarn-env.sh。

在 server1 节点上执行以下命令，修改用户为 root：

```
echo "export YARN_REGISTRYDNS_SECURE_USER=root" >> yarn-env.sh
echo "export YARN_RESOURCEMANAGER_USER=root" >> yarn-env.sh
echo "export YARN_NODEMANAGER_USER=root" >> yarn-env.sh
```

3) 修改 core-site.xml。

- 在节点 server1 上创建目录：

```
mkdir /home/hadoop_tmp_dir
```

- 编辑 core-site.xml 文件：

```
vim core-site.xml
```

- 添加或修改<configuration>标签范围内的参数，在 core-site.xml 文件中编辑以下内容：

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://server1:9000</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/hadoop_tmp_dir</value>
</property>
<property>
  <name>ipc.client.connect.max.retries</name>
  <value>100</value>
</property>
<property>
  <name>ipc.client.connect.retry.interval</name>
  <value>10000</value>
</property>
```

修改结果如下图所示：

```

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://server1:9000</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/hadoop_tmp_dir</value>
</property>
<property>
  <name>ipc.client.connect.max.retries</name>
  <value>100</value>
</property>
<property>
  <name>ipc.client.connect.retry.interval</name>
  <value>10000</value>
</property>
</configuration>

```

4) 修改 hdfs-site.xml。

- 在节点 agent1、agent2、agent3 上分别创建 dfs.datanode.data.dir 对应的目录：

```
mkdir -p /data/data1/hadoop
```

- 在 server1 节点上修改 hdfs-site.xml 文件：

```
vim hdfs-site.xml
```

- 添加或修改<configuration>标签范围内的参数，在 hdfs-site.xml 文件中编辑以下内容：

```

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/data/data1/hadoop/nn</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/data/data1/hadoop/dn</value>
</property>
<property>
  <name>dfs.http.address</name>
  <value>server1:50070</value>
</property>
</property>

```

```
<name>dfs.namenode.http-bind-host</name>
<value>0.0.0.0</value>
</property>
<property>
  <name>dfs.datanode.handler.count</name>
  <value>600</value>
</property>
<property>
  <name>dfs.namenode.handler.count</name>
  <value>600</value>
</property>
<property>
  <name>dfs.namenode.service.handler.count</name>
  <value>600</value>
</property>
<property>
  <name>ipc.server.handler.queue.size</name>
  <value>300</value>
</property>
```

5) 修改 mapred-site.xml。

- 在 server1 节点上编辑 mapred-site.xml 文件：

```
vim mapred-site.xml
```

- 添加或修改<configuration>标签范围内的参数，在 mapred-site.xml 文件中编辑以下内容：

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
  <final>true</final>
  <description>The runtime framework for executing MapReduce jobs</description>
</property>
<property>
  <name>mapreduce.job.reduce.slowstart.completedmaps</name>
  <value>0.88</value>
</property>
<property>
  <name>mapreduce.application.classpath</name>
  <value>
    /usr/local/hadoop/etc/hadoop,
    /usr/local/hadoop/share/hadoop/common/*,
    /usr/local/hadoop/share/hadoop/common/lib/*,
    /usr/local/hadoop/share/hadoop/hdfs/*,
    /usr/local/hadoop/share/hadoop/hdfs/lib/*,
    /usr/local/hadoop/share/hadoop/mapreduce/*,
    /usr/local/hadoop/share/hadoop/mapreduce/lib/*,
    /usr/local/hadoop/share/hadoop/yarn/*,
```

```
    /usr/local/hadoop/share/hadoop/yarn/lib/*
  </value>
</property>
<property>
  <name>mapreduce.map.memory.mb</name>
  <value>6144</value>
</property>
<property>
  <name>mapreduce.reduce.memory.mb</name>
  <value>6144</value>
</property>
<property>
  <name>mapreduce.map.java.opts</name>
  <value>-Xmx5530m</value>
</property>
<property>
  <name>mapreduce.reduce.java.opts</name>
  <value>-Xmx2765m</value>
</property>
<property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx2048m -Xms2048m</value>
</property>
<property>
  <name>mapred.reduce.parallel.copies</name>
  <value>20</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
```

6) 修改 yarn-site.xml。

- 在节点 agent1、agent2、agent3 分别创建\${yarn.nodemanager.local-dirs}对应的目录：

```
mkdir -p /data/data1/yarn
```

- 在 server1 节点上编辑 yarn-site.xml 文件：

```
vim yarn-site.xml
```

- 添加或修改<configuration>标签范围内的参数，在 yarn-site.xml 文件中编辑以下内容：

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
  <final>true</final>
</property>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>server1</value>
</property>
<property>
  <name>yarn.resourcemanager.bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>yarn.scheduler.maximum-allocation-mb</name>
  <value>65536</value>
</property>
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>102400</value>
</property>
<property>
  <name>yarn.nodemanager.resource.cpu-vcores</name>
  <value>48</value>
</property>
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
<property>
  <name>yarn.client.nodemanager-connect.max-wait-ms</name>
  <value>300000</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-pmem-ratio</name>
  <value>7.1</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>false</value>
</property>
<property>
```

```
<name>yarn.nodemanager.pmem-check-enabled</name>
<value>>false</value>
</property>
<property>
  <name>yarn.scheduler.minimum-allocation-mb</name>
  <value>3072</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.resource.mb</name>
  <value>3072</value>
</property>
<property>
  <name>yarn.nodemanager.local-dirs</name>
  <value>/data/data1/yarn </value>
</property>
<property>
  <name>yarn.scheduler.maximum-allocation-vcores</name>
  <value>48</value>
</property>
<property>
  <name>yarn.application.classpath</name>
  <value>
    /usr/local/hadoop/etc/hadoop,
    /usr/local/hadoop/share/hadoop/common/*,
    /usr/local/hadoop/share/hadoop/common/lib/*,
    /usr/local/hadoop/share/hadoop/hdfs/*,
    /usr/local/hadoop/share/hadoop/hdfs/lib/*,
    /usr/local/hadoop/share/hadoop/mapreduce/*,
    /usr/local/hadoop/share/hadoop/mapreduce/lib/*,
    /usr/local/hadoop/share/hadoop/yarn/*,
    /usr/local/hadoop/share/hadoop/yarn/lib/*
  </value>
</property>
```

7) 修改 workers 文件。

在 server1 节点中，修改 workers 文件：

```
vim workers
```

只保存所有 agent 节点的 IP 地址（可用主机名代替），其余内容均删除：

```
agent1
agent2
agent3
```

修改结果如下图所示：

```
agent1
agent2
agent3
█
```

步骤 3 同步配置到其它节点

1) 在所有节点上（包括 server1、agent1、agent2、agent3）分别创建目录。

```
mkdir -p /usr/local/hadoop/journaldata
```

2) 拷贝 server1 节点上的 hadoop 到 agent1、agent2、agent3 节点的 “/usr/local” 目录。

```
scp -r /usr/local/hadoop root@agent1:/usr/local
scp -r /usr/local/hadoop root@agent2:/usr/local
scp -r /usr/local/hadoop root@agent3:/usr/local
```

步骤 4 启动 Hadoop 集群

首次启动 Hadoop 集群需要进行格式化操作（即 2-3 步操作），之后启动集群，只需要执行第 1、4、5、6 步操作即可。

1) 启动 JournalNode。

分别在 agent1，agent2，agent3 节点上执行以下命令启动 JournalNode：

```
cd /usr/local/hadoop/sbin
./hadoop-daemon.sh start journalnode
```

2) 格式化 HDFS。

在 server1 节点上格式化 HDFS：

```
hdfs namenode -format
```

格式化后集群会根据 core-site.xml 配置的 hadoop.tmp.dir 参数生成目录，本文档配置目录为 “/home/hadoop_tmp_dir”。

3) 格式化 ZKFC。

在 server1 节点上格式化 ZKFC：

```
hdfs zkfc -formatZK
```

4) 启动 HDFS。

在 server1 节点上执行以下命令启动 HDFS：

```
cd /usr/local/hadoop/sbin
./start-dfs.sh
```

5) 启动 Yarn。

在 server1 节点上执行以下命令启动 Yarn：

```
cd /usr/local/hadoop/sbin
./start-yarn.sh
```

6) 查看进程是否都正常启动。

在所有节点上执行 jps 命令查看进程是否正常启动。

server 节点应启动的进程见下列 server1 的截图，agent 节点应启动的进程见下列 agent1 的截图：

```
[root@server1 sbin]#
[root@server1 sbin]# jps
6658 Jps
6291 ResourceManager
1747 WrapperSimpleApp
4954 NameNode
5979 SecondaryNameNode
[root@server1 sbin]#
```

```
[root@agent1 sbin]#
[root@agent1 sbin]# jps
7536 DataNode
8321 NodeManager
8547 Jps
7035 QuorumPeerMain
1755 WrapperSimpleApp
7405 JournalNode
[root@agent1 sbin]#
```

步骤 5 验证 Hadoop

在浏览器中输入 URL 地址，访问 Hadoop Web 页面，URL 格式：http://your server ip:50070

通过观察 Live Nodes 是否与 agent 数目相等（本文是 3）、Dead Nodes 是否为 0，判断集群是否正常启动。

Configured Capacity:	47.98 GB
Configured Remote Capacity:	0 B
DFS Used:	24 KB (0%)
Non DFS Used:	21.68 GB
DFS Remaining:	23.94 GB (49.89%)
Block Pool Used:	24 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Fri Aug 27 11:07:18 +0800 2021
Last Checkpoint Time	Fri Aug 27 11:07:18 +0800 2021
Enabled Erasure Coding Policies	RS-6-3-1024k

1.6 部署 Spark

步骤 1 下载并解压 Spark

1) 下载 Spark 软件包。

登录 server1 节点，执行以下命令获取 Spark 软件包：

```
cd /usr/local
wget https://repo.huaweicloud.com/apache/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
```

2) 输入以下命令解压 Spark 软件包：

```
tar -zxvf spark-2.3.2-bin-hadoop2.7.tgz
mv spark-2.3.2-bin-hadoop2.7 spark
```

步骤 2 添加 Spark 到环境变量

依次登录每个 server 和所有 agent 节点，配置环境变量。

1) 编辑 “/etc/profile” 文件：

```
vim /etc/profile
```

2) 在 profile 文件末尾添加以下环境变量：

```
export SPARK_HOME=/usr/local/spark
export PATH=$SPARK_HOME/bin:$SPARK_HOME/sbin:$PATH
```

3) 使环境变量生效：

```
source /etc/profile
```

步骤 3 修改 Spark 配置文件

以下 Spark 配置文件均需要在 server1 节点进行配置。

1) 登录 server1 节点，进入 \$SPARK_HOME/conf 目录：

```
cd $SPARK_HOME/conf
```

2) 修改 spark-env.sh。

- 以 spark-env.sh.template 为模板，拷贝一份命名为 spark-env.sh：

```
cp spark-env.sh.template spark-env.sh
```

- 编辑 spark-env.sh 文件：

```
vim spark-env.sh
```

- 在 spark-env.sh 文件的末尾添加以下三行内容，修改环境变量 JAVA_HOME 为绝对路径，并指定 Hadoop 目录、Spark master 的 IP 和端口号、Spark 所在目录：

```
export JAVA_HOME=/usr/lib/jvm/java
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

3) 修改 spark-defaults.conf 文件：

```
echo "spark.master yarn" >> spark-defaults.conf
echo "spark.eventLog.enabled true" >> spark-defaults.conf
echo "spark.eventLog.dir hdfs://server1:9000/spark2-history" >> spark-defaults.conf
echo "spark.eventLog.compress true" >> spark-defaults.conf
echo "spark.history.fs.logDirectory hdfs://server1:9000/spark2-history" >> spark-defaults.conf
```

修改结果如下图所示：

```
spark.master yarn
spark.eventLog.enabled true
spark.eventLog.dir hdfs://server1:9000/spark2-history
spark.eventLog.compress true
spark.history.fs.logDirectory hdfs://server1:9000/spark2-history
```

4) 创建/spark2-history 目录：

```
hadoop fs -mkdir /spark2-history
```

5) 同步 hadoop 的 core-site.xml 和 hdfs-site.xml:

```
cp /usr/local/hadoop/etc/hadoop/core-site.xml /usr/local/spark/conf
cp /usr/local/hadoop/etc/hadoop/hdfs-site.xml /usr/local/spark/conf
```

步骤 4 同步配置到其它节点

在 server1 节点上, 拷贝 spark 到 agent1、agent2、agent3 的 “/usr/local” 目录:

```
scp -r /usr/local/spark root@agent1:/usr/local
scp -r /usr/local/spark root@agent2:/usr/local
scp -r /usr/local/spark root@agent3:/usr/local
```

步骤 5 运行 yarn-client

在 server1 节点上, 提交任务给 yarn, 进入 spark-shell 模式:

```
spark-shell --master yarn --deploy-mode client
```

1.7 部署算法库

步骤 1 获取算法包及数据集

首先, 使用 ctrl+c 退出的 spark-shell 模式, 以下算法库有关操作仅需要在 server1 节点上进行配置。

1) 通过 obs 下载运行算法所需要的 spark_algorithm.zip 包, 并解压到指定目录 (如 /home) :

```
cd /home
```

使用 wget 命令从 obs 桶中获取所需的算法包:

```
wget https://chatlab.obs.cn-north-4.myhuaweicloud.com:443/spark_algorithm.zip?AccessKeyId=R5ONPGDIP4HMBWO8L06Z&Expires=1697936945&Signature=xxkr4lsmUcF4OUe4eCp6c1JhMkw%3D
```

若 wget 无法下载, 将桶链接复制到浏览器进行下载。spark_algorithm.zip 需要通过 scp 命令上传至 server1 节点的 home 目录。

先使用 cmd 进入到算法包的下载目录, 然后在 cmd 命令行窗口输入以下命令, 将算法包上传至 server1 节点的 home 目录中。输入完 scp 命令后, 需要输入 server1 节点的密码。

```
scp spark_algorithm.zip root@server1 的 ip:/home
```

使用以下命令解压下载好的算法包:

```
unzip spark_algorithm.zip
```

2) 上传算法所需要的数据集:

```
cd spark_algorithm/dataset
hdfs dfs -mkdir -p /tmp/ml/dataset/
```

```
hdfs dfs -mkdir -p /tmp/graph/dataset/
hdfs dfs -put pca_svd_10M1K/ /tmp/ml/dataset/
hdfs dfs -put cit-Patents.txt/ /tmp/graph/dataset/
```

步骤 2 运行算法

进入 Spark-ml-algo-lib 目录下运行机器学习 SVD 算法，对比优化后的运行时长和原生运行时长。

1) 运行机器学习 svd 算法:

```
cd /home/spark_algorithm/
```

2) 运行优化后算法，查看结果:

```
sh bin/ml/svd_run.sh D10M1K
```

运行结果如下图所示:

```
Results have been saved at hdfs:///tmp/ml/output/svd/D10M1K
Exec Successful: costTime: 12.609s
```

3) 运行原生算法，查看结果:

```
sh bin/ml/svd_run_raw.sh D10M1K
```

运行结果如下图所示:

```
Results have been saved at hdfs:///tmp/ml/output/svd/D10M1K
Exec Successful: costTime: 93.083s
```

4) 运行图 PageRank 算法，对比优化后的运行时长和原生运行时长。

运行优化后算法，查看结果:

```
sh bin/graph/pr_run.sh cit_patents run
```

运行结果如下图所示:

```
pagerank costTime = 18.752s
Exec Successful: costTime: 18.752s
```

运行原生算法，查看结果:

```
sh bin/graph/pr_run_raw.sh cit_patents run
```

运行结果如下图所示:

```
pagerank costTime = 257.928s
Exec Successful: costTime: 257.928s
```

步骤 3 调优 Spark 参数

由于现在用到的集群资源比较少，性能结果对比并不可信，需要在集群性能发挥到最大的时候去对比原生算法和优化后算法的性能，所以需要调整 Spark 参数来调优算法的性能。

可调整参数包括:

numExectuors、executorCores、executorMemory、driverCores、driverMemory。

调整原则:

- $\text{numExectuors} * \text{executorCores} + \text{driverCores} < \text{集群 CPU 总核数};$

- $\text{numExecutors} * \text{executorMemory} + \text{driverMemory} < \text{集群总内存}$ 。

若违背了这两个原则，算法将因为资源不够而停在资源分配的步骤上，无法计算出结果。但不是说符合这两个原则资源就一定足够，设置参数时建议留出一定的调整空间。

设置参数的文件所在路径如下：

调整原生和优化后的 svd 的参数：

```
vim conf/ml/svd/svd_spark.properties
```

调整优化后的 PageRank 的参数：

```
vim conf/graph/pr/pr_spark.properties
```

调整原生的 PageRank 的参数：

```
vim conf/graph/pr/pr_spark_raw.properties
```

调优后参数可参考以下配置：

svd_spark.properties:

```
driverCores_D10M1K_aarch_64=3
driverMemory_D10M1K_aarch_64=10G
numExecutors_D10M1K_aarch_64=9
executorCores_D10M1K_aarch_64=10
executorMemory_D10M1K_aarch_64=25G
extraJavaOptions_D10M1K_aarch_64=-Xms25g
execMemOverhead_D10M1K_aarch_64=1G
```

pr_spark.properties:

```
# Spark parameters
deployMode=client

run_cit_patents_driverCores_aarch_64=3
run_cit_patents_driverMemory_aarch_64=10G
run_cit_patents_numExecutors_aarch_64=9
run_cit_patents_executorCores_aarch_64=10
run_cit_patents_executorMemory_aarch_64=25G
run_cit_patents_extraJavaOptions_aarch_64=-Xms25G
run_cit_patents_numPartitions_aarch_64=100
```

pr_spark_raw.properties:

```
# Spark parameters
deployMode=client

run_cit_patents_driverCores=3
run_cit_patents_driverMemory=10G
run_cit_patents_numExecutors=9
run_cit_patents_executorCores=10
run_cit_patents_executorMemory=25G
run_cit_patents_extraJavaOptions=-Xms25G
run_cit_patents_numPartitions=100
~
```

步骤 4 运行参数调优后的算法

运行机器学习 svd 算法：

```
cd /home/spark_algorithm/
```

运行优化后算法及结果：

```
sh bin/ml/svd_run.sh D10M1K
```

运行结果如下图所示：

```
Results have been saved at hdfs:///tmp/ml/output/svd/D10M1K
Exec Successful: costTime: 9.167s
```

运行原生算法及结果：

```
sh bin/ml/svd_run_raw.sh D10M1K
```

运行结果如下图所示：

```
Results have been saved at hdfs:///tmp/ml/output/svd/D10M1K
Exec Successful: costTime: 82.066s
```

2、运行图 PageRank 算法，对比优化后的运行时长和原生运行时长。

运行优化后算法及结果：

```
sh bin/graph/pr_run.sh cit_patents run
```

运行结果如下图所示：

```
pagerank costTime = 15.61s
Exec Successful: costTime: 15.61s
```

运行原生算法及结果：

```
sh bin/graph/pr_run_raw.sh cit_patents run
```

运行结果如下图所示：

```
pagerank costTime = 106.459s
Exec Successful: costTime: 106.459s
```

1.8 思考题

实验过程中用到的服务器配置是否可以更改为 4 核 16G？

答：不可以，在运行参数调优的时候会出现内存不足的错误。