# 分布式系统 Lab4 Hadoop 文档

**杨健邦**

**515030910223**

**yangjianbang112@gmail.com**

## Table of Contents

## 一、项目目录介绍

```
.
├── app
│   ├── dvalueavg
│   └── dvaluesum
├── data
│   ├── device.txt
│   ├── dumpdata.sql
│   ├── dvalues.txt
│   └── output
├── doc
│   └── doc_lab4.docx
├── hadoop-docker
│   ├── README.md
│   ├── docker-compose-template.yml
│   ├── docker-compose.yml
│   ├── hadoop-base
│   ├── hadoop-master
│   └── hadoop-slave
└── scripts
    ├── run_app.sh
    └── sync_to_server.sh

11 directories, 9 files
```

- app 目录下是***两个 Mapreduce app 的程序源码***，分别对应作业要求里的两条查询语句，使用 gradle 进行包管理和编译打包。
- data 目录下是输入数据文件以及***输出数据文件***，还有从 mysql 导出数据的脚本。
- doc 目录下是文档。
- hadoop-docker 目录下是搭建 hadoop 的 docker 集群的相关文件。
- scripts 目录下是两个脚本文件，一个是一键运行 mapreduce app 程序的脚本，另一个是将 项目目录同步到服务器的脚本(docker 跑在服务器上，开发在本地)

# 二、Hadoop 环境的配置

## 1. 综述

此次 Lab 使用 Docker 以及 Docker Compose 来搭建 Hadoop 集群，通过编写 Dockerfile 文件以及 docker-compose.yml 文件来实现集群的自动化搭建和部署。使用的库版本为 ***Hadoop 2.7.6*** 以及 ***Java 8***，使用 ***Yarn*** 框架来进行 Mapreduce。此次共搭建 4 个节点，包含 1 个 master 节点和 3 个 slave 节点，hdfs 使用默认的 3 replica 模式，各个节点的功能和角色如下：

| Node Name(Host Name) | HDFS | Yarn |
|---|---|---|
| **hadoopmaster** | NameNode<br>Secondary NameNode | ResourceManager |
| **hadoopslave1** | DataNode | NodeManager |
| **hadoopslave2** | DataNode | NodeManager |
| **hadoopslave3** | DataNode | NodeManager |

## 2. 搭建过程

此次 Lab 共创建三个 Docker Image，分别为 hadoop-base，hadoop-master 和 hadoop-slave。

### *a. Hadoop-base Image*

hadoop-base 镜像被 hadoop-master 和 hadoop-slave 所共同依赖，其作用是搭建 Hadoop 环境，包括安装 Java 库以及 Hadoop 库，还要配置 ssh 环境，Dockerfile 文件如下:

```
FROM ubuntu:16.04
MAINTAINER Jianbang Yang <yangjianbang112@gmail.com>

# Install openjdk
RUN apt-get update && apt-get install -y software-properties-common && add-apt-
repository ppa:openjdk-r/ppa
RUN apt-get update && apt-get install -y openjdk-8-jdk
ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64

RUN apt-get update && apt-get -y install wget

# Download hadoop. For fast downloading, you can use tsinghua mirror.
# https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/
ENV HADOOP_VERSION 2.7.6
RUN wget -P /tmp/ https://www.apache.org/dist/hadoop/common/hadoop-
$HADOOP_VERSION/hadoop-$HADOOP_VERSION.tar.gz \
    && tar -xvf /tmp/hadoop-$HADOOP_VERSION.tar.gz -C /opt/ \
    && rm /tmp/hadoop-$HADOOP_VERSION.tar.gz*

RUN ln -s /opt/hadoop-$HADOOP_VERSION/etc/hadoop /etc/hadoop
RUN mkdir /opt/hadoop-$HADOOP_VERSION/logs

ENV HADOOP_PREFIX=/opt/hadoop-$HADOOP_VERSION
ENV HADOOP_CONF_DIR=/etc/hadoop
ENV YARN_CONF_DIR=/etc/hadoop

ENV PATH $HADOOP_PREFIX/bin/:$PATH

# SSH connect without key
```

```
RUN apt-get update && apt-get install -y openssh-server \
    && ssh-keygen -t rsa -f ~/.ssh/id_rsa -P '' \
    && cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys \
    && touch ~/.ssh/config
ADD etc/ssh/config /root/.ssh/config
```

第一步：将 ubuntu16:04 镜像作为基本镜像。

第二步：安装 openjdk8。因为 hadoop 是基于 java 的，所以要安装 jdk，这里要注意 hadoop 版本和 jdk 版本要兼容，在这里，hadoop2.7.6 和 jdk8 是兼容的。

第三步：安装 hadoop2.7.6，安装到 opt/hadoop2.7.6 中，并且创建/etc/hadoop 到 hadoop 配置文件目录的软连接，方便以后修改配置。如果官方源下载太慢的话，可以采用清华的镜像。

第四步：设置环境变量 HADOOP_PREFIX，HADOOP_CONF_DIR 和 YARN_CONF_DIR，这里设置环境变量的原因是以后使用 Hadoop 自带的自动脚本的时候需要通过这些环境变量来找到 hadoop 的二进制运行程序。

第五步：安装 ssh-server，并且修改/root/.ssh/config 文件使得 ssh 连接不用检查 ECDSA key，否则使用 Hadoop 自带的启动脚本会失败。这里要注意的一点是，Dockerfile 中的 ADD 命令中的路径要使用绝对路径，ADD 中的~并不代表/root 目录，~代表着一个叫作~的目录。

### *b. Hadoop-master Image*

hadoop-master 镜像是基于 hadoop-base 镜像的，更新了适合 master node 的配置文件和运行脚本， Dockerfile 文件如下:

```
FROM dynamicheart/hadoop-base:1.0.0-hadoop2.7.6-java8
MAINTAINER Jianbang Yang <yangjianbang112@gmail.com>

# Copy configuring data
COPY etc/hadoop/* $HADOOP_CONF_DIR/

RUN mkdir -p /hadoop/dfs/name

ADD run.sh /run.sh
RUN chmod a+x /run.sh

ENTRYPOINT service ssh restart && ./run.sh && bash
```

第一步：将 hadoop-base 作为基本镜像。

第二步：复制配置文件到 image。

第三步：创建 namenode 所需要的数据目录，用于存放 nodename 的数据。

第四步：复制运行脚本到 image。

第五步：设置 entrypoint。entrypoint 首先要开启 ssh 服务，因为 ubuntu 的 docker 镜像是被精简化的，每次创建容器都需要启动 ssh 服务。接着，运行启动脚本，启动 hadoop 集群。最后运行 bash，保持容器处于 alive 的状态，使得不会自动关闭，以便我们之后进行 mapreduce 操作。

Hadoop-master 镜像需要 6 个配置文件，分别是 core-site.xml，hadoop-env.sh，hdfs-site.xml、yarn-site.xml、mapred-site.xml 和 slaves。此外，还添加了一个运行脚本 run.sh。

core-site.xml 中配置了 hdfs 的 URL，使得各个节点都能找得到 hdfs，并且与其它通信，其具体配置如下:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoopmaster:8020</value>
    </property>
</configuration>
```

hdfs-site.xml 的配置告知了 hdfs 中的 namenode 其数据目录在哪，而且为了便于之后测试，关闭了文件权限设置，其具体配置如下：

```
<configuration>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///hadoop/dfs/name</value>
        <description>NameNode directory for namespace and transaction logs
storage.</description>
    </property>

    <property>
      <name>dfs.permissions.enabled</name>
      <value>false</value>
      <description></description>
    </property>
</configuration>
```

yarn-site.xml 的配置设置了 ResourceManager 应该启动在哪个节点，各个 slave 节点也能知道 ResourceManager 在哪，其具体配置如下：

```
<configuration>

<!-- Site specific YARN configuration properties -->
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoopmaster</value>
    </property>

</configuration>
```

hadoop-env.sh 是启动脚本所需要的，其中要设置 JAVA_HOME 为具体路径，否则使用启动脚本启动集群的时候会找不到 JAVA_HOME 变量。

slaves 文件是启动脚本，其中设置了要启动的 slave 节点的 hostname，以便于启动脚本用 ssh 来启动 slave 节点。

```
hadoopslave1
hadoopslave2
hadoopslave3
```

mapred-site.xml 配置了 mapreduce 框架是 yarn

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>

</configuration>
```

run.sh 脚本的功能是格式化 namenode，以及运行启动脚本启动 hdfs，然后再启动 yarn。

```
#!/bin/bash

echo "Hadoop Master start up"

namedir="/hadoop/dfs/name"

if [ "`ls -A $namedir`" == "" ]; then
  echo "Formatting namenode name directory: $namedir"
  $HADOOP_PREFIX/bin/hdfs --config $HADOOP_CONF_DIR namenode -format
$CLUSTER_NAME
fi

$HADOOP_PREFIX/sbin/start-dfs.sh
$HADOOP_PREFIX/sbin/start-yarn.sh
```

### *c. Hadoop-slave Image*

hadoop-slave 镜像也是基于 hadoop-base 镜像的，其 Dockerfile 以及各种配置文件基本和 hadoop-slave 镜像一样。除了 Dockerfile 中要创建的是 datanode 的数据目录而不是 namenode 的数据目录，以及 hdfs-site.xml 中配置 datanode 的目录，不是配置 namenode 的目录。

```
<configuration>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///hadoop/dfs/data</value>
        <description>DataNode directorydescription>
    </property>

    <property>
      <name>dfs.permissions.enabled</name>
      <value>false</value>
      <description></description>
    </property>
</configuration>
```

还有，yarn-site.xml 要增加两项，配置 shuffle 的方式。

```
<configuration>

<!-- Site specific YARN configuration properties -->
    <property>
        <name>yarn.resourcemanager.hostname</name>
```

```
        <value>hadoopmaster</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>

</configuration>
```

### *d. docker-compose.yml*

使用 docker compose 可以自动将上面的 Dockerfile 自动构建成 docker image，并且根据各个 container 的依赖关系，按顺序运行 docker container，docker-compose.yml 文件如下：

```
version: '2'

services:
  hadoopbase:
    build: ./hadoop-base
    image: dynamicheart/hadoop-base:1.0.0-hadoop2.7.6-java8
    container_name: hadoopbase
  hadoopmaster:
    build: ./hadoop-master
    image: dynamicheart/hadoop-master:1.0.0-hadoop2.7.6-java8
    container_name: hadoopmaster
    environment:
      - CLUSTER_NAME=test
      - APP_PATH=/home/workstation/app
      - DATA_PATH=/home/workstation/data
    depends_on:
      - hadoopbase
      - hadoopslave1
      - hadoopslave2
      - hadoopslave3
    volumes:
      - hadoopmaster_name:/hadoop/dfs/name
      - /your/local/workstation/dir:/home/workstation
    stdin_open: true
    tty: true
  hadoopslave1:
    build: ./hadoop-slave
    image: dynamicheart/hadoop-slave:1.0.0-hadoop2.7.6-java8
    container_name: hadoopslave1
    stdin_open: true
    tty: true
    depends_on:
      - hadoopbase
    volumes:
      - hadoopslave2_data:/hadoop/dfs/data
    stdin_open: true
    tty: true
  hadoopslave2:
    build: ./hadoop-slave
```

```
    image: dynamicheart/hadoop-slave:1.0.0-hadoop2.7.6-java8
    container_name: hadoopslave2
    depends_on:
      - hadoopbase
    volumes:
      - hadoopslave2_data:/hadoop/dfs/data
    stdin_open: true
    tty: true
  hadoopslave3:
    build: ./hadoop-slave
    image: dynamicheart/hadoop-slave:1.0.0-hadoop2.7.6-java8
    container_name: hadoopslave3
    depends_on:
      - hadoopbase
    volumes:
      - hadoopslave3_data:/hadoop/dfs/data
    stdin_open: true
    tty: true

volumes:
  hadoopmaster_name:
  hadoopslave1_data:
  hadoopslave2_data:
  hadoopslave3_data:
```

docker-compose.yml 文件有几个编写的细节：

1. 所有的 service 都依赖于 hadoopbase service，这样保证在构建其它 image 的时候，hadoop-base 这个被其它 image 所共同依赖的 image 已经构建成功，依赖关系不出错。

2. 使用 stdin_open:true 和 tty:true 配合 Dockerfile 中 entrypoint 的最后的 bash 命令来保持容器在启动之后能一直保持 alive 的状态而不会自动退出。

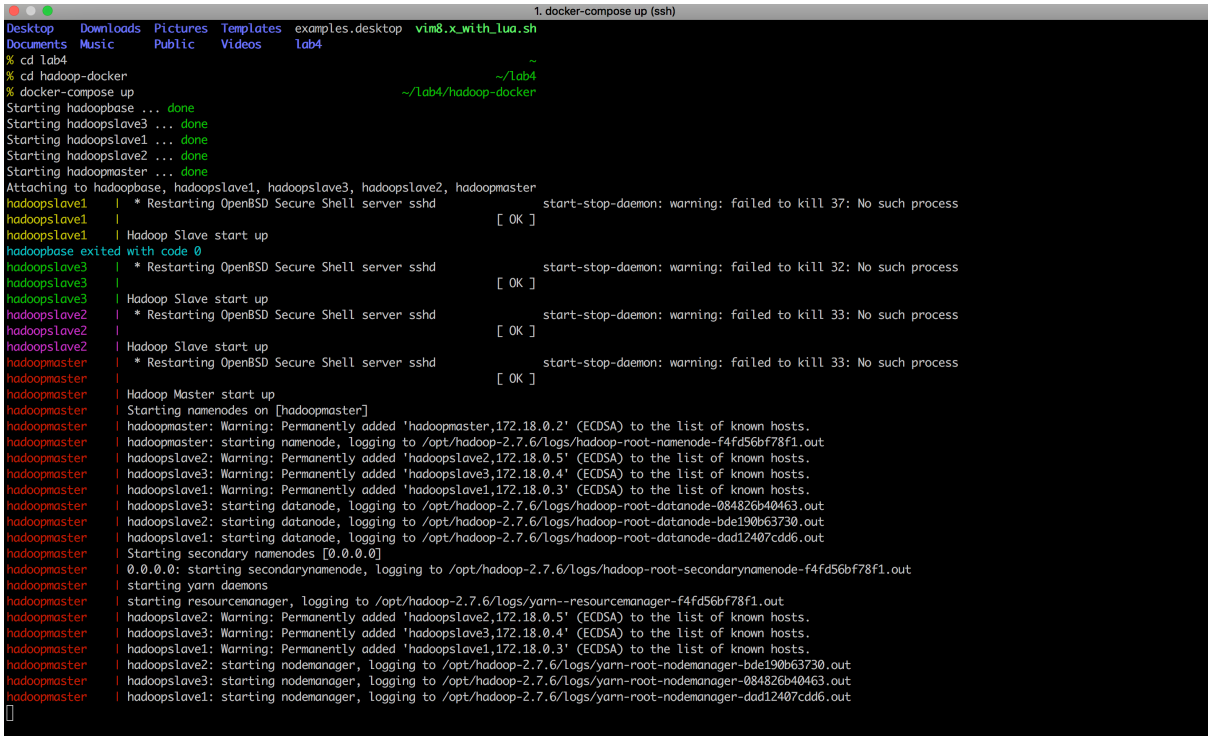
3. hadoopmaster 要依赖于所有的 hadoopslave，这样保持了在 hadoopmaster 运行启动脚本的时候，所有的 slave 节点都是处于启动状态的，只有这样，集群才能成功启动起来。

4. 使用 volume 来持久化 hdfs 的数据，包括 name 数据和 data 数据。

5. 将本地的工作目录(workstation)挂载到 hadoopmaster 容器中，通过这种方式，可以方便读将输入数据以及 map-reduce 程序“上传”到容器内。

## 3. 搭建结果演示

### a. 在 hadoop-docker 目录下运行 docker-compose up 启动集群




### b. 进入 hadoopmaster 容器和其它 hadoopslave 容器检查是否启动成功

```
% docker exec -it hadoopmaster /bin/bash
root@f4fd56bf78f1:/# jps
371 SecondaryNameNode
553 ResourceManager
153 NameNode
877 Jps
root@f4fd56bf78f1:/#
```

```
% docker exec -it hadoopslave1 /bin/bash
root@920614eaf06f:/# jps
177 NodeManager
65 DataNode
324 Jps
root@920614eaf06f:/#
```

预计的线程均启动，启动成功！

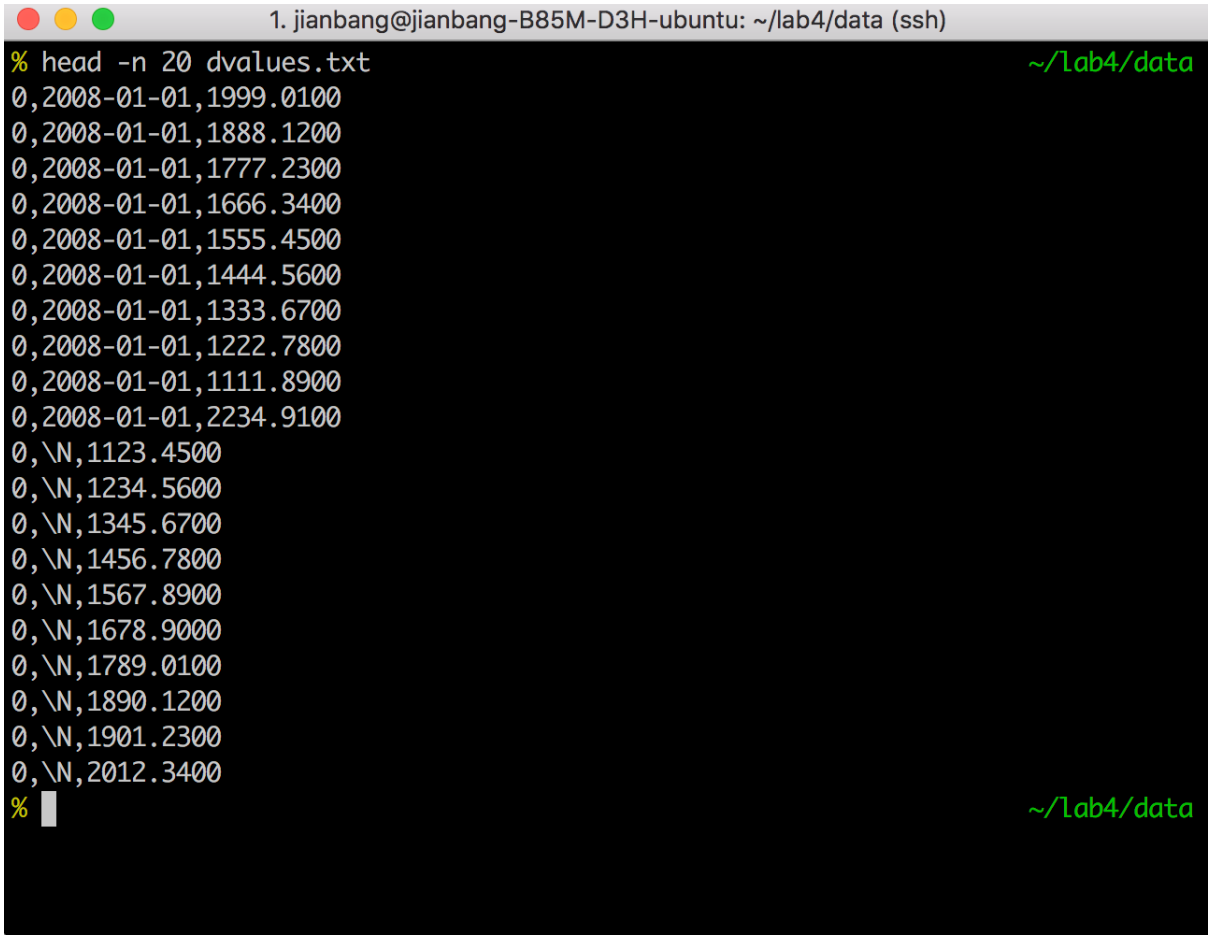
## 三、测试数据的导出

先将数据导入 MYSQL 数据库中，再运行如下两个 SQL 脚本，即可将数据导出成文本模式：

```
select * from device into outfile '/var/lib/mysql-files/device.txt'
fields terminated by ','
optionally enclosed by ''
lines terminated by '\n';

select * from dvalues into outfile '/var/lib/mysql-files/dvalues.txt'
fields terminated by ','
optionally enclosed by ''
lines terminated by '\n';
```

每一张表导出成一个文件，每一行为一个 Record，Record 的 Field 之间用**逗号**分隔，为 NULL 的 Field 在文件中用**\N** 表示：



# 四、MapReduce 程序的设计

## 1. 整体设计

两个查询都是需要先 join 然后再进行聚集操作，有两种做法。

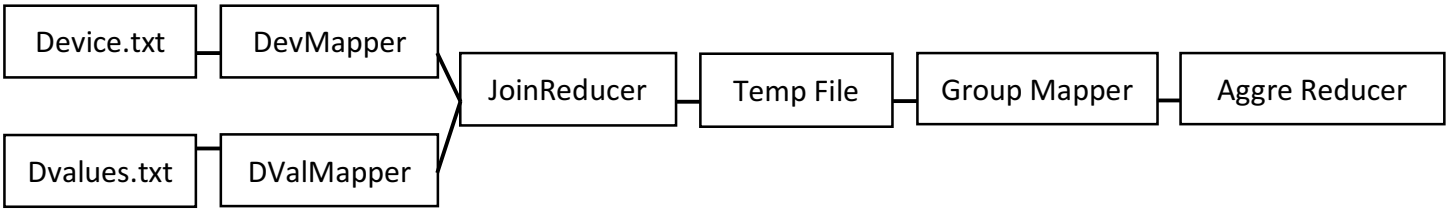
第一种方法是，在 Map 端 Join，即将较小表 device 的数据读进内存中，在 map 的时候一同 join，筛选数据，然后在 reduce 的时候聚集。相当于在 map 端处理 from，where 和 group 语句，在 reduce 端处理 select 和 order 语句。

第二种方法是，在 Reduce 端 Join，进行两次 mapreduce，第一次 mapreduce 进行 join 操作，产生中间文件，第二次 mapreduce 对 join 的结果进行聚集操作。

Device 表有一万条数据，而 DValues 表有二十万条数据，第一种方法虽然快，但是并不具有普适性，数据量如果增多，便不适用。***因此，采用第二种方法。***

***因为有两个输入文件，因此第一次 mapreduce 过程中需要两个 Mapper 类，但共用同一个 Reduce 类。***

## 2. 实现细节




1. 输入数据是两个文件（两张表），每个文件的格式都不一样，因此需要两类 Mapper 来进行处理，Mapper 的输出打上 tag，来标注其来自哪张表，然后交由同一类的 Reducer 进行处理。使用 MultipleInputs 类将 mapper 和输入文件绑定。

```
MultipleInputs.addInputPath(joinJob, new Path(args[0]), TextInputFormat.class, DeviceJoinMapper.class);
MultipleInputs.addInputPath(joinJob, new Path(args[1]), TextInputFormat.class, DValueJoinMapper.class);
```

2. DevMapper 和 DvalMapper 相当于从 where 中筛选出有用数据，比如根据 1<=did<=1000 或者 date is null 等筛选数据，output Key 为 did，output value 中还要加上一个 table 的 tag 来注明表明这个数据来自于哪个 table，以便 JoinReducer 进行对两表进行 join 操作。

3. JoinReducer 根据 value 中的 tag，将 values 分离成两组，然后进行 join 操作。生成中间表存在 Temp File 中。

```
for (String[] deviceFields: deviceFieldsList) {
    for(String[] dValuesFields: dValuesFieldsList) {
        outputKey.set(dValuesFields[1] + DELIMITER + deviceFields[1]);
        outputValue.set(dValuesFields[2]);
        context.write(outputKey, outputValue);
    }
}
```

4. GroupMapper 读入中间文件，通过设置 output Key 来分好组，然后传给 Aggre Reducer。Aggre Reducer 进行聚集操作和排序操作，产生结果文件。

**5. 排序操作是创建一个 DescendingKeyComparator 类，将比较的结果*(-1)，然后调用 Aggre Reducer 的 setSortComparatorClass(DescendingKeyComparator.class)，即可在最后一步 recude 的过程中排序，而不需要另外一个新的的 mapreduce 操作。**

```
public static class DescendingKeyComparator extends WritableComparator {
    protected DescendingKeyComparator() {
        super(Text.class, true);
    }

    @SuppressWarnings("rawtypes")
    @Override
    public int compare(WritableComparable w1, WritableComparable w2) {
        Text key1 = (Text) w1;
        Text key2 = (Text) w2;
        String keyfields1[] = key1.toString().split(DELIMITER);
        String keyfields2[] = key2.toString().split(DELIMITER);
        if (keyfields1[0].equals(keyfields2[0])) {
            return keyfields1[1].compareTo(keyfields2[1]);
        } else {
            return -1 * keyfields1[0].compareTo(keyfields2[0]);
        }
    }
}
```

**6. 对于空值匹配，只有 DValues 表中的 date 项可能会有空值存在。在导出的 txt 文件中，空值用字符串“\N”来代替，可以用字符串匹配的方法来判断空值。**

**7. 对于空值聚集，查询 1 不存在空值聚集的情况，而查询 2，因为使用了 left join，因此需要考虑空值聚集的情况， 由于聚集函数 sum 和 avg 都不考虑空值，因此可以在 JoinReducer 的时候丢弃聚集项为空的 Record(通过 for 循环，不会由于 left join 导致产生空值的 record)。**

```
for (String[] deviceFields: deviceFieldsList) {
    for(String[] dValuesFields: dValuesFieldsList) {
        outputKey.set(dValuesFields[1] + DELIMITER + deviceFields[1]);
        outputValue.set(dValuesFields[2]);
        context.write(outputKey, outputValue);
    }
}
```

**8. 由于一个完成的程序要经过两次 Mapreduce，第二次 mapreduce 要等到第一次 mapreduce 结束之后才能开始，因此使用 job.waitForCompletion(true)来阻塞，直到第一次 mapreduce 结束时候才会执行第二次 mapreduce。**

## 3. 结果演示

运行 scripts 目录下的 run_app.sh 脚本，即可运行两个 mapreduce 程序并且显示结果。HDFS 启动时会进入 safe mode，大概 30s 后才会退出 safe mode，这时候才能进行正常测试。

**a. 查询一结果：**

下图分别为不使用 Yarn 框架的 Mapreduce 程序、使用 Yarn 框架的 Mapreduce 程序和 mysql 查询的结果， 他们结果一致（精确度不一样，但是结果是相同的）。由于筛选集大，不使用 YARN 框架的 mapreduce 程序的耗时比 mysql 查询要少(real 的时间比 user 的少是因为并行操作)，使用 YARN 框架的 mapreduce 程序则会非常慢，可能是框架太过于庞大的原因，对于数据集太少的 mapreduce，overhead 太高。

```
real    0m3.735s
user    0m5.872s
sys     0m0.304s
[Hadoop APP Controller]: Dvalue sum app ends successfully
[Hadoop APP Controller]: Print the result of dvalue sum app
ZZZZZ   4850278.099999994
YYYYY   4829898.099999995
XXXXX   4829518.099999996
WWWWW   4809138.099999995
VVVVV   4808758.099999996
UUUUU   4788378.099999996
TTTTT   4787998.099999996
SSSSS   4767618.099999996
RRRRR   4767238.099999995
QQQQQ   4746858.099999994
PPPPP   4746478.099999995
OOOOO   4726098.099999997
NNNNN   4725718.099999995
MMMMM   4715338.099999995
LLLLL   4940948.049999998
KKKKK   4930558.049999997
JJJJJ   4930168.049999996
IIIII   4919778.049999995
HHHHH   4919388.049999996
GGGGG   4908998.049999999
FFFFF   4908608.049999998
EEEEE   4898218.049999997
DDDDD   4897828.049999996
CCCCC   4887438.049999998
BBBBB   4887048.049999998
AAAAA   4850658.099999996
```

```
real    0m33.157s
user    0m4.309s
sys     0m0.204s
[Hadoop APP Controller]: Dvalue sum app ends successfully
[Hadoop APP Controller]: Print the result of dvalue sum app
ZZZZZ   4850278.099999994
YYYYY   4829898.099999995
XXXXX   4829518.099999996
WWWWW   4809138.099999995
VVVVV   4808758.099999996
UUUUU   4788378.099999996
TTTTT   4787998.099999996
SSSSS   4767618.099999996
RRRRR   4767238.099999995
QQQQQ   4746858.099999994
PPPPP   4746478.099999995
OOOOO   4726098.099999997
NNNNN   4725718.099999995
MMMMM   4715338.099999995
LLLLL   4940948.049999998
KKKKK   4930558.049999997
JJJJJ   4930168.049999996
IIIII   4919778.049999995
HHHHH   4919388.049999996
GGGGG   4908998.049999999
FFFFF   4908608.049999998
EEEEE   4898218.049999997
DDDDD   4897828.049999996
CCCCC   4887438.049999998
BBBBB   4887048.049999998
AAAAA   4850658.099999996
```

```
mysql> select type,sum(value) from device, dvalues
    -> where id = did and did > 0 and did < 1000 and date is null
    -> group by type order by type desc;
+-------+-------------+
| type  | sum(value)  |
+-------+-------------+
| ZZZZZ | 4850278.1000 |
| YYYYY | 4829898.1000 |
| XXXXX | 4829518.1000 |
| WWWWW | 4809138.1000 |
| VVVVV | 4808758.1000 |
| UUUUU | 4788378.1000 |
| TTTTT | 4787998.1000 |
| SSSSS | 4767618.1000 |
| RRRRR | 4767238.1000 |
| QQQQQ | 4746858.1000 |
| PPPPP | 4746478.1000 |
| OOOOO | 4726098.1000 |
| NNNNN | 4725718.1000 |
| MMMMM | 4715338.1000 |
| LLLLL | 4940948.0500 |
| KKKKK | 4930558.0500 |
| JJJJJ | 4930168.0500 |
| IIIII | 4919778.0500 |
| HHHHH | 4919388.0500 |
| GGGGG | 4908998.0500 |
| FFFFF | 4908608.0500 |
| EEEEE | 4898218.0500 |
| DDDDD | 4897828.0500 |
| CCCCC | 4887438.0500 |
| BBBBB | 4887048.0500 |
| AAAAA | 4850658.1000 |
+-------+-------------+
26 rows in set (4.14 sec)
```

**b. 查询二结果：**

下图分别为不使用 Yarn 框架的 Mapreduce 程序、使用 Yarn 框架的 Mapreduce 程序和 mysql 查询的结果，结果一致（精确度不一样，但是结果是相同的）。很明显 Mysql 比不使用 YARN 框架 mapreduce 程序用时少，因为筛选集小了很多。(real 的时间比 user 的少是因为并行操作)，而使用 Yarn 框架的 Mapreduce 程序仍然是最慢的。

```
real    0m3.756s
user    0m5.226s
sys     0m0.333s
[Hadoop APP Controller]: Dvalue avg app ends successfully
[Hadoop APP Controller]: Print the result of dvalue avg app
2008-01-04      IIIII   2631.3959999999997
2008-01-04      JJJJJ   2632.3959999999997
2008-01-03      DDDDD   2626.3959999999997
2008-01-03      EEEEE   2627.3959999999997
2008-01-03      FFFFF   2628.3959999999997
2008-01-03      GGGGG   2629.3959999999997
2008-01-03      HHHHH   2630.3959999999997
2008-01-02      BBBBB   2624.396
2008-01-02      CCCCC   2625.3959999999997
root@1cb2fcd49699:/home/workstation/scripts#
```

```
real    0m33.442s
user    0m4.163s
sys     0m0.266s
[Hadoop APP Controller]: Dvalue avg app ends successfully
[Hadoop APP Controller]: Print the result of dvalue avg app
2008-01-04      IIIII   2631.3959999999997
2008-01-04      JJJJJ   2632.3959999999997
2008-01-03      DDDDD   2626.3959999999997
2008-01-03      EEEEE   2627.3959999999997
2008-01-03      FFFFF   2628.3959999999997
2008-01-03      GGGGG   2629.3959999999997
2008-01-03      HHHHH   2630.3959999999997
2008-01-02      BBBBB   2624.396
2008-01-02      CCCCC   2625.3959999999997
```

```
mysql> select date, type, avg(value) from device left join dvalues on id = did
    -> where did > 0 and did < 10 and date is not null
    -> group by date, type order by date desc, type;
+------------+-------+--------------+
| date       | type  | avg(value)   |
+------------+-------+--------------+
| 2008-01-04 | IIIII | 2631.39600000 |
| 2008-01-04 | JJJJJ | 2632.39600000 |
| 2008-01-03 | DDDDD | 2626.39600000 |
| 2008-01-03 | EEEEE | 2627.39600000 |
| 2008-01-03 | FFFFF | 2628.39600000 |
| 2008-01-03 | GGGGG | 2629.39600000 |
| 2008-01-03 | HHHHH | 2630.39600000 |
| 2008-01-02 | BBBBB | 2624.39600000 |
| 2008-01-02 | CCCCC | 2625.39600000 |
+------------+-------+--------------+
9 rows in set (0.11 sec)
```

# 五、Hadoop 集群容错验证

## 测试 Yarn 的框架的容错性

**实验目的：**验证 Hadoop 集群单 Slave 节点失效容错性

**实验预期结果：**由于 HDFS 是多备份，Yarn 也有容错机制来实时监听各个 slave 节点，因此，HDFS 仍能继续读写数据，而 Yarn 可以重新规划分配计算任务，最后产生正确结果。

**实验配置：**1 个 Master Node，角色是 ResourceManager 和 NameNode。3 个 Slave Node，角色是 NodeManager 和 DataNode。HDFS 使用 3 备份模式。使用的程序是 hadoop 官方 wordcount 样例程序。使用的数据集是 https://archive.ics.uci.edu/ml/datasets/Bag+of+Words。

**测试方法：找一个几百兆大小的文件，这样才能使得文件会分成多个 split，然后跑在多个 Slave 上，**在运行 mapreduce 程序的时候，关掉其中一个 Slave Node，查看 Log 和 mapreduce 的结果，判断 mapreduce 能否继续进行以及产生正确的结果。

**测试辅助脚本：**scripts 目录下 download_dataset.sh 和 run_wordcount.sh

**实验过程：**

**步骤零：**配置 Log 程序。由于 hadoop 框架配备了 log 相关的应用，timelineserver 以及 historyserver，它们能让我们更好地浏览 Log。因此需要修改配置文件来启动 timelineserver 和 historyserver。

a. 在 yarn-site.xml 里面增加如下条目来配置 timelineserver，其中 yarn.log-aggregation-enable 条目是使 log 存放到 hdfs 中。相关条目的所代表的含义可以参考 https://www.jianshu.com/p/e73e6ba60668 和 http://www.aboutyun.com/thread-16690-1-1.html

```
    <property>
        <name>yarn.log-aggregation-enable</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.nodemanager.remote-app-log-dir</name>
        <value>app-logs</value>
    </property>
    <property>
        <name>yarn.log.server.url</name>
        <value>http://hadoopmaster:8188/applicationhistory/logs/</value>
    </property>

    <property>
        <name>yarn.timeline-service.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.timeline-service.generic-application-
history.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.resourcemanager.system-metrics-publisher.enabled</name>
        <value>true</value>
    </property>
```

```
    <property>
        <name>yarn.timeline-service.hostname</name>
        <value>hadoopmaster</value>
    </property>


    <property>
        <name>yarn.timeline-service.leveldb-timeline-store.path</name>
        <value>/hadoop/yarn/timeline</value>
    </property>
```

b. 在 mapred-site.xml 里面添加如下条目，用于配置 historyserver。

```
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>hadoopmaster:10020</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>hadoopmaster:19888</value>
    </property>
```

c. 在启动 master 脚本 run.sh 里面添加如下条目，用于启动 timelineserver 和 history server。

```
$HADOOP_PREFIX/sbin/yarn-daemon.sh start timelineserver
$HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh start historyserver
```

d. 此外还需要添加一个 Volume 来持久化 Log，并且要映射某些容器端口到主机端口，使得我们能够从浏览器来浏览 Log 记录。

**步骤一**：从 https://archive.ics.uci.edu/ml/machine-learning-databases/bag-of-words/docword.nytimes.txt.gz 中下载目标数据集，解压后的大小为 958M。

**步骤二**：上传数据集到 hdfs。

```
root@caff0c32c916:/home/workstation# hdfs dfs -ls /
Found 1 items
-rw-r--r--   3 root supergroup 1004435682 2018-06-04 12:53 /docword.nytimes.txt
```

**步骤三：**运行官方样例 Wordcount Mapreduce，得到正确结果。

```
root@caff0c32c916:/home/workstation# ./scripts/run_wordcount.sh /docword.nytimes.txt /out/baseline
rm: `/out/baseline': No such file or directory
18/06/04 12:54:59 INFO client.RMProxy: Connecting to ResourceManager at hadoopmaster/172.18.0.5:8032
18/06/04 12:54:59 INFO input.FileInputFormat: Total input paths to process : 1
18/06/04 12:55:00 INFO mapreduce.JobSubmitter: number of splits:8
18/06/04 12:55:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1528116799818_0001
18/06/04 12:55:00 INFO impl.YarnClientImpl: Submitted application application_1528116799818_0001
18/06/04 12:55:00 INFO mapreduce.Job: The url to track the job: http://hadoopmaster:8088/proxy/application_1528116799818_0001/
18/06/04 12:55:00 INFO mapreduce.Job: Running job: job_1528116799818_0001
18/06/04 12:55:06 INFO mapreduce.Job: Job job_1528116799818_0001 running in uber mode : false
18/06/04 12:55:06 INFO mapreduce.Job:  map 0% reduce 0%
18/06/04 12:55:19 INFO mapreduce.Job:  map 11% reduce 0%
18/06/04 12:55:22 INFO mapreduce.Job:  map 13% reduce 0%
18/06/04 12:55:25 INFO mapreduce.Job:  map 20% reduce 0%
18/06/04 12:55:28 INFO mapreduce.Job:  map 21% reduce 0%
18/06/04 12:55:31 INFO mapreduce.Job:  map 29% reduce 0%
18/06/04 12:55:37 INFO mapreduce.Job:  map 38% reduce 0%
18/06/04 12:55:40 INFO mapreduce.Job:  map 42% reduce 0%
18/06/04 12:55:43 INFO mapreduce.Job:  map 49% reduce 0%
18/06/04 12:55:49 INFO mapreduce.Job:  map 56% reduce 0%
18/06/04 12:55:52 INFO mapreduce.Job:  map 56% reduce 4%
18/06/04 12:55:55 INFO mapreduce.Job:  map 63% reduce 4%
18/06/04 12:56:01 INFO mapreduce.Job:  map 68% reduce 4%
18/06/04 12:56:04 INFO mapreduce.Job:  map 70% reduce 4%
18/06/04 12:56:07 INFO mapreduce.Job:  map 79% reduce 4%
18/06/04 12:56:08 INFO mapreduce.Job:  map 92% reduce 4%
18/06/04 12:56:09 INFO mapreduce.Job:  map 100% reduce 4%
```

**步骤四：**运行官方样例 Wordcount mapreduce，在运行到一半的时候，运行 docker container kill 命令，关掉其中一个 Slave 节点 。

```
root@caff0c32c916:/home/workstation# ./scripts/run_wordcount.sh /docword.nytimes.txt /out/fault_slave1
rm: `/out/fault_slave1': No such file or directory
18/06/04 12:56:47 INFO client.RMProxy: Connecting to ResourceManager at hadoopmaster/172.18.0.5:8032
18/06/04 12:56:48 INFO input.FileInputFormat: Total input paths to process : 1
18/06/04 12:56:48 INFO mapreduce.JobSubmitter: number of splits:8
18/06/04 12:56:48 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1528116799818_0002
18/06/04 12:56:48 INFO impl.YarnClientImpl: Submitted application application_1528116799818_0002
18/06/04 12:56:48 INFO mapreduce.Job: The url to track the job: http://hadoopmaster:8088/proxy/application_1528116799818_0002/
18/06/04 12:56:48 INFO mapreduce.Job: Running job: job_1528116799818_0002
18/06/04 12:56:52 INFO mapreduce.Job: Job job_1528116799818_0002 running in uber mode : false
18/06/04 12:56:52 INFO mapreduce.Job:  map 0% reduce 0%
18/06/04 12:57:04 INFO mapreduce.Job:  map 1% reduce 0%
18/06/04 12:57:05 INFO mapreduce.Job:  map 11% reduce 0%
18/06/04 12:57:07 INFO mapreduce.Job:  map 12% reduce 0%
18/06/04 12:57:08 INFO mapreduce.Job:  map 14% reduce 0%
18/06/04 12:57:11 INFO mapreduce.Job:  map 19% reduce 0%
18/06/04 12:57:14 INFO mapreduce.Job:  map 21% reduce 0%
18/06/04 12:57:17 INFO mapreduce.Job:  map 26% reduce 0%
18/06/04 12:57:20 INFO mapreduce.Job:  map 30% reduce 0%
18/06/04 12:57:23 INFO mapreduce.Job:  map 32% reduce 0%
18/06/04 12:57:26 INFO mapreduce.Job:  map 38% reduce 0%
```

```
% docker container kill hadoopslave1
hadoopslave1
% docker ps
CONTAINER ID        IMAGE                                          COMMAND                  CREATED             STATUS              PORTS               NAMES
caff0c32c916        dynamicheart/hadoop-master:1.0.0-hadoop2.7.6-java8   "/bin/sh -c 'service…"   4 minutes ago       Up 4 minutes                            hadoopmaster
dd5f540edd66        dynamicheart/hadoop-slave:1.0.0-hadoop2.7.6-java8    "/bin/sh -c 'service…"   4 minutes ago       Up 4 minutes                            hadoopslave2
3ba5e04ea1c2        dynamicheart/hadoop-slave:1.0.0-hadoop2.7.6-java8    "/bin/sh -c 'service…"   4 minutes ago       Up 4 minutes                            hadoopslave3
```

```
hadoopslave1    | root@a1981b2e1504:/# hadoopslave1 exited with code 137
```

**步骤五：**观察是否能得到结果，并且将此结果和正确结果比较，看结果是否一致。从结果来看，即使 kill 掉了 slave1，wordcount 程序还是能够成功跑出结果的(_SUCCESS)。

```
root@caff0c32c916:/home/workstation# hdfs dfs -get /out/fault_slave1
root@caff0c32c916:/home/workstation# ls
app  baseline  data  doc  docword  fault_slave1  hadoop-docker  scripts  tmp
root@caff0c32c916:/home/workstation# cd fault_slave1/
root@caff0c32c916:/home/workstation/fault_slave1# ls
_SUCCESS  part-r-00000
```

对比两个文件，两个文件没有差异，说明 Mapreduce 程序得到了正确执行。

```
root@caff0c32c916:/home/workstation# diff -s fault_slave1/part-r-00000 baseline/part-r-00000
Files fault_slave1/part-r-00000 and baseline/part-r-00000 are identical
root@caff0c32c916:/home/workstation#
```

**步骤六：**分析 Log 的内容。

a. 访问 historyServer（端口 8188）的 WEB UI，可以发现 mapreduce 程序都执行成功了 ，下面三次运行程序的 log，从下到上分别代表①正常运行的程序的 log ②“运行到一半有一个 slave 被 kill 掉的程序”的 log ③“运行到一半有两个 slave 被 kill 掉的程序”的 log。

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| application_1528179580532_0001 | root | word count | MAPREDUCE | default | Tue Jun 5 14:20:28 +0800 2018 | Tue Jun 5 14:34:29 +0800 2018 | FINISHED | SUCCEEDED | History |
| application_1528178508022_0002 | root | word count | MAPREDUCE | default | Tue Jun 5 14:16:44 +0800 2018 | Tue Jun 5 14:18:45 +0800 2018 | FINISHED | SUCCEEDED | History |
| application_1528178508022_0001 | root | word count | MAPREDUCE | default | Tue Jun 5 14:11:10 +0800 2018 | Tue Jun 5 14:12:20 +0800 2018 | FINISHED | SUCCEEDED | History |

b. 首先看正常运行的程序的 log。只进行了一次 appattempt，而且 container 的退出状态均一致，为-105，这里的状态码显示可能有点问题，进到 appattempt 的 log(实际上是 application master 的 log)查看，发现 container 是被 application master kill 掉的，属于正常退出的一种方式。因此-105 状态码代表的是正常退出。

Show 20 entries Search:

| Attempt ID | Started | Node | Logs |
|---|---|---|---|
| appattempt_1528178508022_0001_000001 | Tue Jun 5 14:11:11 +0800 2018 | http://e27a288e6243:8042 | Logs |

Showing 1 to 1 of 1 entries First Previous 1 Next Last

Show 20 entries Search:

| Container ID | Node | Container Exit Status | Logs |
|---|---|---|---|
| container_1528178508022_0001_01_000012 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528178508022_0001_01_000011 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0001_01_000010 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000009 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000008 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000007 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000006 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000005 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000004 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000003 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000002 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528178508022_0001_01_000001 | http://e27a288e6243:8042 | 0 | Logs |

Showing 1 to 12 of 12 entries First Previous 1 Next Last

```
2018-06-05 06:12:16,303 INFO [AsyncDispatcher event handler] org.apache.hadoop.mapreduce.v2.app.job.impl.JobImpl: Num completed Tasks: 4
2018-06-05 06:12:16,427 INFO [IPC Server handler 6 on 35243] org.apache.hadoop.mapred.TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1528178508022_0001_m_000001_0 is : 0.6377456
2018-06-05 06:12:16,492 INFO [IPC Server handler 8 on 35243] org.apache.hadoop.mapred.TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1528178508022_0001_m_000001_0 is : 0.667
2018-06-05 06:12:16,665 INFO [IPC Server handler 9 on 35243] org.apache.hadoop.mapred.TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1528178508022_0001_m_000000_1 is : 0.23863098
2018-06-05 06:12:16,749 INFO [IPC Server handler 21 on 35243] org.apache.hadoop.mapred.TaskAttemptListenerImpl: MapCompletionEvents request from attempt_1528178508022_0001_r_000000_0. startIndex
2018-06-05 06:12:16,951 INFO [IPC Server handler 17 on 35243] org.apache.hadoop.mapred.TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1528178508022_0001_m_000000_0 is : 0.6419763
2018-06-05 06:12:17,010 INFO [RMCommunicator Allocator] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Before Scheduling: PendingReds:0 ScheduledMaps:0 ScheduledReds:0 AssignedMaps:
2018-06-05 06:12:17,013 INFO [RMCommunicator Allocator] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Received completed container container_1528178508022_0001_01_000004
2018-06-05 06:12:17,013 INFO [RMCommunicator Allocator] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: After Scheduling: PendingReds:0 ScheduledMaps:0 ScheduledReds:0 AssignedMaps:6
2018-06-05 06:12:17,013 INFO [AsyncDispatcher event handler] org.apache.hadoop.mapreduce.v2.app.job.impl.TaskAttemptImpl: Diagnostics report from attempt_1528178508022_0001_m_000002_0: Container
Container killed on request. Exit code is 143
Container exited with a non-zero exit code 143
```

c. 接着看“运行到一半有一个 slave 被 kill 掉的程序”的 log，这里出现了两个 exit status 为-100 的 container，而这两个 container 都属于同一个节点，应该是被 kill 掉的 slave 节点，而其它节点的 container 的 exit status 均为-105，说明其它节点都正常。由于 ApplicationMaster 仍然处于 alive 状态，因此虽然有部分 container 失效了，但是 mapreduce 不需要重新开始做。

Show 20 entries Search:

| Attempt ID | Started | Node | Logs |
|---|---|---|---|
| appattempt_1528178508022_0002_000001 | Tue Jun 5 14:16:45 +0800 2018 | http://20dc85fe0710:8042 | Logs |

Showing 1 to 1 of 1 entries First Previous 1 Next Last

Show 20 entries Search:

| Container ID | Node | Container Exit Status | Logs |
|---|---|---|---|
| container_1528178508022_0002_01_000012 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000011 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000010 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000009 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528178508022_0002_01_000008 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528178508022_0002_01_000007 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000006 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000005 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000004 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000003 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000002 | http://20dc85fe0710:8042 | -105 | Logs |
| container_1528178508022_0002_01_000001 | http://20dc85fe0710:8042 | 0 | Logs |

Showing 1 to 12 of 12 entries First Previous 1 Next Last

d. 由上图可以知道 container8 和 container9 失效了。Attempt7_0 被分配给了 container9，由于 attempt7_0 长时间没有完成，因此 ApplicationMaster 重新分配了一个相同的 attempt7_1 给 container12，当 attempt7_1 完成之后，kill 掉 attempt7_0。用这种方法来容错。

```
rAllocator: Assigned container container_1528178508022_0002_01_000002 to attempt_1528178508022_0002_m_000000_0
rAllocator: Assigned container container_1528178508022_0002_01_000003 to attempt_1528178508022_0002_m_000001_0
rAllocator: Assigned container container_1528178508022_0002_01_000004 to attempt_1528178508022_0002_m_000002_0
rAllocator: Assigned container container_1528178508022_0002_01_000005 to attempt_1528178508022_0002_m_000003_0
rAllocator: Assigned container container_1528178508022_0002_01_000006 to attempt_1528178508022_0002_m_000004_0
rAllocator: Assigned container container_1528178508022_0002_01_000007 to attempt_1528178508022_0002_m_000005_0
rAllocator: Assigned container container_1528178508022_0002_01_000008 to attempt_1528178508022_0002_m_000006_0
rAllocator: Assigned container container_1528178508022_0002_01_000009 to attempt_1528178508022_0002_m_000007_0
rAllocator: Recalculating schedule, headroom=<memory:14336, vCores:1>
rAllocator: Reduce slow start threshold not met. completedMapsForReduceSlowstart 1
```

```
27a288e6243 to /default-rack
)dc85fe0710 to /default-rack
:1df6a3673d to /default-rack
nptImpl: attempt_1528178508022_0002_m_000007_1 TaskAttempt Transitioned from NEW to UNASSIGNED
r: Before Scheduling: PendingReds:0 ScheduledMaps:1 ScheduledReds:0 AssignedMaps:3 AssignedReds:1 CompletedMaps:6 CompletedR
r: getResources() for application_1528178508022_0002: ask=5 release= 0 newContainers=0 finishedContainers=0 resourcelimit=<m
apCompletionEvents request from attempt_1528178508022_0002_r_000000_0. startIndex 6 maxEvents 10000
r: Got allocated containers 1
r: Assigned container container_1528178508022_0002_01_000012 to attempt_1528178508022_0002_m_000007_1
r: After Scheduling: PendingReds:0 ScheduledMaps:0 ScheduledReds:0 AssignedMaps:4 AssignedReds:1 CompletedMaps:6 CompletedRe
)dc85fe0710 to /default-rack
nptImpl: attempt_1528178508022_0002_m_000007_1 TaskAttempt Transitioned from UNASSIGNED to ASSIGNED
Impl: Processing the event EventType: CONTAINER_REMOTE_LAUNCH for container container_1528178508022_0002_01_000012 taskAttem
Impl: Launching attempt_1528178508022_0002_m_000007_1
tocolProxy: Opening proxy : 20dc85fe0710:45443
Impl: Shuffle port returned by ContainerManager for attempt_1528178508022_0002_m_000007_1 : 13562
nptImpl: TaskAttempt: [attempt_1528178508022_0002_m_000007_1] using containerId: [container_1528178508022_0002_01_000012 on
nptImpl: attempt_1528178508022_0002_m_000007_1 TaskAttempt Transitioned from ASSIGNED to RUNNING
Speculator: ATTEMPT_START task_1528178508022_0002_m_000007
apCompletionEvents request from attempt_1528178508022_0002_r_000000_0. startIndex 6 maxEvents 10000
successful for job_1528178508022_0002 (auth:SIMPLE)
VM with ID : jvm_1528178508022_0002_m_000012 asked for a task
VM with ID: jvm_1528178508022_0002_m_000012 given task: attempt_1528178508022_0002_m_000007_1
r: getResources() for application_1528178508022_0002: ask=5 release= 0 newContainers=0 finishedContainers=0 resourcelimit=<m
ogress of TaskAttempt attempt_1528178508022_0002_m_000006_1 is : 0.42788288
```

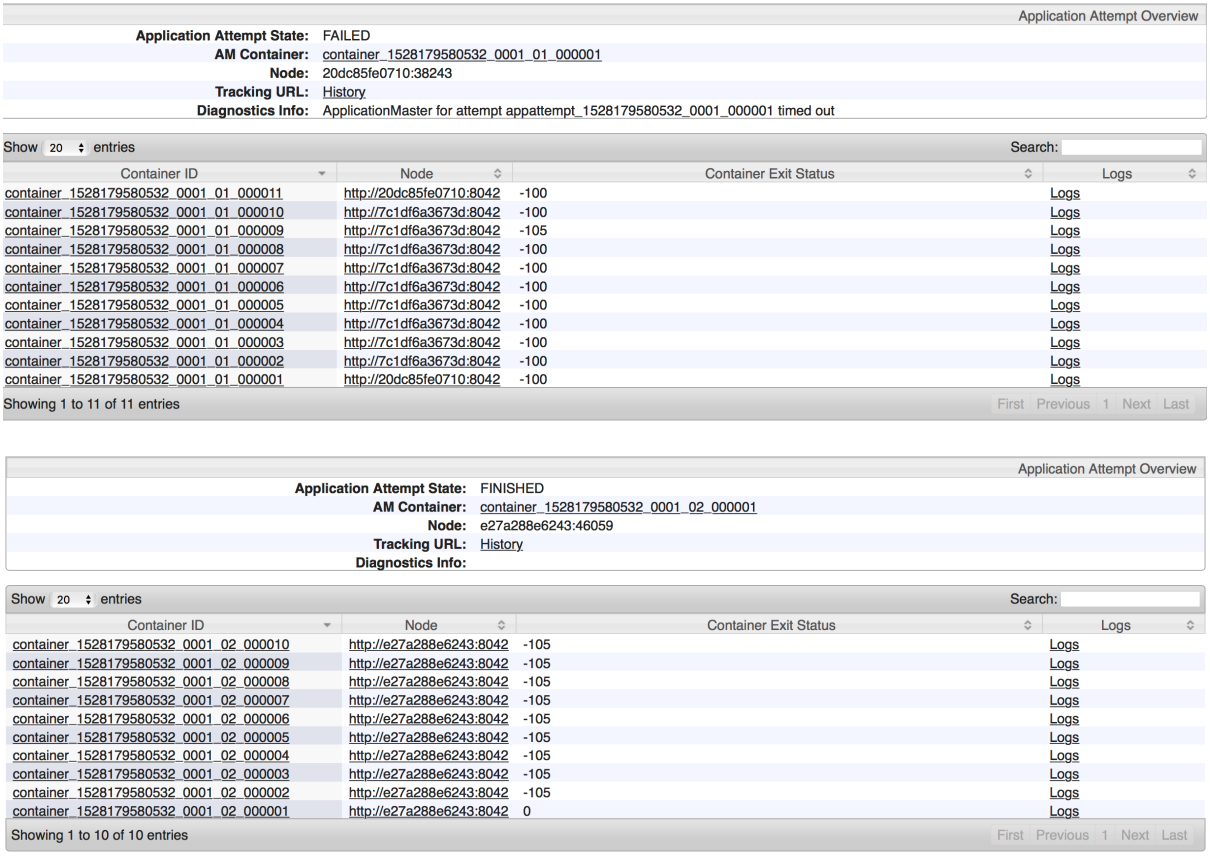
```
Task succeeded with attempt attempt_1528178508022_0002_m_000007_1
Issuing kill to other attempt attempt_1528178508022_0002_m_000007_0
```

e. 然后看“运行到一半有两个 slave 被 kill 掉的程序”的 log，可以发现这次进行了两次 appattempt，第一次失败了，原因是被 kill 掉的了两个 slave 中有一个是担任 ApplicationMaster 的角色，导致程序不能继续再进行；因此经过一段时间之后，ResourceManager 发现 timeout 了，便重新制定一个新的 ApplicationMaster ，重头开始进行 mapreduce 任务。可以发现新的 appattempt 的 container 均属于同一个节点（因为其它两个节点被 kill 掉了，只剩下一个节点）。

Show 20 entries Search:

| Attempt ID | Started | Node | Logs |
|---|---|---|---|
| appattempt_1528179580532_0001_000002 | Tue Jun 5 14:33:01 +0800 2018 | http://e27a288e6243:8042 | Logs |
| appattempt_1528179580532_0001_000001 | Tue Jun 5 14:20:28 +0800 2018 | http://20dc85fe0710:8042 | Logs |

Showing 1 to 2 of 2 entries First Previous 1 Next Last



Application Attempt Overview

| | |
|---|---|
| **Application Attempt State:** | FAILED |
| **AM Container:** | container_1528179580532_0001_01_000001 |
| **Node:** | 20dc85fe0710:38243 |
| **Tracking URL:** | History |
| **Diagnostics Info:** | ApplicationMaster for attempt appattempt_1528179580532_0001_000001 timed out |

Show 20 entries Search:

| Container ID | Node | Container Exit Status | Logs |
|---|---|---|---|
| container_1528179580532_0001_01_000011 | http://20dc85fe0710:8042 | -100 | Logs |
| container_1528179580532_0001_01_000010 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000009 | http://7c1df6a3673d:8042 | -105 | Logs |
| container_1528179580532_0001_01_000008 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000007 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000006 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000005 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000004 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000003 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000002 | http://7c1df6a3673d:8042 | -100 | Logs |
| container_1528179580532_0001_01_000001 | http://20dc85fe0710:8042 | -100 | Logs |

Showing 1 to 11 of 11 entries First Previous 1 Next Last



Application Attempt Overview

| | |
|---|---|
| **Application Attempt State:** | FINISHED |
| **AM Container:** | container_1528179580532_0001_02_000001 |
| **Node:** | e27a288e6243:46059 |
| **Tracking URL:** | History |
| **Diagnostics Info:** | |

Show 20 entries Search:

| Container ID | Node | Container Exit Status | Logs |
|---|---|---|---|
| container_1528179580532_0001_02_000010 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000009 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000008 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000007 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000006 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000005 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000004 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000003 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000002 | http://e27a288e6243:8042 | -105 | Logs |
| container_1528179580532_0001_02_000001 | http://e27a288e6243:8042 | 0 | Logs |

Showing 1 to 10 of 10 entries First Previous 1 Next Last

**步骤七：**得出结论。Yarn 框架能很好地处理 slave 节点失效的情况。当失效的节点不是 ApplicationMaster 节点时，ApplicationMaster 会重新转移分配计算任务，使得 Mapreduce 任务能够继续进行。当失效的节点是 ApplicationMaster 节点时，ResourceManager 会重新分配一个新的 ApplicationMaster 节点，然后重头开始 Mapreduce 任务。

## 六、经验总结

这个 Lab 花了我大概 4 个整天的时间，搭建 Hadoop 的 Docker 集群花了一天，编写 Mapreduce 程序花了一天，容错性验证花了一天，最后编写文档花了一天 。搭建 Hadoop Docker 集群看似简单，但实际上有很多坑要踩，比如每次开启容器之后都需要启动一下 ssh 服务、如何让容器启动之后不会立即关闭而保持 alive 状态等等，有很多细节要查询很多资料，琢磨很多时间。

虽然之前就有过用 docker 搭建 hadoop 集群，但是这次实验，我仍收获不少。通过这次实验，我对 docker 的使用更加熟练了，学会了如果使用 docker compose，对 hadoop 的理解更加深入，对 hadoop 的各种配置文件也更新熟悉了，明白了各个配置项的含义，对 hadoop 的容错机制有了一定的了解，还学会了如何编写 mapreduce 程序，总之，收获很多。