Hadoop 大数据平台架构与实战

笔记源自慕课网Hadoop学习视频《Hadoop大数据平台架构与实践--基础篇》<http://www.imooc.com/learn/391>。

1、初步认识Hadoop

1.1 简介

目标：

掌握大数据存储和处理技术的原理（理论知识）

掌握Hadoop的使用和开发能力（实践能力）

内容：

大数据相关的概念

Hadoop的架构和运行机制

Hadoop的安装和配置

Hadoop的开发

学习建议

使用书本知识，全面的学习；动手安装和配置，以及编程实践

需要的基础 Java开发知识&linux常见命令的使用

1.2 hadoop出现的需求

应对PB级数据的存储和处理，单机系统存在性能瓶颈（存储、读写和计算）

Google 的大数据技术： MapReduce BigTable GFS 三大技术革命变化，带来了优势：

降低了成本，使用不同的计算机，而不是大型机；

软件容错硬件故障成为常态，实用软件来容错；

简化并行分布式计算，无需控制节点同步和数据交换。

模仿google开源的实现Hadoop技术。

1.3 Hadoop的功能和优势

开源的分布式存储和分布式计算：

HDFS：分布式存储

MapReduce：分布式计算

用于搭建大型数据仓库、PB级数据的存储、处理、分析和统计，比如搜索引擎、商业智能、日志分析。

性能：高扩展性（增加硬件即可），低成本（普通的PC机即可，通过软件容错实现高可靠性）、成熟的生态圈（hbase hive zookeeper等）。

Hadoop的开发和运维需求比较大，而是未来互联网发展的趋势.

1.4 Hadoop的生态系统

Hive 简单的SQL语句转换成为Hadoop任务去执行,降低使用门槛

Hbase: 提供数据的随机读写和实时访问,实现对于表数据的读写功能,放弃事务的特性.

Zookeeper: 监管平台，监控每一个节点的状态，维护数据的一致性。

Hadoop版本的选择：2.6版本还是1.X版本，1.2版本的选择，比较简单

2. hadoop 安装

2.1 获取linux操作系统

需要Linux、Java开发环境、配置Hadoop环境

2.2 Linux安装Java环境

sudo tar zxvf jdk\*.tar.gz -C /usr/lib/jvm

cd /usr/lib/jvm

sudo ln -s jdk\*/ java8

vim /etc/profile

add text at the end of the file

#java environment

export JAVA\_HOME=/usr/lib/jvm/java8

export JRE\_HOME=$JAVA\_HOME/jre

export CLASSPATH=$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

soure /etc/profile

sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/java8/bin/java 300

sudo update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java8/bin/javac 300

sudo update-alternatives --install /usr/bin/javah javah /usr/lib/jvm/java8/bin/javah 300

sudo update-laternatives --install /usr/bin/jar jar /usr/lib/jv,/java8/bin/jar 300

2.3 install hadoop 1.1.2

这里使用的是Hadoop1.2.1版本

sudo useradd –m hadoop -s /bin/bash

sudo passwd hadoop

sudo adduser hadoop sudo // hadoop 用户增加管理员权限

mkdir /usr/dahua

sudo tar hadoop-1.2.1.tar.gz -C /usr/dahua/

sudo ln –s hadoop-1.2.1.tar.gz hadoop

sudo chown -R hadoop:hadoop ./dahua //改变其用户组、用户为hadoop

之后需要告诉系统我们的Hadoop的目录，并且把hadoop/bin加入到PATH

在/etc/profile中添加

export $HADOOP\_HOME=/opt/hadoop

export $PATH=$HADOOP\_HOME/bin:$PATH

source /etc/profile

配置文件方面，有四个Hadoop文件需要配置：

1.hadoop-env.sh 文件告诉其java环境目录

export JAVA\_HOME=/usr/lib/jvm/java8

2.core-site.xml

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

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

<configuration>

<property>

<name>hadoop.tmp.dir></name>

<value>/opt/hadoop </value>

</property>

<property>

<name>dfs.name.dir</name>

<value>/opt/hadoop/name</value>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://192.168.1.103:9000</value>

</property>

</configuration>

3. hdfs-site.xml

<configuration>

<property>

<name>dfs.data.dir</name>

<value>/opt/hadoop/data</value>

</property>

4. mapred-site.xml

<property>

<name>mapred.job.tracker</name>

<value>192.168.1.103:9001</value>

</property>

我们这里使用的是hadoop账户运行的hadoop程序，所以保证日志目录，hadoop文件目录等等都是可以在hadoop账户下可以有读写执行的权限，所以使用sudo chown 更改文件夹的用户组，不然会出现权限问题，而导致运行失败。

使用命令 hadoop 格式化namenode

hadoop namenode -format

start-all.sh to start the hadoop

then use the command **jps** to view the process, which followed by :

JobTracker

Jps

DataNode

NameNode

SecondaryNameNode

sudo chown -R hadoop:hadoop ./hadoop

使用hadoop fs -ls / 查看的根目录下面的hadoop文件

<http://192.168.1.103:50070/dfshealth.jsp>

3. hadoop 的核心 HDFS

3.1 核心基本概念

HDFS设计架构： 块（Block） 文件被分成块进行存储，默认大小是64MB，他是文件的逻辑单元。

在HDFS中有两种节点：NameNode和DataNode

NameNode是管理节点，存放文件的元数据：文件与数据块的映射表；数据块与数据节点的映射表；NameNode是唯一的节点，数据存放在哪一个节点。

DataNode存放数据块的节点。

3.2 HDFS数据管理和容错

每一个数据都存放三份，来保证数据的恢复，这样一个节点坏掉，可以在同一个机架中找到，整一个机架坏掉，可以在另一个机架上面找到，来保证数据的可靠。

心跳检测：DataNode定期向NameNode发送心跳信息，确保集群中的DataNode是正常的；SecondaryNode用于备份NameNode，在NameNode宕机，SecondaryNode根据之前的同步信息，进行接管NameNode的服务。

3.3 HDFS 文件读取流程

客户端向NameNode请求，NameNode返回文件的元数据给客户端，然后客户端直接从不同的DataNode读取数据。

写文件：把文件拆分成块，然后通知NameNode，然NameNode根据目前的DataNode所有节点的状况，告知客户端那些数据节点的那些块是可以进行写的，然后客户端把数据写入DataNode，之后在数据节点之间通过流水线复制技术，备份数据块；最后更新元数据到NameNode状态。然后在写第二块数据。

3.4 HDFS特点:

数据冗余，来实现硬件的容错；

流式数据的访问：写一次，读多次

适合存储大文件：小文件会给NameNode造成很大的压力

这些特点：

HDFS适合数据批量的读写操作，而不适合交互性的

HDFS适合一次写入，多次读取，顺序读写，不适合多用户并发写相同的文件

3.5 HDFS使用

HDFS 命令行.

hadoop namenode -fortmat 在hadoop启动之前需要把hadoop的NameNode格式化.

hadoop fs –ls / 查看HDFS的文件夹展示出来。

hadoop fs -mkdir input

hadoop fs -put file dir\_HDFS

hadoop fs -ls /dir\_HDFS

hadoop fs -cat file

hadoop fs -get input/hadoop.sh hadoop-env.2.sh

hadoop dfsadmin –report

4.MapReduce的原理

MapReduce是hadoop的计算框架，整体的思想就是分治思想，然后在合并。

比如统计日志IP访问的平率计算：将数据进行分割，然后每一个节点进行Map映射处理分隔文件中的IP访问数目；然后将所有节点处理的结果进行shuffle交换，然后在对处理的结果进行Reduce，便可以整理出IP的访问次数情况。

MapReduce的基本概念

Job&Task

JobTracker ： 所有的任务都提交给JobTracker，由JobTracker来分配任务。

TaskTracker: Map TaskTracker & Recude TaskTracker

一般情况下，使用NameNode来作为JobTracker， 来实现作业调度、分配任务、监控任务的进度。

而TaskTracker就是执行具体的任务，汇报任务的结果给JobTracker，一般使用DataNode来实现TaskTracker。

MapReduce的容错机制的方法：

重复执行：对于出现错误的节点任务，出现错误之后，就会重复执行，如果重复执行4times，则放弃执行。

推测执行：根据TaskTracker的执行进度，来推测执行，不会因为一个TaskTracker的故障而导致作业的失败。

5. Hadoop的应用程序

1. 应用程序开发 WordCount

1）编写完成WordCount程序之后（Mapper&Reducer）

2）编译程序：javac: javac -classpath /usr/cloud/hadoop/hadoop-core\*.jar:/usr/cloud/hadoop/lib/common-cli.\*jar –d word\_count\_class/ WordCountMapReduce.java

之前需要创建好word\_count\_class文件夹。

3）打包成为一个jar文件：jar -cvf wordcount.jar ./

4）提交hadoop程序执行JAR：

hadoop jar wordcount.jar com/yang/hadoop/mr/module/WordCountMapReduce <INPUT> <OUTPUT>

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configurable;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

/\*

\* MapReduce example

\*/

public class MyWordCount {

    // Mapper

    /\*

     \* Word Count Mapper Class

     \*/

    static class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

        private Text word = new Text();

        private final static IntWritable one = new IntWritable(1);

        @Override

        protected void map(LongWritable key, Text value, Context context)

                throws IOException, InterruptedException {

            // text input

            String lineValue = value.toString();

            StringTokenizer stringTokenizer = new StringTokenizer(lineValue);

            while (stringTokenizer.hasMoreElements()) {

                String wordValue = stringTokenizer.nextToken();

                word.set(wordValue);

                context.write(word, one);

            }

        }

    }

    // Reducer

    /\*

     \* Word Count Reducer Class

     \*/

    static class MyReducer extends

            Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();

        @Override

        protected void reduce(Text key, Iterable<IntWritable> values,

                Context context) throws IOException, InterruptedException {

            int sum = 0;

            for (IntWritable value : values) {

                sum += value.get();

            }

            result.set(sum);

            context.write(key, result);

        }

    }

    // client

    /\*\*

     \* @param args

     \* @throws IOException

     \* @throws InterruptedException

     \* @throws ClassNotFoundException

     \* PriviledgedActionException

     \* FileUtil

     \*/

    public static void main(String[] args) throws IOException,

            ClassNotFoundException, InterruptedException {

        args = new String[]{

                "hdfs://hadoop-master:9000/opt/data/wc/input/",

                "hdfs://hadoop-master:9000/opt/data/wc/output7/"};

        Configuration conf = new Configuration();

        Job job = new Job(conf, "wc");

        // 1.设置Job运行Class

        job.setJarByClass(MyWordCount.class);

        // 2设置 Job 的Mapper & Reducer

        job.setMapperClass(MyMapper.class);

        job.setCombinerClass(MyReducer.class);

        job.setReducerClass(MyReducer.class);

        // 3.设置Job的输入输出目录

        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 4. 输出结果的key & value 类型

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);

        // 5. 提交Job，等待运行结果

        boolean isSuccess = job.waitForCompletion(true);

        String info = isSuccess ? "Success" : "Failed";

        System.out.println(info);

        System.exit(isSuccess ? 0 : 1);

    }

}