**《并行计算》上机报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 王章瀚 | | 学号： | PB18111697 | 日期： | 2021/5/29 |
| 上机题目： | Hadoop编程实验 | | | | | |
| 实验环境：  CPU：8 块 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz ;  内存: 32GB ;  操作系统：Windows 10 ;  软件平台：Hadoop 3.2.2, JDK 1.8.0\_281 ; | | | | | | |
| 1. **Hadoop 配置**   Hadoop的配置流程比较复杂. 这里详述一下.  由于资源有限, 只能配置伪分布式的 Hadoop. 实验文档给出的 Hadoop 版本比较低, 不能很好地适配现在Hadoop, 因此我尝试参考官方文档配置 Hadoop 3.2.2. 这是一个参考链接 :  <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>   1. 安装并配置 OpenSSH Client 和 Server   为了配置Hadoop这样一个分布式计算系统, 我们需要有 SSH 这样的连接方式, 并且要配置免密登录模式, 以便集群中的节点中进行通信.  运行以下命令可以安装:  sudo apt-get install ssh pdsh  安装完成后, 通过ssh-keygen来生成一堆公私秘钥, 一般会生成在 ${HOME}/.ssh里, 即 id\_rsa和id\_rsa.pub:    而后, 将 id\_rsa.pub 的内容放进 authorized\_keys 里, 以完成无秘登录.  之后测试能否 ssh localhost 无秘登录:    见到以上界面就算是成功了.   1. 配置 Java 和 JDK   Hadoop 是基于 Java 的, 所以首先需要安装 Java 和 JDK(Java Development Kit). 可以到Oracle官网下载. 由于Hadoop需要, 必须下载1.8版本的JDK:  <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>  通过以下命令解压到/opt:  tar zxf ~/Downloads/jdk-8u281-linux-x64.tar.gz  然后在 .bashrc 中写入环境变量的配置:    然后 source ~/.bashrc 以激活, 并验证是否正确安装:     1. 下载 Hadoop 并安装   最新版的Hadoop 可以在该链接下载  <https://hadoop.apache.org/releases.html>  下载完成后, 将其解压到所想要的路径. 这里指定的是 /opt. 通过以下命令完成:  cd /opt && tar zxf ~/Downloads/hadoop-3.2.2.tar.gz    将Hadoop的可执行文件加入环境变量, 修改 .bashrc:    source ~/.bashrc后验证是否成功    如此即可.   1. 配置 Hadoop 2. . 配置Hadoop的核心和HDFS:  * 在 ${HADOOP\_HOME}/etc/hadoop/core-site.xml写入     这是配置了HDFS的通信地址.   * 在 ${HADOOP\_HOME}/etc/hadoop/hdfs-site.xml写入     这里配置了HDFS的数据冗余数目以及namenode和datanode的目录.   1. 配置 Map Reduce 引擎:  * 在${HADOOP\_HOME}/etc/hadoop/mapred-site.xml中写入     以配置 mapreduce job 所使用的框架. 此外为了使用 yarn, 还需要对${HADOOP\_HOME}/etc/hadoop/mapred-site.xml 进行配置:     1. 初始化HDFS并做测试   完成了上述配置后, Hadoop的基本配置就完成了. 现在就可以初始化HDFS, 并测试Hadoop了.  首先初始化HDFS:  hdfs namenode -foramt  启动所有Hadoop相关服务并通过jps查看运行的进程:  start-all.sh && jps    在HDFS创建一些文件夹, 以便后续测试:  hdfs dfs -mkdir -p /user/hank/input    往input里放一些测试文件:    运行测试文件代码:        见到这些输出, 基本上就是成功了, 可以输出一下这个信息验证:    发现确实找出了所有满足正则表达式 dfs[a-z.]+的字符串.  至此, Hadoop 配置基本完成!   1. **配置IDEA结合Maven以调试Hadoop代码**   IDEA 是JetBrains下的一个非常好用的Java IDE. 可以配置它以便于我们编写代码.  Maven是一个主要用于Java的工程管理及包管理工具, 现在大多数Java项目都基于或者将基于Maven开发.  这里就不赘述安装IDEA和Maven的过程了. 假设已经安装好了IDEA和Maven.  我们可以基于  org.apache.maven.archetypes:maven-archetype-quickstart  来构建我们的项目.    一路Next之后, 工程就创建好了. 可以把自动生成的 src/main/java/org/hank/App.java替换成实验说明给出的 WordCount.java 及本实验第2步要求的 WordLengthCount.java.  然后配置pom.xml以让Maven能够自动解决包依赖问题. 在 dependencies里加入:    这样, 就可以直接Build这个Project了. 配置 Maven运行:    然后直接运行可以得到这些打包好的文件:    其中我们可以用hadoop命令运行WordLengthCount-1.0-SNAPSHOT.jar来运行我们的程序, 例如:  hadoop jar target/WordLengthCount-1.0-SNAPSHOT.jar org.hank.WordCount input output    此外也可以在IDEA中直接配置运行命令:       1. **算法设计与分析：**   为了统计不同长度单词出现的次数, 我们可以这样布局MapReduce:  Map步将文档Map解析为单词, 然后按Key为长度, Value为1来完成Map步.  Reduce步可以按单词长度为Key做求和的Reduce, 即可得到相应长度的单词数.   1. **核心代码：**   Map步解析每个文档的Text, 利用 StringTokenizer来取出所有单词, 然后通过一个循环遍历所有单词, 通过 context.write(length, one) 完成Map步.    Reduce步将所有前面Map的结果求和即可:    在main函数中提交相应的job. 这需要指定相应过程所用的 class:     1. **结果与分析：**   我把 WikiPedia上关于 Hadoop, MapReduce和Java的内容放进input文件夹里, 以作为输入数据.  然后运行 WordCount 得到:    当然后续还有很多内容, 空间限制就不放进来了.  而对于单词长度的统计, 可以得到如下结果:    这是非常符合预期的, 因为英文单词长度确实大多都集中于 2到9这样的区间内.  可以确认的是原本的WordCount程序没有问题, 那么WordLengthCount仅仅把key改成了长度, 没有引入任何问题. 因此本程序确实也是正确的.  **六、备注（\* 可选）：**  有可能影响结论的因素：  Hadoop版本和MapReduce引擎可能影响性能 | | | | | | |
| **总结：**  本次实验配置了Hadoop, 并利用它完成了单词长度统计, 且熟悉了Java编程, 收获颇丰. | | | | | | |
| 附录（源代码） | | 算法源代码（C/C++/JAVA描述） | | | | |