一 MapReduce 入门

1.1 MapReduce 定义

Mapreduce 是一个分布式运算程序的编程框架，是用户开发“基于 hadoop 的数据分析应用”的核心框架。

Mapreduce 核心功能是将用户编写的业务逻辑代码和自带默认组件整合成一个完整的分布式运算程序，并发运行在一个 hadoop 集群上。

1.2 MapReduce 优缺点

1.2.1 优点

1）MapReduce 易于编程。它简单的实现一些接口，就可以完成一个分布式程序，这个分布式程序可以分布到大量廉价的 PC 机器上运行。也就是说你写一个分布式程序，跟写一个简单的串行程序是一模一样的。就是因为这个特点使得 MapReduce 编程变得非常流行。

2）良好的扩展性。当你的计算资源不能得到满足的时候，你可以通过简单的增加机器来扩展它的计算能力。

3）高容错性。MapReduce 设计的初衷就是使程序能够部署在廉价的 PC 机器上，这就要求它具有很高的容错性。比如其中一台机器挂了，它可以把上面的计算任务转移到另外一个节点上运行，不至于这个任务运行失败，而且这个过程不需要人工参与，而完全是由 Hadoop 内部完成的。

4）适合 PB 级以上海量数据的离线处理。这里加红字体离线处理，说明它适合离线处理而不适合在线处理。比如像毫秒级别的返回一个结果，MapReduce 很难做到。

1.2.2 缺点

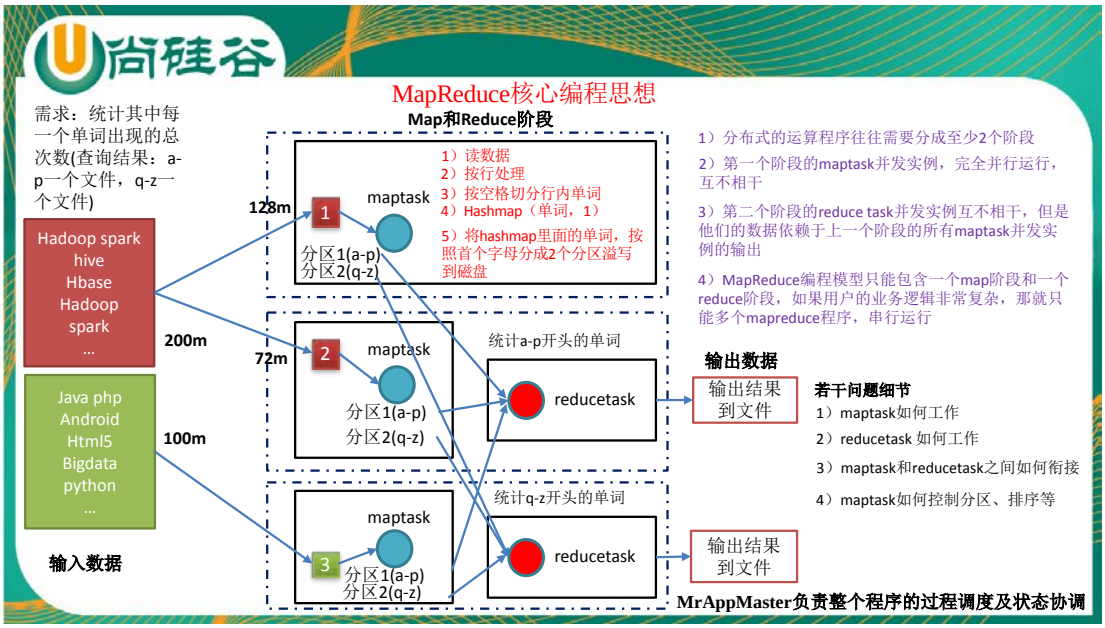
MapReduce 不擅长做实时计算、流式计算、DAG（有向图）计算。

1）实时计算。MapReduce 无法像 Mysql 一样，在毫秒或者秒级内返回结果。

2）流式计算。流式计算的输入数据是动态的，而 MapReduce 的输入数据集是静态的，不能动态变化。这是因为 MapReduce 自身的设计特点决定了数据源必须是静态的。

3）DAG（有向图）计算。多个应用程序存在依赖关系，后一个应用程序的输入为前一个的输出。在这种情况下，MapReduce 并不是不能做，而是使用后，每个 MapReduce 作业的输出结果都会写入到磁盘，会造成大量的磁盘 IO，导致性能非常的低下。

1.3 MapReduce 核心思想



1）分布式的运算程序往往需要分成至少 2 个阶段。

2）第一个阶段的 maptask 并发实例，完全并行运行，互不相干。

3）第二个阶段的 reduce task 并发实例互不相干，但是他们的数据依赖于上一个阶段的所有 maptask 并发实例的输出。

4）MapReduce 编程模型只能包含一个 map 阶段和一个 reduce 阶段，如果用户的业务逻辑非常复杂，那就只能多个 mapreduce 程序，串行运行。

1.4 MapReduce 进程

一个完整的 mapreduce 程序在分布式运行时有三类实例进程：

1）MrAppMaster：负责整个程序的过程调度及状态协调。

2）MapTask：负责 map 阶段的整个数据处理流程。

3）ReduceTask：负责 reduce 阶段的整个数据处理流程。

1.5 MapReduce 编程规范

用户编写的程序分成三个部分：Mapper、Reducer 和 Driver。

1）Mapper 阶段

（1）用户自定义的 Mapper 要继承自己的父类

（2）Mapper 的输入数据是 KV 对的形式（KV 的类型可自定义）

（3）Mapper 中的业务逻辑写在 map()方法中

（4）Mapper 的输出数据是 KV 对的形式（KV 的类型可自定义）

（5）map()方法（maptask 进程）对每一个<K,V>调用一次

2）Reducer 阶段

（1）用户自定义的 Reducer 要继承自己的父类

（2）Reducer 的输入数据类型对应 Mapper 的输出数据类型，也是 KV

（3）Reducer 的业务逻辑写在 reduce()方法中

（4）Reducetask 进程对每一组相同 k 的<k,v>组调用一次 reduce()方法

3）Driver 阶段

整个程序需要一个 Drvier 来进行提交，提交的是一个描述了各种必要信息的 job 对象

1.6 WordCount 案例实操

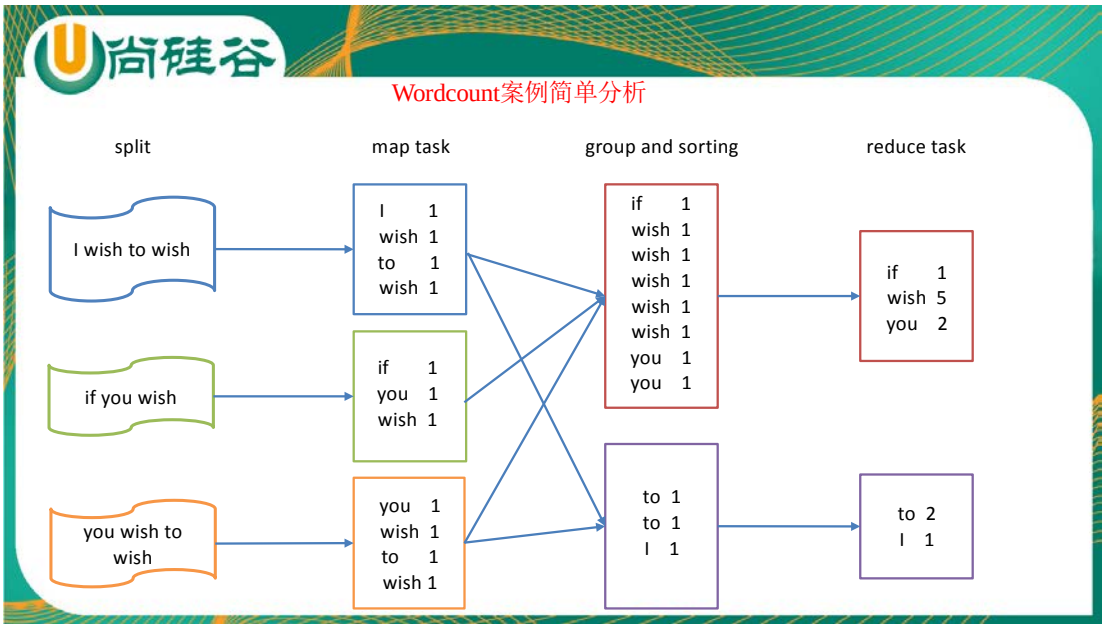
1）需求：在一堆给定的文本文件中统计输出每一个单词出现的总次数

2）数据准备：

hello.txt

3）分析

按照 mapreduce 编程规范，分别编写 Mapper，Reducer，Driver。



4）编写程序

（1）编写 mapper 类

```
package com.atguigu.mapreduce;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordcountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{

	Text k = new Text();
	IntWritable v = new IntWritable(1);

	@Override
	protected void map(LongWritable key, Text value, Context context)
			throws IOException, InterruptedException {

		// 1 获取一行
		String line = value.toString();

		// 2 切割
		String[] words = line.split(" ");

		// 3 输出
		for (String word : words) {

			k.set(word);
```

```
                context.write(k, v);
            }
        }
}
```

（2）编写 reducer 类

```
package com.atguigu.mapreduce.wordcount;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordcountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

    int sum;
    LongWritable v = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> value,
            Context context) throws IOException, InterruptedException {

        // 1 累加求和
        sum = 0;
        for (IntWritable count : value) {
            sum += count.get();
        }

        // 2 输出
        v.set(sum);
        context.write(key,v);
    }
}
```

（3）编写驱动类

```
package com.atguigu.mapreduce.wordcount;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordcountDriver {
```

```
    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

        // 1 获取配置信息以及封装任务
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 2 设置 jar 加载路径
        job.setJarByClass(WordcountDriver.class);

        // 3 设置 map 和 reduce 类
        job.setMapperClass(WordcountMapper.class);
        job.setReducerClass(WordcountReducer.class);

        // 4 设置 map 输出
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // 5 设置 Reduce 输出
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // 6 设置输入和输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 7 提交
        boolean result = job.waitForCompletion(true);

        System.exit(result ? 0 : 1);
    }
}
```

5）本地测试

（1）在 windows 环境上配置 HADOOP_HOME 环境变量

（2）在 eclipse 上运行程序

（3）注意：如果 eclipse 打印不出日志，在控制台上只显示

```
1.log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
2.log4j:WARN Please initialize the log4j system properly.
3.log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

需要在项目的 src 目录下，新建一个文件，命名为“log4j.properties”，在文件中填入

```
log4j.rootLogger=INFO, stdout
```

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=target/spring.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n
```

6）集群上测试

（1）在依赖中添加

```
<build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin </artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <manifest>
                            <mainClass>com.atguigu.mr.WordCountReduce</mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

（2）将程序打成 jar 包，然后拷贝到 hadoop 集群中

（3）启动 hadoop 集群

（4）执行 wordcount 程序

```
[atguigu@hadoop102 software]$ hadoop jar wc.jar com.atguigu.wordcount.WordcountDriver /user/atguigu/input /user/atguigu/output1
```