Python 实践之

400 行 Python 写一个类 Hadoop 的 MapReduce 框架

Max

Google 的老三篇已经快问世十年了，但这 10 年里我只是肤浅的了解了一下 Mapreduce 的原理，并无深入的实践过程真是惭愧。最近 2 天，抱恙在家，正好开始了学习 Hadoop 的工作原理。说起来 Hadoop 极其复杂，各种派生类盘根错杂。但是其实 MapReduce 的本身工作原理却并不太复杂。可以说很好懂，简单的来说就是将数据分块，并行的进行 MAP 操作，再将并行进行 Reduce 操作。利用并行计算的优势，减少业务的处理时间。

为此我花了 2 天时间，用 python 实现了一套类 Hadoop 的 Mapreduce 框架。更确切的说这只是一个简单的模型，用户只要写一个 python 文件，将业务的 InputSplit , Map, Reduce, OutputFormat 四个接口简单的实现。框架会根据传入的参数，依次调用这几次函数，完成一个最简单 Mapreduce 的程序了。

例子一共用了 400 行，主要包括以下几个模块：

一个 sort 的客户程序， 实现了 InputSplit , Map, Reduce, OutputFormat 这四个接口，完成具体业务相关的。 它完成的业务 就是 sort –k 1 –n xxx.txt | uniq –c | awk ’{print $2” “$1}’

一个 server 程序，它主要起到了一个消息队列的作用，主要缓存每次业务做完中间结果。
一个 task_master 主程序，他是业务的灵魂，它负责启动各次业务的启动，启动多个子进程模拟多节点情况，并行地运行 MAP 和 Reduce 操作。同时并关注每次业务完成情况，负责整体业务流程。
其他一些工具函数。比如序列化对象保存到本地文件。

从宏观的角度看 MapReduce 的过程 （图摘自 Hadoop 技术内幕）

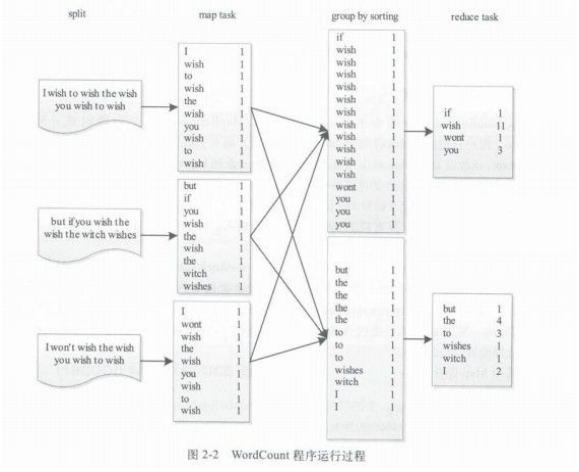

图 2-2 WordCount 程序运行过程

从图中我们可以知道 split ， map （group by sort），reduce task 这几个步骤是靠客户程序

来完成。
而系统则是完成整体业务，并将这几个步骤串起来 就完成了 Hadoop 框架本身的功能。

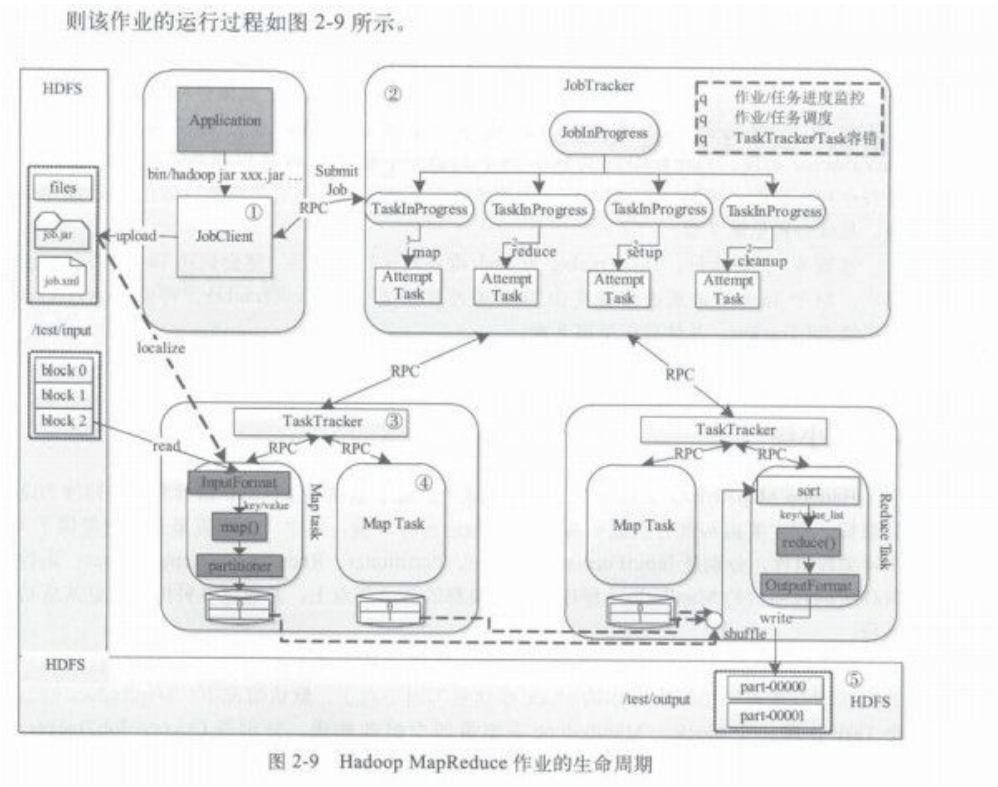

图 2-9 Hadoop MapReduce 作业的生命周期

这个框架本身极其简陋的，不支持分布式文件系统，只能用本地文件模拟。不支持多机处理只能用本地进程模拟 worker。没有单点故障监控，更不支持业务调度这样的复杂功能。

单纯但从一个业余 python 开发者的角度看，我认为用 400 来行代码就实现了一个 Hadoop 的 Mapreduce 的原型说明 python 的表现能力和效果都是非常惊人。是实现项目原型很好的一种选择。

技术实现：

1） 动态调用 python 模块

框架实现以后，就应该不会再修改了。它可以一直运行，客户只需要编写符合要求的 python 模块，这个框架就能自动去调用了。

那么如何让框架去调用一个它甚至不知道的名字的模块？

比如 我要做一个 wordcount 的操作， 完成代码编写以后， 我只需要将字符串 “wordcount” 传入框架， 框架就会自动去执行 wordcount 类里定义的 InputSplit, Map, Reduce, OutputFormat 这样几个函数。 下一次 我又写了一个 “TOP10” 的操作，我只需要将字符串 “TOP10” 传入框架，框架就会自动去执行 TOP10 类里定义的 InputSplit , Map, Reduce, OutputFormat 这样几个函数。

JAVA 里我们有反射能做到这点，Python 里也有类似的方法 （GO 是静态语言，我不知道如何实现，还请高人指点）代码如下

```
def run_task(modename, functionname,arg):
    obj = __import__(modename) # import module
```

```
c = getattr(obj,modename)
obj = c() # new class
fun = getattr(obj,functionname)
fun(arg) # call def
```

自此通过参数将 类名的字符串，函数名，参数都以字符串的形式传进去，一切就都好了。

2) 序列化

每个业务的中间结果都是一些 MAP，但是如何传给下一个工序呢？ 我采用的方法是 使用 python 序列化将中间结果对象，序列化成一个文件，将文件名保存在消息队列中，下一道工序通过消息队列获取相应的文件名，反序列化文件。获得上一次步骤的中间结果，继续操作。

3）消息队列

就是一个全局的 MAP，以 任务 ID 作为主键的一个分层 MAP。

剩下的事情，就是耐心了。我相信一个普通的程序员几个钟头都是能完成的了。

示例 利用这个框架进行验证 是否真的有效果

首先下载 rand.cpp, 这是一个简单的 cpp 代码。主要输出随机的数字。

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(int argc, char* argv[]) {
	srand(time(NULL));
	int linecount = 1000;
	if (argc >= 2) {
		linecount = atoi(argv[1]);
	}
	char* buff = new char[1024 * 1024];
	int len = 0;
	int offset = 0;
	for (int i = 0; i < linecount; ++i) {
		unsigned long long x = (unsigned long long)rand() % 9 + 1;
		*(buff + offset + x) = '\n';
		*(buff + offset + x + 1) = 0;
		for (int j = x - 1; j >= 1; j--) {
			*(buff + offset + j) = rand() % 10 + '0';
		}
		*(buff + offset) = rand() % 9 + 1 + '0';
		offset += x + 1;
		len++;
```

```
            if (len >= 1024 * 1024 / 11) {
                printf("%s", buff);
                len = 0;
                offset = 0;
            }
        }
        if (len > 0) {
            printf("%s", buff);
        }
        delete[] buff;
        return 0;
}
```

编译以后 执行。

./rand 50000000 > 1.txt

表示随机生成 5000 万行记录 把它保存在 1.txt 里面。

验证一下 没有问题

```
root@lotus:~/mapreduce# ls -lah 1.txt
-rw-r--r-- 1 root root 287M Nov 27 16:01 1.txt
root@lotus:~/mapreduce# wc -l 1.txt
50000000 1.txt
```

首先用 shell 进行一次操作，获得基准数据。

```
root@lotus:~/mapreduce# time sort -k 1 -n 1.txt | uniq -c | awk '{print $2" "$1}' > data2

real    0m59.089s
user    2m27.741s
sys     0m5.651s
root@lotus:~/mapreduce# md5sum data2
8003a8211d1caf9e227cd92f5d491872  data2
```

计算出结果的 md5

然后用我的 py mapreduce 试试看呢？

```
root@lotus:~/mapreduce# ./task_master.py abc 1.txt sort InputSplit
```

其中 abc 表示这次业务的唯一标识符

1.txt 表示这次业务的源文件

sort 表示这次业务调用的客户模块名

InputSplit 表示从客户模块的这个函数开始启动整个流程。

```
[2014-11-27 16:07:18,769] root:INFO: GAME OVER
0:01:33.311095
root@lotus:~/mapreduce# md5sum data
8003a8211d1caf9e227cd92f5d491872  data
```

从 md5 上面看，2 者的结果是一致的。但是速度上仍然是 shell 的快，而且快了接近 30%。

我把我的代码放在了 github 上，你可以在 https://github.com/xiaojiaqi/py_hadoop 找到它。