Lab0 实验报告

李睿 2019202254

- Lab0 实验报告
  - 实验结果
    - 1. 完成 Map-Reduce 框架
      - 实现思路
        - doReduce
        - merge
    - 2. 基于 Map-Reduce 框架编写 Map-Reduce 函数
      - 实现思路
      - 与 example 的对比
  - 实验总结
    - 串行优化1
    - 串行优化2
    - 并发优化: 数据倾斜的问题
    - 并发优化:Combiner
    - 修改topk算法

实验结果

1. 完成 Map-Reduce 框架

实现思路

这一部分是完成map reduce的框架,接受的map和reduce是example给的
需要补全的代码分为两部分,一个是worker的doReduce, 另一个是Master的merge

doReduce

首先读取并解析(NewDecoder)由上一步map得到的文件,并将这几个文件整合,得到的keyValuesMap后对其中kv对做reduce

merge

待Map完成后,首先发送nReduece个task信号量,等待Reduce完成,接着我们发送输出文件名给下一轮

`make test_example` 的实验截图

```
服务器端的1,10,100,400,800MB测试(只放了800)
Case0 PASS, dataSize=800MB, nMapFiles=60, cost=55.30050755s
Case1 PASS, dataSize=800MB, nMapFiles=60, cost=13.86143233s
Case2 PASS, dataSize=800MB, nMapFiles=60, cost=12.77643974s
Case3 PASS, dataSize=800MB, nMapFiles=60, cost=9.93136942s
Case4 PASS, dataSize=800MB, nMapFiles=60, cost=17.09674512s
Case5 PASS, dataSize=800MB, nMapFiles=60, cost=45.44751199s
Case6 PASS, dataSize=800MB, nMapFiles=60, cost=39.89379281s
Case7 PASS, dataSize=800MB, nMapFiles=60, cost=44.07095283s
Case8 PASS, dataSize=800MB, nMapFiles=60, cost=33.76309438s
Case9 PASS, dataSize=800MB, nMapFiles=60, cost=23.40785184s
Case10 PASS, dataSize=800MB, nMapFiles=60, cost=14.81220718s
--- PASS: TestExampleURLTop (503.27s)
PASS
ok      talent  503.374s
```

2. 基于 Map-Reduce 框架编写 Map-Reduce 函数

实现思路

思路很简单, 收到值后进行map, map做combiner

与 example 的对比

`cnts[tmp[0]] += n`

mapCount时做combiner,相应的reduceCount时是求sum

`make test_homework` 的实验截图

```
服务器端的1,10,100,400,800MB测试(只放了800)
Case0 PASS, dataSize=800MB, nMapFiles=60, cost=262.29324ms
Case1 PASS, dataSize=800MB, nMapFiles=60, cost=289.7656ms
Case2 PASS, dataSize=800MB, nMapFiles=60, cost=314.02454ms
Case3 PASS, dataSize=800MB, nMapFiles=60, cost=533.68808ms
Case4 PASS, dataSize=800MB, nMapFiles=60, cost=52.53213261s
Case5 PASS, dataSize=800MB, nMapFiles=60, cost=409.52248ms
Case6 PASS, dataSize=800MB, nMapFiles=60, cost=295.26022ms
Case7 PASS, dataSize=800MB, nMapFiles=60, cost=282.51321ms
Case8 PASS, dataSize=800MB, nMapFiles=60, cost=441.00174ms
Case9 PASS, dataSize=800MB, nMapFiles=60, cost=439.52307ms
Case10 PASS, dataSize=800MB, nMapFiles=60, cost=298.03536ms
--- PASS: TestURLTop (87.23s)
PASS
ok      talent  87.346s
```

实验总结

在这部分可以简单谈论自己在实验过程中遇到的困难、对 Map-Reduce 计算框架的理解，字数在 1000 字以内。

优化过程

串行优化1

```
1,10,100MB的测试集中
使用newDecode代替unmarshal加速20%
时间31s->25s(本地)
```

串行优化2

```
1,10,100MB的测试集中
合并allKeyValuePairs和keyValuesMap
时间25s→24s
```

经过以上优化,在1,10,100的规模上的时间从31缩减为24,加速25%
到这一步就已经可以跑完全部的case

并发优化: 数据倾斜的问题

>case0尤为严重,只有一条key

> 我们可以对key加一个扰动,在hash上动手脚,使得出现次数多的key分布在不同reduce执行,这样能够让数据更均衡,但是会增加下面一种情况的开销

```
在1,10,100,400,800的测试集下(本地),
时间240s→220s
```

case4是大量不同的key,本身不需要一个扰动,但是由于为了提高别的case的性能这样做了,使得其本身的性能发生了下降,其时间是其他case的三到四倍,不过最后总体的时间还是ok的

但是上述策略只在本地有效(240s->220s),服务器上的开销太高了,甚至原来只要500s,现在跑不完

并发优化:Combiner

于是我们决定使用combiner
修改mapcount和reducecount,原来是生成一条记录(key,""),现在我们修改生成规则,使得value是有意义的,即为当前文件中该key出现的次数(相当于预先reduce一下),reduce count就只需要把key一样的value相加即可.
这样可加速五倍!

1,10,100,400,800MB的测试集下
通过在MapCount下加combiner
240s->40s

修改topk算法

算法优化,因为只需要top10, 那么过多的记录是没有必要的,我们可以使用中位数partation算法求出前10个url