# Spark性能优化拾遗

## Spark中可以每次shuffle时多次训练，用数组等装多次训练的结果，最终要比外面多次训练性能好

Shuffle(Array.tabulate(I => loop(i, a)) {

…

}要比

Loop() {

Shuffle(a)

}

要快

比价好的例子是spark的KMeans

测试的性能效果如下：

|  |  |  |
| --- | --- | --- |
| 次数id | 执行方式 | 执行时间 |
| 1 | 外层执行10次时间 | 54 678 502 |
| 并行训练10个模型时间 | 11 454 794 |
| 2 | 外层执行10次时间 | 36 555 154 |
| 并行训练10个模型时间 | 5 230 959 |
| 3 | 外层执行10次时间 | 40 251 928 |
| 并行训练10个模型时间 | 7 440 861 |
| 4 | 外层执行10次时间 | 37 958 866 |
| 并行训练10个模型时间 | 5 899 895 |
| 5 | 外层执行10次时间 | 40 952 640 |
| 并行训练10个模型时间 | 6 722 452 |

也就是：

每次shuffle的数据量 \* ( shuffle次数 \* 执行shuffle次数) <

(每次shuffle的数据量 \* 执行shuffle次数) \* shuffle次数

分析：虽然shuffle传递的总数据量一样，数据真正传递的时间一样，但由于shuffle性能消耗中很大一部分是建立shuffle writer和shuffle reader的过程。方案一shuffle次数多，建立的shuffle writer也多，而方案二shuffle次数少，建立的少。

A = 连接消耗

B = 数据传递消耗

T1 = shuffle次数

T2 = 执行shuffle次数

方案1：(A + B)\*T1\*T2

方案2：(A + B\*T2)\*T1