计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(3):218−222 [doi: 10.15888/j.cnki.csa.007351]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 YARN 资源调度器的 MapReduce 作业数调节方法①

廉　华, 刘　瑜

(浙江理工大学 机械与自动控制学院, 杭州 310018)
通讯作者: 刘　瑜, E-mail: liuyu8099@163.com

摘　要: YARN 是 Hadoop 的一个分布式的资源管理系统, 用来提高分布式集群的内存、I/O、网络、磁盘等资源的利用率. 然而, YARN 的配置参数众多, 要对其人工调优并获得最佳的性能费时费力. 本文在现有的 YARN 资源调度器的基础上, 结合了一种闭环反馈控制方法, 可在集群运行状态下动态地对 MapReduce(MR) 作业数进行优化, 省去了人工调整参数的过程. 实验表明, 在 YARN 的容量调度器和公平调度器的基础上使用该方法, 相比于默认配置, MR 作业完成时间分别减少 53% 和 14% 左右.

关键词: 大数据; 资源调度器; MapReduce; 闭环反馈控制

引用格式: 廉华,刘瑜.基于 YARN 资源调度器的 MapReduce 作业数调节方法.计算机系统应用,2020,29(3):218–222. http://www.c-s-a.org.cn/1003-3254/7351.html

Number Adjustment Method of MapReduce Jobs Based on YARN Resource Scheduler

LIAN Hua, LIU Yu

(Faculty of Mechanical Engineering & Automation, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: YARN is a distributed resource management system of Hadoop. It can be used to improve the utilization of memory, I/O, network, disk and other resources of distributed cluster. However, there are many configuration parameters in YARN. Due to this reason, manual tuning of Hadoop performance to get the best performance is difficult and time-consuming. Based on the existing YARN resource scheduler, a successive approximation closed-loop feedback control method is proposed. This method can dynamically tune the parallel number of MapReduce (MR) jobs in the running state of the cluster, and eliminating the process of manual adjustment of parameters. Experiments show that the proposed approach reduces the MR operation time for 53% and 14% based on capacity scheduler and fair scheduler, respectively, compared with the default configuration.

Key words: big data; resource scheduler; MapReduce; closed-loop feedback control

Apache Hadoop 是 Apache 软件基金会的顶级开源项目之一, 它在分布式计算与存储上具有先天的优势. 随着大数据时代的到来, Hadoop 成为了大数据的标签之一, 在很多领域都得到了广泛的应用[1]. 当前 Hadoop 已然发展为拥有 MapReduce (MR)、HDFS、YARN、Pig、Hive、Hbase 等系统的较为完整的大数据生态圈[2]. YARN (Yet Another Resource Negotiator) 资源管理系统是 Hadoop 生态圈中的核心, 很多计算框架都迁移到 YARN 上, 借助 YARN 获得了更好的资源调度、资源隔离、可扩展性以及容灾容错的特性, 比如 MR on YARN, Strom on YARN, Spark on YARN 等[3]. 但是, YARN 有上百个参数供用户配置, 而且大多数参

① 收稿时间: 2019-08-07; 修改时间: 2019-09-05, 2019-09-23; 采用时间: 2019-10-21; csa 在线出版时间: 2020-02-28

数间有着密切联系. 对于生产集群来说, 手动调节任何参数都需要用户对集群管理机制有深入的理解以及集群性能长期的观察, 并且调节参数后还需用户花费很多时间来判断集群性能是否达到预期的效果[4]. 对于开发者来说在面对上百个参数的情况下很难实现 YARN 的最佳性能, 而 YARN 性能对于 MR 作业有很大影响[5]. 因此, 设计一种自动调节 YARN 配置参数的方法不仅有助于减少用户在参数调节上所花费的时间, 同时适当的参数配置还可以减少作业的执行时间, 提高作业的吞吐量. 近年来, 研究人员开发了很多种算法来提高 Hadoop 集群的性能. Bei ZD 等[6]提出了 RFHOC 模型, 利用随机森林方法为 map 和 reduce 阶段构建性能模型集合, 在这些模型的基础上使用遗传算法自动调整 Hadoop 配置参数. Chen CO 等[7]利用基于树的回归方法对集群的最优参数进行选取. 这类利用机器学习构建性能模型来寻找最优参数的方法需要一个费事的模型训练阶段. 同时, 若作业的类型发生改变或出现集群中节点增加或减少等变化时, 使用这类方法就需要重新训练模型. Zhang B 等[8]发现作业在运行过程中会出现 under-allocation 和 over-allocation 两种情况会降低集群性能. Under-allocation 表示 MR 作业数被设置的过少, 导致集群的资源利用率过低. Over-allocation 表示 MR 作业过多, 大部分资源分配给了 AppMaster, 导致分配给 Container 计算任务的资源不足, 甚至会出现 AppMaster 只占用资源却不进行任何计算的情况, 导致集群性能下降. 为克服以上两种情况, 张博等人提出一种闭环反馈控制方法, 以内存使用状况作为输入, 以最小化资源竞争和最大化资源利用率为目标来调整 MR 作业数, 但对内存的使用状况考虑不够细腻, 会出现无法跳出 over-allocation 状态的情况, 例如当集群中计算任务占用总内存的比率小于 T2 时集群处于 over-allocation 状态, 此时向集群提交大量 MR 作业后, 又会将集群认定为 under-allocation 状态, 导致控制量不发生变化, 这时集群实际上处于 MR 作业数被设置的过少, 集群的资源利用率过低的 over-allocation 状态. 本文提出了一种对 YARN 容量调度器和公平调度器参数调节的闭环反馈控制方法, 该方法采用二分法原则, 逐次缩小被控参数的调节步长, 具有调节速度快, 控制精度高的优点. 相比于文献 8 本文将公平调度器参数也纳入调节范围, 同时修改了集群状态的判定条件防止无法跳出 over-allocation 状态. 本文方法有效解决了 MR 作业运行过程中 AppMaster 的 under-allocation 与 over-allocation 的问题, 相比于现有的参数自整定方法具有如下优点: 1) 当集群中有节点增加减少或更换集群时, 无需花费任何时间调整方法模型; 2) 不必事先对 YARN 框架或被提交的 MR 作业进行修改.

1 相关知识

1.1 YARN 体系架构

YARN 主要依赖于 3 个组件 ResourceManager、NodeManager 和 AppMaster 来实现所有的功能[9]. 组件 ResourceManager 是集群的资源管理器, 整个集群只有一个, 负责集群整个资源的统一管理和调度分配, 包含两个部分: 可插拔的资源调度器和 ApplicationManager, 用于管理集群资源和作业. 组件 NodeManager 位于集群中每个计算节点上, 负责监控节点的本地可用资源, 故障报告以及管理节点上的作业, 它将节点上的资源信息上报给 ResourceManager. 组件 AppMaster 是每个作业的主进程, 用于管理作业的生命周期. 一个作业由一个 AppMaster 及多个 Container 组成, 其中 Container 是节点上的一组资源的组合, 例如 (1 GB, 1 core), 用于运行作业中的计算任务. 当客户端有作业提交时, ResourceManager 会启动一个 AppMaster, 之后再由 AppMaster 根据当前需要的资源向 ApplicationManager 请求一定量的 Container. ApplicationManager 基于调度策略, 尽可能最优的为 AppMaster 分配 Container 的资源, 然后将资源请求的应答发给 AppMaster[10]. 上述作业提交流程如图 1 所示.

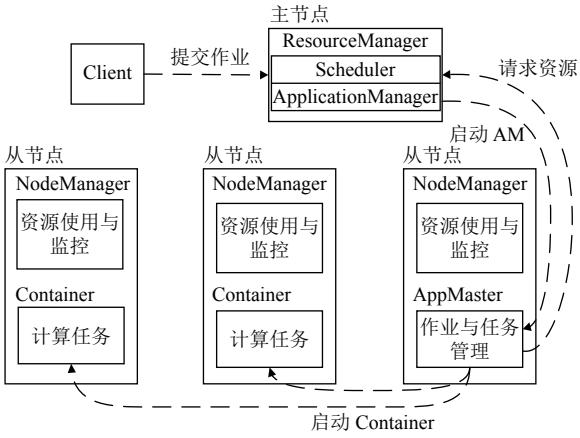

图 1 作业提交流程

1.2 YARN 的资源调度器

YARN 中自带的可插拔资源调度器有先入先出调度器 (FIFO scheduler)、容量调度器 (capacity scheduler) 和公平调度器 (fair scheduler)[11]. 其中先入先出调度器以“先来先服务”的原则, 按照作业提交的先后时间进行调度, 无需任何配置. 但这种调度策略不考虑作业的优先级, 只适合低负载集群, 当使用大型共享集群时, 大的作业可能会独占集群资源, 导致其他应用阻塞. 在大型共享集群中更适合容量调度器或公平调度器, 这两个调度器都允许大作业和小作业同时运行并都获得一定的系统资源[12]. 因此本文利用这两种调度器调节 MR 作业数. 在容量调度器中可设定参数 yarn. scheduler. capacity. maximum-am-resource-percent (AMRP), 来调节集群中可分配于 AppMaster 的资源量, 这个参数影响着集群中 MapReduce 作业数. 在公平调度器中可设定参数 maxAMShare, 调节 MapReduce 作业数, 它限制了 AppMastter 可占用队列资源的比例.

2 MapReduce 作业数控制方法

本文提出的 MR 作业数控制方法是基于容量资源调度器和公平资源调度其实现的, 是一种闭环反馈控制, 控制系统的方框图如图 2 所示. y_{sp}为用户所设定的域值 $T1$、$T2$、$T3$, 将监控器得到的内存使用情况与设定值y_{sp}相比较, 以此调节控制量 A, 防止 Hadoop 集群中 MR 作业出现 under-allocation 和 over-allocation 两种状态, 并使被控变量 Hadoop 集群中 MR 作业数保持在最佳值. 阈值 $T1$、$T3$ 与集群中所有运行作业的内存使用率相比较, 阈值 $T2$ 与集群中所有计算任务的内存使用率相比较.

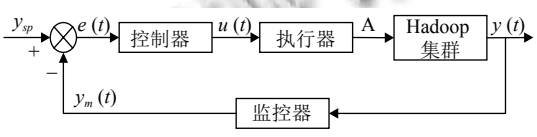

图 2 闭环反馈控制系统的方框图

本文提出的闭环反馈控制方法主要分为 3 个模块: 监控器、控制器和执行器:

(1) 监控器: 在集群主节点中运行, 负责周期性监控集群资源使用情况和 MR 作业运行情况, 并将监控到的数据传输至控制器中. YARN ResourceManager 有一个 Web 端口, 用这个端口可以查看 YARN 监控页面, 使用户可以方便地了解集群的运行情况和作业的运行情况. 监控器得到的数据是由 Python 脚本爬取 YARN 监控页面获得, 而未使用 YARN 自带的用户命令编写 shell 脚本. 原因是由 YARN 自带的用户命令获取数据相较于使用 Python 爬取较慢, 且受集群运行状况影响较大. 监控器具体获得的指标有: 队列中等待提交的作业数、队列中正在运行的作业数、集群中所有运行作业占用内存、集群中所有从节点总内存.

(2) 控制器: 负责根据从监控器接收到的数据对资源调度器的参数进行周期性地修改. 本文调度器参数控制算法以集群内存使用情况和队列中的作业状态作为判断条件, 对集群是否处于 over-allocation 或 under-allocation 状态做出判断, 若集群处于上述两种状态则采用二分法原则逐次调节被控参数直至集群从这两种状态中跳出. 参数控制算法的流程描述如算法 1.

算法 1. 调度器参数控制算法

输入: *P*: 队列中等待提交的作业数
R: 队列中正在运行的作业数
M_AMs: 集群中所有 AppMaster 占用内存
M_tasks: 集群中所有计算任务所占用内存
M_used: 集群中所有运行作业占用内存
M_total: 集群中所有从节点总内存
A: 参数 AMRP 或 maxAMShare 的值

```
1.  detect()//检测调度器类型, 读取调度器参数
2.  If P=0 Then
3.      If R(t)<R(t−1) Then
4.          decrease_A()
5.          If R=0 and A=A_min Then
6.          n=1
7.          End If
8.      End If
9.  Else
10.     If M_used/M_total<T1 Then
11.         If P(t)>P(t−1) Then
12.             increase_A()
13.         Else If M_used/M_total>T3 and M_tasks/M_total<T2
Then
14.             decrease_A()
15.         End If
16.     Else
17.         If M_tasks/M_total<T2 Then
18.             decrease_A()
19.         End If
20.     End If
21. End If
22. n=n+1
```

算法 1 中P、R、M_used、M_total的值由监控器

传入, M_AMs、M_tasks由如下公式获得:

$$M\_AMs = R \times Unit \times 2 \quad (1)$$

$$M\_tasks = M\_used - M\_AMs \quad (2)$$

其中, $Unit$为每个 Container 的内存大小. $detect()$ 通过读取 YARN 配置文件检测调度器类型, 随后读取相应调度器配置文件得到变量 A 的值. A_min和A_max是调度器参数 AMRP 或 maxAMShare 允许达到的最小值和最大值.

当$M_used/M_total < T1$且集群中等待提交的作业在增加时 (第 11 行), 可认为没有足够的内存分配给 AppMaster, 集群中并行的 MR 作业数量较少, 导致出现 MR 作业占用内存小于设定阈值$T1$的情况. 此时集群处于 under-allocation 状态, 需增加A的值, 提升 MR 作业数, 充分利用集群的资源.

集群处于 over-allocation 状态可分为如下 3 种情况:

① 队列中无等待作业且队列中正在运行的作业数比上一时刻少, 即 P=0 且$R(t) < R(t-1)$, 表示集群中并 MR 作业数较低或正在降低, 这种状态判断为 over-allocation. 此时可减少分配到 AppMaster 的资源, 使计算任务可使用的资源增多, 以此来减少 MR 作业总体完成时间.

②$T3 < M_used/M_total < T1$且$M_tasks/M_total < T2$(第 13 行), 当集群中提交的任务较少时很容易满足$M_used/M_total < T1$和$M_tasks/M_total < T2$条件设置阈值$T3$表示当被使用的内存足够多时才会判断集群是否为 over-allocation 状态. 若集群处于 over-allocation 状态, 则需减小 A 的值, 使集群跳出此状态.

③ $M_used/M_total > T1$且$M_tasks/M_total < T2$ (第 17 行), 表示 MR 作业占用内存大于设定阈值 $T1$, 但 MR 作业中的计算任务占用内存小于设定阈值 $T2$. 这种情况下集群 MR 作业数过高, 集群中计算任务得不到足够的资源, 此时需要减小 A 的值, 降低 MR 作业数, 使计算任务获得更多的资源.

算法 1 中调节变量 A 的方法 $increase\_A()$ 和 $decrease\_A()$ 分别如算法 2 和算法 3 所示. 当增加或减小变量 A 时, 将确定$\alpha/2^n$是否大于用户设置的 $step$, 若大于则以$\alpha/2^n$作为此次调节的步长, 若小于则以 $step$ 作为此次调节的步长.

(3) 执行器: 负责将修改对应调度器配置文件的 AMRP 或 maxAMShare 参数, 命令 Hadoop 集群重新加载调度器配置.

算法 2. $increase\_A()$

```
1.  α=A_max−A
2.  If α/2^n>step Then
3.      A=A+α/2^n
4.  Else
5.      A=A+step
6.  End If
```

算法 3. $decrease\_A()$

```
1.  α=A−A_min
2.  If α/2^n>step Then
3.      A=A−α/2^n
4.  Else
5.      A=A−step
6.  End If
```

3 实验分析

3.1 实验环境

为了验证 MR 作业数控制方法的效果, 选取了 5 台普通 PC 机来搭建测试集群. 其中一台作为主节点, 包含 1 个 NameNode 角色和 1 个 ResourceManager 角色. 其余 4 台作为从节点, 每台节点包含 1 个 DataNode 角色和 1 个 NodeManager 角色. 本文选取 Grep, Terasort, Wordcount 这 3 种常见的 MR 作业用于测试, 其中 Terasort 为 IO 密集型作业, Grep 为计算密集型作业, Wordcount 作业在 Map 阶段为计算密集型, Reduce 阶段为 IO 密集型, 如表 1 所示.

表 1 实验作业类型

MR 作业名称	任务数 (Map: Reduce)	任务大小 (MB)
Grep(GR)	6(5:1)	100
Terasort(TS)	6(5:1)	100
Wordcount(WC)	6(5:1)	100

3.2 实验结果

本实验分别在将 YARN 集群资源调度器配置为容量调度器和公平调度器的情况下进行测试, 将四组作业提交到集群中 (第一组 30 个 Grep 作业、第二组 30 个 Terasort 作业、第三组 30 个 Wordcount 作业、第四组每种类型作业各 10 个), 对比在集群使用默认配置、最优配置和本文方法后作业的完成时间. 实验中 AMRP 和 maxAMShare 的最优值由穷举法得到, 穷举范围为{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}, 一

组作业完成时间越短可认为参数 AMRP 或 maxAMShare 越优.图 3 中在使用容量调度器的情况下, 每组作业完成时间最短时, AMRP 分别为 0.6, 0.5, 0.3, 0.5. 图 4 中在使用公平调度器的情况下, 每组作业完成时间最短时 maxAMShare 分别 0.8, 0.9, 0.6, 0.8. 实验中 *T*1、*T*2、*T*3、*step*、*A_ min*、*A_ max* 的值分别为 1.0, 0.5, 0.8, 0.05, 0.05, 0.95.

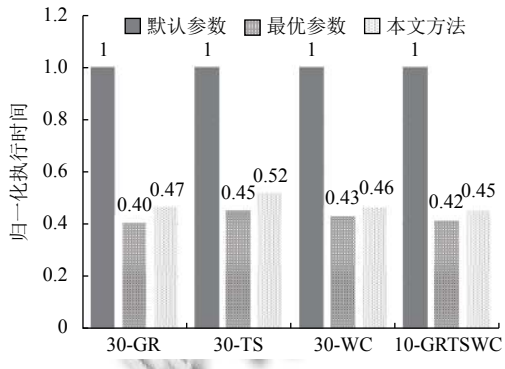

图 3 使用容量调度器时作业完成时间对比

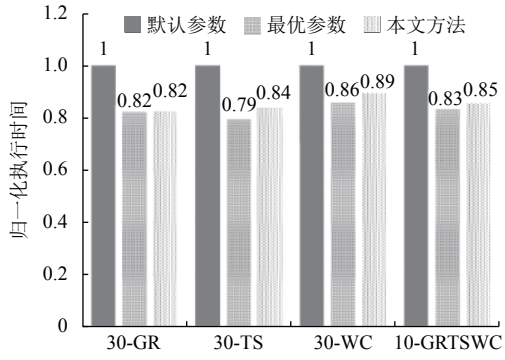

图 4 使用公平调度器时作业完成时间对比

从图 3、图 4 中可以看出使用本文提出的方法调节 MR 作业数相比于默认的配置, MR 作业整体完成时间会明显的减少, 在使用容量调度器和公平调度器的情况下平均减少了 53% 和 14%. 同时在参数最优的情况下, 与使用本文提出的方法在作业完成时间上相差较小. 两者与默认配置相比, 作业完成时间最多有 7% 的差异. 由于对于不同的作业组合, 都需要做多组实验才能确定最优的参数, 所以在实际使用集群时要达到明确的最优参数并不现实, 因此本文提出的方法与最优参数下的作业完成时间相比略有差异可以接受.

4 结论与展望

本文在现有调度器的基础上提出了一种闭环反馈控制方法对 MR 作业数进行动态调节. 通过实验证明本文提出的方法能够在省去人工调节参数的情况下有效的降低 MR 作业完成时间, 提升了集群的性能. 下一步的工作是研究集群虚拟核使用率与集群 CPU 使用率的关系, 将节点虚拟核数也纳入调节的范畴.

参考文献

1 吴岳. 基于 Hadoop 平台的云计算节能研究. 计算机系统应用, 2015, 24(11): 235–241.
2 李媛祯, 杨群, 赖尚琦, 等. 一种 Hadoop Yarn 的资源调度方法研究. 电子学报, 2016, 44(5): 1017–1024.
3 Murthy AC, Vavilapalli VK, Eadline D, 等 . Hadoop YARN 权威指南. 罗韩梅, 洪志国, 杨旭, 等译. 北京: 机械工业出版社, 2015. 31–33.
4 Lee GJ, Fortes JAB. Hadoop performance self-tuning using a fuzzy-prediction approach. Proceedings of 2016 IEEE International Conference on Autonomic Computing. Wurzburg, Germany. 2016. 55–64.
5 Babu S. Towards automatic optimization of MapReduce programs. Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis, IN, USA. 2010. 137–142.
6 Bei ZD, Yu ZB, Zhang HL, *et al*. RFHOC: A random-forest approach to auto-tuning Hadoop's configuration. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(5): 1470–1483. [doi: 10.1109/TPDS.2015.2449299]
7 Chen CO, Zhuo YQ, Yeh CC, *et al*. Machine learning-based configuration parameter tuning on hadoop system. Proceedings of 2015 IEEE International Congress on Big Data. New York, NY, USA. 2015. 386–392.
8 Zhang B, Krikava F, Rouvoy R, *et al*. Self-configuration of the number of concurrently running MapReduce jobs in a hadoop cluster. Proceedings of 2015 IEEE International Conference on Autonomic Computing. Grenoble, France. 2015. 149–150.
9 张垚杰. 基于 YARN 的混合结构调度器的研究和优化[硕士学位论文]. 哈尔滨: 哈尔滨工业大学, 2018.
10 于金良, 朱志祥, 李聪颖. Hadoop MapReduce 新旧架构的对比研究综述. 计算机与数字工程, 2017, 45(1): 83–87.
11 王荣丽, 侯秀萍. 基于优先级权重的 Hadoop YARN 调度算法. 吉林大学学报 (信息科学版), 2017, 35(4): 443–448.
12 董春涛, 李文婷, 沈晴霓, 等. Hadoop YARN 大数据计算框架及其资源调度机制研究. 信息通信技术, 2015, (1): 77–84.