文章编号:1009-3087(2015)06-0104-09 DOI:10.15961/j.jsuese.2015.06.015

基于垃圾回收的 MapReduce 作业内存调优

罗永刚,陈兴蜀*,王煜骢
(四川大学 计算机学院 网络与可信计算研究所,四川 成都 610065)

摘 要:针对合理管理 MapReduce 作业内存资源困难的问题,提出评估方法并给出优化配置建议。首先分析 Java 虚拟机的内存分配与垃圾回收的原理,给出垃圾回收重要指标;其次提出内存分配合理性评估的 3 种指标和评估方法;最后根据评估结果给出 2 种优化配置建议:一是通过使用聚类算法和统计信息来估计晋升对象大小阈值,优化 Java 虚拟机的对象分配和垃圾回收性能;二是使用回归模型和搜索算法来预测作业合理的内存配置。实验结果表明,提出的方法能自动发现作业内存配置的不足并给出优化的配置建议。与采用机器学习方法相比,提出的方法不需要运行大量的测试,因此该方法能很好适用于 MapReduce 的生产集群环境。

关键词:MapReduce;Hadoop;Java 虚拟机;垃圾回收;资源优化
中图分类号:TP309 **文献标志码**:A

GC-based MapReduce Job Memory Tuning

LUO Yonggang, CHEN Xingshu, WANG Yucong*
(Network and Trusted Computing Inst., College of Computer Sci., Sichuan Univ., Chengdu 610065, China)

Abstract: Different Job requires different memory resources, and it is difficult to assess the rationality for a memory allocation to a MapReduce Job. In order to solve this problem, an assessment method was presented and recommended for memory settings of JVM where Job's tasks run. Firstly, some important GC metrics were introduced based on the analysis of JVM's memory allocation and GC workflow in-depth. Then, three kinds of indicators and memory allocation rationality evaluation method were introduced based on the three indicators. Finally, two kinds of optimal JVM configuration were recommended, which are using K-means algorithm and statistical information to estimate the threshold value of the object size which should have been allocate in old generation, and modeling GC pause time and using search algorithm to predict the size of young generation and the old generation, respectively. Experimental results showed that the proposed approach can automatically find insufficient of memory configuration of a Job. Compared with using machine learning methods, the proposed method does not need to run a large number of test cases, so it can apply to production cluster of MapReduce.

Key words: MapReduce; Hadoop; JVM; GC; Memory tuning

谷歌公司在文献[1]中提出一种处理大数据[2]的编程模式 MapReduce,其开源版本的实现 Apache Hadoop(包含 HDFS 和 MapReduce 2 个核心组件)得到了学术界、工业界的积极支持,已成为离线分析大数据的事实标准。Mahout、HBase、Pig[3]、Hive[4]等框架的出现完善了 Hadoop 的整个生态圈,使 Hadoop 的适用范围越来越广,Hadoop 集群规模也不断扩大,每个集群运行的作业越来越多样化,很难为大量不同作业找出共同的合理资源配置。

MapReduce 的作业(以下简称作业)性能与资源有关,已有文献研究了其性能优化的问题。作业的性能(如执行时间)可看成是 MapReduce 程序(p)、输入数据(d)、可用资源(r)和 MapReduce 相关的配置参数(c)的函数,即 $perf = F(p,d,r,c)$[5],

收稿日期:2014-12-10
基金项目:国家科技支撑计划资助项目(2012BAH18B05)
作者简介:罗永刚(1980—),男,博士生. 研究方向:大数据信息安全. E-mail:iamlyg98@163.com
* 通信联系人 E-mail:chenxsh@scu.edu.cn

求解此函数关系的方法很多,包括黑盒模型[6]、白盒模型[5]和灰盒模型[7-8]。不管采用什么方法,内存资源是影响 r 和 c 的重要因素。这些方法仅涉及 MapReduce 框架和作业层的资源相关配置,未考虑作业运行环境 Java 虚拟机(即 JVM)的配置。

文献[9]使用数据流、JVM 堆的使用情况来预测作业的内存需求,由于采用固定时间间隔来采样 JVM 堆的使用情况,不能全面掌握 JVM 内存的使用信息。文献[10]使用机器学习算法实现 JVM 内存管理的调优。预测性能受训练数据的取值范围影响。文献[11]提出分析 JVM 垃圾回收性能的工具 Shrek。文献[12]提出一种通用的 JVM 垃圾回收日志的分析系统。文献[11-12]很难直接应用到 MapReduce 环境。

JVM 的内存管理策略对 MapReduce 作业具有重大影响。当内存分配不足时可能降低作业的运行性能甚至引起内存溢出错误。当晋升对象阈值设置不合理时会造成不必要的对象晋升,增加垃圾回收发生的次数和应用暂停的时间,增加作业的运行时间。JVM 分代内存大小设置不合理也会造成资源浪费或降低作业的性能。在 MapReduce 的生产集群中很难使用不同的配置参数进行大量测试,进一步增加了内存调优的困难。

为解决这些问题,作者在分析 JVM 垃圾回收管理器的内部实现原理的基础上,通过分析垃圾回收日志、使用回归模型、分代大小预测算法、晋升对象阈值预测算法,实现作业内存配置的优化,具体包括:

1)提出内存分配合理性的评估方法和 3 种指标;

2)使用回归模型建模垃圾回收暂停时间,使用分代大小预测算法估计年轻代和年长代的合理大小;

3)使用 K-means 聚类算法预测晋升对象阈值大小。

1　MapReduce 作业与垃圾回收

MapReduce 作业是对特定输入数据集进行处理,并产生特定输出数据的程序运行实例,每个作业由 Map 阶段和 Reduce 阶段组成,Map 阶段和 Reduce 阶段分别由一个或多个 Map 任务和 Reduce 任务构成。Map 任务和 Reduce 任务运行在单独 JVM 之上。

在 Map 任务和 Reduce 任务运行过程中,不断从 JVM 所管理的堆内存(如无特殊说明,文中等价使用堆内存、内存和堆)上申请空间,当对象不再有效时,由 JVM 执行垃圾回收,释放这些对象所占用的空间。

1.1　MapReduce 作业内存使用概述

Map 任务的主要工作流程是:Map 任务会不断读取需要处理的数据,处理后写入缓冲区,当缓冲区写满后,将这些数据写入本地文件[13]。从内存使用角度看,Map 任务会开辟一块大的缓冲区域来缓存 Map 函数的输出记录,这部分缓存区域占据绝大多数的年长代空间。

Reduce 任务包括 3 个主要阶段,分别是数据拷贝阶段(即 Shuffle 阶段)、排序阶段(即 Sort 阶段)和 Reduce 阶段(即执行用户 Reduce 函数的阶段)。在拷贝阶段中,Reduce 任务不断申请较大空间的缓存区来保存拷贝的数据,当空闲内存不够时将缓存的数据进行排序,写入本地文件,形成中间数据文件,然后释放这些缓存数据所占用的空间。在排序阶段,Reduce 任务会对所有已拷贝的数据进行排序。在 Hadoop1.2.1 版本中,Reduce 任务排序操作只是根据每个 Map 任务的输出数据的第一条记录的键的大小构建一个堆结构。在 Reduce 阶段,Reduce 任务不断读取记录,处理完成后将结果写入分布式文件系统。

1.1.1　JVM 分代内存结构

JVM 首先将内存分为堆和非堆 2 个区域来进行管理,对于堆内存,JVM 采用分代管理机制[14-15]。JVM 使用自动内存管理技术管理堆内存,其主要功能是为新对象分配所需空间,当内存紧张时自动执行垃圾回收操作,将无效对象所占空间回收。JVM 使用恒久代(即 Perm 代)保存 Java 程序的字节码等数据。

JVM 的堆内存布局如图 1 所示。堆内存划分为年轻代(Young Generation)和年长代(Old Generation or Tenured Generation)区域。年轻代通常为小对象分配所需存储空间,年长代通常为“大对象”或长期有效的对象分配存储空间[14]。根据功能的不同,年轻代进一步划分为 2 类区域,即:1)分配对象存储空间的区域(即 Eden 区域);2)保存垃圾回收后仍有效的对象的区域(即对象幸存区——Survivor 区域,包括 From 和 To 两个区域)。

1.1.2　内存分配与垃圾回收

当 Java 程序使用 new 关键字实例化一个对象或者数组时,由 JVM 的内存管理模块为其分配所需

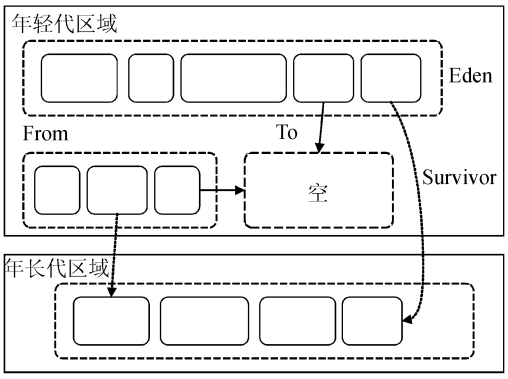


图1 JVM分代内存布局

Fig. 1 JVM generation memory arrangement

内存空间,当没有足够空闲内存供分配时触发一次垃圾回收事件。为便于描述,定义以下几个术语:

1)垃圾对象:没有被引用的对象称为垃圾对象[16],简称垃圾。

2)垃圾回收(GC):发现并释放垃圾对象的过程为垃圾回收,绝大多数的垃圾回收是因为不能满足对象分配请求导致。

3)Minor GC:年轻代不能满足对象分配请求时触发,对年轻代执行垃圾回收。Minor GC的时间通常很短。

4)Major GC:年长代或Perm代不能满足对象分配请求时触发,对年长代、持久代执行垃圾回收,有些垃圾回收算法会扫描整个堆空间。与Minor GC相比,Major GC的时间通常要长很多。

在执行Minor GC时,垃圾回收器首先识别有效的对象,将Eden和From区域中的有效对象拷贝到To区域。如果To空间不足以保存年轻代中的有效对象时,将对象直接拷贝到年长代。Minor GC执行完成后To和From区域角色互换。如果年轻代中的对象在经历多次Minor GC后仍然有效,则将这些对象拷贝到年长代,此过程称为对象晋升。对象晋升需要执行对象拷贝操作,占用CPU资源。对于有效时间较长的"大对象",应该直接在年长代中分配。

1.2 垃圾回收指标

垃圾回收指标展现了连续两次GC之间、以及一次GC执行前后对象的变化情况等重要信息。明确GC指标是分析GC性能的基础,文献[17]提出了一些指标,作者在此基础之上提出更加全面的指标。

主要指标解释如下:

年轻代中分配对象 S^i_{new}:表示在第 $i-1$ 次和第 i 次GC之间,年轻代区域分配的对象大小。年长代中分配对象 S^i_{old}:表示在第 $i-1$ 次和第 i 次GC期间分配在年长代中的对象大小。

晋升对象 S^i_{promo}:表示在第 i 次Minor GC执行时晋升对象的大小。

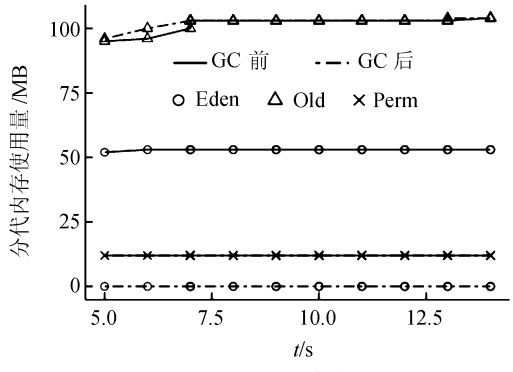
年轻代垃圾对象 S^i_{minor}:表示第 i 次Minor GC回收的对象大小。

年长代垃圾对象 S^i_{major}:表示第 i 次Major GC回收的对象大小。

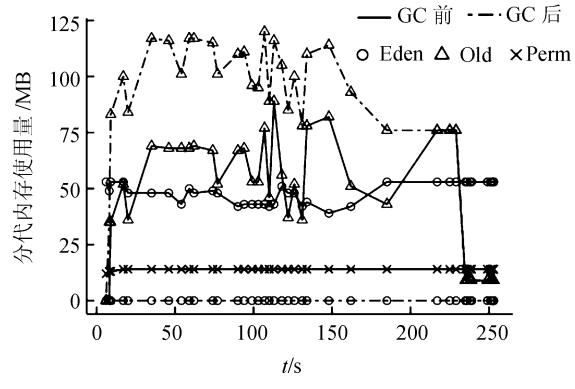
1.3 作业的GC分析

MapReduce以及JVM均使用默认设置,运行PUMA[18]的WordCount测试程序,其主要功能是统计10GB的XML文件中各单词出现的次数。同一个作业的Map任务和Reduce任务的GC特征相似,因此只选择其中1个Map任务和Reduce任务的GC数据进行分析。

图2中(a)是1个Map任务的虚拟机内存各区域使用时序图,(b)为Reduce任务的时序图。其中Perm区域的GC之前和GC之后的线条几乎重合。



(a) Map任务



(b) Reduce任务

图2 Map任务和Reduce任务JVM的各内存区域使用量时序图

Fig. 2 Generations' usage timing plot for a Map and Reduce task

从图2可以得知:

1)对于Map任务:①年长代使用量比较稳定,且没有触发Major GC。年长代空间未被充分利用

(从图 2 中可知 Map 任务大约有 12MB 的空间没有被使用)。②Eden 空间几乎耗尽后才触发 Minor GC,说明年轻代分配的小对象占多数。③Perm 区域的使用量几乎没有发生变化。

2)对于 Reduce 任务:①年长代的使用量出现了多次大的波动,说明年长代中出现了大量的较短时间的"大对象",且发生在 Reduce 任务的开始阶段,此时 Reduce 任务正在拷贝 Map 任务的输出数据。②出现多次 Eden 还有大量的空闲空间时触发 GC 操作的情况。说明有"大对象"分配在年轻代。③Perm 区域的使用量几乎没有发生变化。

图 3 为一个 Map 任务和 Reduce 任务的对象晋升时序图,从图 3 易知:①Map 任务中分配在 Eden 中的对象绝大多数为临时对象,不需要晋升到年长代,且几乎每次 GC 后 Eden 空间被完全释放。②Reduce 任务存在大量对象需要晋升到年长代,从时间上看,大量对象晋升操作发生在 Reduce 的 Shuffle 阶段。

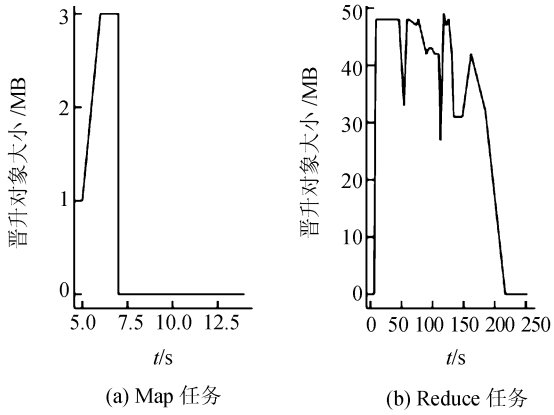


图 3　Map 和 Reduce 任务对象晋升时序图

Fig. 3　Object promotion timing plot for a Map and Reduce task

从图 2 和 3 不难发现使用 JVM 的默认配置时,存在多处需要优化的地方,概括起来主要有:

1)Map 任务和 Reduce 任务的内存需求不同,需要使用不同的配置;

2)Map 任务年轻代中大量对象是临时对象,应该根据这个特点进一步减少年轻代的空间;

3)Reduce 任务存在大量的晋升操作,降低了 JVM 的性能,需要设置一个对象晋升阈值,使大于该阈值的对象直接分配在年长代。

2　作业内存评估与调优

2.1　内存分配情况评估

作者定义内存评估指标包括:

1)GC 代价:GC 暂停时间占任务运行时间的比率,取值范围(0,1);

2)最大内存使用率:各分代内存占用量的最大值与各分代内存最大值的比率,取值范围(0,1);

3)From 中分配对象:逻辑值,为真时表示发生了在 From 区域中直接分配对象的事件。

在理解 JVM 内存管理原理和对象分配工作流程基础之上,通过分析 GC 指标将内存分配情况分为以下几类:

1)内存分配过剩:最大内存使用率小于设定阈值(文中设置为 0.9)。对于 Map 任务和 Reduce 任务来讲,内存分配过剩对减少运行时间没有贡献,造成内存资源浪费。

2)内存分配紧张:没有发生内存溢出错误(即 OOM 错误),且以下 2 个条件任何一个成立:

①From 中分配对象为真;

②GC 代价大于阈值。

3)内存分配不足:发生了 OOM 错误。

4)内存分配合理:其他情况视为内存分配合理。

2.2　作业内存调优

内存调优包括预测作业 Map 任务和 Reduce 任务 JVM 的各分代内存的大小、计算 Perm 的大小、估计对象晋升阈值。

通过图 2 可以看出,Perm 使用量在整个 Map 任务和 Reduce 任务运行期间表现非常稳定。为了避免 Perm 代内存紧张触发 Major GC 操作,Perm 代的大小取 Map 任务和 Reduce 任务运行期间使用量的最大值。

2.2.1　内存大小预测

1)年长代和年轻代大小预测方法

定义 T_{Task} 表示 Map 任务或 Reduce 任务的执行时间,$T_{\text{Threshold}}$ 表示 GC 吞吐率阈值,由此可以得到期望的 GC 暂停时间 $T_{\text{GC}} = T_{\text{Task}} \times T_{\text{Threshold}}$,$T_{\text{Threshold}}$ 默认设置为 0.01。

在默认设置情况下,年轻代的分配通常是过剩的,因此算法 2 只考虑保持 Minor GC 暂停时间不变的情况下减少年轻代的大小。

过程 1:年轻代和年长代大小预测框架

①分别计算年轻代和年长代 GC 暂停时间的期望值,年轻代的期望时间为当前 Minor GC 的总暂停时间,用 T_{Young} 表示,年长代的期望时间为 $T_{\text{Old}} = T_{\text{GC}} - T_{\text{Young}}$。

②检查内存评估输出结果,如果内存分配紧张

执行步骤3,否则执行步骤4。

③调用算法1预测年长代的大小,然后调用算法2预测年轻代的大小。

④检查年长代是否分配过剩,过剩时将新的年长代的大小设置为当前年长代大小和推荐大小的最小值。推荐值为当前年长代占用量的最大值加10 MB的冗余量。然后调用算法2预测年轻代的大小。

算法1 年长代内存预测算法

M_{old}^{cur} 为年长代当前的大小,S_{old} 为分配在年长代中对象的总大小, ceiling 为取大于实数值参数的最小的整数操作,f_{Major} 表示式(2)的 $f_{Major}^{R}(M_{old})$。

输入:S_{old},M_{old}^{cur},f_{Major},允许的时间阈值 T,且 $T = T_{Old}$

输出:年长代预测大小。

1. IF JVM's heap is insufficient,THEN
2. $Step <- M_{old}^{cur}$
3. ELSE
4. $Step <- 50 * 1\ 024$
5. ENDIF
6. $M_{old}^{cur} <- M_{old}^{cur} + Step$
7. $t <- f_{Major}(M_{old}^{cur})$
8. $T_{Major} <- t * \text{ceiling}(S_{old}/M_{old}^{cur})$
9. IF $T_{Major} <= T$,THEN
10. RETURN M_{old}^{cur}
11. ELSE
12. GOTO 6
13. ENDIF

算法2 年轻代内存预测算法

M_{eden}^{cur} 表示 Eden 当前的大小,S_{Young} 表示分配在年轻代中对象的总大小,Map 任务的 S_{Young} 由式(4)计算, Reduce 任务的 S_{Young} 由式(5)计算; LIMIT 表示年轻代允许的最小值,由回归模型训练数据的最小值确定,ceiling 的含义与算法1相同。

输入:S_{Young},M_{eden}^{cur},期望的时间 T,且 $T = T_{Young}$, Minor GC 回归模型 f_{Minor}, Map 任务时为式(3),Reduce 任务时为式(1)

输出:年轻代的预测大小。

1. $Step <- 10 * 1\ 024$
2. $S_{Pre} <- M_{eden}^{cur}$
3. $M_{eden}^{cur} <- M_{eden}^{cur} - Step$
4. IF $M_{eden}^{cur} <= LIMIT$
5. $S_{Pre} <- LIMIT$
6. GOTO 15
7. $t <- f_{Minor}(M_{eden}^{cur})$
8. $T_{Minor} <- t * \text{ceiling}(S_{Young}/M_{eden}^{cur})$
9. IF $T_{Minor} >= T$,THEN
10. GOTO 15
11. ELSE
12. $S_{Pre} <- M_{eden}^{cur}$
13. GOTO 3
14. ENDIF
15. RETURN($S_{Pre} + 2 * S_{Pre}/8$)

算法1和算法2需要使用回归模型来估计给定内存大小下 Minor GC 或 Major GC 的平均暂停时间。为了保证回归模型的预测精度,分别训练 Map 任务的 Minor GC 的和 Reduce 任务的回归模型。回归模型的自变量为 Eden 或 Old 的大小,因变量为 GC 平均暂停时间,即:

$$t_{Minor}^{R} = f_{Minor}^{R}(M_{eden}) \tag{1}$$

$$t_{Major}^{R} = f_{Major}^{R}(M_{old}) \tag{2}$$

$$t_{Minor}^{M} = f_{Minor}^{M}(M_{eden}) \tag{3}$$

其中,t_{Minor}^{R} 和 t_{Major}^{R} 分别表示 Reduce 任务的 Minor GC 和 Major GC 的平均暂停时间,S_{eden} 和 S_{old} 表示 Eden 和 Old 区域的大小,t_{Minor}^{M} 表示 Map 任务的 Minor GC 的平均暂停时间。

2)估计 S_{Young}

Map 任务的 S_{Young} 近似等于直接分配在年轻代的对象大小之和,即:

$$S_{Young} = \sum_{i=1}^{n} S_{new}^{i} \tag{4}$$

式中,n 为垃圾回收的次数。

预测 Reduce 任务的 S_{Young} 比较复杂,其主要原因是 Reduce 任务存在大量对象晋升现象。默认情况下 JVM 的配置参数 PretenureSizeThreshold = 0,对于 Reduce 任务来讲这个设置不合理,因此需要将该参数设置为一个大于零的正整数(算法3)。本参数的设置会使得原先分配在 Eden 中的"大对象"直接分配到年长代,如果只有默认配置下作业运行的 GC 日志数据,很难预测本参数设置后直接分配在年轻代的对象大小。作者将晋升对象分为两类:第一类是缓存已拷贝的 Map 任务输出数据的"大对象",另一类是 MapReduce 框架或作业运行过程中需要的"大对象"或长期有效的对象。假设设定 PretenureSizeThreshold 后大对象直接分配在年长代,且第一类对象的大小与 Reduce 已拷贝的字节数呈线性关系,则有:

$$S_{\text{Young}} = \sum_{i=1}^{n} (S_{\text{new}}^{i} + S_{\text{old}}^{i}) - \alpha S_{\text{shuffle}},\ 0.9 \leq \alpha \leq 1.0 \tag{5}$$

其中,S_{shuffle} 表示 Reduce 任务从 Map 端拷贝的数据大小,可通过 Reduce 任务的计数器值获得。

3)估计 S_{old}

根据 JVM 对象分配原理,对象分配到年长代有以下几种情况:

· 对象是"大对象",直接分配到年长代;

· 对象从年轻代晋升到年长代。

因此,Map 任务或 Reduce 运行完成后,年长代中对象的大小之和为:

$$S_{\text{old}} = \sum_{i=1}^{n} (S_{\text{promo}}^{i} + S_{\text{old}}^{i}) \tag{6}$$

2.2.2　对象晋升阈值预测

默认情况下,只要待分配的对象不大于 Eden 的大小,串行垃圾回收器会尽力将对象分配在 Eden 区域。由图 2 知道,Reduce 任务的 Shuffle 阶段会频繁出现对象晋升现象,由此带来 2 个方面的不利影响:1)触发更多的 Minor GC 操作;2)增加每次 Minor GC 的暂停时间。

有效的解决方法是将 JVM 提供的配置参数 PretenureSizeThreshold 设定为适当值。当该参数设置为一个大于零的值后,JVM 在每次分配对象时首先判断申请空间是否大于本参数的设定值,如果大于则直接分配在年长代。理想情况是通过设置本参数使得所有缓存 Map 数据"大对象"直接分配在年长代。在生产集群中很难获得 Reduce 任务缓存 Map 数据的缓冲区的最小值,因此需要通过分析 GC 数据来估计合适的值。

预测晋升对象大小的思路基于以下 2 点:

1)如果缓存 Map 输出数据的"大对象"分配在 Eden 区域,则在为这些"大对象"分配空间的时候触发 Minor GC 的概率也较大,且此刻的 Eden 空闲空间必然小于"大对象"的大小。

2)触发 Minor GC 的新对象可以分为 2 类,一类是小对象,另一类是"大对象"。当为小对象分配空间时触发 Minor GC,此时 Eden 的空闲空间通常非常小(如只有几百 kB 等)。当为"大对象"分配空间时触发 Minor GC,此时的 Eden 的空闲空间通常较大。

算法 3　晋升对象阈值估计算法

输入:Minor GC 指标,每行表示一条 Minor GC 记录,记录条数等于 Minor GC 次数。

输出:对象晋升阈值的估计值。

1. 计算 Minor GC 时 Eden 的空闲空间向量 $\boldsymbol{a}$;

2. 计算初始的聚类中心 $\boldsymbol{c}$,$\boldsymbol{c}[1] = \max(a)$,$\boldsymbol{c}[2] = \min(a)$;

3. 使用 K-means[19] 聚类算法对 $\boldsymbol{a}$ 进行聚类,初始聚类中心为 $\boldsymbol{c}$;

4. 计算各聚类的摘要统计信息(包括最小值、均值、中位数、最大值等)

5. 返回选择均值较大的一个聚类的最小值。

3　实验与分析

Hadoop 测试集群有 6 个从节点,每个从节点分配一个 Map 任务槽和 1 个 Reduce 任务槽,Hadoop 版本为 1.2.1。测试程序为 PUMA 的 WordCount 测试程序,输入数据为 10 GB 的 XML 文件,HDFS 数据块大小为 64 MB,Reduce 任务在所有 Map 处理完成后才启动。每个从节点分配 4 GB 内存,排他性使用一个 E5 - 2609@2.4 GHz 的物理核。由于 Map 任务和 Reduce 任务运行的节点只有一个 CPU 核,因此 Map 任务和 Reduce 任务均选择使用串行垃圾回收器。JVM 使用 64 位 Linux 系统下 JDK1.7.0 update 45。

3.1　GC 回归模型分析

3.1.1　Reducce 任务 GC 回归模型

训练数据集通过运行 WordCount 后收集每个 Reduce 任务的垃圾回收数据获得。每次运行的 Reduce 任务的 JVM 堆大小设置为不同值,测试的堆大小包括 400、500、…、3 000 MB。年长代与年轻代的比例为 9,晋升阈值设置为 5 MB,Eden 与 From 的比值为 8。

回归模型选择简单线性回归模型,模型可表示为:

$$T = \beta_0 + \beta_1 x + \varepsilon \tag{7}$$

其中,x 表示每次 GC 回收的空间,kB;T 为 GC 平均暂停时间,μs;ε 为随机误差,且服从均值为 0,方差为 σ^2 的正太分布。x 的取值可以用下式近似:

$$x = \begin{cases} M_{\text{eden}}, & \text{for Minor GC;} \\ M_{\text{old}}, & \text{for Major GC} \end{cases} \tag{8}$$

Reduce 任务 Minor GC 的模型拟合残差标准误差为 1 846 μs,R^2 为 0.924 1,Major GC 的模型拟合残差标准误差为 8 999 μs,R^2 为 0.992 5。R^2 越接近 1,说明模型的拟合度越好。由 R^2 和残差标准误差得知 Reduce 任务的 GC 模型拟合度非常优秀[20-21]。

模型的预测精度如图 4 所示,图 4(a)是 Minor

GC 的预测值与实验值的对比图,(b)图是 Major GC 的对比图。实线表示实验数据,虚线表示预测数据。对于 Minor GC,实验值与预测值之间的平均误差小于 2 ms,Major GC 的平均误差小于 20 ms。

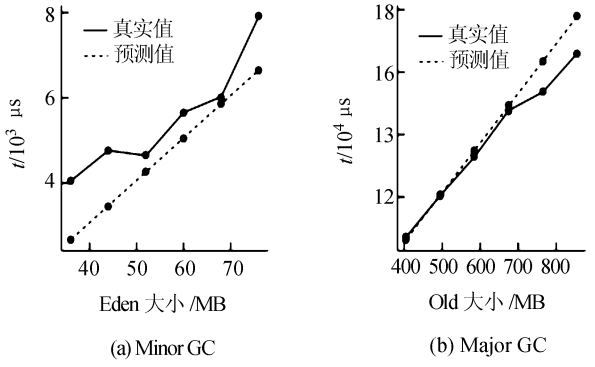


图 4 Reduce 任务的 GC 时间均值的预测值与实验值对比

Fig. 4 Prediction precision of Reduce GC regression models

3.1.2 Map 任务 GC 回归模型

Map 任务 JVM 的年长代空间主要用来保存 Map 的输出记录,需要晋升到年长代的对象很少。因此 Map 任务在执行过程中几乎没有发生 Major GC,因此本文只考虑 Map 任务的 Minor GC 的回归建模。

Map 任务 GC 回归模型仍然使用简单线性回归模型,自变量为 Eden 的大小,单位为 kB,因变量为给定 Eden 下所有 Map 任务的 Minor GC 的平均暂停时间,单位为 μs。作者收集了 18 个不同的 Eden 大小情况下 Map 任务 Minor GC 的平均暂停时间作为模型训练数据。Eden 的最小值为 12 288 kB,最大值为 81 920 kB。

拟合模型的 R^2 值为 0.972 2,残差标准误差为 141 μs,远小于 Minor GC 的毫秒级暂停时间。

为验证 Map 任务的 GC 预测模型的精度,分别使用 200、300 和 400 MB 的 JVM 堆大小进行测试。测试结果如表 1 所示。其中,*Eden* 表示 Eden 区域的大小,单位是 kB;Minor GC 平均暂停时间的实验值用 *Expr* 表示,单位 μs;预测值用 *Prediction* 表示,两者的差值用 *Err* 表示。

表 1 Map 任务 Minor GC 暂停时间预测模型精度

Tab. 1 Prediction precision of Minor GC for a Map task

Eden/kB	*Expr*/μs	*Prediction*	*Err*
16 384	1 428	1 524	-96
24 576	1 919	1 827	92
32 768	2 076	2 130	-54

3.2 对象晋升阈值分析

将算法 3 应用到默认设置的 Reduce 任务的 GC 指标后,得到聚类的统计信息如表 2 所示。其中,*Cluster* 表示簇号,*mean*、*min*、*max* 分别表示 Minor GC 发生时 Eden 空闲空间(kB)的均值、最小值和最大值。从表 2 统计数据易知簇 1 为“大对象”申请空间时 Eden 的剩余空间数据,因此使用聚类 1 的最小值 5 071 作为晋升对象大小的阈值。

表 2 默认设置下 Eden 空闲空间使用 K-means 聚类后的统计信息

Tab. 2 Statistics of Eden free space for both clusters

Cluster	*mean*	*min*	*max*
1	9 446	5071	14 606
2	637	0	4 977

通过 Reduce 任务的计数器值得知 Reduce 任务拷贝的数据为 2 572 068 kB。将该任务的 JVM 的对象晋升阈值设置为 5 MB 后,直接分配在年长代的对象大小为 2 506 207 kB(默认设置下为 0 kB),由此可知对象晋升阈值预测准确。

将对象晋升阈值设置为 5 MB 后,Minor GC 性能得到极大地提升,如图 5 所示。图 5 中 4 个子图分别表示 Reduce 任务 Minor GC 的总暂停时间(ms)、发生次数、均值(ms)和中位数(ms)。黑色表示未设置对象晋升阈值的 Reduce 任务,灰色表示将对象晋升阈值设置为预测的 5MB 的 Reduce 任务。

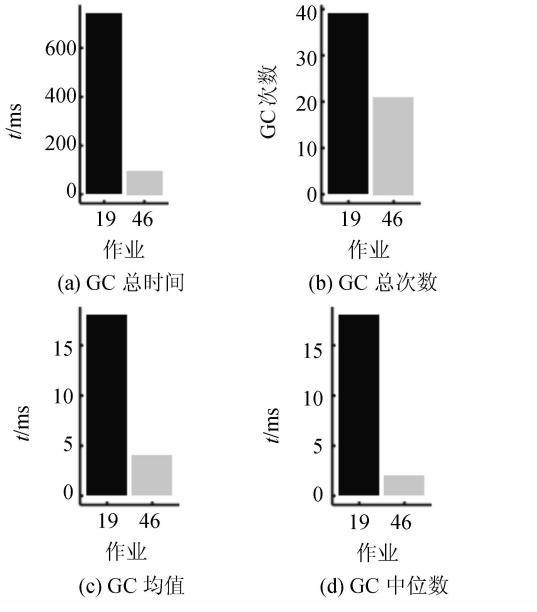


图 5 对象晋升阈值设置前后 Minor GC 性能对比

Fig. 5 Comparison of Minor GC performance between setting JVM's PretenureSizeThreshold and no setting

3.3　作业内存预测性能评估

使用已获得的 Reduce 任务的 Minor GC、Major GC 的回归模型和 Map 任务的 Minor GC 回归模型,使用过程 1、算法 1 和算法 2 分别预测 Reduce 任务和 Map 任务的年轻代、年长代的大小。默认配置与预测值如表 3 所示,相应的 GC 性能对比如表 4 所示。表 3 中,Y、O 分别表示年轻代和年长代大小。表 4 中,MD、MP、RD、RP 分别表示 Map 任务默认设置、Map 任务推荐设置、Reduce 任务默认设置和 Reduce 任务推荐设置。表 3 和 4 数据表明提出的方法能:

1)发现年长代分配紧张的情况,并增加到合理的大小;

2)能精细调整年轻代的大小;

3)发现年长代内存分配过剩的情况,并将其减小到合理值(减少后保持 Major GC 的次数为 0);

4)能确保在推荐的内存配置下垃圾回收的吞吐率满足用户设定值(文中设置为 1%)。

表 3　预测配置与默认配置对比

Tab. 3　Generaton arrangement for the default and recommended job

作业配置	Reduce 任务		Map 任务	
	Y/MB	O/MB	Y/MB	O/MB
默认	66	133	66	133
预测	23	267	67	115

表 4　预测配置与默认配置任务 GC 性能对比

Tab. 4　GC performance of the default and recommended job

任务	Minor GC		Major GC	
	次数	时间/μs	次数	时间/μs
MD	26	71 190	—	—
MP	26	76 354	—	—
RD	39	740 337	40	2 461 356
RP	61	138 530	18	1 526 868

使用预测的内存大小使 Reduce 任务的 GC 代价从默认配置下的 1.2% 下降到 0.7%,Map 任务的 GC 代价保持在 0.4%。

4　总　结

调优 MapReduce 作业资源的常见方法是使用机器学习算法,此种方法存在 2 个方面的限制:1)在 Hadoop 生产集群中不易开展大量的测试来获得机器学习所需的训练数据;2)机器学习的性能受训练参数范围限制。

作者以生产集群容易获得的作业历史记录和 JVM 垃圾回收日志为分析对象,在深入分析 JDK1.7 的串行垃圾回收器的工作原理基础之上提出详细的 GC 指标;提出对象分配合理性的评估指标,为内存评估与调优提供方向性指导;构建 Minor GC 和 Major GC 回归模型,实现特定内存配置下的 GC 时间预测;提出年长代和年轻代的预测算法;使用 K-means 算法预测晋升对象阈值。实验结果表明,提出的调优方法能在分配较少的内存条件下减少 GC 数量,提升 GC 吞吐率。作者基于作业的历史运行数据进行分析,易实现迭代优化,能为 MapReduce 生产集群提供内存自动调优的功能。

参考文献:

[1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107 – 113.

[2] Zikopoulos P, Eaton C. Understanding big data: Analytics for enterprise class Hadoop and streaming data[EB/OL]. (2011 – 10)[2015 – 06 – 10]. http://public.dhe.ibm.com/common/ssi/ecm/im/en/iml14296usen/IML14296USEN.PDF.

[3] Olston C, Reed B, Srivastava U, et al. Pig latin: A not-so-foreign language for data processing[C]//Proceedings of the 2008 ACM SIGMOD international conference on Management of data. Vancouver, Canada: ACM, 2008: 1099 – 1110.

[4] Thusoo A, Sarma J S, Jain N, et al. Hive-a petabyte scale data warehouse using hadoop[C]//2010 IEEE 26th International Conference on Data Engineering (ICDE). Long Beach, California, USA: IEEE, 2010: 996 – 1005.

[5] Herodotou H, Babu S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs[J]. Proceedings of the VLDB Endowment, 2011, 4(11): 1111 – 1122.

[6] Yang H L, Luan Z Z, Li W J, et al. MapReduce workload modeling with statistical approach[J]. Journal of Grid Computing, 2012, 10(2): 279 – 310.

[7] Zhou Shilong, Chen Xingshu, Luo Yonggang. Hadoop mapreduce job performance analysis and prediction based on grey-box model[J]. Journal of Sichuan University: Engineering Science Edition, 2014, 46(Supp 1): 146 – 154. [周世龙,陈兴蜀,罗永刚. 基于灰盒模型的 Hadoop MapReduce job 参数性能分析与预测[J]. 四川大学学报:工程

科学版,2014,46(Supp 1):146 – 154.]

[8] Kadirvel S, Fortes J A B. Grey-box approach for performance prediction in map-reduce based platforms[C]//21st International Conference on Computer Communications and Networks (ICCCN), 2012. Washington DC, USA: IEEE Communication Society, 2012.

[9] Xu L, Liu J, Wei J. FMEM: A fine-grained memory estimator for MapReduce jobs[C]//ICAC, 10th International Conference on Autonomic Computing. San Jose, California: USA USENIX in Cooperation with ACM SIGARCH, 2013: 65 – 68.

[10] Singer J, Kovoor G, Brown G, et al. Garbage collection auto-tuning for java mapreduce on multi-cores[J]. ACM SIGPLAN Notices, 2011, 46(11): 109 – 118.

[11] Kejariwal A. A tool for practical garbage collection analysis in the cloud[C]//2013 IEEE International Conference on Cloud Engineering (IC2E). San Francisco, California, USA: IEEE, 2013: 46 – 53.

[12] Angelopoulos V, Parsons T, Murphy J, et al. GcLite: An expert tool for analyzing garbage collection behavior [C]//Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual. Swissotel Grand EfesIzmir, Turkey: IEEE, 2012: 493 – 502.

[13] White T. Hadoop: The definitive quide[M]. 3rd ed. California, USA: O'Reilly, 2012.

[14] Sun Microsystems. Memory Management in the Java HotSpot virtual machine[EB/OL]. [2014 – 08 – 28]. http://www. oracle. com/technetwork/java/javase/memorymanagement-whitepaper-150215. pdf.

[15] Hunt C, Binu J. Java Performance[M]. Upper saddle River, NJ, USA: Prentice Hall Press, 2011.

[16] Jones R, Hosking A, Moss E. The garbage collection handbook: The art of automatic memory management[M]. New York, USA: Chapman & Hall/CRC, 2011.

[17] Alka Gupta. GC Portal[EB/OL]. [2014 – 10 – 09]. http://www. oracle. com/technetwork/articles/java/gcportal – 136937. html.

[18] Ahmad F, Lee S, Thottethodi M, et al. PUMA: Purdue MapReduce benchmarks suite[EB/OL]. [2013 – 09 – 26]. http://web. ics. purdue. edu/ ~ fahmad/benchmarks. htm.

[19] Hartigan J A, Wong M A. Algorithm AS 136: A k-means clustering algorithm[J]. Applied Statistics, 1979, 28: 100 – 108.

[20] Robert I, kabacoff. R 语言实战[M]. 高涛, 译. 北京: 人民邮电出版社, 2013: 162 – 164.

[21] Faraway J J. Linear models with R[M]. New York: Chapman & Hall/CRC, 2004.

(编辑 张 琼)