一种Mapreduce作业内存精确预测方法

罗永刚，陈兴蜀，杨 露

(四川大学网络空间安全研究院 成都 610065)

【摘要】针对准确预测mapreduce作业内存资源需求困难的问题，根据Java虚拟机(JVM)的分代(JVM将堆内存划分为年轻代和年长代)内存管理特点，该文提出一种分代内存预测方法。建立年轻代大小与垃圾回收时间的模型，将寻找合理年轻代大小的问题转换为一个受约束的非线性优化问题，并设计搜索算法求解该优化问题。建立mapreduce作业的map任务和reduce任务性能与内存的关系模型，求解最佳性能的内存需求，从而获得map任务和reduce任务的年长代内存大小。实验结果表明，本文提出的方法能准确预测作业的内存需求；与默认配置相比，能提供平均6倍的性能提升。

关 键 词 垃圾回收； Java虚拟机； mapreduce； 资源管理

中图分类号 TP393.08 **文献标志码** A doi:10.3969/j.issn.1001-0548.2016.06.019

An Innovative Memory Prediction Approach for Mapreduce Job

LUO Yong-gang, CHEN Xing-shu, and YANG Lu

(Cybersecurity Research Institutes, Sichuan University Chengdu 610065)

Abstract It is difficult to predict the amount of memory for a mapreduce job. Based on the fact that Java virtual machine (JVM) divides the heap space managed by the JVM garbage collector into young and old generations, a generational memory prediction method is put forward. We build up a function that models the relationship between the amount of young generation and the total garbage collection time, and then we use a constrained nonlinear optimization model to find the rational footprint of young generation. The memory model for the map phase is established, the phase of a mapreduce job is reduced, then a relationship between map/reduce tasks' performance (runtime of a task) and the amount of memory of the old generation is set up, and finally, the reasonable old generation memory size is obtained. The experimental results show that the proposed approach can accurately predict the memory size of map and reduce the tasks of a mapreduce job. In comparison with the default configuration, the proposed approach can give us 6 times performance improvement than default settings.

Key words garbage collection; JVM; mapreduce; resource management

Apache Hadoop[1](包含HDFS和mapreduce两个核心组件，本文只考虑mapreduce，因此在没有特殊说明情况下Hadoop和mapreduce等价使用)是谷歌公司提出的mapreduce编程模式的开源版本，得到了学术界、工业界的大力支持。随着mapreduce应用规模不断扩大，为Hadoop集群分配合理的资源以及优化mapreduce应用的运行性能一直是研究热点[2]。Hadoop官方路线图规划在Hadoop3.0版本中增加了内存自动预测机制[3]。

已有大量研究工作关注mapreduce性能与配置参数的关系。这些研究的核心思想是通过构建性能模型获得特定作业在某个配置下的运行时间，然后使用优化算法获得最优的配置参数。文献[4]使用基于代价的方式构建性能模型[5]，使用递归随机搜索算法找到优化的配置参数。文献[6]的性能模型与文献[4]类似，但使用的是遗传算法寻找最优配置参数。文献[7]未构建性能模型，通过修改hadoop框架在线获得任务运行时间，使用拉丁超立方体抽样方法进行配置参数采样，使用智能爬山算法搜索最优配置参数。这些研究存在以下不足：1) 构建性能模型增加了不必要的复杂度；2) 搜索算法需要限定内存资源配置参数的搜索空间，增加用户使用难度。

Hadoop应用程序运行在Java虚拟机之上，JVM的配置对Hadoop应用的性能和稳定性有直接影响。文献[8]通过定期采样作业的内存使用量来预测内存的使用，没有说明预测的内存配置是否提升作业性能。文献[9]通过随机森林算法优化JVM配置，但给出的优化配置的年轻代与年长代比例仅有1:2与1:8

收稿日期：2015－11－03；修回日期：2016－07－07
基金项目：国家科技支撑计划(2012BAH18B05)
作者简介：罗永刚(1980－)，男，博士，主要从事大数据安全方面的研究.

这两种配置。文献[10-11]没有考虑mapreduce作业的特殊性，也没有给出JVM内存大小的建议。

为解决这些问题，根据JVM分代内存管理原理，结合年轻代和年长代对mapreduce作业性能影响不同，提出分代内存预测方法，具体包括：

1) 使用回归模型建模年轻代垃圾回收平均暂停时间，将寻找合理的年轻代大小问题转换为非线性优化问题，并设计求解算法；

2) 分析map任务和reduce任务的性能与内存配置的关系，建立内存模型，通过内存模型求解年长代内存需求。不需要构建性能模型和使用优化算法，也不需要用户指定内存范围。

1　分代内存大小预测方法

1.1　分代内存预测

Java虚拟机使用分代内存管理[12]，将管理的堆内存划分为年轻代(young)、年长代(old)两个区域。按照一定的比例(如默认为8:1)，JVM将年轻代进一步划分为eden、from和to这3个区域。

JVM首先尝试将Java对象分配在eden区域，仅当Java对象的大小超过某个阈值时，JVM才直接从年长代分配该对象所需内存。当eden或年长代内存空间不够时，分别触发minor GC或major GC，即分别回收年轻代或年长代中不再被使用的对象。执行垃圾回收期间，JVM暂停Java应用程序的运行。通常minor GC暂停时间较短，major GC暂停时间较长。

一个mapreduce作业由多个map任务和reduce任务组成。每个任务可以看成是一个Java应用。map或reduce任务分配的对象可以分为以下两类。

1) 临时对象：具有单个对象占用空间小、有效时间短(即有效期通常不会超过连续两次minor GC的时间)、对象数量庞大等特征。

2) 任务缓存对象：map任务或reduce任务运行期间用于缓存数据的对象通常很大，且几乎在整个任务生命周期内均有效。

第1类对象分配在eden，且minor GC后被回收，因此eden内存空间可以被重用。第2类对象可以进一步分为mapreduce框架缓存对象(S_{mr})和用户定义的缓存对象(S_u)。S_{mr}的1个实例是mapreduce框架用于缓存map函数输出记录的对象。用户缓存对象表示用户在自定义的map类或reduce类中定义的数据缓存对象。例如使用mapreduce实现K-means聚类算法时，reduce函数需要缓存属于同一簇的原始数据，以便从中选择新的聚类中心。S_u与用户设计的map函数和reduce函数直接相关，不具有通用性，且大量mapreduce作业的map任务和reduce任务的S_u较小，因此不予考虑。

因此，本文使用以下2种思路分别预测年轻代与年长代的大小。

1) 年轻代：根据map任务或reduce任务的运行时间确定期望的minor GC总时间，然后寻找满足该时间要求的eden最小值；

2) 年长代：年长代主要分配mapreduce框架的缓存对象，缓存空间的大小会影响map任务与reduce任务的运行性能，因此需要根据map任务与reduce任务的数据流和工作流来计算年长代的大小。

1.1.1　预测年轻代内存大小

假设任务的运行时间为T_{task}，设置任务minor GC总暂停时间占任务运行时间的比值为p，则希望的minor GC总时间$T_{gc}=pT_{task}$。假设eden大小为e，当任务处理程序不变，且处理数据相当的情况下，任务运行过程中第1类对象的总大小s基本保持不变，则寻找合理的eden大小问题可以转换为满足以下约束条件的优化问题。

最小化目标函数$f(e)=e$，约束条件为：

$$\begin{cases} g(e,s)\leqslant T_{gc} \\ l\leqslant e\leqslant u \\ l=\min(e_{train}) \\ u=\max(e_{train}) \end{cases} \tag{1}$$

式中，e_{train}表示minor GC回归模型训练数据集中eden的取值；$g(e,s)$表示在已知第1类对象的总大小s和给定的eden条件下minor GC的总时间。约束条件的核心是minor GC时间小于等于计算的期望时间T_{gc}。

通过实验发现，在其他条件不变的情况下，增加eden将减少minor GC总时间，减少eden会增加minor GC总时间。基于上述发现，设计算法1来求解上述优化问题。算法1的主要思路为：随机选择eden大小，计算对应的GC时间，大于期望值时增加eden，小于期望值时减少eden，直到找到接近期望时间所对应的eden或超过设定的最大值或最小值为止。算法1中l和u与式(1)含义相同，m_0表示每次增加或减少内存的大小。

算法1：eden搜索算法

输入：$g(e,s)$，l，u，m_0，T_{gc}

输出：预测eden的大小

1) 随机选择x_1，使得$l\leqslant x_1\leqslant u$，将$x_1$设置为

起始搜索点

2) 求 $T_1 = g(x_1, s)$，如果 $T_1 < T_{\text{th}}$，$\text{step} = -m_0$，执行步骤 3)；如果 $T_1 > T_{\text{th}}$，$\text{step} = m_0$，执行步骤 4)；否则返回 x_1

3) $x_2 = x_1 + \text{step}$，求 $T_2 = g(x_2, s)$，如果 $T_2 < T_{\text{th}}$，依次检查 $x_3, x_4, \cdots$，直到 x_i 使得 $T_i \geqslant T_{\text{th}}$，返回 x_{i-1}；如果 $T_2 \geqslant T_{\text{th}}$，返回 x_1

4) $x_2 = x_1 + \text{step}$，求 $T_2 = g(x_2, s)$，如果 $T_2 > T_{\text{th}}$，依次检查 $x_3, x_4, \cdots$，直到 x_i 使得 $T_i \leqslant T_{\text{th}}$，返回 x_i；如果 $T_2 \leqslant T_{\text{th}}$，返回 x_2。

1.1.2 预测年长代内存大小

mapreduce提供了多个配置参数分别设置map任务和reduce任务的缓存大小。这些配置参数设置不合理时会导致map任务和reduce任务发生多次溢写操作，直接影响任务性能和作业性能。

map任务或reduce任务缓存对象用 S_{mr} 表示，在整个任务运行期间有效，因此这些对象将驻留年长代，通过计算 S_{mr} 的大小即可获得年长代的大小。

对map任务，S_{mr} 缓存map函数输出记录；对reduce任务，S_{mr} 缓存shuffle数据。单个map任务或reduce任务的 S_{mr} 大小需要结合任务的数据流和工作流确定。

同一个mapreduce作业的所有map任务和reduce使用相同的配置。因此需要根据同一作业下不同map任务和reduce任务的期望大小计算合理的统一值。

1) map任务的 S_{mr}

map函数输出记录序列化后保存在 S_{rd} 中，该记录的分区号及在 S_{rd} 中的地址索引保存在 S_{idx} 中，因此 S_{mr} 为 S_{rd} 与 S_{idx} 之和。第i个map任务的记录索引缓存 $S_{\text{idx}}^{(i)}$ 和记录数据缓存 $S_{\text{rd}}^{(i)}$ 通过下文计算方法获得。$S_{\text{idx}}^{(i)}$ 和 $S_{\text{rd}}^{(i)}$ 能确保第i个map不发生溢写操作，因此分别选择 $S_{\text{idx}}^{(i)}$ 和 $S_{\text{rd}}^{(i)}$ 的最大值 S_{idx} 和 S_{rd} 作为统一配置，可以确保每个任务均不发生溢写。定义 P_{idx} 表示map任务的索引数据所占百分比，则有：

$$\begin{cases} S_{\text{idx}} = \max(S_{\text{idx}}^{(i)}) \\ S_{\text{rd}} = \max(S_{\text{rd}}^{(i)}) \\ P_{\text{idx}} = \lceil 100 S_{\text{idx}} / (S_{\text{idx}} + S_{\text{rd}}) \rceil / 100 \end{cases} \tag{2}$$

此时必然有 $S_{\text{idx}} \geqslant S_{\text{idx}}^{(i)}, S_{\text{rd}} \geqslant S_{\text{rd}}^{(i)}$，即每个 map 任务均不会执行溢写操作。为了避免发生内存溢出错误，当 S_{idx} 和 S_{rd} 的占用比例超过 β 时触发溢写操作，为了避免溢写发生，实际分配的缓存大小为计算所得缓存大小 S'_{mr} 除以 β。为了避免内存资源浪费，将 β 设置为接近于1的值，设置为0.99，因此有：

$$\begin{aligned} S'_{\text{mr}} &= \max \begin{cases} \lceil S_{\text{idx}} / P_{\text{idx}} \rceil \\ \lceil S_{\text{rd}} / (1 - P_{\text{idx}}) \rceil \end{cases} \\ S_{\text{mr}} &= S'_{\text{mr}} / \beta \end{aligned} \tag{3}$$

2) reduce任务的 S_{mr}

reduce任务将拷贝的数据缓存到 S_{R} 中，只要 S_{R} 大于每个任务的实际缓存数据大小，reduce任务就不会发生溢写，从而获得最佳运行性能。设 $S_{\text{R}}^{(j)}$ 表示第j个reduce任务的内存需求，为了确保每个reduce任务不发生溢写操作，S_{R} 必须取 $S_{\text{R}}^{(j)}$ 的最大值，即：

$$S_{\text{R}} = \max(S_{\text{R}}^{(j)}), \quad j \in [1, \text{m}] \tag{4}$$

2 求解 $g(e, \text{s})$

设每次minor GC平均暂停时间为 t_{gc}，通过实验数据分析发现 t_{gc} 是e的函数，即：

$$t_{gc} = \psi(e) \tag{5}$$

ψ 可以通过回归建模获得。当已知年轻代对象的总大小s和eden大小e时，可计算出需要执行的minor GC的次数 N_{gc}，即：

$$N_{\text{gc}} = \lceil s / e \rceil \tag{6}$$

则minor GC的总时间估计值 $g(e, \text{s})$ 为：

$$g(e, s) = N_{\text{gc}} t_{\text{gc}} \tag{7}$$

式(7)为在年轻代对象总需求不变、给定eden大小的条件下，minor GC的总暂停时间。

将式(5)、式(6)带入式(7)，得：

$$g(e, s) = \lceil s / e \rceil \psi(e) \tag{8}$$

s可以通过对minor GC日志信息进行统计分析获得，即：

$$s = \sum_{i=1}^{n} O_i \tag{9}$$

式中，O_i 表示第i-1次minor GC与第i次minor GC期间分配在年轻代对象的大小；n表示minor GC发生的次数。使用式(8)可以计算任务在某个eden大小下的GC暂停时间。

3 作业性能分析与优化

3.1 map任务性能分析与优化

map任务的数据流和工作流如图1所示，其中灰色方框表示操作，白色方框表示文件。当缓存(即 S_{mr})设置过小时，map任务执行过程会发生多次不必要的溢写操作，形成多个中间文件。当map处理完所有输入数据时，将缓存中的记录以及中间文件中的记录进行归并排序，形成单个输出文件。每条记录

发生1次溢写操作将增加1次本地文件读写操作，增加归并排序的时间，因此优化目标通过计算合理配置参数达到不执行图1中虚线框部分的操作的目的。

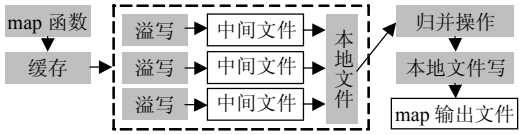


图1　map任务详细工作流程

map任务优化的实现思路是根据map任务的计数器值计算该任务所需的 $S_{\mathrm{idx}}^{(i)}$ 和 $S_{\mathrm{rd}}^{(i)}$ 大小，并分配相应的缓存空间，因此本文优化map任务时不需要构建性能模型。

每条map输出记录需要占用16 byte的索引空间，令 $N_{\mathrm{rd}}^{(i)}$ 表示第i个map任务输出的记录数，则有：

$$S_{\mathrm{idx}}^{(i)}=16N_{\mathrm{rd}}^{(i)} \tag{10}$$

$S_{\mathrm{rd}}^{(i)}$ 可以直接从map任务的计数器中获得。

例如某个作业的第i个map任务的输出记录数为10 485 760，输出记录数据为200 MB，则该map任务的 $S_{\mathrm{idx}}^{(i)}=10\,485\,760\times16=160$ MB，$S_{\mathrm{rd}}^{(i)}=200$ MB，则 $P_{\mathrm{idx}}^{(i)}=45\%$。

3.2　reduce任务性能分析与优化

reduce阶段包含3个子阶段，即shuffle阶段、排序阶段和执行reduce函数阶段，如图2所示。shuffle阶段主要功能是从map端拷贝数据并保存到缓存 $S_{\mathrm{R}}^{(i)}$ 中，当 $S_{\mathrm{R}}^{(i)}$ 的使用比例超过 $P_{\mathrm{sf}}P_{\mathrm{sp}}$ 后将数据溢写到中间文件，当中间文件数量超过阈值后执行归并操作，形成单个更大的中间文件。shuffle阶段完成后进入排序阶段，主要功能是使用归并排序算法将中间文件及保留在 $S_{\mathrm{R}}^{(i)}$ 中的数据进行归并排序。在执行归并排序之前，mapreduce框架首先将超过JVM最大堆与 P_{red} 乘积的数据溢写到中间文件。排序阶段完成后进入执行reduce函数的阶段，此阶段主要将记录交由reduce函数处理，将结果写入HDFS文件。本文主要对shuffle阶段和排序阶段进行优化。

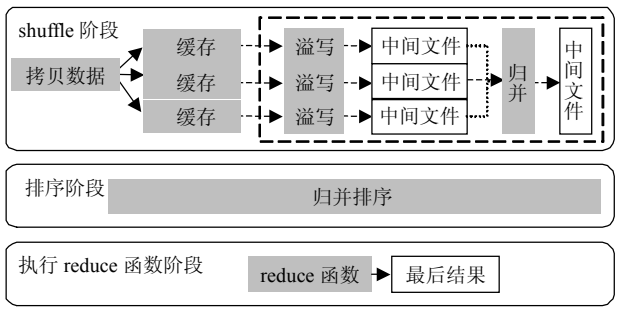


图2　reduce任务详细工作流程

本文采用以下两项措施优化reduce的数据流和工作流：1) 修改Hadoop框架，取消原有框架中reduce任务缓存使用量不能超过2 GB的限制，为实现reduce阶段零溢写提供架构支持；2) 通过reduce任务历史数据计算最优的内存配置，获得零溢写的最优执行流程。优化后的reduce任务工作流程减少了图2中的虚线框内的操作。

每个reduce任务的优化配置计算过程如下：令 $S_{\mathrm{rd}}^{(i)}$ 表示第i个reduce任务拷贝的数据大小(通过历史数据获得)，P_{sf} 为shuffle操作能使用内存的百分比，P_{sp} 为发生溢写的阈值，为获得最优的执行流程，则必须有：

$$S_{\mathrm{R}}^{(i)}>\frac{S_{\mathrm{red}}^{(i)}}{P_{\mathrm{sf}}P_{\mathrm{sp}}} \tag{11}$$

即保证第i个reduce任务实际拷贝的数据小于 $S_{\mathrm{R}}^{(i)}$，以确保该任务不执行溢写操作，从而获得最优的运行性能。为了避免内存资源浪费，取 $P_{\mathrm{sf}}=P_{\mathrm{sp}}=0.99$。为了避免将已缓存的数据在进入排序阶段之前被溢写到中间文件，设置 P_{red} 为1.0。

4　实验与分析

4.1　实验环境

Hadoop测试集群有8个从节点，每个从节点分配5 GB内存，排他性使用一个E5-2609@2.4GHz的物理核。每个从节点分配2个map任务槽和1个reduce任务槽，Hadoop以apache发布版本1.2.1为基础，修改了reduce任务内存管理，修改日志保存机制。本文选择PUMA测试套件[14]的wordcount(WD)、invertedindex(ID)、TeraSort(TS)、adjlist(ADJ)这4个测试程序。wordcount、invertedindex测试程序的输入数据为多个文件组成的10 GB wikipedia数据集。adjlist输入数据集为从图数据集中随机选出10 GB数据。TeraSort输入数据使用mapreduce提供的程序自动生成，总共10 GB左右。

4.2　任务年轻代内存分析

使用loess[15]对WD、ID、TS、ADJ的map任务和reduce任务的minor GC进行回归建模。使用算法1预测的年轻代大小进行测试，统计了4种测试程序的所有map和reduce任务GC暂停时间占任务运行时间的百分比，如图3所示。可得知除WD和ADJ的reduce任务的GC暂停时间的比例大于1%以外，其他任务的GC暂停时间均小于1%。

为避免年轻代分配过大导致不必要的资源浪费，分析了WD的map任务和reduce任务运行时间平均值与年轻代的关系，如图4所示。

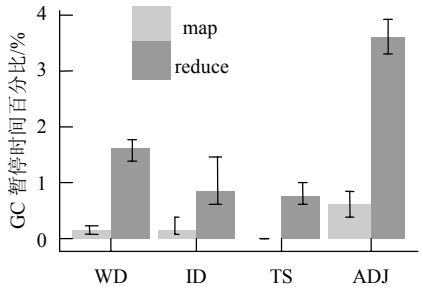


图3 年轻代大小预测精度

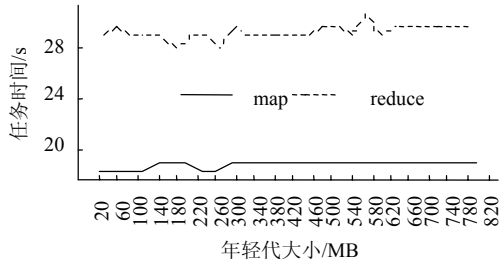
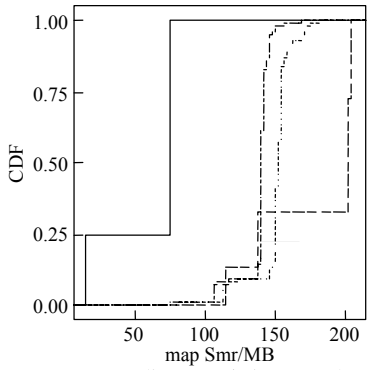


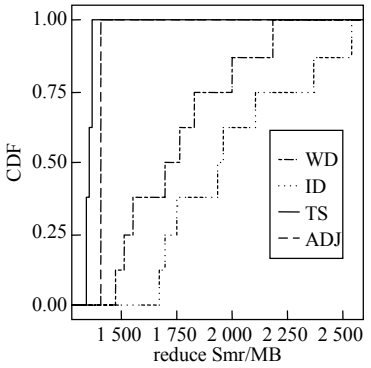
图4 wordcount map和reduce任务时间与年轻代的关系

从图4得知，年轻代为180 MB左右能获得较好的运行性能。ID、TS和ADJ也呈现相似的的特征。图4中获得的年轻代合理值可以和算法1所得值综合考虑，选择两者的最小值为最终的年轻代大小。

4.3 任务年长代内存分析



a.不同作业 map 任务 Smr 分布



b. 不同作业 reduce 任务 Smr 分布

图5 4种测试程序的map和reduce任务的 S_{mr} 分布

统计WD、ID、TS和ADJ的map任务和reduce任务的 S_{mr} 累计分布函数，其中map任务 S_{mr} 非常集中，ADJ和TS的reduce任务的 S_{mr} 也非常集中，WD和ID较分散，但差距也较小，如图5所示。因此即使采用最大内存需求也不会造成过多内存资源浪费。

pred(即本文提出的内存预测方法)和starfish的优化内存配置对比如表1所示。从表1得知pred给出的内存配置均比starfish小。为评价 S_{mr} 分配是否合理，分别统计了默认配置、starfish和pred这3种配置条件下发生溢写的任务数量，如表2所示，其中MSC、MNSC、RSC和RNSC分别表示发生溢写的map任务数、未发生溢写的map任务数、发生溢写的reduce任务数和未发生溢写的reduce任务数。未发生溢写是任务获得最佳运行性能的基本条件。从表1和表2得知，pred方法能在比starfish更少的内存配置条件下获得最佳的任务执行流程，即未发生溢写操作。

表1 starfish和pred推荐的内存配置

测试程序	starfish		pred	
	map/MB	reduce/MB	map/MB	reduce/MB
WD	500	4 000	350	2 260
ID	1 200	4 000	350	2 680
TS	500	4 000	250	1 500
ADJ	1 200	4 000	410	2 000

表2 默认配置、starfish推荐配置和pred推荐配置下4种测试程序溢写任务统计

测试程序	MSC	MNSC	RSC	RNSC
WD	162|0|0	0|162|162	1|8|0	0|0|8
ID	162|6|0	0|156|162	1|8|0	0|0|8
TS	150|0|0	50|200|200	1|1|0	0|31|8
ADJ	0|0|0	183|183|183	1|16|0	0|0|8

注：表格数据格式为：默认配置|starfish|pred。

4.4 作业性能分析

mapreduce配置参数调优以starfish最为全面，效果最佳。与基于经验的配置参数优化相比，starfish能获得更好的性能提升[4]。在starfish0.3.0版本的基础之上增加了adjlist和invertedindex测试程序的支持，在相同的集群资源下分别比较4种测试程序在默认配置、starfish优化配置和pred配置下的运行时间，结果如图6所示。starfish和pred给出的配置参数显著提升了作业性能，且pred性能在4种测试程序下均优于starfish。与默认配置相比，starfish在测试集群上平均加速比为3.5，最大加速比为6.1。与默认配置相比，pred的平均加速比为6.4，最大加速比为8。

starfish目前只支持Hadoop0.20版本，要求mapreduce必须使用新API编写。pred只需要作业的历史记录文件和任务JVM输出的垃圾回收日志，mapreduce的各个版本均支持通过简单配置获得这些数据，因此从部署难度及适应Hadoop版本更新两

个方面讲，pred比starfish更易于使用。

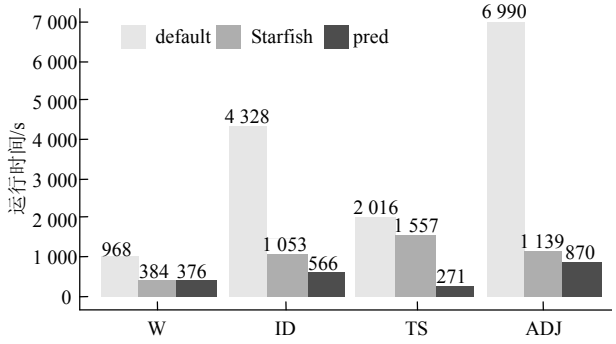


图6　4种测试程序在不同配置下的运行时间对比

文献[16-17]表明，在facebook、雅虎和淘宝的Hadoop生产集群中，绝大部分的mapreduce作业的输入数据只是GB级别，当前的Hadoop集群内存资源足够将这些待处理的数据全部缓存在内存中。对于这种数据规模的mapreduce作业，pred能提供很好的性能优化。

5　结 束 语

本文提出一种分代内存大小的预测方法。根据JVM的分代内存管理特点，分别使用不同的方法预测JVM不同分代内存区域的大小。本文提出的分代内存预测方法综合考虑了mapreduce层和JVM层的内存配置，是一种更为全面mapreduce作业内存配置参数优化方法。测试结果表明，与默认配置相比，本文给出的内存优化配置平均性能提升至6倍，最大达到8倍。与starfish相比，本文提出的方法在提供更好的性能前提下，使用更少的内存资源。

参 考 文 献

[1] DEAN J, GHEMAWAT S. Mapreduce: Simplified data processing on large clusters[C]//Proceedings of the 6th Conference on Eperating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2004, 6: 137-150.

[2] POLATO I, RÉ R, GOLDMAN A, et al. A comprehensive view of Hadoop research—a systematic literature review[J]. Journal of Network and Computer Applications, 2014, 46: 1-25.

[3] GERA S. Derive heap size or mapreduce.*.memory.mb. automatically[EB/OL]. [2014-03-08]. https://issues.apache.org/jira/browse/MAPREDUCE-5785.

[4] 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.

[5] HERODOTOU H. Hadoop performance models[EB/OL]. [2014-12-04]. http://arxiv.org/pdf/1106.0940v1.pdf .

[6] LIU C, ZENG D, YAO H, et al. MR-COF: a genetic mapreduce configuration optimization framework[M]. [S.l.]: Springer International Publishing, 2015: 344-357.

[7] LI M, ZENG L, MENG S, et al. MRONLINE: Mapreduce online performance tuning[C]//Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing. Vancouver, Canada: ACM, 2014: 165-176.

[8] XU L, LIU J, WEI J. FMEM: a fine-grained memory estimator for mapreduce jobs[C]//Proceedings of the 10th International Conference on Autonomic Computing. California, USA: USENIX in Cooperation with ACM SIGARCH, 2013: 65-68.

[9] 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.

[10] KEJARIWAL A. A tool for practical garbage collection analysis in the cloud[C]//2013 IEEE International Conference on Cloud Engineering (IC2E). Boston, USA: IEEE, 2013: 46-53.

[11] ANGELOPOULOS V, PARSONS T, MURPHY J, et al. GcLite: an expert tool for analyzing garbage collection behavior[C]//2012 36th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW). Lzmir, Turkey: IEEE, 2012: 493-502.

[12] SUN M. Memory management in the Java hotspot virtual machine[EB/OL]. [2014-08-28]. http://www. oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf.

[13] RAO S S. Engineering optimization: Theory and practice[M]. New Jersey, USA: John Wiley & Sons, 2009.

[14] FARAZ A, SEYONG L, MITHUNA T，et al. PUMA: Purdue mapreduce benchmarks suite[EB/OL]. [2013-09-26]. http://web.ics.purdue.edu/~fahmad/benchmarks. htm.

[15] CLEVELAND W S, DEVLIN S J. Locally weighted regression: an approach to regression analysis by local fitting[J]. Journal of the American Statistical Association, 1988, 83(403): 596-610.

[16] REN Z, XU X, WAN J, et al. Workload characterization on a production hadoop cluster: A case study on taobao[C]// 2012 IEEE International Symposium on Workload Characterization (IISWC). California, USA: IEEE Computer Society, 2012: 3-13.

[17] CHEN Y, GANAPATHI A, GRIFFITH R, et al. The case for evaluating mapreduce performance using workload suites[C]//2011 19th IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS). Singapore: IEEE, 2011: 390-399

编　辑　叶　芳