异构环境下自适应 reduce 任务调度算法的研究*

付彦卓，张树东，李 辉
（首都师范大学 信息工程学院，北京 100048）

摘 要：异构环境下的 Hadoop 平台对 reduce 任务的调度存在随机性，在分配任务时既没有考虑数据本地性，也没有考虑计算节点对当前任务的计算能力。针对以上问题，提出一种异构环境下自适应 reduce 任务调度算法（SARS）。算法根据 reduce 任务的输入数据分布选择所含数据量最大的机架，在选择计算节点的过程中，结合节点所含任务的数据量、节点的计算能力和当前节点的忙碌状态来选出任务的执行节点。实验结果表明，SARS 算法减少了 reduce 任务执行时的网络开销，同时也减少了 reduce 任务的执行时间。

关键词：Hadoop；reduce；异构环境；数据本地性；计算能力
中图分类号：TP301 **文献标志码**：A **文章编号**：1001-3695(2018)07-1989-03
doi:10.3969/j.issn.1001-3695.2018.07.015

Research on self-adaptive reduce task scheduling algorithm in heterogeneous environment

Fu Yanzhuo, Zhang Shudong, Li Hui
(*College of Information Engineering*, *Capital Normal University*, *Beijing* 100048, *China*)

Abstract: The scheduling of reduce tasks in Hadoop platform under heterogeneous environment is random, which neither consider the data locality nor the computing power of the computing node when assigning tasks. To solve the above problems, this paper proposed a self-adaptive reduce task scheduling algorithm (SARS), which selected the rack with the largest amount of data according to the input data distribution of the reduce task. In the process of selecting the computing node, selecting the execution node of the task bases on the amount of data contained in the node, the computing power of the node and the busy state of the current node. The experimental results show that the SARS algorithm reduces the network overhead and the execution time of the reduce task.

Key words: Hadoop; reduce; heterogeneous; data locality; computing power

0 引言

随着互联网的快速发展，数据的规模得到了快速增长，海量数据的存储与处理显得尤为重要，进而诞生的 Hadoop[1] 得到了快速的发展与应用。Hadoop 是一种能够对海量数据进行分布式处理的软件框架，其主要包含了分布式文件系统（HDFS）和分布式计算模型（MapReduce）两个部分。HDFS 解决了 Hadoop 进行分布式文件存储等问题，而 MapReduce 编程模型是对 HDFS 上的数据进行分布式计算。在 MapReduce 进行分布式计算的过程中，任务调度算法的选择往往对整个作业的执行效率、网络拥塞的控制以及计算机资源的利用率等方面产生很大的影响。

任务调度是分布式计算的核心之一。一个好的任务调度算法在作业的执行效率、网络带宽和资源利用率上都会有很大的改变。MapReduce 在处理作业时，将作业分为 map 阶段和 reduce 阶段。Map 任务是处理数据，reduce 任务是处理 map 的输出结果。Map 和 reduce 任务的调度会对整个作业产生至关重要的影响。因此，设计一个好的 MapReduce 的任务调度算法越来越重要。

Hadoop 默认调度算法 FIFO（first in first out）简单明了，却存在一定的局限性。为此，Yahoo 和 Facebook 先后提出计算能力调度器（CS）和公平份额调度器（FS）。这两种调度器分别在计算资源分配上和作业处理公平性上得到一定的改善，但是无法保证良好的数据本地性[2]。为了更好地提高作业处理性能，研究人员针对不同作业提出了不同的任务调度算法。

针对 map 任务，众多研究人员提出了提高数据本地性的概念。例如，Seo 等人[3] 提出了 HPMR 算法，通过数据预取提高数据本地性；Sun 等人[4] 提出了 HPSO 算法，通过任务预测和数据预取提高数据本地性；Zaharia 等人[5] 通过延迟任务调度来提高数据本地性；还有研究人员以减少网络拥塞为目标[6]、以改善任务备份为目标[7]、以及根据节点计算能力为目标[8]，分别对 MapReduce 任务调度进行改进。

针对 reduce 任务调度的研究主要是从提高数据本地性与减少 reduce 阶段数据传输来考虑。Fujishima 等人[9] 提出动态放置文件策略来提高 reduce 阶段的 I/O 性能；Hammoud 等人[10] 提出位置感知的 reduce 调度算法，通过依次选择数据量最多的机架与节点来提高 reduce 任务的数据本地性；Arslan 等人[11] 提出 LoNARS 算法来减少 reduce 任务过程中的数据传输。以上方法虽然考虑了 reduce 数据本地性和数据传输，但是没有考虑节点对当前任务的计算能力。

在异构环境下，集群中各个节点配置差异，集群中会出现网络带宽和计算能力的差异。网络带宽会影响 reduce 任务的数据传输，计算能力的差异会影响 reduce 任务的执行效率。在分配任务时选择合适的节点，不仅能达到高效的资源利用率还能达到负载均衡的效果。

本文研究了异构环境下 reduce 任务的调度特点，指出了 reduce 任务选择计算节点上的缺陷，然后提出在选择 reduce 任务执行的节点过程中，考虑当前网络带宽、map 输出结果的网络传输量和节点的计算能力，从而达到一种综合高效的任务执行效果。

收稿日期：2017-03-15；**修回日期**：2017-04-18 **基金项目**：国家自然科学基金资助项目（31571563）；高可靠嵌入式系统技术北京市工程研究中心资助项目（2013BAH19F01）

作者简介：付彦卓（1992-），男，河南信阳人，硕士，主要研究方向为大数据、云计算（fuyanzhuo.love@163.com）；张树东（1969-），男，教授，博士，主要研究方向为大数据、云计算、传感器网络；李辉（1990-），男，硕士，主要研究方向为数据挖掘、机器学习.

本文从三个方面来决策 reduce 任务的执行节点:a)异构环境下的 reduce 任务调度策略,将异构环境下各个计算节点的计算能力作为考虑一方面;b) reduce 的输入数据,即各个 map 的输入结果;c)当前环境下忙碌的计算节点达到空闲状态所需的时间。

1 机架的选择

Reduce 任务是从各个节点的 map 任务读取一片数据,经过排序后,交给用户编写的 reduce 函数处理。Reduce 任务的 shuffle 阶段,需要从各个节点的 map 结果中取出属于该 reduce 所需的一部分,传输到任务的计算节点。然而 map 任务存在于不同机架上的不同节点,所以在数据传输过程中会占用大量的网络资源,如果不合理地分配 reduce 任务,可能会出现大量的跨节点以及跨机架的数据传输,从而导致网络拥塞[11]。

Hadoop 原生调度算法 FIFO(先进先出)在执行任务时,采用的是一种"拉"的机制。空闲的 TaskTracker 向 JobTracker(任务调度器)请求一个任务,JobTracker 调用 TaskScheduler 为该 TaskTracker 分配任务。Map 任务由于有数据本地性,其任务选择策略是按着 node-local、rack-local 和 off-switch 的优先级依次选择任务[12]。由于认为 reduce 任务不存在数据本地性,所以其调度策略非常简单。Reduce 任务在执行时是根据当前空闲节点的请求顺序而分配。为了便于说明,用图 1 简单表示 reduce 任务分配策略。

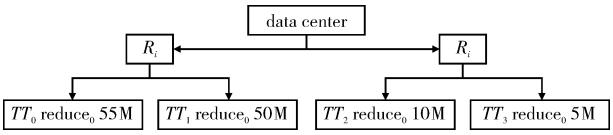

图 1 Reduce 任务所需数据分布示例

图 1 只是简明地列出了一个数据中心 data center,两个机架 R_0、R_1 和四个计算节点 $TT_i(i=0,1,2,3)$,以及一个 reduce$_0$ 任务。其中 TT_0 和 TT_1 属于机架 R_0,TT_2 和 TT_3 属于机架 R_1;reduce$_i$ 是指第 i 个 reduce 任务。假设当前有一个 reduce 任务待执行,该任务所需的数据在各个节点上分别为 TT_0:55M、TT_1:50M、TT_2:10M、TT_3:5M。此时空闲节点为 TT_3 和 TT_1,并且节点向 JobTracker 请求任务的顺序依次为 TT_3 和 TT_1。按照 FIFO 调度策略,JobTracker 调用 TaskScheduler 为 TT_3 分配 reduce$_0$ 任务。然而 TT_3 上只有 reduce$_0$ 所需的 5M 数据,需要跨机架传输 105M(TT_0、TT_1),需要跨节点传输 10M(TT_2)。在实际处理海量数据的过程中,跨机架传输和同机架跨节点传输的数据量远远多于图 1,这可能会使任务处于等待数据阶段而影响作业的执行时间,也可能会增加网络负担,甚至可能造成网络拥塞。

为了避免 reduce 阶段数据传输带来的网络拥塞等问题,本文在选择 reduce 任务的计算节点时,优先包含该 reduce 所需数据最多的机架。由于同一机架中网络带宽最高[10],所以在 reduce 任务远程获取数据时相同数据量的情况下,同一机架的数据传输效率远高于跨机架的数据传输。因此,在图 1 的示例中,空闲节点为 TT_3 和 TT_1,分别属于机架 R_0 和 R_1,按照选择所含数据量最多的机架策略,在此处优先选择 R_0,从而避免大量的数据跨机架传输。

2 计算节点的选择

计算节点的选择是从机架中选择最合适的计算节点来执行 reduce 任务。在获取已知的机架后,节点的选择主要考虑数据本地性和节点的计算能力两个方面。

2.1 数据本地性

数据本地性是指执行任务的节点包含其所需的数据,在本文中是 reduce 任务执行的节点包含该任务所需的部分数据。选择机架是选择包含数据量最多的机架。同理,在已选的机架中,如果一个节点所含一个任务的数据量比较多,则其任务的本地性比较高。如果 reduce 任务在该节点上执行,则需要传输的数据量相对较少。正如图 1 所示,各个节点所含 reduce 任务的数据分布情况如表 1 所示。

表 1 reduce$_0$ 在不同节点上执行所需传输的数据量

数据	TT_0	TT_1	TT_2	TT_3
本地数据	55M	50M	10M	5M
传输数据	65M	70M	110M	115M

Hadoop 的一个基本原则是:移动计算比移动数据更高效[13]。由表 1 可知,TT_0 与其他节点相比,所含该任务的数据最多且需要传输的数据最少,所以表 1 中会优先考虑 TT_0 作为 reduce$_0$ 的计算节点。

2.2 节点计算能力

节点的计算能力是指节点处理任务的效率。异构环境下任务所在节点的计算差异性会对整个作业产生直接的影响[14]。所以节点的计算能力也是任务执行时不可忽略的因素之一。

由于任务的差异性会导致计算性能的差异性,所以节点的计算能力是相对某一特定的任务来说明的。节点相对于某一任务可以分为快节点和慢节点[15]。节点处理任务的能力与其任务类型、硬件配置(CPU、内存等)与任务负载有一定的联系[16]。在本文中是指处理 reduce 任务的能力,以及在任务执行前获取节点的计算能力。本文用该节点历史任务的平均处理速率来确定其计算能力。

用 p_i 表示第 i 个节点的计算能力,data$_j$ 表示第 i 节点上已处理的第 j 个任务的数据,cost$_j$ 表示第 j 个任务的执行时间,则节点的计算能力为

$$p_i = \frac{1}{n}\sum_{j=1}^{n}\frac{\text{data}_j}{\text{cost}_j} \tag{1}$$

计算节点的理想选择有两点:a)该节点上的任务本地性最高,从而需要跨节点传输的数据量最少;b)该节点的计算能力最强,使得数据传输完毕以后,该节点能快速执行完任务。在实际应用中,节点的配置和 reduce 任务所需数据的分布式是不确定的,所以 reduce 任务的计算节点同时满足以上两个条件的概率很小。考虑不同时满足以上两点的情况,本文提出一种新的选择计算节点策略,以任务所需数据的传输时间和任务执行时间来决定是否作为计算节点。

用 TT_i 表示第 i 个计算节点,TT_i^{data} 表示 TT_i 节点上的 reduce 数据量,在 reduce 任务执行过程中,dataSize 表示 reduce 任务跨节点传输数据的大小,则 dataSize 为

$$\text{dataSize} = \sum_{i=0}^{n-1} TT_i^{\text{data}} \tag{2}$$

a)传输时间。用 tranTime 表示跨节点数据的传输时间,tranRate 表示网络的传输速率,则 reduce 任务所需数据的传输时间(只考虑同机架中的节点)为

$$\text{tranTime} = \frac{\text{dataSize}}{\text{tranRate}} \tag{3}$$

b)任务执行时间。任务执行时间就是 reduce 任务的全部输入数据需要处理的时间,用 execTime 表示,根据式(1),则

$$\text{execTime} = \frac{1}{p_j}\sum_{i=1}^{n} TT_i^{\text{data}} \tag{4}$$

用 W_j 表示其他节点向第 j 个节点传输数据所需的时间和所有数据在第 j 个节点上的计算时间的总和,根据式(3)(4)可得

$$W_j = \frac{1}{\text{tranRate}}\sum_{i=0}^{n-1} TT_i^{\text{data}} + \frac{1}{p_j}\sum_{i=0}^{n} TT_i^{\text{data}} \tag{5}$$

本文将 W_j 队列从小到大排序,并每隔一个心跳更新一次,取最小的节点,即 min(W_j) 作为对应 reduce 任务执行的节点。但是在实际过程中,所选中的时间最短节点可能处于忙碌状态,不能作为当前 reduce 任务的候选节点。Hammoud 等人[10]提出的 LARTS 算法是在发现节点处于忙碌状态时,拒绝执行该 reduce 任务,拒绝次数到达一个阈值后随机选择其他节点。该做法会使当前 reduce 任务处于等待状态,并且任务不确定

性导致阈值不确定性。本文在处理忙碌节点时放弃分配,直接从排序好的 W_j 队列选择次优节点,依此类推,从而避免了等待的时延。

3　实验及结果分析

实验环境是基于 Hadoop-1.2.1 构建的集群。该集群由五台 PC 中的九个虚拟机和三个交换机构成。其中与 slave 相连的交是百兆交换机,与 master 相连的是千兆交换机。实验集群的网络拓扑结构如图 2 所示。

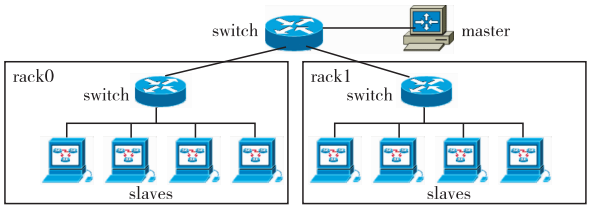

图 2　集群网络拓扑结构

本文研究的是异构环境下 reduce 任务的调度策略。为了体现异构环境的特点,表 2 为实验集群中节点的配置情况。其中 slave0 ~ slave3 属于一个机架,slave4 ~ slave7 属于另一个机架,master 节点与主交换机相连,九个节点的磁盘大小为 50 GB。

表 2　集群节点配置情况

节点名称	CPU	内存大小/GB	节点名称	CPU	内存大小/GB
master	Intel i7, 2 核心	2	slave4	Intel i7, 4 核心	4
slave0	Intel i7, 1 核心	1	slave5	Intel i3, 1 核心	1
slave1	Intel i7, 1 核心	1	slave6	Intel i3, 2 核心	2
slave2	Intel i7, 2 核心	2	slave7	Intel i3, 2 核心	1
slave3	Intel i7, 2 核心	2			

为了评估本文提出的异构环境下自适应 reduce 任务调度策略,本实验用修改后的 Terasort 和 WordCount 作为实验的基准测试。实验基准测试信息如表 3 所示。表 3 中的 sort1 和 sort2 是 Hadoop 自带的 Terasort 生成的数据,WordCount 是微博上爬取的数据,都为非规则数据。本文为了模拟真实场景,并没有运用 LARTS 算法中的方法(通过不同节点中的 mapper 产生的数据限制)来达到数据倾斜的效果。

表 3　实验基准测试信息

作业类型	数据大小/GB	map 数量	reduce 数量
sort1	1	30	5
sort2	5	76	10
WordCount	2.37	41	6

将表 3 中的作业使用原有 Hadoop 调度策略(FIFO)、LARTS 调度算法和本文的自适应调度算法(self-adaption)依次执行,其中的 map 和 reduce 数量不包括状态为 killed 和 failed 的数量,同时 EarlyShuffle 也为 Hadoop 默认配置。通过观察实验运行情况,得出如下结果。

3.1　reduce 平均执行时间(图 3)

图 3 中 reduce 的平均时间是从 shuffle 开始到 reduce 结束的平均执行时间。可以看出本文提出的 SARS 调度策略相比于 FIFO 调度算法,在 reduce 阶段的平均执行时间平均减少了 7.8% 左右;相比于 LARTS 调度算法,平均减少了 3% 左右,从而证明了本文提出的自适应 reduce 任务调度的可行性。

3.2　Shuffle 过程数据量

三种作业 shuffle 过程中分别产生的数据量如图 4 所示。该数据量是 shuffle 过程中跨节点数据量,而不是 Hadoop 系统原有计数器 reduce shuffle bytes 的统计(计数器 reduce shuffle bytes 中包含本地节点的数据)。可以发现,本文提出的 SARS 算法相比于 FIFO 算法,在 shuffle 过程中平均减少了 6.3% 左右。LARTS 算法在 reduce 调度过程中尽量选择所含数据量最多的节点作为计算节点,相比本文提出的 SARS 算法,LARTS 算法在 shuffle 过程中有一定的优越性。由图 4 可见,LARTS 算法比本文的 SARS 算法减少了 3.4% 左右。

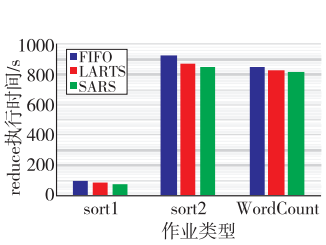

图 3　Reduce 平均执行时间

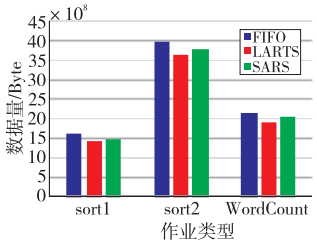

图 4　Shuffle 过程的数据总量

3.3　Shuffle 跨机架传输数据量(图 5)

Shuffle 阶段的跨机架传输,可能会导致网络拥塞。本文在此处只是研究 reduce 阶段,所以在此处统计的是 shuffle 阶段跨机架传输的数据对比。由图 5 可知,本文提出的 SARS 算法在 shuffle 过程中跨机架传输上减少了 9.8%;相比于 LARTS 算法,在 sort1 和 sort2 分别增加了 0.2% 和 0.4%,作业在 WordCount 中,本文提出的 SARS 算法比 LARTS 算法减少了 1.7%。可见在 shuffle 跨机架过程中,本文 SARS 算法与 LARTS 算法相差不大,却比 FIFO 算法有显著提高,证明了该算法对减少网络传输有一定的意义。

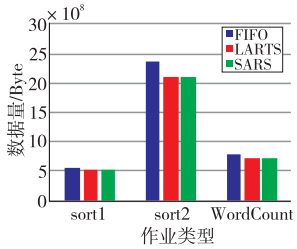

图 5　Shuffle 过程跨机架传输数据总量

综上所述,通过以上实验表明,本文提出的自适应 reduce 任务调度算法,对 reduce 任务的数据本地性、机架本地性有一定的提高,从而减少了 reduce 任务的执行时间;同时减少了 shuffle 阶段的跨机架传输,降低了网络拥塞的可能性。

4　结束语

通过分析异构环境下 Hadoop 原有调度算法在分配 reduce 任务时的缺点,以及考虑到异构环境下计算节点的差异性,提出了自适应 reduce 调度算法(SARS)。SARS 算法在选择 reduce 计算节点时不仅考虑数据本地性,也考虑节点计算能力。通过实验对比 FIFO 与 LARTS 算法,可得出本文提出的 SARS 算法在 reduce 任务的执行时间上分别提高了 7.8% 和 3% 左右;在 shuffle 传输过程中,SARS 算法虽然优于 FIFO 算法,但是相比于 LARTS 算法却有一点差距。最后通过对比 shuffle 过程中跨机架传输数据量,得出本文提出的 SARS 算法比 FIFO 算法减少了 9.8%,与 LARTS 算法相差不大。可见本文在减少 reduce 任务执行时间和减少数据传输方面有一定的优越性。

参考文献:

[1] White T. Hadoop: the definitive guide[M]. [S. l.]: O'Reilly Media, Inc, 2011.

[2] 万兵, 黄梦醒, 段茜. 一种基于资源预取的 Hadoop 作业调度算法[J]. 计算机应用研究, 2014, 31(6): 1639-1643.

[3] Seo S, Jang I, Woo K, *et al*. HPMR: prefetching and pre-shuffling in shared MapReduce computation environment[C]//Proc of IEEE International Conference on CLUSTER Computing and Workshops. 2009: 1-8.

[4] Sun Mingming, Zhuang Hang, Li Changlong, *et al*. Scheduling algorithm based on prefetching in MapReduce clusters[J]. Applied Soft Computing, 2015, 38(1): 1109-1118.

[5] Zaharia M, Borthakur D, Sarma J S, *et al*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling [C]//Proc of the 5th European Conference on Computer Systems. 2010: 265-278.

(下转第 2000 页)

algorithms	results	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
CMOGSO	mean	0.011 4	0.025 6	0.374	0.025 6	0.007 4	0.004 9
	variance	0.003 5	0.002 5	0.691 20	0.002 5	0.001	0.000 7
MGSO	mean	–	0.092	0.095 4	0.090 9	0.021 2	0.049 7
	variance	–	0.009 7	0.023 1	0.010 9	0.007	0.108 9
NSGA-Ⅱ	mean	0.621	0.058 5	8.041 6	0.056 5	0.009 4	0.012 6
	variance	3.029 5	0.004 9	10.773 2	0.004 1	0.001 0	0.000 7
MOBFO	mean	0.006 2	0.015 4	0.655 4	0.011 1	0.005 9	0.004 6
	variance	0.002 8	0.002 3	0.952 6	0.001 6	0.000 8	0.000 7

从表3、4的实验数据可以看出,对以上六个三目标测试函数,本文算法仅对DTLZ3的寻优效果略差于对比算法,对其余测试函数的寻优均值都优于对比算法。

综合以上图表的实验数据可以看出,在两目标测试函数中,本文算法仅对非连续函数ZDT3和KUR的SP指标略差,这与所求函数前沿面的连续性有关,在不连续的前沿面上,点与点之间的距离本就不均匀,因此会导致计算出的SP指标增大,对两目标测试函数的其余指标都优于对比算法;在所有的三目标优化测试函数中,本文算法仅对DTLZ3函数优化的指标GD均值弱于MGSO,而优于其余两种对比算法,SP值弱于CMOGSO和MGSO,而优于NSGA-Ⅱ,对其余函数的优化指标均优于对比算法。

考虑算法的复杂度,对比文献[2,17]中的算法,初始寻优种群规模均为100,存放最优解集的空间大小也为100,与本文中种群规模相同。对比文献[2]算法,函数评价次数为50 000,本文改进算法对函数的评价次数为30 000,时间复杂度上优于文献[2];文献[17]中,函数评价次数与本文相同。综合考虑算法的时间和空间复杂度,本文算法优于文献[2],与文献[17]相同。

综上所述,本文算法求解多目标优化问题可以得到较好的收敛性指标,对于分散性指标,除开连续的ZDT3和KUR函数外,比其他对比算法有明显优势。

4 结束语

对传统的单目标细菌觅食优化算法进行改进,设计了针对多目标优化问题的多目标细菌觅食优化算法MOBFO。针对多目标优化问题的特征,对传统单目标细菌算法的每一步操作进行调整。a)在寻优过程中得到两个解互不支配而无法判定优劣时,给出了归一化的择优策略,解决了多目标优化中解的优劣判定问题;b)在复制操作中引入差分思想,通过个体间的差分运算来完成复制,最大限度地提高种群的多样性;c)采用栅格划分迁徙法,将迁徙操作与解集的分散性结合,有目的将解迁徙到稀疏区域,取代了传统的随机迁徙过程,提高了解集的分散性;d)使用外部集来存储找到的非劣解,并设计了完整的外部集存放策略,使外部集中存储的非支配解集不断优化。实验证明,本文设计的MOBFO算法具有良好的性能,能够有效解决多目标优化问题。

参考文献:

[1] Deb K. Multi-objective optimization using evolutionary algorithms [M]. Hoboken: Wiley, 2001.

[2] 公茂果, 焦李成, 杨咚咚, 等. 进化多目标优化算法研究 [J]. 软件学报, 2009, 20(3): 272-289.

[3] Deb K, Pratap A, Agarwal S, *et al.* A fast and elitist multi-objective genetic algorithms: NSGA-Ⅱ [J]. IEEE Trans on Evolutionary Computation, 2002, 6(2): 182-197.

[4] Zitzler E, Laumanns M, Thiele L. SPEA2: improving the strength Pareto evolutionary algorithm, TIK-report 103 [R]. Zurich: Swiss Federal Institute of Technology, 2001: 126-140.

[5] Coello C A, Pulido G T, Lechuga M S. Handling multiple objectives with particle swarm optimization [J]. IEEE Trans on Evolutionary Computations, 2004, 8(3): 256-279.

[6] Gong Maoguo, Jiao Licheng, Du Haifeng, *et al.* Multi objective immune algorithm with nondominated neighbor-based selection [J]. Evolutionary Computation, 2008, 16(2): 225-255.

[7] 肖菁, 陈凤莲, 汤建超. 基于蚁群算法的多目标优化技术研究 [J]. 华南师范大学学报: 自然科学版, 2014, 46(1): 1-6.

[8] Zhang Qingfu, Li Hui. MOEA/D: a multiobjective evolutionary algorithm based on decomposition [J]. IEEE Trans on Evolutionary Computation, 2007, 11(6): 712-731.

[9] Passino K M. Biomimicry of bacterial foraging for distributed optimization and control [J]. IEEE Control Systems, 2002, 22(3): 52-67.

[10] 周雅兰. 细菌觅食优化算法的研究与应用 [J]. 计算机工程与应用, 2010, 46(20): 16-21.

[11] 王帆. 面向高维及多目标优化的协同细菌觅食算法研究 [D]. 大连: 大连理工大学, 2013.

[12] 岑雪婷. 基于细菌觅食优化算法的多目标资源受限项目调度问题研究 [D]. 广州: 华南理工大学, 2013.

[13] 李珺, 党建武. 细菌觅食优化算法求解高维优化问题 [J]. 计算机应用研究, 2016, 33(4): 1024-1027, 1033.

[14] Zitzler E, Deb K, Thiele L. Comparison of multi-objective evolutionary algorithms: empirical results [J]. Evolutionary Computation, 2000, 8(2): 173-195.

[15] Kursawe F. A variant of evolution strategies for vector optimization [C]//Proc of International Conference on Parallel Problem Solving from Nature. Berlin: Springer-Verlag, 1991: 193-197.

[16] Deb K, Thiele L, Laumanns M, et al. Scalable multi-objective optimization test problems [C]//Proc of Congress on Evolutionary Computation. Washington DC:IEEE Computer Society,2002: 825-830.

[17] 李亚洲. 多目标群搜索算法研究及其应用[D]. 济南: 山东师范大学, 2016.

(上接第1991页)

[6] Wang Weina, Ying Lei. Data locality in MapReduce: a network perspective[J]. Performance Evaluation, 2016, 96(2): 1-11.

[7] Chen Quan, Zhang Daqiang, Guo Minyi, *et al.* SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment [C]//Proc of the 10th IEEE International Conference on Computer and Information Technology. Washing DC: IEEE Computer Society, 2010: 2736-2743.

[8] Wang Bo, Jiang Jinlei, Yang Guangwen. ActCap: accelerating MapReduce on heterogeneous clusters with capability-aware data placement[C]// Proc of IEEE Conference on Computer Communications. 2015: 1328-1336.

[9] Fujishima E, Yamaguchi S. Dynamic file placing control for improving the I/O performance in the reduce phase of Hadoop[C]//Proc of the 10th International Conference on Ubiquitous Information Management and Communication. New York: ACM Press, 2016: 1-7.

[10] Hammoud M, Sakr M F. Locality-aware reduce task scheduling for MapReduce[C]// Proc of 10 3rd IEEE International Conference on Cloud Computing Technology and Science. Washington DC: IEEE Computer Society, 2011: 570-576.

[11] Arslan E, Shekhar M, Kosar T. Locality and network-aware reduce task scheduling for data-intensive applications[C]//Proc of the 5th International Workshop on Data-Intensive Computing in the Clouds. Piscataway, NJ: IEEE Press, 2014: 17-24.

[12] 董西成. Hadoop技术内幕: 深入解析MapReduce架构设计与实现原理(Hadoop internals: in-depth study of mapreduce)[M]. 北京: 机械工业出版社, 2013.

[13] Kao Y C, Chen Yashu. Data-locality-aware mapreduce real-time scheduling framework [J]. Journal of Systems and Software, 2016, 112: 65-77.

[14] Yang S J, Chen Y R. Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds[J]. Journal of Network and Computer Applications, 2015, 57(11): 61-70.

[15] Zaharia M, Konwinski A, Joseph A D, *et al.* Improving MapReduce performance in heterogeneous environments[C]// Proc of USENIX Conference on Operating Systems Design and Implementation. [S. l.]: USENIX Association, 2008: 29-42.

[16] Wang Wenzhu, Wu Qingbo, Tan Yusong, *et al.* Optimizing the MapReduce framework for CPU-MIC heterogeneous cluster[M]//Proc of the 11th International Symposium on Advanced Parallel Processing Technologies. Berlin: Springer International Publishing, 2015: 33-44.