小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2016 年 12 月 第 12 期
Vol. 37 No. 12 2016

MapReduce 模型下的分布式差分进化算法

董小刚[1],邓长寿[1],袁斯昊[1],吴志健[2],张忠平[3]

[1](九江学院 信息科学与技术学院,江西 九江 332005)
[2](武汉大学 软件工程国家重点实验室,武汉 430072)
[3](燕山大学 信息科学与工程学院,河北 秦皇岛 066004)
E-mail:dxg110@jju.edu.cn

摘 要: 差分进化算法简单、高效且鲁棒性好.然而在求解大规模优化问题时,其性能随着问题维度的增加会迅速降低.针对此问题,提出一种基于 MapReduce 编程模型的分布式差分进化算法.算法采用改进的精英学习策略和岛模型两种机制,提高算法的收敛精度.利用 MapReduce 并行编程模型,构建分布式差分进化算法,并将其部署到分布式集群 Hadoop 上.利用 13 个标准测试问题进行仿真实验,实验结果表明该算法求解精度高,且具有较好的加速比和扩展性,是求解大规模优化问题的有效方法.

关 键 词: 大规模优化;分布式差分进化;岛模型;精英学习
中图分类号: TP301 **文献标识码:** A **文 章 编 号:** 1000-1220(2016)12-2695-07

Distributed Differential Evolution Algorithm Based on MapReduce Model

DONG Xiao-gang[1], DENG Chang-shou[1], YUAN Si-hao[1], WU Zhi-jian[2], ZHANG Zhong-pin[3]
[1](School of Information and Technology, Jiujiang University, Jiujiang 332005, China)
[2](State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)
[3](School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China)

Abstract: Differential Evolution is very simple, efficient and robust. However, when dealing with large scale optimization problem, the performance of Differential Evolution will deteriorate rapidly as the dimensionality of the search space increases. To overcome this problem, a Distributed Differential Evolution algorithm Based on MapReduce Model was proposed. Firstly, the elite learning strategy and island Model were used to improve the convergence accuracy. And Secondly, with MapReduce model, the Distributed Differential Evolution algorithm was constructed. Then it is deployed on Hadoop cluster. The proposed algorithm has been tested on 13 Benchmark Functions, and experimental result shows that the performances of the new algorithm is competitive, and has good performances of speedup and scalability, thus it is an effective method for solving large scale optimization problem.

Key words: large scale optimization; distributed differential evolution; island model; elite learning

1 引 言

互联网时代各领域的数据规模急速增长,由此产生的各种优化问题的规模(维度)也呈指数级增长,使得大规模优化问题成为当前的研究热点之一.

差分进化[1]算法结构简单、性能高效、鲁棒性强,是具有竞争力的智能优化算法之一.过去二十多年中,科学领域的研究成果以及工程领域的应用情况表明差分进化算法在低维优化问题上表现优秀[2-6].然而,传统的差分进化算法在求解大规模优化问题时,其效率将随着问题维度增加而恶化.相关研究表明分布式计算框架是解决大规模数据处理的必然选择,它将工作任务分配到各个工作节点,实现分布式并行处理,通过各节点的协同合作,共同完成计算任务[7].

MapReduce 是由 Google 实验室提出的分布式框架,广泛用于处理大规模数据集[8].该模型下用户仅需考虑 Map 和 Reduce 两个函数的实现,从而降低了分布式计算的实现复杂度.Apache 的 Hadoop 平台为 MapReduce 提供了开源实现[9].借助该平台用户可以充分利用集群资源,实现大规模数据的高速运算和存储.目前在各领域借助 MapReduce 模型进行大规模数据处理已经有了一些研究成果[10-13].此外,已有研究人员将 MapReduce 模型和智能进化算法进行结合,实现进化过程并行化.例如吴昊[14]等人提出基于 MapReduce 的蚁群算法,并应用于大规模旅行商问题的求解,取得了一定的效果.倪志伟[15]等人提出了一种云和声搜索算法,利用 MapReduce 模型和分布式文件系统 HDFS 实现了和声算法的并行化,实验结果表明该算法的有效性.

为了提高差分进化算法求解大规模优化问题的性能,本文提出一种 MapReduce 模型下的分布式差分进化算法(Dis-

收稿日期:2015-11-03 收修改稿日期:2016-01-22 基金项目:国家自然科学基金项目(61364025)资助;武汉大学软件工程国家重点实验室开放基金项目(SKLSE2012-09-39)资助;江西省教育厅科学技术项目(GJJ13729, GJJ14742)资助;九江学院科研项目(2013KJ27, 2014KJYB034, 2015LGYB29)资助. 作者简介:**董小刚**,男,1979 年生,硕士,讲师,CCF 会员,研究方向为智能计算;**邓长寿**,男,1972 年生,博士,教授,CCF 会员,研究方向为智能计算、云计算;**袁斯昊**,男,1979 年生,硕士,讲师,研究方向为智能计算;**吴志健**,男,1963 年生,博士,教授,博士生导师,研究方向为智能计算、并行计算和智能信息处理;**张忠平**,男,1972 年生,博士后,教授,CCF 高级会员,研究方向为大数据、数据挖掘、半结构化数据等.

tributed Differential Evolution based on MapReduce Model, DDE-MR). DDE-MR 算法采用改进的精英学习变异模式,并结合多岛协作进化策略来提高算法的收敛速率. 采用 Hadoop 集群进行 DDE-MR 算法的部署,验证了算法的整体性能.

2 差分进化

DE 算法是一种基于实数编码的群体优化算法主要过程为初始化种群、变异、交叉及选择四个步骤. 具体描述如下:

1)*初始种群*:DE 首先在一个 D 维空间上随机产生 NP 个受上下界约束的个体 $x_{i,0}$ 构成初始种群,其中 $i \in [1, NP]$.

2)*变异算子*:DE 算法针对种群中的每一个个体,利用若干个随机选择的个体的差向量产生变异向量 $v_{i,g+1}$. *DE/rand/1/bin* 是 DE 算法最常用的一种变异算子,如公式(1)所示.

$$v_{i,g+1} = x_{r1,g} + F \cdot (x_{r2,g} - x_{r3,g}) \tag{1}$$

其中 $r1 \neq r2 \neq r3 \neq i$, $r1, r2, r3 \in [1, NP]$,缩放因子 $F \in [0,2]$

3)*交叉算子*:DE 算法依据交叉算子产生试验向量 u,如式(2)所示.

$$u_{ij,g+1} = \begin{cases} v_{ij,g+1} & if\ rand < CR\ or\ j = R(i) \\ x_{ij,g} & otherwise \end{cases} \tag{2}$$

其中 $i = 1, 2, \cdots, NP$; $j = 1, 2, \cdots, D$; $rand \in [0,1]$ 是一个均匀分布的随机数; $R(i)$ 是 $[1, D]$ 之间的随机整数; $CR \in [0,1]$ 为交叉概率. 该算子保证至少有一位基因来自变异个体.

4)*选择算子*:DE 算法的选择操作,是让试验向量 $u_{i,g+1}$ 和当前种群的 $x_{i,g}$ 依据目标函数适应度值进行竞争,二者中胜出的进入下一代种群,如式(3)所示.

$$x_{i,g+1} = \begin{cases} u_{i,g+1} & if\ \ f(u_{i,g+1}) \leqslant f(x_{i,g}) \\ x_{i,g} & otherwise \end{cases} \tag{3}$$

3 基于 MapReduce 的分布式差分进化算法 DDE-MR

3.1 新型精英学习变异策略

2008 年 Rahnamayan 等人[16] 首次将反向学习策略(Opposition-Based Learning, OBL)引入到差分进化算法中. 实验证明反向学习策略能有效提高差分进化算法性能. 然而 OBL 策略对每个个体进行反向求解,带有一定的盲目性. 受 OBL 启发,周新宇等人[17] 提出一种精英反向学习策略(Elite Opposition-Based Learning, EOBL),并依据该策略实现了精英反向学习差分进化算法 EOBDE.

EOBDE 算法改进了 OBL 策略的盲目性,使得个体的反向解通过学习当前精英而产生,从而加速了 DE 算法的收敛速率. 然而,EOBDE 算法采用学习当前最优解的方式,与经典 DE/Best/1 变异算子有一定的相似性,进化过程易陷入局部收敛. 另外, EOBDE 算法采用动态边界限定搜索空间,使得局部收敛的可能性进一步增加,不利于提高 DE 算法在高维空间中的搜索能力.

本文提出一种新的改进的精英学习(New Elite learning, NEL)变异策略,如式(4)所示.

$$\breve{x}_{i,j} = k \cdot (a_j + b_j) + \sin(\omega) \cdot (c \cdot x_{e,j} + (1 - c) \cdot x_{r1,j}) \tag{4}$$

其中, $k \in [0,1]$ 为服从均匀分布的随机数; a_j, b_j 分别为当前种群的边界的上下界; $x_{e,j}$ 为当前精英解的第 j 个基因; $x_{r1,j}$ 为种群中第 $r1$ 个个体的第 j 个基因, $r1 \neq i$, $r1 \in [1, NP]$; $\omega \in [0, 2\pi]$, c 为固定值 0.3.

公式(4)所表示的精英学习策略中,采用固定的全局边界,利于保持种群的多样性. 其次,新的变异策略按照固定比率分别向精英个体和一个随机选择的个体进行学习,使得后代个体的产生,向精英靠近的同时,能够利用社会学习的方式保持一定的多样性,降低陷入局部最优的几率. 最后,采用定义域为 $[0, 2\pi]$ 的正弦函数对学习空间进行扩展. $[0, 2\pi]$ 区间内正弦函数值域为 $[-1, 1]$,且正弦函数是一种均匀分布函数,故利用正弦函数进行扩展,使得学习能够在更广的空间上均匀地进行.

在 NEL 策略的基础之上,本文提出改进的精英学习差分进化算法(Differential Evolution Based on New Elite learning, DE-NEL),DE-NEL 算法的伪代码如下所示:

Step 1. Initialization population P and set parameters
Step 2. Evaluate the population P
Step 3. find the Elite individual x_e
Step 4. $FES = NP$;
Step 5. while $FES < MaxFES$
Step 6. for i = 1:NP
Step 7. if (rand < ep)
Step 8. carry out NEL according to equation (4)
Step 9. else
Step 10. carry out DE/rand/1/bin according to (1) and (2)
Step 11. end if
Step 12. Evaluate the trial vector $u_{i,g}$
Step 13. select $u_{i,g}$ and $x_{i,g}$ according to equation 3.
Step 14. update x_e
Step 15. end for
Step 16. end while
Step 17. output best individual

在上述 DE-NEL 算法中,ep 是 NEL 策略的执行控制参数,设置为 0.05. rand 是 $[0,1]$ 之间的随机数. 本文算法 DDE-MR 中,每个子种群均使用 DE-NEL 算法.

从时间复杂度上来看,DE-NEL 算法第 5 到第 16 步计算规模最大. 其中,执行 NEL 策略的第 8 步和变异交叉的第 10 步,时间复杂度为 $O(D)$. for 循环依次遍历每一个个体,所以,6 到 15 步的时间复杂度为 $O(NP * D)$. while 循环迭代 $(MaxFES - NP)/NP$ 次(即进化代数 iter), 5 至 16 步的时间复杂度为 $O(iter * NP * D)$. 其它各步计算规模较小,略去不计. 因此,DE-NEL 算法的时间复杂度为 $O(iter * NP * D)$,与 DE 算法时间复杂度保持一致.

3.2 岛模型进化策略

岛模型是一种粗粒度的分布式模型. 岛模型不仅可以有效降低计算时间消耗,而且可以明显改善算法的全局搜索能力[18]. DDE-MR 算法采用岛模型实现,涉及岛的数目(子种群数目)、迁移拓扑结构、迁移数量、迁移频率及迁移方法五个参数的设置,其中核心参数是迁移拓扑结构和迁移方法.

迁移拓扑结构是指子种群之间依据何种结构进行个体的

迁移. 图 1 给出两种常见的环形拓扑结构,其中左侧(a)图为一般环形拓扑结构. 右侧(b)图为一种扩展的环形拓扑结构. 在这种扩展的环形结构中,每一个种群均向邻接其后的两个种群进行迁移操作,称之为 Ring +1 +2 结构. 文献[19]的研究指出 Ring +1 +2 拓扑是一种可靠且更适合于并行进化算法的拓扑结构. 因此,在 DDE-MR 算法的实现过程中,采用 Ring +1 +2 结构来进行子种群之间的个体迁移操作.

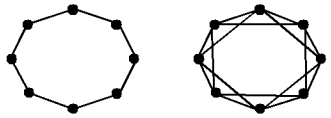

图 1 Ring (a),Ring +1 +2 (b)

Fig. 1 Ring (a),Ring +1 +2 (b)

迁移方法采用的是源种群和目标种群依据"优-差"进行替换,即用源种群最优的 m(m 代表迁移数量)个体替换目标种群最差的 m 个体. 具体的替换条件为式(5)所示:

$$f_k(x_{best}) < f_{k+n}(x_{worst}), n = 1, 2 \tag{5}$$

其中,k 为当前子种群索引. $k+n$ 是第 k 个子种群后相邻的第 n 个子种群的索引.

3.3 DDE-MR 的实现

基于 MapReduce 构建并行计算程序的两个主要任务是定义 Map 和 Reduce 函数. 其中,Map 负责对整个任务进行分解,而 Reduce 负责把分解后的多任务处理结果进行归约汇总. Map 以 key-value 数据对作为输入,返回另外的 key-value 作为中间输出数据. Reduce 以 key 及对应的 value 列表作为输入,通过合并 key 相同的 value 后,返回另外一系列 key-value 对,作为最终的输出结果写入 HDFS.

在 Map 和 Reduce 之间存在关键的 Shuffle 过程. Shuffle 过程是整个 MapReduce 编程框架的核心,它从 Map 的数据输出开始,到 Reduce 接收到数据启动执行结束. 其主要工作可描述如下:

1)数据写入缓冲区. 在 Map 任务的最后阶段,首先将中间数据写入到内部缓冲区.

2)数据溢写. 当数据缓冲区的使用率达到阀值后,启动一次溢写操作,对中间数据进行分区、排序,并最终写入磁盘形成一个新的溢写文件.

3)数据合并. 当 Map 任务的所有数据处理完成后,对所有中间文件进行合并操作,最终形成一个中间数据文件.

4)数据复制. 当集群上 Map 任务的完成数达到一定数目之后,Reduce 任务开始从 Map 的任务节点上复制数据. 在复制数据的同时,启动两个后台线程对内存和磁盘上的数据文件做合并操作.

5)数据排序. 在前期与排序的基础上, Reduce 再做一次归并排序,保证复制数据的整体有序性.

执行完合并与排序操作后,将数据交给 reduce( )方法处理,Shuffle 过程结束.

依据 MapReduce 编程模型,DDE-MR 算法主要工作如下:

第一:由于 MapReduce 要求所处理数据为 key-value 对,且 DDE-MR 采用多岛进化模型. 为此,在 DDE-MR 实现过程中 key-value 对采用式(6)所示形式:

$$[\text{key}_i, \text{value}_i], i = 1, 2, \cdots, m \tag{6}$$

其中,m 为岛模型中独立岛的数目;key_i 是一个整数,是第 i 个独立岛的索引;value_i 是一个列表,包含第 i 个独立岛中所有个体.

第二:利用 Map 的并行操作,将总任务分解,并分配给集群运行. 每个 Map 任务都对一个岛进行独立进化,所产生中间数据直接存入 HDFS,作为下一轮进化的输入. 其中,完成每个岛独立进化的 Map 函数结构如下:

```
Function Map(key1,value1)
{
    indexSubPop = key1;
    //读取传递进来的子种群索引
    tempPop = value1;
    //读取传递进来的子种群信息
    MaxIter = M;
    //设置最大进化代数
    Iter = 0;
    //设置初始进化代数
    while(iter < maxIter)
    {
    //采用 DE-NEL 作为优化器进化子种群 tempPop
    Evolve the tempPop by DE-NEL.
    }
    Read the target Subpopulation information
    //读取待迁移的目标子种群信息;
    execute the migration operation
    //执行子种群间的迁移操作;
    output of Subpopulation information to the HDFS
    //输出种群信息到 HDFS;
}
```

第三:DDE-MR采用岛模型进化机制,当一次迁移操作完成之后,每一个岛所对应的新种群数据直接作为中间数据

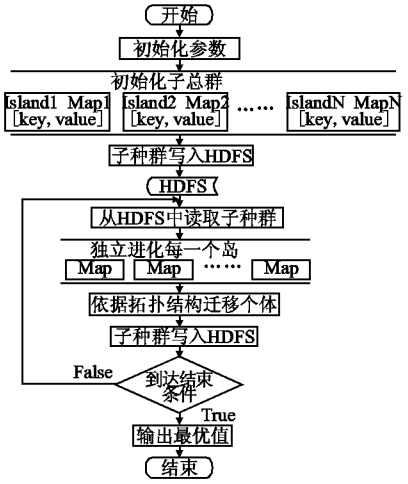

图 2 DDE-MR 流程

Fig. 2 DDE-MR Flow

直接写入 HDFS,下一轮进化直接读取 HDFS 即可.无需对每个岛所对应的子种群进行重新归约.因此,DDE-MR 在实现过程没有指定 Reduce 函数的规约任务,采用系统默认的空 Reduce 函数,即 Reduce 函数不做任何处理.

依据以上思路,DDE-MR 的如图 2 所示.

DDE-MR 算法采用 DE-NEL 作为优化器并行进化每一个岛.因此,时间在复杂度与 DE-NEL 保持一致.空间复杂度为岛的数目和子种群规模(个体数量)乘积.

4 实验与分析

4.1 测试问题

为了验证 DDE-MR 算法对大规模优化问题的求解性能,选取 13 个广泛采用的标准测试函数作为仿真测试集.测试集中的 13 个函数特征各不相同,其中 f_1 ~f_5 为单峰函数,仅有一个极值点;f_6 为阶梯函数,不连续且仅有一个极值点;f_7 为带噪声的四次函数,仅有一个极值点;f_8 ~f_{13} 均为多峰函数,具有多个局部极值点;所有 13 个函数的详细特征如表 1 所示.

表 1 测试问题

Table 1 Test problem

Fun	名 称	定义域	最优解
f_1	Sphere Model	[-100,100]	0
f_2	Schwefel's Problem 2.22	[-10,10]	0
f_3	Schwefel's Problem 1.2	[-100,100]	0
f_4	Schwefel's Problem 2.21	[-100,100]	0
f_5	Generalized Rosenbrock's Function	[-30,30]	0
f_6	Step Function	[-100,100]	0
f_7	Quartic with Noise	[-1.28,1.28]	0
f_8	Generalized Schwefel's Problem 2.26	[-500,500]	D*418.983
f_9	Generalized Rastrigin's Function	[-5.12,5.12]	0
f_{10}	Ackley's Function	[-32,32]	0
f_{11}	Generalized Griewank Function	[-600,600]	0
f_{12}	Generalized Penalized Functions 1	[-50,50]	0
f_{13}	Generalized Penalized Functions 2	[-50,50]	0

4.2 实验环境配置

DDE-MR 算法采用 Hadoop 2.2.0 进行集群构建.集群结构采用主从式,由 8 台 x86 计算机组成.具体硬件配置为 64bit corel7 cpu,主频 3.4G,内存 8G.1 台作为集群主控节点,7 台作为数据节点.软件环境采用 Ubuntu13.10

操作系统和 Eclipse 编程环境.为充分使用集群资源,提高集群执行效率,DDE-MR 的实现,按式(7)对数据节点上执行 Map 任务的最大个数(M)进行设置.

$$M = \text{数据节点最大内存} / \text{单个 } Map \text{ 任务内存} \quad (7)$$

集群设置中每个 Map 任务的内存设置为 2GB,因此,每一个数据节点上所能并行的最大 Map 任务数为 4.

4.3 算法收敛性实验

为验证 DDE-MR 算法求解大规模优化问题的有效性,设置 13 个函数的维度为 1000,每个函数独立执行 20 次,记录 DDE-MR 所获得的最好解,最差解,平均值及方差.并将结果和经典 DE 算法、求解大规模优化问题的知名算法 DECCG [20]进行比较.实验参数设置为,DDE-MR 算法的缩放因子 F=0.5,交叉概率 CR=0.5,子种群个数为 20,子种群规模为 50,迁移频率为 500 代(每隔 500 代进行一次迁移),迁移数量为 15 个个体.DE 算法的缩放因子 F=0.5,交叉概率 CR=0.5,种群规模为 1000;DECCG 算法采用原文献参数设置.结束条件均为目标函数评价次数达到 D*5000 次.仿真结果见表 2.

表 2 收敛性实验结果

Table 2 Experimental results of convergence

fun	算法	最优解	最差解	平均值	方差
	DE	1.19e+06	1.31e+06	1.27e+06 -	2.79e+04
f_1	DECCG	4.96e-29	1.88e-28	9.53e-29 -	3.51e-29
	DDE-MR	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	DE	2.32e+170	Inf	Inf -	Inf
f_2	DECCG	2.84e-015	3.86e-14	1.41e-14 -	1.07e-14
	DDE-MR	2.90e-187	1.24e-183	1.67e-184	0.00e+00
	DE	1.51e+07	2.12e+07	1.89e+07 -	1.36e+06
f_3	DECCG	4.22e-04	3.40e-03	1.60e-03 -	9.77e-04
	DDE-MR	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	DE	9.44e+01	9.53e+01	9.49e+01 -	2.16e-01
f_4	DECCG	2.47e-02	4.14e-02	3.26e-02 -	4.70e-03
	DDE-MR	4.20e-123	1.57e-78	1.06e-79	3.56e-79
	DE	3.61e+09	4.06e+09	3.88e+09 -	1.18e+08
f_5	DECCG	9.85e+02	9.87e+02	9.86e+02 +	4.19e-01
	DDE-MR	9.89e+02	9.90e+02	9.90e+02	1.48e-01
	DE	1.20e+06	1.31e+06	1.27e+06 -	2.90e+04
f_6	DECCG	0.00e+00	0.00e+00	0.00e+00≈	0.00e+00
	DDE-MR	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	DE	5.20e+04	6.14e+04	5.79e+04 -	2.56e+03
f_7	DECCG	1.40e-03	3.70e-03	2.60e-03 -	6.72e-04
	DDE-MR	5.62e-06	4.32e-05	2.38e-05	1.12e-05
	DE	-1.27e+05	-1.21e+05	-1.24e+05	-1.90e+03
f_8	DECCG	-4.19e+05	-4.19e+05	-4.19e+05	-0.00e+00
	DDE-MR	-6.49e+04	-6.65e+04	-6.53e+04	3.18e+03
	DE	1.29e+04	1.32e+04	1.31e+04 -	6.95e+01
f_9	DECCG	5.33e-15	2.84e-14	1.49e-14 -	6.00e-15
	DDE-MR	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	DE	1.96e+01	1.98e+01	1.97e+01 -	3.50e-02
f_{10}	DECCG	1.22e-13	1.39e-13	1.28e-13 -	5.87e-15
	DDE-MR	4.44e-16	4.44e-16	4.44e-16	0.00e+00
	DE	1.06e+04	1.17e+04	1.14e+04 -	2.75e+02
f_{11}	DECCG	5.55e-16	1.22e-15	9.27e-16 -	1.74e-16
	DDE-MR	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	DE	7.37e+09	8.35e+09	7.95e+09 -	2.71e+08
f_{12}	DECCG	6.16e-28	1.73e-27	1.26e-27 +	3.18e-28
	DDE-MR	6.99e-02	1.03e-01	8.37e-02	7.11e-03
	DE	1.55e+10	1.69e+10	1.60e+10 -	3.89e+08
f_{13}	DECCG	5.77e-24	4.39e-02	4.40e-03 +	1.03e-02
	DDE-MR	9.76e+01	9.88e+01	9.84e+01	2.85e-01

从表 2 三种算法对 13 大规模个问题的求解结果来看,DDE-MR 算法对其中的 f_1,f_3,f_6,f_9,f_{11} 五个问题均能找到全局最优解,DDE-MR 算法对这 5 个问题的求解性能非常稳定;DECCG 算法,仅在问题 f_6 上找到了全局最优解(与 DDE-MR 算法性能相当).对于问题 f_1,f_2,f_3,f_4,f_7,f_8,f_9,f_{10},f_{11} 9 个问题的求解结果 DECCG 均弱于 DDE-MR 算法,且在 f_1,f_2,

f_3,f_4,f_{11}这 5 个问题上的差距非常明显. 在 f_5,f_{12},f_{13}3 个问题上,DECCG 所得结果略胜于 DDE-MR 算法. 总体来看 DECCG 对 13 个问题问题的求解性能明显弱于 DDE-MR 算法; DE 算法对所有 13 个问题的求解结果均不理想,尤其是在问题 f_2 上,DE 算法几乎不能获得有效解. 这说明经典 DE 算法在面对大规模优化问题时,全局寻优能力下降明显. DDE-MR 算法在 13 个大规模优化问题的求解上均优于 DE 算法.

表 3 Wilcoson 检验结果统计

Table 3 Results statistical of wilcoson test

算法\类别	优	劣	相似
DE	0	13	0
DECCG	3	9	1

本文采用显著水平为 0.05 的 Wilcoxon 检验方法,对三种算法的求解结果进行检验. 检验结果位于表中均值之后,其中符号“+”表示优于 DDE-MR,“-”表示差于 DDE-MR,“≈”表示与 DDR-MR 算法性能相似. 检验的统计结果见表 3. 从 Wilcoxon 检验的统计结果来看,DDE-MR 算法在 13 个问题上均优于 DE 算法,在一个问题(f_6)上与 DECCG 算法性能相当,在三个问题(f_5,f_{12},f_{13})上差于 DECCG 算法,在其余 9 个问题上优于 DECCG 算法. 检验统计结果表明 DDE-MR 算法的求解性能总体优于其它两种算法.

图 3 至图 7 的 5 幅收敛曲线图描述了三种算法对 5 个大规模问题的求解过程(受篇幅限制,仅选取5个问题绘制收敛曲线),从收敛曲线来看,DDE-MR算法整体收敛能力较

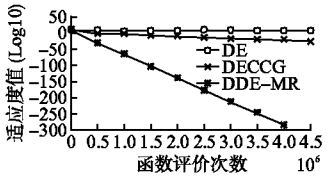

图 3 f1 函数收敛曲线图

Fig. 3 Convergence curve of Function f1

强,在三个单峰问题上 DDE-MR 算法的收敛速度明显快于 DE 和 DECCG 两种算法;在带噪声函数 f_7 上,DDE-MR 算法

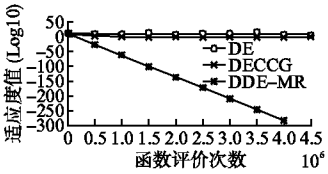

图 4 f3 函数收敛曲线图

Fig. 4 Convergence curve of function f3

的收敛速度大幅度的快于 DE 算法. 与 DECCG 相比,在进化前期都有很好的收敛速率,后期二者均出现一定程度上的局部收敛现象,但DDE-MR算法最终收敛结果优于DECCG算

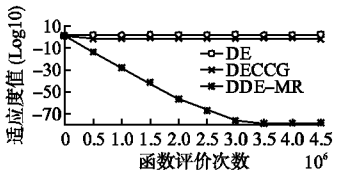

图 5 f4 函数收敛曲线图

Fig. 5 Convergence curve of function f4

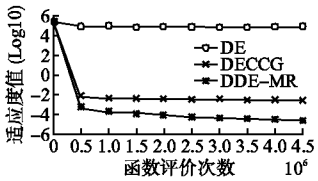

图 6 f7 函数收敛曲线图

Fig. 6 Convergence curve of function f7

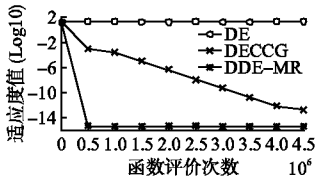

图 7 f10 函数收敛曲线图

Fig. 7 Convergence curve of Function f10

法;在多峰问题 f_{10} 的收敛曲线上,DDE-MR 算法的收敛速率均快于 DE,DECCG 两种算法.

最后,采用 Friedman 检验方法,对三种算法求解 13 个大规模问题的性能进行排名检验,结果见表 4. 其中,DDE-MR 算法的排名检验值为 1.46,均小于 DE,DECCG 两种算法的排名检验值,排名第一.

表 4 Friedman 检验

Table 4 Friedman test

算法	DE	DECCG	DDE-MR
检验值	2.92	1.63	1.46

以上实验结果表明,在收敛精度和收敛速率上 DDE-MR 算法总体优于 DE、DECCG 两种算法,说明 DDE-MR 算法较好地提升了 DE 算法求解大规模优化问题的整体性能.

4.4 集群加速性能实验

加速性能是衡量一个系统在扩展性方面优劣的主要指标, 主要考察两个方面的性能:一是当计算硬件资源增加时,对于相同规模的作业,系统的处理能力; 二是当计算资源和处理作业的规模保持相近比例增长时,系统的处理能力[21].

因此,本实验从两个方面进行:

实验 1. 当计算机硬件资源增加,集群处理同规模作业的加速性能检验. 为此,本实验针对 13 个 1000 维函数的优化任务(结束条件为 FES = D * 5000),分别选择 1、2、3、4、5、6、7 个计算节点进行 20 次独立试验,记录每个函数不同数据节点下完成任务的平均消耗时间,具体结果见下页表 5.

从表 5 的结果来看,在所有 13 个函数上,随着节点数的逐步增加,算法执行所消耗的时间在不断减少,尤其是在逐步增加节点的前期,每增加一个节点,程序执行时间减少幅度较大,加速效果非常明显. 从表 5 底部的均值来看,当数据节点增加到 6 个时,集群的加速效果达到峰值,算法执行消耗时间最少. 再增加第 7 个节点,算法消耗时间反而出现增加. 这与 DDE-MR 算法和集群的设置相关. DDE-MR 算法采用的岛模型进化机制设置子种群数为 20,而在基于 MapReduce 模型实现时,子种群和 Map 的任务数一一对应,即一个子种群对应一个 Map 任务. 如前文所述,Hadoop 集群中每个数据节点 Map 任务的最大执行数为 4. 因此,当集群中数据节点数为 5 个时,正好每个节点上达到最大 Map 任务数,集群资源得到充分利用,加速效果明显. 当数据节点继续增大时,依据 DDE-MR 的设置,20 个

子种群分配于大于5个结点上执行,其中必然有部分结点不能充分使用资源,而多出来的结点却带来通信上的消耗.6个数据节点时,时间消耗均值继续减小,说明第6个节点的增加带来的加速效果整体上仍然大于通信消耗.但是,部分函数(f_9,f_{10})的时间消耗相对于5个数据节点已出现明显增加.说明此时数据节点增加所带来的加速效果大部分被通信消耗抵消.7个数据节点时,时间消耗均值出现增加情况,说明通信消耗已经大于增加节点所带来的加速效果.

加速比[22]是评价并行算法效率的常用指标,其具体定义如公式(8)所示.

表5 加速性能实验(单位:毫秒)

Table 5 Acceleration experiment(unit:ms)

问题/节点数	1	2	3	4	5	6	7
f_1	374812	267216	214226	205800	194855	193341	194429
f_2	371438	266230	212239	203194	193253	192408	191930
f_3	2859499	1717703	1244552	1149524	1037532	809812	817033
f_4	387120	263837	212596	201406	191671	190801	190196
f_5	385556	264043	211358	202446	191971	191149	192248
f_6	385680	265758	212614	202817	192655	191031	191953
f_7	385244	266334	213481	203816	191957	191048	190748
f_8	524795	347475	266979	254133	243278	242366	242863
f_9	423366	289022	227092	214931	203868	204223	204293
f_{10}	424312	289058	227026	214444	203494	204184	204398
f_{11}	415660	294312	231776	218970	206104	204368	205889
f_{12}	500010	348117	269145	253482	242829	242147	241938
f_{13}	400940	288841	226392	214354	204687	202770	203989
mean	559888	369139	283534	267094	249869	232832	233708

$$S_m(N)=\frac{T_m(1)}{T_m(N)} \tag{8}$$

其中,$T_m(1)$表示数据节点为1个时算法的执行时间;$T_m(N)$表示数据节点为N个时算法的执行时间.

依据表5求得加速比,并绘制加速比曲线,如图8所示.图8的加速比曲线图验证了上述分析.

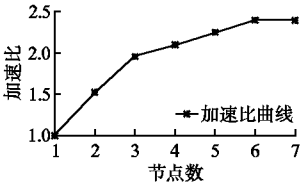

图8 加速性能曲线图

Fig. 8 Acceleration curve

实验2.当计算资源和处理作业的规模保持相近比例增长时,集群处理能力检验.为此,本实验选取8个问题(f_1,f_2,f_4,f_5,f_6,f_7,f_9,f_{10}),分别采用1000,2000,3000三种维度规模进行实验.并分别部署于1个、2个、3个数据节点的集群上执行,结束条件统一设置为FES = D * 5000.记录三种规模,三种集群上每个函数的执行时间(执行20次,记录平均时间),数据仿真结果见表6,表6最后一行为平均值.为清晰的描述平均消耗时间的变化,依据其均值绘制柱状图,如图9所示.

表6 平均时间消耗(单位:毫秒)

Table 6 Average time consumption (unit:ms)

问题	1000维 1个数据节点	2000维 2个数据节点	3000维 3个数据节点
f_1	374812	381785	415528
f_2	371438	379204	414002
f_4	387120	367244	400208
f_5	385556	369984	404492
f_6	385680	371815	405489
f_7	385244	369527	401348
f_9	423366	453824	529066
f_{10}	424312	451392	525890
mean	392191	393097	437003

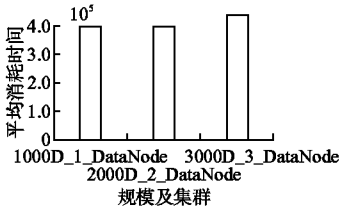

图9 平均时间消耗柱状图

Fig. 9 Histogram of average time cost

从表6的数据和图9的柱状图来看,节点资源和优化问题的规模保持同比例增长的时候,DDE-MR执行消耗时间基本接近,没有数量级上的差别.这表明DDE-MR算法在优化问题规模增长时,通过增加相似比例的节点资源,取得了较好的求解效率,表明DDE-MR算法扩展性较好.

5 结 论

本文提出一种分布式差分进化算法DDE-MR,该算法采用改进的精英学习策略和岛模型进化机制,提高了算法的收敛精度;此外,为提高算法对大规模优化问题的求解效率,采用分布式并行编程模型MapReduce实现了算法,并将算法部署在Hadoop集群上执行.收敛性实验表明DDE-MR算法具备很好的收敛精度,能有效求解大规模优化问题;集群加速性能实验表明DDE-MR算法具备良好的加速性能和扩展性能,是差分进化算法的一种有效的并行化改进.未来研究中,我们将在DDE-MR的基础之上进一步探索提高算法性能的进化

策略,并不断改进其并行化方案,提高算法对大规模优化问题的求解效率和求解精度.

References:

[1] Storn R, Price K. Differential evolution-a simple an efficient heuristic for global optimization over continuous spaces [J]. Journal of Global Optimization, 1997, 11(4): 341-359.

[2] Wang Y, Cai Z, Zhang Q, et al. Differential evolution with composite trial vector generation strategies and control parameters [J]. IEEE Transactions on Evolutionary Computation, 2011, 15(1): 55-66.

[3] Qin A K, Huang V L, Syganthan P N, et al. Differential evolution algorithm with strategy adaptation for global numerical optimization [J]. IEEE Transactions on Evolutionary Computation, 2009, 13(2): 398-417.

[4] Zhang J, Sanderson A C. JADE: adaptive differential evolution with optional external archive [J]. IEEE Transactions on Evolutionary Computation, 2009, 13(5): 945-958.

[5] Brest J, Greiner S, Boskovic B, et al. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems [J]. IEEE Transactions on Evolutionary Computation, 2006, 10(6): 646-657.

[6] Wang H, Wu Z, Rahnamayan S, et al. Enhanced opposition-based differential evolution for solving high-dimensional continuousoptimization problrm [J]. Soft Computing, 2011, 15(11): 2127-2140.

[7] Wang Xian-wei, Dai Qing-yun, Jiang Wen-chao, et al. Retrieval of design patent images based on Mapreduce model [J]. Journal of Chinese of Computer System, 2012, 33(3): 626-632.

[8] Lu Wei-ming, Du Chen-yang, Wei Bao-gang, et al. Distributed affinity propagation clustering based on MapReduce [J]. Journal of Computer Research and Development, 2012, 49(8): 1762-1772.

[9] Apache software foundation. hadoop [EB/OL]. http://hadoop.apache.org/, 2008.

[10] Jeffrey Dean, Sanjay Ghemawat. Map reduce: a flexible data processing tool [J]. Communications of the ACM, 2010, 53(1): 72-77.

[11] Tian Xia. Large-scales SMS messages mining based on MapReduce [C]. Proceedings of the International Symposium on Computational Intelligence and Design, 2008: 7-12.

[12] Aaron McKenna, Matthew Hanna, Eric Banks, et al. The genome analysis tookit: a MapReduce framework for analyzing next-generation DNA Sequencing data [J]. Genome Research, 2010, 20(9): 1297-1303.

[13] Yuan Yu-lai, Wu Yong-wei, Feng Xiao, et al. VDB-MR: MapReuce-based distributed data integration using virtual databse [J]. Future Generation Computer Systems, 2010, 26(8): 1418-1425.

[14] Wu Hao, Ni Zhi-wei, Wang Hui-ying. MapReduce-based ant colony optimization [J]. Computer Integrated Manufaceturing Systems, 2012, 18(7): 1503-1509.

[15] Ni Zhi-wei, Wu Hao, Yin Dao-ming, et al. Clund harmony sereach algorithms for optimization of composite knowledge as a service [J]. Application Research of Computers, 2013, 30(3): 806-809, 813.

[16] Rahnamayan S, Tizhoosh H R, et al. Opposition-based differential evolution [J]. IEEE Transactions on Evolution Computation, 2016, 10(6): 646-657.

[17] Zhou Xin-yu, Wu Zhi-jian. Elite opposition based particle swarm optimization [J]. Acta Electronoca Sinica, 2013, 41(8): 1647-1652.

[18] Gong Y J, Chen W N, Zhan Z H, et al. Distributed evolution algorithms and their models: A survey of the state-of-the-art [J]. Application of Soft Compute, 2015, 34(9): 286-300.

[19] Ruciński M, Izzo D, Biscani F. On the impact of the migration topology on the Island Model [J]. Parallel Computing, 2010, 36(10-11): 555-571.

[20] Yang Zhen-yu, Tang Ke, Yao Xin. Large scale evolutionary optimization using cooperative coevolution [J]. Information Sciences, 2008, 178(15): 2985-2999.

[21] Jiang Xiao-ping, Li Cheng-hua, Xiang Wen, et al. Parallel implementing k-means clustering algorithm using MapReduce programming mode [J]. Jornal of Huazhong University of Science and Technology (Natural and Science Edition), 2011, 39(S1): 120-123.

[22] Grama A, Gupta A, Karypis G, et al. Introduction to parallel computing [M]. Addison-Wesley, USA, 2000.

附中文参考文献:

[7] 王贤伟,戴青云,姜文超,等. 基于 Mapreduce 的外观设计专利图像检索方法[J]. 小型微型计算机系统,2012,33(3):626-632.

[8] 鲁伟明,杜晨阳,魏宝刚,等. 基于 MapReduce 的分布式近邻传播聚类算法[J]. 计算机研究与发展,2012,49(8):1762-1772.

[14] 吴 昊,倪志伟,王会颖. 基于 MapReduce 的蚁群算法[J]. 计算机集成制造系统,2012,18(7):1503-1509.

[15] 倪志伟,吴 昊,尹道明,等. 云和声搜索算法及其在知识服务组合中的应用[J]. 计算机应用研究,2013,30(3):806-809,813.

[17] 周新宇,吴志健. 一种精英反向学习的粒子群优化算法[J]. 电子学报,2013,41(8):1647-1652.

[21] 江小平,李成华,向 文,等. k-means 聚类算法的 MapReduce 并行化实现[J]. 华中科技大学学报(自然科学版),2011,39(S1):120-124.