基于MapReduce的蚁群优化算法实现方法

王诏远 李天瑞 易修文
(西南交通大学信息科学与技术学院 成都 610031)
(四川省云计算与智能技术高校重点实验室 成都 610031)

摘 要 探讨了蚁群算法的几种并行方式与适用场景以及结合云计算编程框架MapReduce的可行性,并将局部搜索类蚁群优化算法抽象为几个组件,分别与MapReduce框架的几个接口对应实现,从而为该类蚁群优化算法在MapReduce框架下实现并行化提供了一种灵活、扩展性好的解决方案。最后通过旅行商问题的仿真实验验证了所提方法的有效性。

关键词 蚁群优化算法,MapReduce,Hadoop,旅行商问题
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.07.054

Approach for Development of Ant Colony Optimization Based on MapReduce

WANG Zhao-yuan LI Tian-rui YI Xiu-wen
(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China)
(Key Laboratory of Cloud Computing and Intelligent Technology of Sichuan Province, Chengdu 610031, China)

Abstract This paper discussed several parallel ways of ant colony optimization and their application scenarios as well as the feasibility of combination with the cloud computing framework MapReduce. Ant colony optimization with the local search feature was abstracted into several components, and then several interfaces based on MapReduce were built to implement each component. It provides a flexible, scalable solution for ant colony optimization with the local search feature to implement parallelism with the MapReduce framework. Finally, simulation results validate the proposed approach on the traveling salesman problem.

Keywords Ant colony optimization, MapReduce, Hadoop, Traveling salesman problem

1 引言

昆虫学科学家发现由群居性生物产生出来的一种集体行为虽然是自发的、简单的,但是却可以解决复杂的问题。这种集群智能引起了计算机科学家们的注意,其中1991年意大利学者Dorigo等人提出模拟蚁群行为的模拟算法——蚁群算法[1]。蚁群在觅食时,蚂蚁会在巢穴-食物路径上留下一种称之为信息素的物质,群体中其他蚂蚁会根据该物质选择路径,较短的路径容易积累更多的信息素,也使得更多的蚂蚁选择该路径,最终几乎所有的蚂蚁都集中在路程最短的路径上。Dorigo等人对该行为进行抽象、总结,最终提出蚁群算法。该算法具有启发式搜索、信息正反馈和分布式计算等特点[2]。但蚁群算法由于存在易早熟收敛的问题,求解质量不高,近年来出现了很多对蚁群算法进行优化的研究,提高了算法解的质量,这些优化后的算法统称为蚁群优化算法。蚁群优化算法应用于旅行商(Traveling Salesman Problem, TSP)问题、调度和二次分配等诸多经典的组合优化类问题,即已取得了较好的效果[3]。

随着大数据时代的来临,数据的规模数量级增长,传统的单机处理方式在处理大规模增长的数据集时,即使通过增加或者更换CPU、内存、GPU等硬件以扩展计算能力,也难以满足实际的需求。蚁群算法也存在同样的问题,即当数据集增大到一定程度后,传统单机方式求解所需的空间和时间代价都是巨大的,难以满足人们的需求。云计算作为一个新兴的并行处理技术,融合了网格计算、分布式计算和并行计算等特点,在大数据计算和网络存储方面具有卓越的性能表现[4],因而在云计算平台下优化算法成为一种可行、可靠的解决方案。

新兴的MapReduce并行编程框架定位于处理大数据,将大数据处理任务划分为Mapper(映射)阶段和Reducer(规约)阶段。MapReduce隐藏了冗繁的底层分布式计算实现的细节,具有良好的扩展性和容错性;而且给用户提供了简单易用的功能接口以及和性能相关的调优参数[5]。因此,MapReduce成为目前进行大规模并行处理的主流方法之一。本文旨在将蚁群算法与MapReduce并行计算框架融合以实现并行蚁群算法,从而解决大数据情况下蚁群算法的求解空间和时间代价过高的问题。

本文的主要贡献点如下:

到稿日期:2013-09-24 返修日期:2014-01-03 本文受国家自然基金项目:基于粒计算的动态更新知识理论与高效算法研究(61175047)资助。
王诏远(1989—),男,硕士生,主要研究方向为数据挖掘和云计算等,E-mail:wang_zhaoyuan@foxmail.com;李天瑞(1969—),男,博士后,教授,博士生导师,主要研究方向为智能信息处理、数据挖掘和云计算等;易修文(1991—),男,硕士生,主要研究方向为数据挖掘和云计算等。

(1)探讨了3种蚁群算法并行方式及这3种并行方式与MapReduce分布式编程框架相结合的可行性与适用场景。

(2)将蚁群优化算法中加入局部搜索这一大类蚁群优化算法抽象为各个组件与MapReduce框架接口对应实现,特别是将蚁群优化算法的优化部分剥离出来抽象成一个独立的组件,与MapReduce的Map部分结合,形成可插拔的接口,只要符合输入输出格式,优化算法可以随意替换,使实现方式灵活而有效。本文将采用遗传算法对传统蚁群算法进行优化来对接口的使用进行举例说明,验证了上述解决方案的合理性,为加入局部搜索类蚁群优化算法在MapReduce框架下实现并行提供一种通用的解决方案。

本文第2节介绍相关工作;第3节讨论了蚁群算法的几种并行方式以及与MapReduce编程框架结合的可行性与使用场景;第4节介绍了蚁群优化算法在MapReduce框架下的设计与实现;第5节是实验设计与结果分析。

2 相关工作

2.1节介绍蚁群算法;2.2节讨论蚁群优化算法;2.3节介绍MapReduce编程框架与基于MapReduce实现蚁群算法的相关工作。

2.1 蚁群算法

本小节以求解TSP问题为例介绍传统蚁群算法[6-9],其搜索过程大致如下:

步骤1 初始化,对蚂蚁进行初始化,将m只蚂蚁随机放置于n个城市中,并初始化各条路径上的信息素值,一般选择:$\tau_{ij}(0)=\tau_0$为信息素初值,其中$\tau_0=m/L_m$,L_m是每只蚂蚁由所在城市出发根据贪心算法构造的路径长度。

步骤2 初始化之后,蚂蚁$k(k=1,2,\cdots,m)$按照随机比例规则选择下一步要转移的城市,其选择概率为:

$$p_{ij}^k(t)=\begin{cases}\dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{s\in allowed_k}\tau_{is}^{\alpha}(t)\eta_{is}^{\beta}(t)}, & j\in allowed_k\\ 0, & \text{otherwise}\end{cases} \tag{1}$$

式中,τ_{ij}为边(i,j)上的信息素,$\eta_{ij}=\dfrac{1}{d_{ij}}$,$d_{ij}$为城市$i$到城市$j$的距离,$\eta_{ij}$反映由城市$i$转移到城市$j$的期望程度。$allowed_k$为蚂蚁$k$下一步可以访问城市的集合,蚂蚁已访问过的城市将不出现在该集合中。α为信息启发式因子,反映了蚂蚁在运动过程中信息素的相对重要性,该值越大,则蚂蚁越倾向选择其他蚂蚁选择过的路径。β为期望启发式因子,反映了蚂蚁在运动过程中启发信息的相对重要性,该值越大,蚂蚁在寻找路径过程中越倾向于贪心算法进行寻路径。

禁忌表$tabu_k$记录蚂蚁已经走过的路线,其目的是不让蚂蚁选择已经访问过的路径。蚂蚁k经过t时刻走过一条路径,禁忌表即进行记录,记录后不允许蚂蚁在本次循环中再访问该路径。当蚂蚁都完成一次周游时,在进入下次循环前,将禁忌表清空,开放所有路径的访问权限。

步骤3 蚂蚁k完成一次循环,如果所有蚂蚁都完成步骤2,跳到步骤4,否则选择下一只蚂蚁,跳到步骤4。

步骤4 所有蚂蚁完成一次循环,找出其中的最佳路径,对最佳路径上的信息素进行更新,更新参照式(2)与式(3)进行:

$$\tau(r,s)=(1-\alpha)\tau(r,s)+\alpha\Delta\tau(r,s) \tag{2}$$

$$\Delta\tau(r,s)=\begin{cases}(L_{gb})^{-1}, & \text{if}(r,s)\in\text{global\_best\_tour}\\ 0, & \text{otherwise}\end{cases} \tag{3}$$

其中,α为信息素挥发参数,$0<\alpha<1$;L_{gb}为当前全局最短路径。

步骤5 检验是否满足退出迭代次数,满足则退出,不满足则迭代次数加1,跳到步骤1,继续运行直到满足迭代次数。

2.2 蚁群优化算法

由于传统蚁群算法易早熟收敛,导致解质量不高。近年来出现了很多对蚁群算法的优化的研究,我们将这些优化蚁群算法的方法统称为蚁群优化算法。这些算法大致可以分为两类:

(1)对信息素管理进行优化,如蚁群系统[9]、最大-最小蚂蚁系统[10]等;

(2)加入局部搜索算法,如使用模拟退火[11]、遗传变异[12]、模糊控制[13]等局部搜索算法。

第2类算法可以抽象为一个共同的流程,如图1所示,即这类蚁群优化算法的流程图。在传统蚁群算法的基础上加入了局部搜索步骤,经过该优化步骤之后,可以有效地防止早熟收敛,改善解的质量。本文针对第2类蚁群优化算法,基于MapReduce编程框架实现提出一种通用的解决方案。当然对于第1类算法,如该优化算法未增加蚂蚁构造路径过程中对信息素的修改,也可使用本文框架实现。

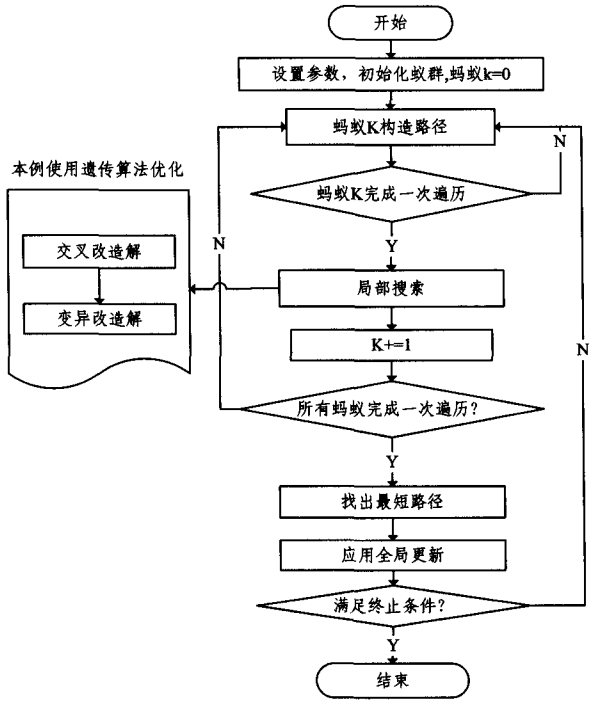

图1 蚁群优化算法流程图

本文的局部搜索将使用遗传算法[14]的思想对蚁群算法的解进行优化,对该优化过程进行举例说明,以验证本设计的可行性。遗传算法是1975年John Holland教授根据达尔文的生物进化论和孟德尔的遗传变异理论提出的仿生类优化算法。图1将优化过程由注释框中步骤替代,即为经过遗传算法融合和改造后的蚁群算法流程图。本文交叉算子采用随机的双点交叉;变异算子采用随机的双点变异。

2.3 MapReduce

MapReduce[15]模型是Google实验室提出的分布式并行编程模型或框架,可方便地编写应用程序运行在上千个节点(异构/同构)的大型集群上,提供一个可靠的、容错的方式并行处理大量的数据集,成为云计算平台主流的并行数据处理

模型。Apache 开源社区的 Hadoop 项目用 Java 语言实现了该模型并开放了源代码，是目前学术界和工业界事实上的大数据并行处理标准。

MapReduce 的编程原理是基于“分而治之”的思想。MapReduce 将复杂的并行计算过程高度地抽象为两个函数：Mapper(映射)和 Reducer(规约)，简单地说就是“任务的分解与结果的汇总”。一个 MapReduce 作业通常会把输入的数据设置成独立的数据块，以完全并行的方式处理 Mapper 任务。该框架对 Mapper 任务的输出进行排序，然后被输入到 Reducer 任务，MapReduce 的任务数据流图如图 2[16] 所示。同时输入和输出的工作通常被存储在一个文件系统。整个框架负责任务的调度、监控，以及重新执行失败的任务。

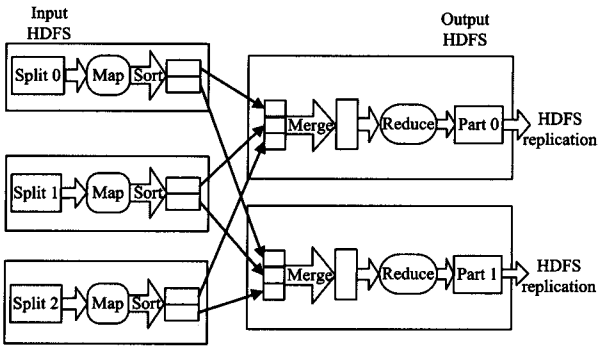

图 2 MapReduce 运行图[16]

目前关于蚁群算法结合 MapReduce 实现的研究方面也有一些成果，例如文献[17]介绍了基本的蚁群算法在Map-Reduce 框架下的一种实现方式；文献[18]针对蚁群算法解决 0-1 背包问题在 MapReduce 框架下的实现，并在实现过程中采用轮盘赌、交叉和变异等思想对蚁群算法进行优化；文献[19]针对蚁群算法解决 TSP 问题在 MapReduce 框架下的实现，并使用了模拟退火的思想对蚁群算法进行优化，都取得了不错的效果。本文不具体针对哪一类特定的问题、哪种特定的优化方式，而是为相比基本蚁群算法更为复杂且更有应用价值的蚁群优化算法在 MapReduce 框架下的实现提供了一种通用的解决方案，其中优化过程与 MapReduce 接口对应，用户可以针对特定的问题定制特定的优化方式。

3 蚁群算法并行的几种方式及其适用场景

本小节首先介绍几种蚁群算法并行化的可能的方式[20]，并分析其在 MapReduce 框架下的可行性，以及本文的选择与理由。

本算法的并行方式大致可分为 3 种：

方式 1 并行独立蚁群，即蚁群和蚁群之间的并行。

方式 2 并行交互蚁群，即将一个蚁群分为几个部分，并在这些部分之间进行并行。

方式 3 并行蚂蚁，即对蚁群中各只蚂蚁进行并行。

在传统的多线程并行编程中，上述 3 种方式都可以实现，但是传统方式受限于单机的硬件计算能力，而 MapReduce 是通过增加计算节点数量提高计算能力，更经济可行。在MapReduce框架下，由于该编程框架没有全局变量，Mapper 任务或 Reducer 任务间不存在任何依赖，无交互地在不同的数据切片上独立执行。但是在蚁群算法中存在信息素这一全局公共变量，并需要对该公共变量实现资源抢占式访问，所以信息素更新无法并行处理。即方式 1 无法实现。

方式 2 与方式 3 比较类似，方式 2 的极端方式即为方式 3，即将蚁群分为 N(N=蚂蚁数量)个部分，这时方式 2 等同于方式 3。这两种方式都可行，因为二者都是同一蚁群内的并行，并不需要在 Mapper 任务或是 Reducer 任务过程中发生信息素交互，而是在 MapReduce 结束后进行信息素更新。

本文选择了第 3 种并行方式进行实现，原因是实验资源有限，如果选择第 2 种方式，对单个 Mapper 任务执行需要的计算空间更大，而实验资源中的计算机性能较差，为了减轻计算机压力选择了第 3 种方式进行并行。由于 MapReduce 开启 Mapper 和 Reducer 任务会产生作业启动开销(包括作业分发、输入划分、任务划分等一些初始化操作，根据实验数据分析这部分的代价占到整个作业执行代价的 7%～10%左右)[21]，在实际应用中根据集群组成和计算能力，在 Mapper 任务数量和 Mapper 任务中的蚂蚁数量之间进行平衡，合理地选择方式 2 或方式 3，从而提高生产效率。本文第 4 节将对蚁群算法第 3 种并行方式的设计与实现进行详细介绍。

4 基于 MapReduce 的蚁群优化算法设计

本节将局部搜索类蚁群优化算法拆分、抽象为几个部分，每个部分与 MapReduce 接口相对应，并定义接口间的输入、输出规则。将算法接口化的好处是易于维护与扩展，特别是局部搜索部分，本文采用遗传算法思想进行优化，如使用其他算法如模拟退火、模糊控制等算法进行优化，只需改动或替换相应的接口而不需对其他部分进行修改。

本文将蚁群优化算法抽象为 3 个部分：蚁群构造路径、局部搜索、全局更新和寻找最优路径。在 MapReduce 框架中蚁群构造路径对应 InputFormat 接口；局部搜索对应 Map 接口；全局更新和最优路径寻找对应 Reduce 接口。

根据第 3 节中并行蚂蚁的方式，每一个 MapReduce Job 即一次一个蚁群寻径过程，每一个 Mapper 任务为一只蚂蚁构建路径过程，有多少蚂蚁就构建多少 Mapper 任务。其中 InputFormat 接口负责蚂蚁构造路径，将构造好的路径传给 Map 接口，在 Map 接口中进行优化，最后 Reducer 任务中 Reduce 接口获得所有蚂蚁所构建的路径并找出最短路径。蚁群的迭代即为 MapReduce Job 的迭代。前文提到，MapReduce编程框架在开启 Mapper 或者 Reducer 任务时有一定的自身开销，为了降低这种开销，本文的设计思路是使用相对少量的蚂蚁，但每只蚂蚁承担一定量的工作——不超出单机的计算能力范围，即少开启 Mapper 任务，但增大每个 Mapper 任务中的计算量。本文使用遗传算法优化对优化过程进行举例。优化过程在 Map 接口中实现，其中交叉和变异都要循环进行多次，最后输出最优值。其中循环次数 T 对实验结果有较大的影响，若循环次数过多，将导致搜索空间、时间的浪费；若循环次数太少，可能导致搜索不重复，无法找到更优解。

定义 1 搜索次数 T 定义为：

$$T=(N/2)^2/n$$

其中，n 为蚂蚁的个数，也就是 Mapper 任务的个数。一条 N 个元素的路径组合可进行的交换次数为 $(N/2)^2$，这样定义 T，保证每个任务有充分的循环交换过程，合理利用了空间与时间。

定义接口之间的输入和输出规则：在 MapReduce 框架中接口之间传输〈key，value〉对，InputFormat 接口负责处理原始输入数据并将处理后的〈key，value〉对输入给 Map 接口，

Map 接口将该键值对处理后生成新的〈key，value〉对输入给 Reduce 接口，Reduce 接口处理输入键值对后输出最终的结果〈key，value〉对。所以定义接口之间的输入输出规则就是定义〈key，value〉对。本文的〈key，value〉对的设计为：Map 输入的〈key，value〉对为〈当前蚂蚁找到的最短路径值，最短路径集〉；Map 输出的〈key，value〉对为〈经过优化的最短路径值，最短路径集〉；Reduce 输入的〈key，value〉对为 Map 输出的〈key，value〉对经过排序和合并后得到的〈路径长度，该长度的路径集合的集合〉；Reduce 输出的〈key，value〉对为〈全局最短路径长度，最短路径集〉。

5 实验设计与结果分析

本节首先介绍实验环境以及实验目标，随后列出实验结果，并对结果进行分析。最后讨论本设计的并行性能。

5.1 实验环境与目标

(1)节点信息

节点属性信息详见表 1。

表 1 节点信息表

属性名	属性值
节点个数	12
Hadoop 版本	1.0.4
Java 版本	1.7.0
节点 CPU	Intel(R) Xeon(TM) CPU 2.80GHz * 4
节点内存	2G

其中 Hadoop 集群中有一个主节点，其余为从节点。主节点没有 Mapper 任务槽，每个从节点的 Mapper 任务槽数量为 1。

(2)实验测试数据集

本实验共测试了文献[22]中 3 组基于欧氏距离的数据集，分别为 EIL51 数据集、ch130 数据集和 TSP225 数据集。

(3)实验参数设定

根据文献[23]，本文选取 $\alpha=0.999$，$\beta=5.0$，$\rho=1.0$ 作为本实验蚁群算法的参数组合。选择的蚂蚁数量为 10，即 10 个 Mapper 任务。

(4)实验目标

本实验需完成两点目标：

a. 针对不同数据集分别求解实验，验证设计的有效性；

b. 验证同一数据集云计算环境下节点的增减对结果、运行时间的影响，说明设计具有较好的可扩展性。

5.2 实验结果与分析

本实验首先运行局部搜索类蚁群优化算法求解 3 组 TSP 数据集，求解得出结果。随后将优化接口拔除，变为蚁群算法求解 3 组 TSP 数据。表 2 列出了本实验蚁群优化算法解决各 TSP 数据集的最优解和平均解，运行蚁群算法的最优解和平均解以及当前数据集的理论最优解。

表 2 运行结果表

数据集	蚁群优化算法		蚁群算法		数据集
	最优解	平均解	最优解	平均解	最优解
EIL51	431.91599	440.73770	503.73965	532.63043	426
ch130	6378.47776	6475.00378	7668.77128	8098.1366	6110
TSP225	4266.76953	4414.62796	5451.75592	5557.4115	3916

从表 2 可以得出由于蚁群优化算法是启发式算法，通过实验数据与理论最优值的对比可以验证本设计的有效性，而且优化接口的方法对抑制蚁群算法的早熟收敛起到了明显的效果。但蚁群优化算法求解结果与理论最优解仍有一定的差距，我们可以继续优化该局部搜索方法，即遗传算法，或者是更换接口、局部搜索算法。

图 3－图 5 为由蚁群优化算法运行 3 组数据集分别在不同节点下得到结果集绘制而成的箱图。横坐标为运行实验集群的节点数，纵坐标为实验结果。从 3 幅箱图中可以看出，节点数为 3－11 的过程中，箱子显下降趋势，说明在这个过程中随着计算节点的增加，结果集中解的整体质量逐渐提高，并到 11 个节点之后趋于稳定。这说明了在一定条件下节点的增多可以在一定程度上提升解的整体质量，具体的原因我们在 5.3 小节进行分析。

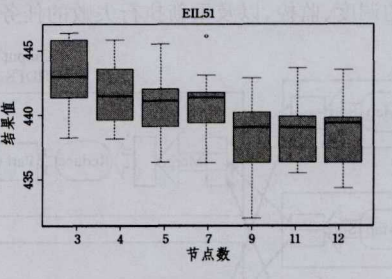

图 3 EIL51 结果箱图

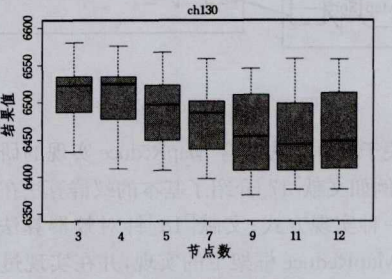

图 4 ch130 结果箱图

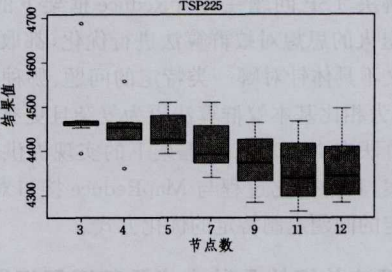

图 5 TSP225 结果箱图

5.3 并行性能分析

表 3 列出了本实验各个数据集在不同节点下所需要的运行时间，其中需要特别指出，本运行时间是指当实验结果收敛到一个较为合理值时的时间：EIL51 为收敛到 450 的时间；ch130 为收敛到 6500 的时间；TSP225 为收敛到 4500 的时间。另外表 3 中一个节点是指在 Hadoop 单机伪分布条件下的运行时间。

表 3 运行时间表

节点数	EIL51	ch130	TSP225
1	70 * 72s/Job=5040s	130 * 75s/Job=9750s	140 * 90s/job=12600s
3	60 * 67s/Job=4020s	100 * 70s/Job=7000s	120 * 80s/job=9600s
4	50 * 60s/Job=3000s	85 * 65s/Job=5525s	100 * 70s/job=7000s
5	40 * 55s/Job=2200s	70 * 58s/Job=4060s	80 * 62s/job=4960s
7	37 * 53s/Job=1961s	64 * 55s/Job=3520s	75 * 58s/job=4350s
9	35 * 50s/Job=1750s	62 * 53s/Job=3286s	73 * 55s/Job=4015s
11	35 * 36s/Job=1260s	62 * 37s/Job=2294s	72 * 39s/Job=2808s
12	35 * 36s/Job=1260s	62 * 38s/Job=2356s	72 * 39s/Job=2808s

本设计基于 MapReduce 框架实现，即通过并行实现来节约计算时间，同时可以解决单机条件下无法解决的大数据集。以下首先介绍衡量并行算法的指标。

Speedup[24]是衡量同一数据集情况下增加节点时并行算法的表现。

$$Speedup(p)=T_1/T_p, p=1,2,\cdots \quad (4)$$

式(4)描述：固定处理任务量不变，依次增加计算节点的数量 p(如从 1 个节点增加到 12 个节点)。T_1 为单个节点执行任务的时间，T_p 为 p 个节点执行任务的时间。理想情况下的 Speedup 与直线 $y=x$ 重合。

图 6 展示了我们算法的 Speedup 结果。结合实验分析：由于实验共有 10 个 Mapper 任务，且每个从节点的 Mapper 任务槽为 1 个，因此在节点数为 3 时可以同时运行 2 个 Mapper 任务，当节点数为 4 时可以运行 3 个 Mapper 任务，以此类推，当节点数达到 11 个时可以同时运行 10 个 Mapper 任务。图中也反映出当节点数从 1 到 5 时 Speedup 增长较快，因为单个节点需要运行的任务从 10 个减为 2 个；而从 5 到 11 个节点，Speedup 增长比较缓慢，因为这个阶段单个节点运行的 Mapper 数量为 2 到 1。当节点数达到 11 时每个节点运行的任务只有 1 个，此时 Speedup 值达到最大；而节点数大于 11 个以后每个节点运行的任务有 1 个或 0 个，多余的节点反而会造成额外的系统开销(任务调度、网络通信等)，此时可以通过增加蚂蚁的数量即增加 Mapper 任务数量同时减少迭代次数来优化运行时间。同时任务分配的分析也解释了 5.2 小节中在一定情况下解的整体质量会随着节点数的增加而提升，说明了单个节点任务数量对解的整体质量会有一定的影响。因此在仿真实验时，我们需要根据数据的大小来合理分配节点个数，在达到最优解和最优时间的同时不浪费资源。从图 6 中可以看出本实验 Speedup 值达到了线性的结果，可以预期，在处理更大规模的数据时，Speedup 性能会进一步提升，说明了本文提出的方法具有较好的可扩展性。

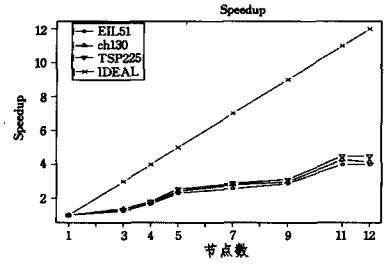

图 6 Speedup 图

结束语 本文针对蚁群优化算法在 MapReduce 并行编程框架下的实现进行了深入的研究。本文首先探讨了传统蚁群算法的 3 种并行方式，以及这 3 种并行方式结合 MapReduce 编程框架的可行性与适用场景；并提出将局部搜索类蚁群优化算法接口化，特别是将优化部分剥离成接口，易于调试、维护和扩展。本文使用遗传算法的思想优化蚁群算法作为优化部分的范例实现，由仿真实验结果可以看出本文的设计是正确可行的，遗传算法在改善蚁群早熟收敛的缺陷上起到了一定的作用，并且具有很好的扩展性，为局部搜索类蚁群优化算法的 MapReduce 下实现提供了通用的解决方案，使该算法突破了传统单机处理方式的瓶颈。MapReduce 编程框架节点数对结果的影响、交互蚁群并行同并行蚂蚁之间的对比以及优化接口的扩展是我们后续工作的主要研究内容。

参 考 文 献

[1] Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies[C]//Proceeding of ECAL91. Paris, France, 1991

[2] 段海滨. 蚁群算法原理及其应用[M]. 北京：科学出版社，2005

[3] 杨学峰. 蚁群算法求解 TSP 问题的研究[D]. 吉林：吉林大学，2010

[4] 张建勋，古志民，郑超. 云计算研究进展综述[J]. 计算机应用研究，2010，27(2)：429-433

[5] 和亮，冯登国，王蕊，等. 基于 MapReduce 的大规模在线社交网络蠕虫仿真[J]. 软件学报，2013(7)：1666-1682

[6] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 1996, 26(1): 29-41

[7] Dorigo M. Optimization, learning and natural algorithms [D]. Politecnico di Milano, Italy, 1992

[8] Dorigo M, Maniezzo V, Colorni A. Positive feedback as a search strategy[R]. Technical report 91-016. Politecnico di Milano, Milano, Italy, 1991

[9] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem[J]. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 53-66

[10] Stutzle T, Hoos H H. MAX-MIN ant system[J]. Future Generations Computer Systems, 2000, 16(8): 889-914

[11] Chen S, Chien C. Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques[J]. Expert Systems with Applications, 2011, 38(12): 14439-14450

[12] Chen S, Chien C. Parallelized genetic ant colony systems for solving the traveling salesman problem[J]. Expert Systems with Applications, 2011, 38(4): 3873-3883

[13] Juang C, Lu C, Lo C, et al. Ant colony optimization algorithm for fuzzy controller design and its FPGA implementation[J]. IEEE Transactions on Industrial Electronics, 2008, 55(3): 1453-1462

[14] Goldberg D E, Holland J H. Genetic algorithms and machine learning[J]. Machine Learning, 1988, 3(2): 95-99

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

[16] White T. Hadoop: the definitive guide[M]. O'Reilly, 2012

[17] 夏卫雷，王立松. 基于 MapReduce 的并行蚁群算法研究与实现[J]. 电子科技，2013，26(2)：146-149

[18] 王会颖，倪志伟，吴昊. 求解多维背包问题的 MapReduce 蚁群优化算法[J]. 计算机工程，2013(4)：248-253

[19] 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

[20] 李士勇，陈永强，李研. 蚁群算法及其应用[M]. 哈尔滨：哈尔滨工业大学出版社，2004

[21] 潘巍，李战怀，伍赛，等. 基于消息传递机制的 MapReduce 图算法研究[J]. 计算机学报，2011(10)：1768-1784

[22] Reinelt G. TSPLIB—A traveling salesman problem library[J]. ORSA Journal on Computing, 1991, 3(4): 376-384

[23] Gaertner D, Clark K L. On Optimal Parameters for Ant Colony Optimization Algorithms[C]//Proceedings of the 2005 International Conference on Artificial Intelligence(ICAI 2005). CSREA Press, 2005: 83-89

[24] Zhang J, Li T, Ruan D, et al. A parallel method for computing rough set approximations[J]. Information Sciences, 2012, 194: 209-223