# 摘 要

基于分解的多目标进化算法（MOEA/D）是2007年提出的一种将传统的数学规划方法与进化算法相结合，求解多目标优化问题的新颖算法。Spark是发源于美国加州大学伯克利分校AMPLab的大数据分析平台，它立足于内存计算，从多迭代批量处理出发，兼顾数流处理、数据仓库和图计算等多种范式，是大数据系统领域的全栈式计算平台。由于大数据行业应用需求日益增长，未来越来越多的研究和应用领域将需要使用大数据并行计算技术，大数据技术将渗透到每个涉及到大规模数据和复杂计算的应用领域，进化算法具有与生俱来的高度并行性，所以可以考虑将MOEA/D算法部署到云平台上以提高算法的执行效率。

本文通过对MOEA/D的深入探索和研究，将MOEA/D算法和云计算技术结合，通过在Spark平台上并行设计与实现了MOEA/D算法并行化方案，包括以下几个方面：1. 邻居群内部并行产生解；2. 并行构造非支配解集。由于非支配解集构造的过程中，非支配个体在解集中的分布情况不同，算法的复杂度也不同，因此本文实现了对排除法和快速排序法两种非支配解集构造方法的并行化。当解集中非支配个体较少时，使用快速排序法。当非支配个体在解集中靠前的位置时，选取排除法。3.并行化MOEA/D算法的评价指标。并在Spark平台上进行试验，在不同数据量和规模的集群里对MOEA/D算法的并行化方法进行测试。最后，通过测试结果来分析MOEA/D在内存并行框架中执行带来的效率提升，从而为多目标优化问题提供更加理想的解决方案。

关键词: 云计算； Spark； MOEA/D； 并行计算； 进化算法

# Abstract

As a combination of mathematical programming methods and evolutionary algorithms, MOEA/D proposed in 2007 is a novel decomposition-based algorithm. Spark is originated AMPLab big data analytics platform in the University of California, Berkeley, which is based on in-memory computing and multiple iterations batch processing. Spark is usually used as the full stack computing platform in the field of big data systems, considering data stream processing, data storage and other computing paradigm. Due to the growing demand for big data business applications, more and more research and future applications will require the use of large data-parallel computing. Big Data technologies will permeate every application field in large-scale data and complex calculations, and evolutionary algorithm has strong parallelism inherently. So the method that deploys MOEA/D algorithm to the cloud platform can be considered to improve the efficiency of the algorithm.

This thesis combines MOEA/D algorithm with cloud computing technologies to improve the efficiency of classification by implementing the parallel MOEA/D algorithm based on Spark. The parallel program of the algorithm includes the following aspects: 1. The solutions are generated by the internal of the neighborhood parallel. 2. The nondominated solution sets are constructed in parallel. While constructing the nondominated solution sets, the complexity of the algorithm is different since the distribution of the nondominated individuals differ in the solution sets. Therefore, this thesis implements the parallelization of the two ways, that is, elimination and quick sort, to construct the nondominated solution sets. The quick sort is used when the nondominated individuals of the solution sets are less while the elimination is selected when the nondominated individuals are at the front of the solution sets. 3. Making the evaluation of the MOEA/D algorithm in parallel. In addition, the thesis also carries experiments on different amounts of nodes using different amounts of data to test the MOEA/D parallel algorithm. Finally, the thesis analyzes the efficiency improvement of MOEA/D in the memory parallel framework. Thus it may provide more ideal solution for the multi-objective optimization problems.

**Keywords:** Cloud Computing; Spark; MOEA/D; Parallel Computing; Evolutionary Algorithms

# 目 录

[摘 要 III](#_Toc422418796)

[ABSTRACT IV](#_Toc422418797)

[第1章 绪论 1](#_Toc422418798)

[1.1 课题背景与意义 1](#_Toc422418799)

[1.2 论文的主要方法 2](#_Toc422418800)

[1.3 论文的研究内容 3](#_Toc422418801)

[1.4 论文的章节安排 3](#_Toc422418802)

[1.5 本章小结 3](#_Toc422418803)

[第2章 预备知识 4](#_Toc422418804)

[2.1 Scala简介 4](#_Toc422418805)

[2.1.1 Scala背景介绍 4](#_Toc422418806)

[2.1.2 Scala函数式编程 4](#_Toc422418807)

[2.1.2 Scala面向对象特性 6](#_Toc422418808)

[2.2 Spark简介 6](#_Toc422418809)

[2.2.1 Spark背景介绍 6](#_Toc422418810)

[2.2.2 Spark生态系统BDAS 8](#_Toc422418811)

[2.2.3 Spark计算模型 9](#_Toc422418812)

[2.3 MOEA/D算法介绍 11](#_Toc422418813)

[2.3.1 多目标优化问题的产生背景 11](#_Toc422418814)

[2.3.2 多目标优化问题的产数学模型以及相关定义 12](#_Toc422418815)

[2.3.3 多目标优化问题的研究现状 13](#_Toc422418816)

[2.3.4 基于分解的多目标进化算法（MOEA/D） 14](#_Toc422418817)

[2.4 多目标0-1背包问题介绍 17](#_Toc422418818)

[2.4.1多目标0-1背包问题背景 17](#_Toc422418819)

[2.4.2多目标0-1背包问题的定义 17](#_Toc422418820)

[2.4.3贪心修复策略 17](#_Toc422418821)

[2.4.4参数设置 18](#_Toc422418822)

[2.5 本章小结 18](#_Toc422418823)

[第3章 MOEA/D算法的并行化设计 20](#_Toc422418824)

[3.1 MOEA/D并行方案 20](#_Toc422418825)

[3.2 MOEA/D并行设计与实现 22](#_Toc422418826)

[3.3 运用MOEA/D求解多目标0-1背包问题 32](#_Toc422418827)

[3.4 本章小结 34](#_Toc422418828)

[第4章 实验及结果分析 35](#_Toc422418829)

[4.1 Spark集群的搭建 35](#_Toc422418830)

[4.1.1 实验环境 35](#_Toc422418831)

[4.1.2 Spark集群搭建的环境准备 36](#_Toc422418832)

[4.1.3 安装配置SSH 37](#_Toc422418833)

[4.1.4 Spark平台的安装与配置 37](#_Toc422418834)

[4.1.5 MOEA/D在Spark上的运行 38](#_Toc422418835)

[4.2 实验说明 42](#_Toc422418836)

[4.3 基于Spark的MOEA/D算法和串行MOEA/D算法对比实验 43](#_Toc422418837)

[4.4 基于Spark平台的MOEA/D算法性能测试 44](#_Toc422418838)

[4.5 实验结果分析 47](#_Toc422418839)

[4.6 本章小结 47](#_Toc422418840)

[结 论 48](#_Toc422418841)

[致 谢 49](#_Toc422418842)

[参考文献 50](#_Toc422418843)

# 第1章 绪论

## 1.1 课题背景与意义

随着科学技术的发展，信息技术和计算机的快速发展和广范应用，各个行业应用系统产生的数据呈爆炸性增长。数据量达到数百TB甚至数十至数百PB规模的大数据，已远远超出了现有传统的计算技术和信息系统的处理能力，因此，寻求有效的大数据处理技术、方法和手段已经成为现实世界的迫切需求。

大数据是大数据是指无法在一定时间内用传统数据库软件工具对其内容进行抓取、管理和处理的数据集合。云计算是继1980年代大型计算机到客户端-服务器的大转变之后的又一种巨变。云计算（Cloud Computing）是分布式计算（Distributed Computing）、（并行计算Parallel Computing）、效用计算（Utility Computing）、 网络存储（Network Storage Technologies）、虚拟化（Virtualization）、负载均衡（Load Balance）、热备份冗余（High Available）等传统计算机和网络技术发展融合的产物。大数据时代的崛起促进了云计算技术的发展。

大数据的产生虽然给我们带来了巨大技术挑战，但同时也带来了巨大的技术创新和商业机遇。不断积累的大数据包含着很多在小数据量时不具备的深度知识和价值，大数据分析挖掘将能为行业/企业带来巨大的商业价值，实现各种高附加值的增值服务，进一步提升行业/企业的经济效益和社会效益。由于大数据隐含着巨大的深度价值，美国政府认为大数据是“未来的新石油”，对未来的科技与经济发展将带来深远影响。因此，在未来，一个国家拥有数据的规模和运用数据的能力将成为综合国力的重要组成部分，对数据的占有、控制和运用也将成为国家间和企业间新的争夺焦点。

大数据的价值是与大数据的容量和种类密切相关得到的。一般看来，数据容量越大，种类越多，信息量越大，获得知识就越多，能够发挥的价值就越大。但这依赖于大数据处理的手段和工具，否则可能造成数据垃圾和信息过剩，失去数据的利用价值。

大数据并行计算技术将应用越来越多的应用和研究领域，每个涉及到复杂计算和大规模数据的应用领域都将会使用大数据技术。不仅如此，以大数据处理为中心的计算技术将对传统计算技术产生革命性的影响，广泛影响程序设计技术和方法、计算机体系结构、数据库、操作系统、编译技术、软件工程技术、多媒体信息处理技术、人工智能以及其他计算机应用技术，并与传统计算技术相互结合产生很多新的研究热点和课题。Spark[1]是一个基于内存计算的大数据并行计算框架，是MapReduce的替代方案，而且兼容HDFS、Hive等分布式存储层，可融入Hadoop生态系统，以弥补MapReduce的不足。

MOEA/D[2]是一种处理多目标优化问题的新颖算法，其创新性地将传统的数学规划方法与进化算法相结合。它将一个多目标优化问题分解为一定数量的单目标优化子问题，然后用进化算法同时求解这些单目标子问题。与传统的多目标进化算法相比，MOEA/D利用子问题间的协同进化机制，不需要重复优化单目标子问题，因而相对来说计算量较小。进化算法具有与生俱来的高度并行性，所以可以考虑将MOEA/D算法部署到云平台上以提高算法的执行效率。本文将MOEA/D算法和云计算技术结合，通过在Spark平台上并行设计与实现MOEA/D算法并行化方案，来提高算法的执行效率。

目前多目标优化问题广泛地应用于控制科学、生物制药、网络交通、集成电路设计、图像处理、分子生物学、财政经济模型、核能和机械设计等领域。但随着这些领域应用需求日益增长，已远远超出了现有传统的计算技术和信息系统的处理能力，因此，将Spark云平台与MOEA/D算法的结合对于多目标优化问题研究具有重要的现实意义。

本文拟基于Spark平台设计MOEA/D并行算法，通过实验测试MOEA/D在内存并行框架中执行带来的效率提升，从而为多目标优化问题提供更加理想的解决方案。

## 1.2 论文的主要方法

本文使用的云计算平台是Spark平台，由于多目标优化问题领域应用需求日益增长，产生数据爆炸式的增长，已远远超出了现有传统的计算技术和信息系统的处理能力。所以本文拟通过运用Spark云计算平台，重新设计MOEA/D算法的实现，将其实现并行化，解决计算量庞大难题，并且对Spark平台上实现的MOEA/D算法进行评价以及结果分析。

算法的研究方法，主要是通过阅读MOEA/D算法的论文、多目标优化问题相关的书籍和并行化编程的书籍。

算法的并行方法，在设计并行的 MOEA/D算法之前，先完成了单机的MOEA/D算法的实现，然后根据算法特点选择并行策略和调整并行算法的一些参数。

解决Spark平台搭建和学习和编程过程中遇到问题的方法，主要是通过请教他人、上网查找官方文档的方式解决。

实验结果分析的方法，主要是从Speedup（加速比）方面来衡量MOEA/D算法的并行效果。

在完成本论文的过程中，主要是参加云计算实验室讨论会和查阅云计算技术和实现并行化编程有关的书籍资料。

## 论文的研究内容

本论文的研究内容包括如下四部分。

第一，Spark 平台相关知识，详细介绍以JVM为目标环境将面向对象和函数式编程有机结合起来的Scala语言，详细介绍Spark计算模型MOEA/D算法的相关概念，并详细介绍了在Spark平台上的并行方案和参数设置。

第二，本论文中针对运筹学中典型的优化难题多目标0-1背包问题，使用MOEA/D算法提供解决方案，论文介绍了多目标背包问题以及其应用领域。

第三，在集群上Spark平台的搭建方法，并给出MOEA/D算法求解多目标0-1背包问题的运行方法。

第四，设计实验，测试算法的运行，分析了MOEA/D求解多目标0-1背包问题的数值实验的实验结果，给出性能分析。

## 1.4 论文的章节安排

本文的内容一共分四章，各章内容如下。

第一章：主要阐述本文的研究意义和背景以及MOEA/D算法的发展现状，也简单介绍Spark云计算平台的特点。

第二章：概述了本文所涉及到的平台算法相关知识，首先通过实例分析介绍了Scala语言的面向对象特性和函数编程特性，然后介绍了Spark平台的计算模型，接着介绍了MOEA/D算法的相关概念，最后介绍了多目标0-1背包问题以及其应用领域。

第三章：详细阐述了在Spark平台上的并行方案的设计和实现过程，并给出了用基于Spark的MOEA/D算法解决多目标0-1背包问题的过程，说明了算法的参数设置。

第四章：本章介绍了Spark云计算平台集群的搭建方式，给出了主要配置文件的参数，并介绍Spark平台下MOEA/D算法的运行步骤，说明了实验过程。本章设计了MOEA/D算法并行和串行的对比实验和MOEA/D算法并行性能分析实验，展示了基于Spark的MOEA/D算法的效率提升。

最后，总结本文得到的结论，并对下一步要研究的内容进行了展望。

## 1.5 本章小结

本章首先对论文的研究做了背景和意义的介绍，介绍了以Spark为代表的云计算技术的兴起背景和MOEA/D算法，展示了将云计算技术与MOEA/D算法相结合的研究意义。最后给出了论文的研究方法和章节安排，便于读者了解了本文的主要结构。

# 第2章 预备知识

## 2.1 Scala简介

### 2.1.1 Scala背景介绍

Scala语言是一门多范式的编程语言，与Java样板代码相比其语法十分简洁，其运行在Java虚拟机上，因此用户可以使用Java现成的海量类库和工具。Scala既拥有函数式编程的特性也没有放弃面向对象的特性。

Scala发展与历史。2001年，联邦理工学院洛桑（EPFL）的Martin Odersky开始设计Scala。2003年底，Java平台的Scala于初发布。2004年6月，.NET平台的Scala发布。2006年3月，Scala语言V2.0发布。2009年4月，Twitter公司部分后端程序从Ruby迁移到Scala，剩余部分也将迁移。此外，Wattzon公司宣称，其整个平台都已经是基于Scala编写。

### 2.1.2 Scala函数式编程

在函数式编程语言中，函数是“头等公民，函数可以像其他任何数据类型一样被传递和操作。在算法中如果你想要传入明细动作时这个特性就会变的非常有用，Scala语言可以将明细动作放在函数中作为参数传入。下面来介绍Scala语言的一些函数特性。

1. 赋值给变量的函数

在Scala中你可以在一个变量中存放函数，就如同变量中存放一个变量那样。

|  |
| --- |
| import scala.math.\_  val *num* = 3.14  val *fun* = *ceil* \_  fun (*num*) // 4.0  Array (3.14 ,1.2 ,9.0).*map*(*fun*) //Array(4.0 , 2.0 ,9.0) |

在上面这段代码中\_将*ceil*方法（*math*库中的方法）转成了函数，在Scala中你无法直接操纵方法，而只能操纵函数。你可以通过函数变量名和参数来调用函数，你还可以把函数变量作为值传给另一个函数，如Array (3.14 ,1.2 ,9.0).*map*(*fun*)这样做把函数*fun*应用到数组中的每个元素，然后返回结果数组为：Array(4.0 , 2.0 ,9.0)。

1. 匿名函数和带函数参数的函数

在Scala中你不需要为每一个函数命名，就如同不要给每个数字命名一样。

|  |
| --- |
| Array (3.14 ,1.2 ,9.0).*map*( ( *x* : Double ) = > *x* \* 2) //Array(6.28 , 2.4 ,18.0) |

在上面这段代码中，( *x* : Double ) = > *x* \* 2是一个匿名函数，他作为参数传*map*函数使得数组中每个元素都乘以二，然后返回一个结果数组。如果有需要可以将函数参数包在花括号中，例如：Array (3.14 ,1.2 ,9.0).*map*{ ( *x* : Double ) = > *x* \* 2}。

1. 柯里化

柯里化（curring）指的是将原来接受两个参数的函数变成新的接受一个参数的函数的过程。

|  |
| --- |
| // 函数接受两个参数  def *m*( *a* :Int, *b* : Int)=*a* \* *b*  // 函数接受一个参数，生成另一个接受单个参数的函数  def *m*1( *a* : Int )=( *b* : Int )=>*a* \* *b*  // m1的另一种写法  def *m*1( *a* : Int )( *b* : Int )=>*a* \* *b*  //调用*m*1函数  *m*1(6)(7) //42 |

在上面这段代码中，函数*m*可以改写成*m*1的形式，调用方法也变为*m*1(6)(7)这种形式，这样做的好处是，有时候，你想要使用柯里化来吧某个函数的参数单取出来，这样就可以获得更多的类型推断的信息了。

1. 闭包

Scala语言中，可以在包、类、和另一个函数或方法中定义函数，在相应作用域中可以访问到任何变量。

|  |
| --- |
| def *mul*( *f* : Int )=(*a* : Int )=>*f* \* *a*  // 调用*mul* 参数为 2  val *t*1= *mul*( 2 )  // 调用*mul* 参数为 3  val *t*2=*mul*( 3)  //调用 *t*1 *t*2  *t*1(2) //4  *t*2(3) //9 |

在上面这段代码中，*mul*首次被调用时变量*f*被设为2，变量在函数(*a* : Int )=>*f* \* *a*的函数体重被使用，并且把这个函数存在*t*1中，这时参数*f*变量从运行时栈中弹出。*Mul*再次被调用时变量*f*被设为3，变量在函数(*a* : Int )=>*f* \* a的函数体重被使用，并且把这个函数存在*t*2中，这时参数*f*变量从运行时栈中弹出。从这个例子可以看出每个函数都有自己的*f*设置，这样的函数就被称为闭包，闭包是由代码和代码中用到的非局部变量定义成的。事实上，这些函数是以类的对象方式实现的。

Java8之前是不支持闭包的，Java8支持一种形式上受限的闭包。

1. 控制抽象

Scala语言中，可以将一系列语句组成不带参数也没有返回值的函数。

|  |
| --- |
| // 一个与while语句类似的函数  def *myWhile*(*cdt* :=> Boolean )(*b* :=> Unit){  if(*cdt*){  *b*  *myWhile*( *cdt* )( *b* )  }  }  // 使用*myWhile*  var *a* = 100  *myWhile* (*a*!=0){  *a* -= 1  println(*a*)  } |

上面这段代码，定义了一个*myWhile*语句，工作原理上类似while语句，这种函数参数叫做换名调用参数，与常规参数不同之处在于，函数被调用时，参数表达式不会被求值，毕竟我们不希望调用*myWhile*时*a*!=0被求值得到true，与之相反参数表达式成为无参数的函数体，该函数被当作参数传递下去。

### 2.1.2 Scala面向对象特性

Scala是一种纯粹的面向对象的语言，每一个值都是一个对象。对象的数据类型和行为都是由类和特征（Trait）描述。类抽象机制的扩展有两种方法：一种是子类继承，另一种是灵活的混入（Mixin）机制。这两种方法能避免多重继承的种种问题。

## 2.2 Spark简介

### 2.2.1 Spark背景介绍

Spark[3]是一个基于内存计算的大数据并行计算框架。Spark基于内存计算的特点，提高了大数据环境下处理数据的实时性，同时也保证了高容错性和高科伸缩性，允许用户将Spark部署在大量廉价的硬件上，形成集群。

Spark在2009年诞生于加州大学伯克利分校AMPLab。目前已经成为Apache基金会下的顶级开源项目。

1. Spark发展与历史
2. 2009年，Spark诞生于AMPLab。
3. 2010年，Spark开源。
4. 2013年6月，Spark成为Apache的孵化器项目。
5. 2014年2月，Spark成为Apache的顶级项目。
6. 2014年2月，知名大数据公司Cloudera宣布将加大Spark框架的投入来取代MapReduce。
7. 2014年4月：知名大数据公司MapR加入Spark阵营，Apache Mahout放弃MapReduce，使用Spark作为计算引擎。
8. 2014年5月，Pivotal Hadoop项目集成Spark全栈。
9. 2014年6月，在旧金山召开Spark2014峰会。
10. 2014年7月，Hive on Spark项目启动。

从Spark的发展和历史可以看出，Spark的发展是飞速的，目前AMPLab和Databricks负责Spark项目的开发维护，很多公司参与到Spark的开发中如Yahoo！、Intel等，同时还有许多开源爱好者积极参与Spark的更新和维护。

AMPLab开发以Spark为核心的BDAS的目标是：只需要一套软件栈就可以完成各种大数据分析任务。就MapReduce上的批量计算、迭代计算以及基于Hive的SQL查询来说，Spark可以带来上百倍的性能提升。目前Spark的生态系统日益完善，以及大量大数据公司对于Spark的全栈的支持，让Spark数据分析范式更加丰富。

1. Spark相比Hadoop MapReduce的优势
2. 中间结果输出

基于MapReduce[4]的计算框架需要将中间结果写在磁盘上来进行存储和容错，然而读写磁盘是一件非常耗时的事。Spark是将执行模型抽象为有向无环图（DAG），这样就可以将多个Stage的任务并行执行或者串联，而不用将Stage的中间结果输出到HDFS上，因此相对于MapReduce计算框架来说性能有质的提升。

1. 内存的布局和数据格式

由于“MapReduce Schema on Read”的处理方式会引起较大的处理开销，Spark抽象出分布式内存存储结构弹性分布式数据集RDD，RDD支持粗粒度的写操作，但是对于读操作，RDD可以精确到每条记录，因此RDD可以用来作为分布式的索引。

1. 执行策略

MapReduce计算框架需要在数据Shuffle之前花费大量的时间来排序，而Spark任务在Shuffle中不是所有的情景都要排序，所以支持基于Hash的分布式聚合，调度过程中采用任务执行计划图（DAG），每一轮的输出结果在内存中缓存，因此Spark可以减少排序带来的开销。

1. 任务调度的开销

传统的基于MapReduce框架的系统，如Hadoop，是为运行较长时间的批量作业而设计的，在某些极端的情况下，提交一个任务的延迟很高。Spark采用了事件驱动类库AKKA来启动任务，使用线程池复用线程来避免进程或线程启动和切换带来的开销。

1. Spark发展优势

Spark的发展优势包括[1]。

Spark是全栈式的多计算范式的高效数据流水线。Spark除了支持简单的“map”和“reduce”操作之外，还支持SQL查询，流式计算、图算法和机器学习，同时还可以在一个工作流中无缝的搭配这些计算范式。

轻量级快速处理Spark1.0的核心代码只有4万余行。这得益于Scala语言的丰富和简介的表达力，并且Spark充分利用和集成Hadoop等其他的第三方组件，同时Spark又着眼于大数据处理，数据速度是很重要的，Spark通过把中间结果缓存在内存里这样做减少了磁盘I/O使性能得到提升。

同HDFS等存储层兼容。Spark不仅可以运行在当下的YARN等集群管理系统之上外，还可以读取已有的任何Hadoop数据。可以说这是一个非常大的优势，这个特性让用户可以轻易的迁移已经存在的持久化层数据。

Spark支持多语言，易于使用。Spark支持Scala、Python和Java语言编程，开发者可以使用自己熟悉的语言进行开发，Spark拥有80多个算子，可以在Shell中进行交互式计算。用户可以很容易的像写单机程序那样来写分布式程序，轻松地搭建Spark大数据内存计算平台实现大数据的实时处理。

社区活跃度高。Spark起源于2009年，当下已有超过260个工程师和50个机构贡献过代码，对于开源系统的发展来说，一个活跃的社区和强大的生态系统是非常重要的。

当然没有哪种技术是十全十美的，Spark并不是完美的，弹性分布数据集适合的全局的粗粒度的数据并行计算，和适合细粒度的、异步更新的计算。在某些方面，针对特定的工作要达到最优性能，仍需要使用其他的大数据处理系统。比如说，流计算领域中的Storm框架在实时性要求很高的场合要比Spark Streaming更胜一筹。

### 2.2.2 Spark生态系统BDAS

目前，Spark已经发展成为包含众多子项目的大数据计算平台，Spark的整个生态系统统称为伯克利数据分析栈（BDAS）。其核心框架是Spark，同时BDAS支持结构化数据SQL查询与分析的查询引擎Spark SQL和Shark，支持并行图计算框架GraphX、流式计算框架Spark Streaming、分布式机器学习库MLlib、采样近似计算查询引擎BlinkDB、内存分布式文件系统Tachyon和资源管理框架Mesos等子项目。针对这些子项目在Spark上有更高层的、更丰富的计算范式。

### 2.2.3 Spark计算模型

1. Spark编程模型之RDD

在Spark集群中，RDD是一个非常重要的分布式数据架构[1]，即弹性分布数据集（Resilient Disturbed Dataset，RDD），它是逻辑集中的实体，它能够减少机器之间数据的重排（Data Shuffling），因为它在集群中多台机器上进行了数据分区，通过对多台机器上不同RDD分区的控制来减少数据重排。

RDD是Spark的核心数据结构，通过RDD的依赖关系形成Spark的调度顺序，通过对RDD的操作形成了整个Spark程序。

RDD有如下特点。

第一，在集群节点上不可变的，它是已分区的集合对象。

第二，是并行转换的方式创建如Map，Filter等。

第三，RDD失败会自动从父RDD重建。

第四，能够控制存储级别（内存、本地磁盘等）来进行重用。

第五，RDD必须是可序列化的。

RDD的重要内部属性。

第一，分区列表。

第二，计算每个分片的函数。

第三，对父RDD的依赖列表。

第四，对Key-Value RDD的分区器，控制分区数目和分区的策略。

第五，每个数据分片的地址列表（如HDFS上的数据块的地址列表）。

RDD具有两种创建方式。

第一，由父RDD经过转换得到RDD。

第二，由Hadoop文件系统（或与Hadoop兼容的其他持久化的存储系统如Hbase）输入（例如HDFS）创建；RDD是分布式的内存抽象，他和分布式共享内存（Distributed Shared Memory，DSM）存在着差异。DSM[6]是通用的内存数据抽象，其在商用集群上实现高一致性和容错性有些困难。RDD和DSM相比区别在于，RDD可以批量“写”，还可以对任意的内存位置读写操作。

1. Spark应用程序编程模型
2. Spark Driver Program（SparkContext)
3. 导入Spark的类和隐式转换。
4. 构建Spark应用程序的运行环境SparkConf。
5. 初始化SparkContext。
6. 关闭SparkContext。

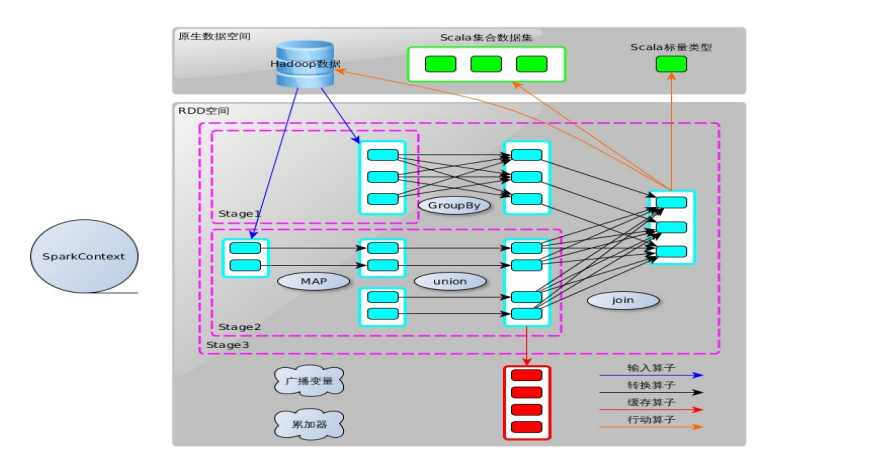


图2.1 Spark编程模型[5]



图2.2 RDD算子转换示意图

1. Spark Executor（RDD 操作)
2. 输入算子（Base->RDD）在Spark程序运行中，数据从外部空间（如分布式文件系统，Scala数据集合）输入Spark，数据进入Spark运行时数据空间，转化为Spark中的数据块。如图2.1中的蓝色箭头是从Hadoop文件系统（或与Hadoop兼容的其他持久化存储系统）输入（例如HDFS）创建。
3. 转换算子（Transformation RDD->RDD）如图2.2所示，从现有的RDD通过关系转换生成一个新的RDD，转换是延时执行（lazy）的。
4. 行动算子（Action RDD->driver or Base）如图2.1所示，在RDD上运行计算后，返回结果给驱动程序或写入文件系统。
5. 缓存算子（Persist or cache）有些RDD是计算的中间结果，其分区并不一定有相应的内存或存储与之对应，如果需要（如以备未来使用），可以调用缓存算子将分区物化下来存下来。
6. Spark广播变量和累加器
7. 广播变量
8. 广播变量（Broadcast Variables）被缓存到各个节点的内存中（注意不是缓存到每个Task中）。
9. 广播变量是只读的，一旦广播后就不能修改。
10. 对于大数据集的广播Spark还没有高效的算法。
11. 使用方法，例如广播一个字符数组到各个节点具体实现如下：

|  |
| --- |
| //定义广播变量  val *broadcastVar* = *sc*.broadcast(Array(“I”, “have”, “a”,”dream”))  //使用广播变量名取值 |
| *broadcastVar*.value |

1. 累加器
2. 累加只能用于加法计数（加操作）。
3. 累加器能够实现高效的并行，可用于实现变量求和。
4. 想要获取累加器的值必须通过驱动程序。
5. 使用方法，例如定义一个加计数器实现如下。

|  |
| --- |
| //定义累加器  val *accumlator* = *sc*.*accumulator*(0)  //实现加计数 |
| sc.*parallelize*(Array(1, 2, 3, 4,6,7)).*foreach*(*num* => *accumlator* += *num*) |

## 2.3 MOEA/D算法介绍

### 2.3.1 多目标优化问题的产生背景

优化问题是一个古老的问题，是人们现实生活中经常遇到的问题。最优化问题可以追溯到拉格朗日关于函数在一组等式约束的极值问题。

随着优化问题的不断发展，优化问题主要分为两类：单目标优化问题和多目标优化问题。单目标优化问题是需要使单个目标函数在给定区域上都可能达到最优的问题。多目标优化问题是需要使多个目标函数在给定区域上都可能达到最优的问题。

多目标优化问题与单目标优化的不同[6]。

第一，多目标优化是向量函数的优化，目标优化是标量函数的优化。

第二，对于单目标优化的问题，任何两个解是可以比较优劣的，而对于多目标优化问题，任何两个解不一定可以比较优劣。

在现实生活中，多目标优化的问题的目标是使利益最大化，成本最小化，随着科技的飞速发展，对于多目标优化的研究问题已经成了一种迫切的需求和研究热点。

### 2.3.2 多目标优化问题的产数学模型以及相关定义

一个有*n*个决策变量，*m*个目标的多目标优化问题（MOP）可以表述为：

(2-1)

式中 *X*—*n*维决策空间；

*Y*—*m*维目标空间；

*—n*维决策矢量；

*—m*维目标矢量。

目标函数定义了*m*个决策空间到目标空间的映射函数。定义了*q*个不等式约束，定义了*p*个等式约束。

以下是多目标优化问题的数学模型及相关定义：

1. 可行解

对于某个，如果*x*满足多目标优化约束条件（）则称*x*为可行解。

1. Pareto占优

假设,（为可行解集合）是多目标优化问题的两个可行解，则称与相比，是Pareto占优的，当且仅当：

且 (2-2)

记做也称支配。

1. Pareto最优解

一个解被称为Pareto最优解集（或非支配解），当且仅当满足如下条件：

1. Pareto最优解集（PS）

Pareto最优解集是所有Pareto最优解得集合，定义如下：

(2-3)

1. Pareto前沿（*PF*）

Pareto最优解集PS中的所有Pareto最优解对应的目标向量组成的集合称为Pareto前沿：

(2-4)

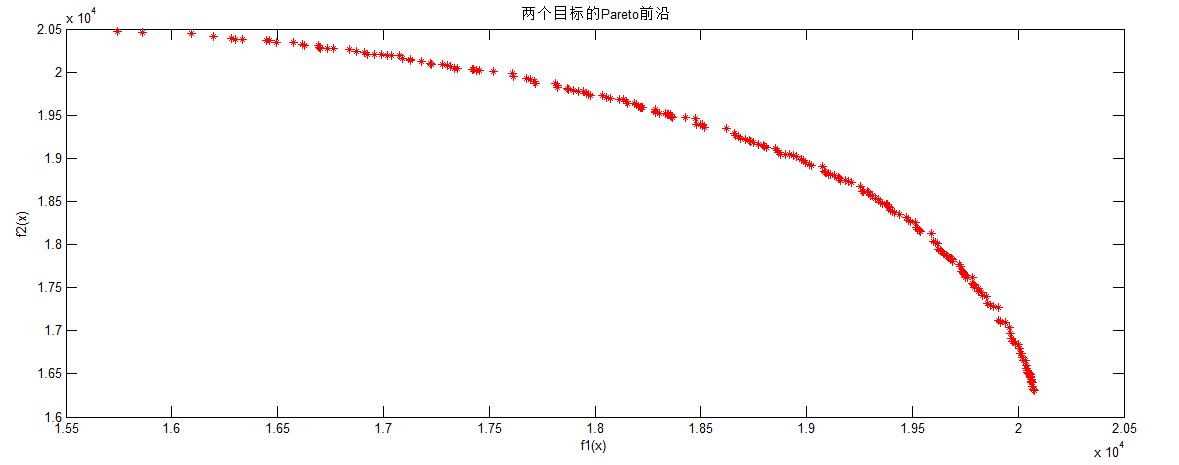


图3-1 两个目标的Pareto前沿

### 2.3.3 多目标优化问题的研究现状

1. 传统的多目标优化方法
2. 加权和方法[7]（Weighted Sum Method）：为每个目标函数分配带有实际意义的权重，再将所有的目标函数聚合成一个单目标函数进行优化。
3. ε-约束法[8]（ε-Constrained Method）先将多目标优化问题转化成单目标问题，然后通过一般的数学规划方法或者模拟退火等方法进行求解。
4. 目标规划法[9]（Goal Programming Method）决策者必须根据需要求解问题的具体情况给出每个目标做期望获得的目标值，把这些目标值作为附加的约束条件加入到原问题中，这样原问题就转化成求解目标函数值到期望函数目标值之间的绝对偏差最小的问题。
5. 极小极大法[10]（Min-Max Approach）极小极大法源自博弈论法，是通过最小化各个目标函数值与预设期望目标之间最大偏移量来寻找问题的最优解。

从以上几种传统的多目标优化方法可以看出，传统的方法的优点的使用了单目标优化的机制，然而他们在求解多目标优化问题时存在很多问题：

1. 设置这些算法必须根据问题的一些先验知识来获取取合适的参数，不同的参数设置会产生不同的最优解。
2. 大多数情况下，采用这些方法一般只能得到一个Pareto最优解，而这个解的获取和转换过程中的参数设定有着非常大的关系。为了获取更多的Pareto最优解，就得不断的调整参数，多次按照单目标优化方法来进行寻优，这往往是很费时间并且很难决策。
3. 有些算法对Pareto前沿的形状很敏感，比如加权和方法，在处理Pareto最优前沿为非凸的问题时，无论参数如何调整，非凸区域都无法找到Pareto最优解。
4. 解决多目标优化问题的新方法

进化算法[11]是基于自然选择和自然遗传等生物进化机制的一种搜索算法。进化计算是一种成熟的具有高鲁棒性和广泛适用性的全局优化方法，具有自组织、自适应、自学习的特性，能够不受问题性质的限制，有效地处理传统优化算法难以解决的复杂问题，它的出现为解决多目标优化问题开辟了一种新的途径。

下面列举了一些解决多目标优化问题的新方法：

第一代多目标进化算法：

1. MOGA；
2. NSGA；
3. NPGA。

第二代多目标进化算法：

1. SPEA和SPEA2；
2. PAES，PESA，PESA2；
3. NSGA2。

多目标优化新型进化算法：

1. 基于粒子群优化的多目标优化；
2. 基于人工免疫系统的多目标优化；
3. 基于分布估计的多目标进化算法；
4. 基于分解的多目标进化算法（MOEA/D）。

### 2.3.4 基于分解的多目标进化算法（MOEA/D）

MOEA/D是一种将传统的数学规划方法与进化算法相结合的新颖算法，用来处理多目标优化问题。它将一个多目标优化问题分解为一定数量的单目标优化子问题，然后用进化算法同时求解这些单目标子问题。

1. MOEA/D分解策略
2. 加权和法（Weighted Sum Approach）[12]

设为一个权向量，其中，转化后的子问题可描述为：

(2-5)

1. 切比雪夫法（Tchebycheff Approach）[12]

设N为种群的规模，为一个均匀的权向量集，*z\**为参考点，*m*为目标函数的个数，MOEA/D将求解PF的MOP问题使用切比雪夫法分解后，第*i*个子问题可以描述为：

(2-6)

1. 传统串行MOEA/D算法的实现[2]

对于第*t*代种群，使用切比雪夫的MOEA/D包含以下初始条件：

1. 大小为*N*的种群，，其中是第*i*个子问题的当前最优解；
2. ，其中是的目标函数值，*)*；
3. ，是目前搜索到的目标函数的最佳值（或者说最大值）；
4. 一个外部种群(*EP*)，是用来存储当前搜索到的非支配解。

**算法2.1：MOEA/D**

**输入：**

1. 多目标优化问题(MOP)；
2. 算法终止条件；
3. *N*：MOEA/D定义的种群大小（或分解子目标的个数）；
4. 均匀分布的*N*个权重向量：；
5. *T*：权向量的邻居个数；

**输出：** *EP*

**Step 1：初始化：**

**Step 1.1：**Set *EP*=。

**Step 1.2：**计算任意两个权重向量间的欧式距离，查找每个权重向量最近的*T*个权重向量作为它的邻居。对于每个，最近的*T*个权重向量。

**Step 1.3：**初始化种群，设。

**Step 1.4：**采用基于问题的特定方法初始化

**Step 2 更新：**

*For i=1,…,N do*

**Step2.1：**基因重组：从*B(i)*中随机选取两个序号*k*, *l*，运用遗传算子由产生一个新的解*y*。

**Step2.2：**改进：对*y*运用基于测试问题的修复和改进启发产生。

**Step2.3：**更新*z*：

*，令*

**Step2.4：**更新邻域解：

令

**Step2.5：**更新*EP*：从*EP*中移除所有被支配的向量，如果*EP*中的向量都不支配，将加入*EP*。

*End for*

**Step3停止判断：**如果满足终止条件：停止并输出*EP*，否则转**Step2**

1. 多目标进化算法的性能评价指标

多目标问题求解的最终目标是得到一个能均匀分布在真实的Pareto前沿的解集，主要从以下两个因素考虑：

1. 分布性：衡量所求的Pareto最优解集在目标空间中的分布均匀性，多样性。
2. 收敛性：评价所求的Pareto最优解集与理想的Pareto前沿趋近的程度。

表2-1 多目标算法评价指标

|  |  |
| --- | --- |
| 多目标进化算法评价指标 | 评价特性 |
| Generational Distance(GD)[13] | 收敛性 |
| Inverted Generational Distance[14](IGD) | 分布性，收敛性 |
| Spacing(S)指标：分布性[15] | 分布性 |
| Hyper volume(HV)[16] | 分布性，收敛性 |

## 2.4 多目标0-1背包问题介绍

### 2.4.1多目标0-1背包问题背景

在实际的优化问题中，决策变量经常会涉及到整型变量，例如调度问题、排序问题、选址问题、投资问题。

离散优化问题处理起来和连续问题相比更加困难，原因在于：许多对于连续优化问题的求解很有效果的算法都是不能直接用于求解离散优化问题，因此有时会将离散优化问题先转化成连续问题来求解，然后再利用连续优化方法求解。

背包问题[17]（KP）是运筹学中典型的优化难题（NP完全问题）。它具有非常广泛的应用领域，如项目选择、预算控制、货物装载等。传统方法对于求解背包问题时比较有效的，如贪心法、动态规划法、分支定界法等，但是对于背包问题一些复杂的变种问题的求解往往效果很不好，进化算法作为一种全局的优化搜索算法，适应范围广，非常适合用来求解各种背包问题。随着大数据时代的来临，Spark计算平台和MOEA/D算法的结合，对多目标优化问题有着重要的意义。

### 2.4.2多目标0-1背包问题的定义

多目标0-1背包问题定义[18]：

(2-7)

式中 *m*—背包数目；

*n*—物品数目；

*w*—背包中物品的重量；

*p*—背包中物品的价值。

对于多目标背包问题来说，选中一个物品放入背包中时，是将这个物品放入到所有背包中。同一物品在不同包中有着不同的价值和重量，每个包的容量也是不同的，对于多目标0-1背包问题，优化的目标就是使各个背包的价值最大化。

### 2.4.3贪心修复策略

贪心修复策略算法流程如下：

**Step 1：**为可行解，则，输出*y*。

**Step 2：**设

**Step 3：**在中选择*k*满足：

(2-8)

设置=0返回**Step 1**

在这种贪心修复策略中，如果物品超出容量的背包，先按照物品的价值重量比递增排序，然后就再按照次序逐一删除背包中的物品直到满足容量约束为止。

### 2.4.4参数设置

1. 权重向量*λ*的设置

本论文实验中权重向量的设置采用配方实验设计方法中的单纯格子点设计。设计方法如下[19]：

1. 先确定一个正整数*H*。
2. 然后在让每个原料取值：

*,*  (2-9)

例如当*m*=3，*H*=1时，试验点为：（1，0，0），（0，1，0），（0，0，1）。当*m*=3，*H*=2时有6个试验点（1，0，0），（0，1，0），（0，0，1），（1/2，0，0），（0，1/2，0），（0，0，1/2）。

这种参数设置方法一般记为设计，一个设计有个试验点。

1. 个体邻居群大小*T*的设置

本论文实验中邻居群的大小*T*的设置参照文献[2]中的设置。

在当代种群中，对于每一个子问题，只有*T*个最相邻子问题被用来优化它。某种意义上来说，两个解只有在它们会为相邻子问题是可以交叉产生一个子代。这个被称为交叉约束。

如果*T*数值过小，那么会非常的接近，因为他们是两个近似子问题的解，因此，在变异后中产生的将会和他们的父代非常接近。因而，这个算法就会陷入局部最优的困局。

如果*T*的数值过大，对于子问题来说可能有用性不大，那么他们的子代也是如此。因而，这个算法的精度就会变低。此外，*T*过大也会导致更新领域解的计算复杂度大大提升。

## 2.5 本章小结

本章主要是对 Scala语言、Spark平台、MOEA/D算法相关知识和多目标0-1背包问题进行了详细的介绍。首先介绍了Scala语言，主要是对Scala语言的背景、发展历史和Scala语言的函数式编程、面向对象编程特性的详细介绍。然后对Spark平台进行介绍，给出了Spark平台的发展历史和Spark平台的发展优势，详细阐述了Spark平台和MapReduce相比的优势，侧重介绍了Spark计算模型。接下来介绍了多目标优化问题的发展现状以及MOEA/D算法的执行过程，并且给出了多目标进化算法的主要评价评价指标。最后详细介绍了本文算法所要求解的实际问题：多目标0-1背包问题，并且给出了实验中参数的设置方法。