分类号 密级

**本科毕业论文**

**基于Spark的笛卡尔遗传程序算法分布式化研究**

**Research on Parallel of Cartesian Genetic Algorithm**

**Based on Spark**

|  |
| --- |
| **学生姓名：夏航宇** |
| **学生学号：2011305200605** |
| **专业班级：计科1204** |
| **指导教师：倪福川 讲师** |

信息学院

二〇一六年六月

目录

[摘要 I](#_Toc453767513)

[Abstract I](#_Toc453767514)

[缩略词 I](#_Toc453767515)

[1 前言 1](#_Toc453767516)

[1.1 研究背景 1](#_Toc453767517)

[1.2 研究内容 1](#_Toc453767518)

[2 笛卡尔遗传规划程序 1](#_Toc453767519)

[2.1 算法介绍 1](#_Toc453767520)

[2.2 算法特点 1](#_Toc453767521)

[2.3 CGP的基因结构 1](#_Toc453767522)

[2.4 CGP解码实例 1](#_Toc453767523)

[2.5 CGP算子分类 1](#_Toc453767524)

[2.6 CGP流程 1](#_Toc453767525)

[3 Spark分布式计算平台 1](#_Toc453767526)

[3.1 Spark简介 1](#_Toc453767527)

[3.2 弹性分布数据集Resilient Distributed Dataset (RDD) 1](#_Toc453767528)

[3.3 Spark架构 1](#_Toc453767529)

[3.3.1 Master-Slave分布式模型 1](#_Toc453767530)

[3.3.2 Spark架构中的基本组件 1](#_Toc453767531)

[3.3.3 Spark的运行逻辑 1](#_Toc453767532)

[3.4 Spark/GraphX 1](#_Toc453767533)

[3.4.1 Graph架构 1](#_Toc453767534)

[3.4.2 GraphX属性图 1](#_Toc453767535)

[3.5.3 GraphX/Pregel 1](#_Toc453767536)

[4 CGP在GraphX中的实现 1](#_Toc453767537)

[4.1 程序采取的构图方式 1](#_Toc453767538)

[4.2 程序的参数设置 1](#_Toc453767539)

[4.3 程序的具体流程 1](#_Toc453767540)

[4.4 实验结果与分析 1](#_Toc453767541)

[5 总结与展望 1](#_Toc453767542)

[5.1 总结 1](#_Toc453767543)

[5.2 展望 1](#_Toc453767544)

[参考文献 1](#_Toc453767545)

[致谢 1](#_Toc453767546)

[附录 1](#_Toc453767547)

# 摘要

笛卡尔遗传规划程序（Cartesian Genetic Programming,CGP）是一种将计算程序编码，以图的形式表现的高效、灵活的进化算法的实现。一般的进化算法关注的是自动进化（如同达尔文进化论一样）的计算模式（如数学方程，计算机程序，数字电路）。这类计算模式往往会进行大量的迭代运算，具有天然的分布式性。Apache Spark是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。本文基于Spark/GraphX图计算框架提出了一种笛卡尔遗传规划程序的并行化方法来解决串行笛卡尔遗传程序无法处理问题规模极大、迭代层次极深的缺点。

**关键词：**笛卡尔遗传规划；分布式计算；Spark；RDD

# Abstract

Cartesian Genetic Programming(CGP) is a highly efficient and flexible form of Genetic Programming that encodes a graph representation of a computer program. Genetic Programmingconcerned with the automatic evolution (as in Darwinian evolution) of computational structures (such as mathematical equations, computer programs, digital circuits, etc.) which involve many iterative computations and are nature for concurrent computation. Apache Spark is a computing framework based on in-memory computing, and it is also applicable to the need of operating specific data sets repeatedly. This article proposed a parallelization method of Cartesian genetic programming algorithm based on Spark/GraphX diagram calculation framework to overcome the shortcoming of serial Cartesian Genetic Programming that can not handle problems with enormous scale and deep iterative level

**Keywords:**Cartesian Genetic Programming; Spark; GraphX; Pregel; RDD

# 缩略词

CGP: Cartesian Genetic Programming, 笛卡尔遗传规划程序

RDD: Resilient Distributed Dataset, 弹性分布数据集

RDPG: Resilient Distributed Property Graph, 弹性分布属性图

# 1 前言

## 1.1 研究背景

进化算法（Genetic Algorithm, GA）是近些年飞速发展起来的一种全新的随机搜索与自进化算法，其基本原理是基于Darwin的进化论和 Mendel的遗传学说。该算法最初由密执安大学的教授Holland和他的学生在1975年创建。此后，国际上召开过多次关于进化算法的学术会议和研讨会，为进化算法的发展提供了很好的环境。作为一种求解通用问题的算法，进化算法将各种复杂的模型转换成简单的编码表表示，并在此基础上模拟生物界遗传操作和优胜劣汰，使得程序能够自动演化和搜索。

在传统的进化算法中，通常主要用大量的群体和交叉方法来从旧的程序开发新的候选解决方案。而进化编程中强调的是变异算子的重要性。虽然遗传程序和进化程序对进化算子的着重点不同，但是它们都是通过将程序表达成解析树来消除基因的表型和基因型的差异，而使用解析树的最初的理由是希望由此找到解决将交叉过程应用到变长基因型上的方法。在这种情况下，一种新的方案顺势而生。这就是笛卡尔遗传规划程序（CGP）。在这个方案中，基因型被表示成一个整型的列表，并且被直接映射到图而不是树中。这么做的原因是图比树更具有基因之间关系的一般性。计算机中的许多数学方程式、电路等复杂的计算结构也都可以用图来表示的。因此，CGP在人工智能，函数优化，电路设计甚至人工智能等很多领域是非常有竞争力的。

但是CGP往往运算在数据量和迭代次数比较庞大的环境里。CGP的自动演化的速度会因此遭到十分严重的挑战。Apache Spark 能通过聚集多台计算节点将计算任务拆解分发给节点分布计算后再整合，从而大大提高CGP的运行速度。

Spark发源于美国加州大学伯克利分校AMPLab的集群计算平台，立足于内存计算，从多迭代批量处理出发，兼收并蓄数据仓库、流处理和图计算等多种计算范式。由于Spark仍然比较年轻，当运用到生产上时，可能发现一些小缺陷。而在代码整洁度方面，也随时在对代码进行着重构。例如，淘宝技术部在2013年就开始尝试将Spark on Yarn应用到生产环境上。他们在执行数据分析作业过程中，先后发现了DAGSchedular的内存泄露，不匹配的作业结束状态等缺陷，从而为Spark库贡献了几个比较重要的Pull Request。

## 1.2 研究内容

本文的主要研究内容是在并行化的框架设计笛卡尔遗传规划程序完成一个1位全加器电路和一个11位的多路选择器的电路设计。涉及的具体内容如下：

（1）串行笛卡尔遗传规划程序的原理

（2）GrapX/Pregel的计算框架

（3）Spark/GraphX的体系架构

（4）CGP并行化

（5）结果分析

# 2 笛卡尔遗传规划程序

## 2.1 算法介绍

笛卡尔遗传规划程序之所以如此命名是因为这个方法主要涉及到笛卡尔坐标系下的节点网络的操作。CGP的节点与分布式分布式遗传规划(PDGP)相似。在早些的版本当中，图的演化过程没有使用（基因型-表型）映射，并且算法定义了许多复杂的交叉算子。到后来，所有的程序都被表示成具有最多N个输出的N节点有向图。

一个笛卡尔遗传规划程序一般会被定义成一个集合P{G F l}。其中G代表一个由整形集合表示的基因型。其中，表示的程序输入索引。表示的节点连接和操作。表示程序输出连接。集合F表示节点的功能。节点网络的行列长度分别为与。最后，程序节点间的相互连接层度由回调参数l定义。l的数值会决定每个节点能的输入能向前引用在它之前的节点的最远间隔的列数。

## 2.2 算法特点

1. 笛卡尔遗传规划算法从问题的串集开始搜索，而不是从单个的解开始。这样使得笛卡尔遗传规划算法有更大的覆盖面，更利于在全局上选择较优解。
2. 笛卡尔遗传规划算法能同时处理多个个体，对搜索空间中多个解进行评估。同时算法本身也易于实现并线化。
3. 笛卡尔遗传规划算不采用确定性规则，而是采用概率的变迁规则来指导搜索方向。
4. 具有自组织、自适应和自学习性。算法利用进化过程获得的信息自行组织搜索时，适应度大的个体具有较高的生存概率，并获得更适应环境的基因结构。
5. 使用笛卡尔集来表示程序逻辑节点。笛卡尔遗传规划算法能以更简单的方式表示电路的逻辑结构。

## 2.3 CGP的基因结构

CGP中每个个体由一条编码后的染色体表示。每个染色体由计算节点网络和输出节点组成。计算节点分为两部分：功能基因和连接基因。输出节点则是随机选择之前的任意计算节点和输入节点得到。

（1）计算节点结构

功能基因：如果一个基因是功能查找表中某个节点计算功能的地址或编号，则称这个基因为功能基因。每个计算节点都有其对应的功能基因。

连结基因：每一个存在的计算节点都有它的数据来源。这些计算在节点上表现了输入数据地址的基因，我们称之为连接基因。连结基因的输入既可以是项目的输入也可以是反馈方式中前面列的输出。一般是前面的计算节点的节点号。一个节点连结基因的个数被选择为最大数量的输入。

一个计算节点的基因构成如图所示：

连接基因1

连接基因2

连接基因3

…

连接基因n农

功能基因

图1计算节点

Fig.1 Compute Node

（2）染色体结构

CGP染色体结构如下图所示。其中0~1表示输入节点，输入节点不属于染色体结构部分；表示计算节点；表示输出节点。输出节点与计算节点的组合基因不同，输出节点只是单个的连接基因并且取值于前面的计算节点或者输入节点。

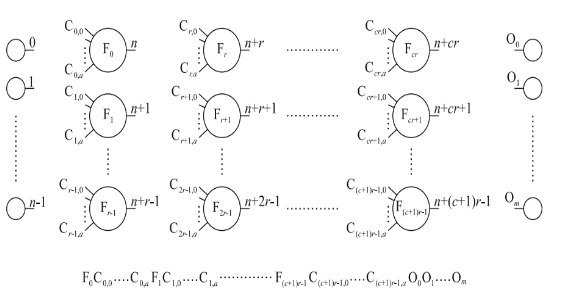


图2 CGP染色体结构

Fig.2 Chromosome structure

## 2.4 CGP解码实例

图描绘了一个有6个输入，3个输出节点的染色体（3行4列）。并且l=1,=3。

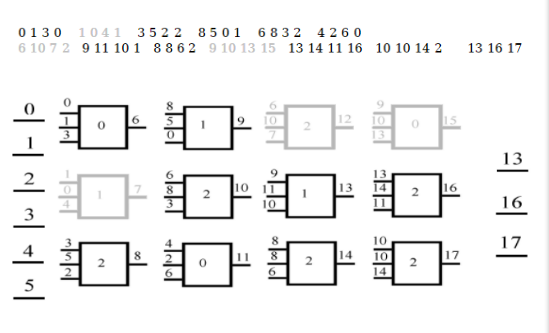


图3染色体实例

Fig.3 Chromosome instance

由图3可以看出个体的染色体基因序列为(0130 1041 3522 8501 6832 4260

61072 911101 8862 9101315 13141116 101014 2131617)。以节点号为6的节点0130为例解释该基因携带信息的意思。节点6的的前三个基因013表示该节点的输入节点参数的节点号，第四个基因0表示节点6进行的操作是事先定义操作0。整个图被分为3个大块，0~5是输入部分，6~17是逻辑处理部分，最后的三个节点输出部分，他们的值是从前17个节点中以一定的规则（可以是完全随机的）直接选出赋予的。

此外，在CGP中同一列的节点不能相互连接，并且每个节点都可以被连接也可以不被连接。程序在l=并且l=时有最大的节点连接自由度。比如，在这种情况下，所有的节点都可以连接到在它左侧任意的节点。将这种网格的图模型编码成一个整形数列如图1所示，是CGP将计算模型转换成程序的关键步骤。算法之后的交叉变异过程也都是基于编码后的数据形式实现的。

## 2.5 CGP算子分类

CGP算子分为两类：选择算子、变异算子。

选择算子：根据各个个体的适应度，按照一定的规则或方法，从第t代群体P(t)中选择出一些优良的个体遗传到下一代群体P(t+1)中。

变异算子：也是产生新个体的主要操作过程，在遗传规划中，变异算子是对染色体上的某个节点上某位的基因按某一较小的概率进行变异。

## 2.6 CGP流程

与一般的进化算法流程相似，如图4所示：

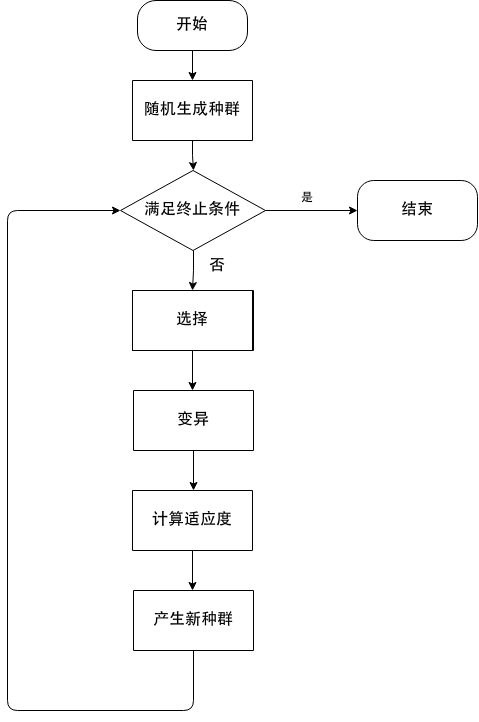


图4进化算法流程

Fig.4 EA process

（1）种群初始化：按照一定的约束条件随机生成一代种群，并初始化。

（2）个体适应度评估：对种群内所有个体进行适应度评估

（3）选择：将本代种群中适应度最高的个体直接放入下一代种群中

（4）交叉、变异：对本代最优的个体进行交叉变异，产出下一代种群中的剩余个体

（5）迭代：判断最优的个体是否产生，或者是否达到最大迭代代数。否则返回步骤（2），继续下一轮演化。

需要注意的是：在第（3）步中如果新产生的基因型中没有比当先适应度最高的基因更优秀的基因。这些基因型会被随机挑出一个基因来进行变异并产生下一代种群。

# 3 Spark分布式计算平台

## 3.1 Spark简介

Spark是UC Berkeley AMP lab开发的一个开源的类似Hadoop MapReduce的一般性分布式框架。Spark具有Hadoop MapReduce的所有优点，但与MapReduce不同的是，Spark Job的中间输出结果可以保存在内存中，从而不需要反复读写HDFS。因此，Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。Spark与Hadoop两者之间还存在其他一些不同之处。Spark 在某些工作负载方面表现得比Hadoop更加优越。Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。

Spark 另外一个特点是它是在 Scala 语言中实现的。它将 Scala 用作其应用程序框架。Spark 和 Scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。这点是Hadoop无法做到的。

虽然创建 Spark 是为了支持分布式数据集上的迭代作业，但它实际上是对 Hadoop 的补充。因此Spark可以在 Hadoop 文件系统上运行，本文的实验环境便是搭载在hadoop2.6.0上的spark1.2.0分布式集群。通过名为 Mesos 或者Yarn等的第三方集群框架可以支持此行为。Spark 由加州大学伯克利分校 AMP 实验室 (Algorithms, Machines, and People Lab) 开发，可用来构建大型的、低延迟的数据分析应用程序。

## 3.2 弹性分布数据集Resilient Distributed Dataset (RDD)

弹性分布数据集(Resilient Distributed Dataset, RDD)是Spark的核心概念。指的是一个只读的，可分区的、容错的、分布式的分布式数据集，这个数据集的全部或部分可以缓存在内存中，并能控制数据的分区。同时，RDD还提供了一组丰富的操作来操作这些数据。在这些操作中，诸如map、flatMap、filter等转换操作实现了monad模式，很好地契合了Scala的集合操作。除此之外，RDD还提供了诸如join、groupBy、reduceByKey等更为方便的操作（注意，reduceByKey是action，而非transformation），以支持常见的数据运算。

一个RDD可以包含多个分区，每个分区就是一个数据片段。RDD可以相互依赖。如果RDD的每个分区最多只能被一个Child RDD的一个分区使用，则称之为narrow dependency；若多个Child RDD分区都可以依赖，则称之为wide dependency。不同的操作依据其特性，可能会产生不同的依赖。例如map操作会产生narrow dependency，而join操作则产生wide dependency。RDD分区的依赖方式是Spark中每个Job中Stage划分的重要依据。

下面，我们将介绍如何在Spark中操作RDD：

1. 获取RDD

1) 从共享的文件系统获取

2) 通过已存在的RDD转换

3) 将已存在scala集合（只要是Seq对象）分布式化，通过调用SparkContext的parallelize方法实现

4) 改变现有RDD的持久性；RDD是懒散，短暂的。

1. 操作RDD的两个动作

1）Actions：对数据集计算后返回一个数值value给驱动程序；例如：Reduce

将数据集的所有元素用某个函数聚合后，将最终结果返回给程序。

2）Transformation：根据数据集创建一个新的数据集，计算后返回一个新RDD；例如：Map将数据的每个元素经过某个函数计算后，返回一个新的转换后的RDD分布式数据集。

## 3.3 Spark架构

### 3.3.1 Master-Slave分布式模型

Spark架构采用了分布式计算中的Master-Slave模型。Master是对应集群中的含有Master进程的节点，Slave是集群中含有Worker进程的节点。Master作为整个集群的控制器，负责整个集群的正常运行；Worker作为计算节点，接收Master分配的计算任务并不断进行状态汇报；Executor负责任务的执行；Client作为用户的客户端，负责提交应用；Driver负责控制一个应用的执行。如图所示：

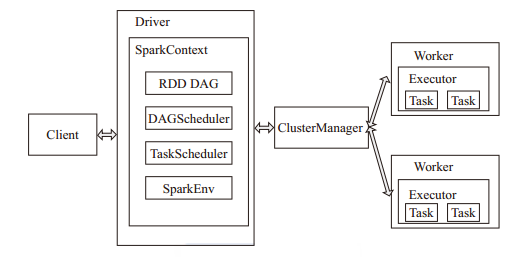


图5 Spark框架图

Fig.5 Spark Frame

部署好Spark集群之后。我们需要执行./sbin/start-all.sh 脚本命令来启动master和所有的worker进程。Driver程序是应用的逻辑执行的起点。Driver程序会负责作业的调度，即分发Task任务到多个worker节点上。Worker程序用来管理所在的计算节点并创建Executor来分布式处理任务。

### 3.3.2 Spark架构中的基本组件

（1）ClusterManager：在Standalone模式中即为Master节点。负责控制整个

集群，调度worker。在 Yarn模式（本文采用的是yarn模式）中，ClusterManager则是资源管理器。

（2）Worker：负责控制计算节点，启动Executor或Driver。在Yarn模式中

为NodeManager，负责计算节点的控制。

（3）Driver：运行应用的main()函数，创建SparkContext。

（4）Executor：由Worker创建，在计算节点上执行任务的组件、用于启动线程池运行任务。每个应用都有一组独立的Executors。

（5）SparkContext：整个应用的上下文。被用来控制应用的生命周期。

（6）RDD：Spark的基本计算单元。一组RDD可以形成一个有向的无环RDD

图。之后程序会将这个图转换为一个更通用的任务执行计划图（DAG）。

（7）DAG Scheduler：根据Job（作业）构建基于Stage的DAG，并提交Stage

给TaskScheduler生成更细化的TaskSet（一个Task的集合）。

（8）TaskScheduler：将Task（任务）分发给Executor执行。

Spark的整体流程为：Client提交应用。Master找到一个Worker启动Driver，然后Driver向Master或者资源管理器申请资源。之后Driver会将应用转化为RDD Graph交予DAGScheduler。DAGScheduler将RDD Graph转化为Stage的有向无环图提交给TaskScheduler，由TaskScheduler提交具体的任务给Executor执行。

### 3.3.3 Spark的运行逻辑

如图所示。在Spark应用中，整个程序的执行流程在Actions算子触发之后会被转化为一个有向无环图（DAG）。之后调度器会调度这个有向无环图的任务进行运算。Spark的调度方式与MapReduce不同。Spark根据RDD之间不同的依赖关系（宽依赖，宅依赖）将有向无环图切分成不同的Stage。具体的切分方法是将拥有持续窄依赖的RDD归并到同一个Stage中，而宽依赖作为一个Stage的结束。此外，每一个Stage还包含一系列函数执行流水线（TaskSet）。图6中的A、B、C、D、E、F分别代表不同的RDD。RDD C上执行map操作转换为一个新的RDD D，RDD B 和RDD E执行join操作转换为新的RDD F。最后，RDD F会通过函数saveAsSequenceFile输出并保存到HDFS中。

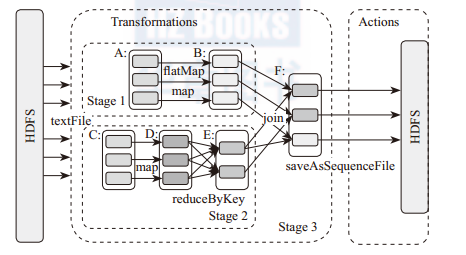


图6 Spark执行的有向无环图

Fig.6 Spark DAG

## 3.4 Spark/GraphX

GraphX是Spark API提供的一个新的高效图计算框架。因为限于底层语言的实现，GraphX在运行速度上无法与使用C语言实现的GraphLab图计算比肩。但是GraphX底层数据表示采用的是扩展的Spark RDD结构RDPG(Resilient Distributed Property Graph)。RDPG是一个带有顶点和边属性的有向图。因此GraphX具有Spark的底层分布式透明的特性，即程序员在运用GraphX进行分布式图计算的过程中可以像编写正常程序一样，而不用考虑复杂的具体分布式操作。GprahX还公开了一组RDPG的基本操作函数，并封装在Graph类以及GraphOps类中。除此外，GraphX对Pregel API也进行了优化。

### 3.4.1 Graph架构

（1）整体架构

GraphX的整体架构一共可以抽象成3个部分。这3个部分如下图所示。

1. 接口层、实现层：以Graph类为中心。内部含有VertexRDD、EdgeRDD以及RDD[EdgeTriplet]的引用。GraphImpl继承Graph类，并实现了Graph的许多操作。
2. 模型层：基于底层的Spark RDD，优化了Google Pregel模型以及BSP消息机制。该层提供了GraphX的Pregel计算接口。
3. 工具包：该层是一个基于GraphX Pregel实现的常用图算法接口集合。具体包括：PageRank、SVDPlusPlus等常用的图计算算法。

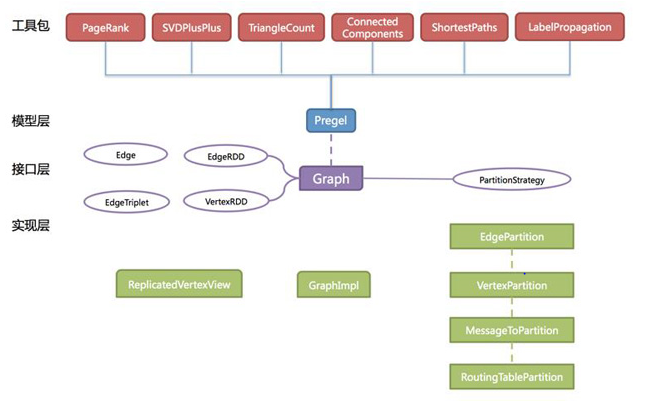


图7 GraphX整体框架

Fig.7 GraphX General Frame

（2）存储架构

GraphX中图的分布式存储分为两种：边分割(edge cut)、点分割(vertex cut)。分割方式如下图所示：

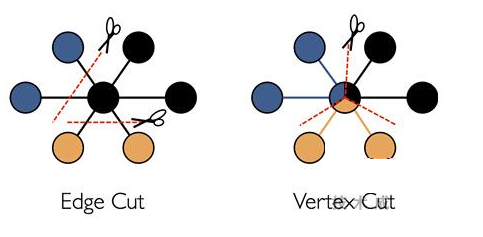


图8图分割

Fig.8 Graph Cut

如上图所示。边分割存储方式的特点是将顶点以边来划分存在不同计算节点上，其中每个顶点都只存储一次但边可以存储多次；点分割存储方式的特点是将边以顶点来划分并存储到计算节点上，其中每条边都只存储一次但顶点可以存储多次。

### 3.4.2 GraphX属性图

（1）Graph的结构

GraphX中的图结构由一个叫做Graph的类实现。Graph是一个有向多重图，其点和边的属性通过vertex(VD)和edge(VD)类型参数化。图中每个顶点都由一个64位的VertexId标识，每条边由两个顶点（起始点src和终点dst ）Id标识。Graph类将顶点和边的信息存放在两张采用RDD结构的表（视图）中。表的定义如下所示：

class Graph[VD, ED] {

val vertices: VertexRDD[VD] //顶点信息表

val edges: EdgeRDD[ED] //边信息表

}

注意：VertexRDD[VD]和EdgeRDD[ED]类分别继承自类RDD[(VertexId,VD)]和RDD[Edge[ED]]。其中Edge类表示一条边的关系，数据元素可以表示为(srcVertexId, dstVertexId, ED)。

一个图的边表、顶点表的转换如下图所示：

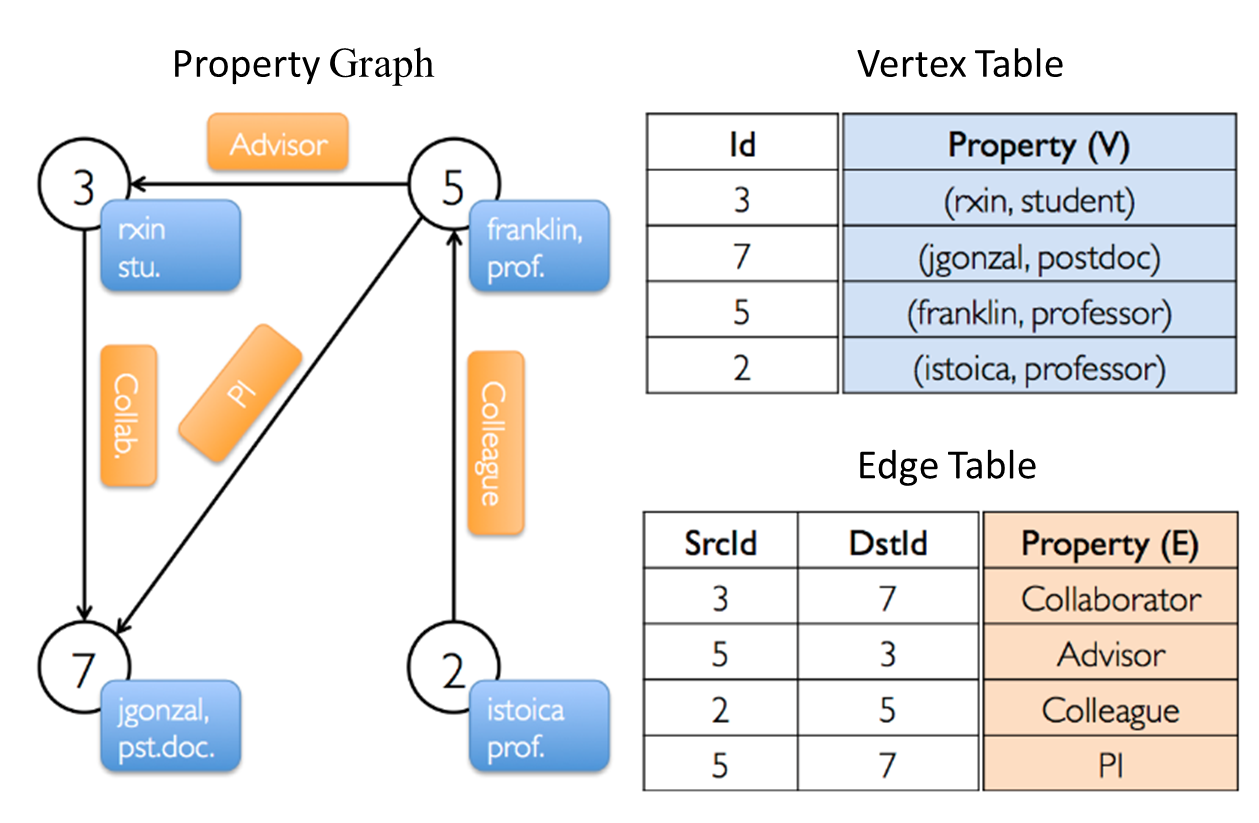


图9边表、顶点表实例

Fig.9 Example of Vertex Table and Edge Table

另一个需要提及的重要结构是Graph包含的边三元组视图。边三元组视图逻辑上将顶点和边的属性保存在一个RDD中，每个边三元组作为RDD的一个元素存储。边三元组视图是一个RDD[EdgeTriplet[VD, ED]]实例。Graph类定义了一套对边三元组试图的操作。通过下面的Sql表达，我们可以了解边三元组视图数据结构的构造方式：

SELECT src.id, dst.id, src.attr, e.attr, dst.attr

FROM edges AS e LEFT JOIN vertices AS src, vertices AS dst

ON e.srcId = src.Id AND e.dstId = dst.Id

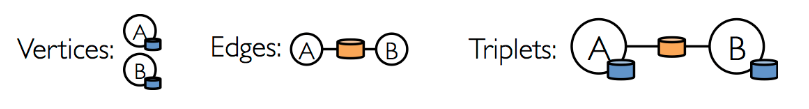
以下面图比较Graph的边视图、顶点试图以及边三元组视图：

图10 Graph三种视图的比较

Fig.10 Comparision between the three different views in Graph

需要提及的是，边三元组EdgeTriplet类继承与Edge类，并在其中添加了类型为VD的srcAttr和dstAttr成员。这两个成员分别包含起始顶点和终止顶点的属性。EdgeTriplet的部分成员申明如下图11所示：



图11 EdgeTriplet的成员

Fig.11 Members of EdgeTriplet

### 3.5.3 GraphX/Pregel

在理解Pregel计算框架之前，我们需要先了解什么是BSP模型。

（1）BSP模型

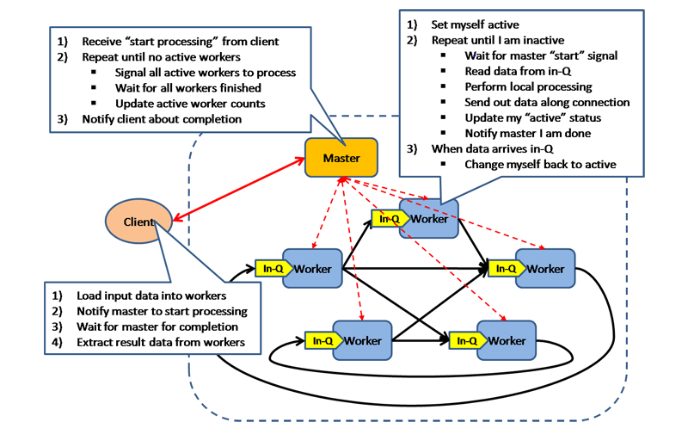
BSP模型全称整体同步分布式计算模型(Bulk Synchronous Parallel Computing Model)，简单点来说就是“计算”-“通信”-“同步”的模型。该模型属于Master-Slave分布式模型。由一个Master协调，所有的Worker同步执行。模型中数据的输入由一个队列管理，所有的Worker都从队列中读取数据。该模型的流程如图：

图13 BSP计算流程

Fig.13 BSP Computation Process

现在，我们通过另外一张图更加细致地解释BSP模型具体的计算框架。首先观察图：

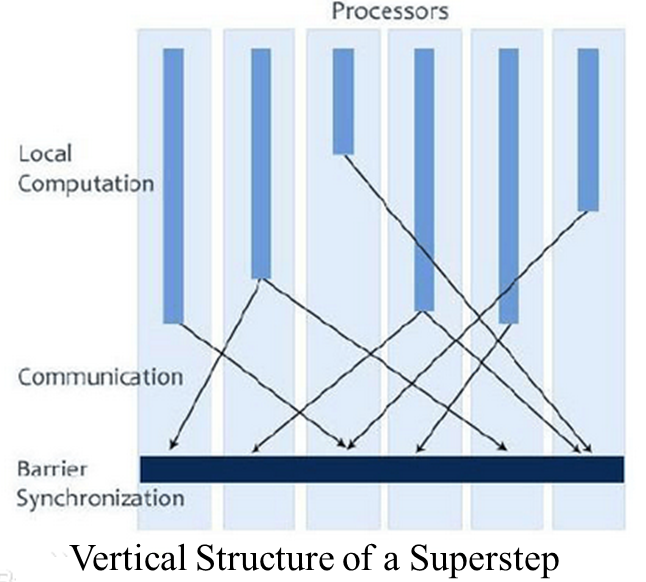


图14 BSP计算框架

Fig.14 BSP Computation Frame

图中一共描述了4个主要的部分：

1）Processors：指的是在集群中多个节点上分别分布式运行的若干计算进程。集群每个节点都可以拥有多个Processor。

2）Local Computation：本地计算指的是单个Processor对存储在本地上的数据进行计算的过程。

3）Communication：在BSP模型中图的计算往往会涉及到递归或者需要使用全局变量。此外，图节点的访问还被分布在不同的Processor中并且并不保证关系紧密的节点能被分布到同一个Processor或者同一个集群节点上。Communication过程便是在这个时候起到在Processor之间通信的作用的。所有需要用到的数据都需要通过这个过程在Processor之间传递。

4）BarrierSynchronization：叫做障碍同步或者栅栏同步。每一次同步都是本次超步的结束和下次超步的开始。

超步是BSP独有的特点。每个超步都是BSP的一次迭代。以图的广度优先遍历为例，我们可以把从起始节点开始的每往下搜索一层的过程看作一个超步。

（2）Pregel框架

在GraphX中，使用的是更高级的Pregel操作。该操作是一个根据图拓扑约束分布式同步消息的抽象模型。Pregel会执行一系列的超级步骤。在每个超级步骤中，每个顶点都有一个消息接收机制来接收上一个超级步骤传给这个顶点的所有消息。顶点会根据这些消息的总和来更新它的属性。在每次超级步骤结束的阶段，每个顶点会向下一次超级步骤中有联系的顶点（邻居顶点）发送消息。消息在处理的过程中是作为一个边三元组参与计算的，这样可以让消息既携带起始顶点的特征又携带终止顶点的特征。没有收到任何消息的顶点会被忽略，进入未激活的状态，而这种状态会被收到的任意消息激活。当所有顶点都处于未激活状态时Pregel操作结束并返回最终的图。Pregel操作申明下所示：

def apply[VD: ClassTag, ED: ClassTag, A: ClassTag]

(graph: Graph[VD, ED],

initialMsg: A,

maxIterations: Int= Int.MaxValue,

activeDirection: EdgeDirection = EdgeDirection.Either)

(vprog: (VertexId, VD, A) => VD,

sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],

mergeMsg: (A, A) => A)

: Graph[VD, ED]

可以看出，Pregel操作有两个参数列表。第一个参数列表包含操作的图对象、初始消息、最大迭代数以及发送消息的边的方向；第二个参数列表包含一个由用户自定义的消息接收函数(vprog)、通讯函数(sendMsg)以及一个消息的合并函数(mergeMsg)。

Pregel函数的处理逻辑是先通过vprog函数接收收到的消息，并用这些消息更新节点的图信息。之后，更新后的图会根据计算函数和合并函数的逻辑产生新的消息。新消息产生后会被发送给相应的图节点（相邻节点）并进入下一次消息处理过程（当前超步结束，下一个超步开始）。下面用一个传播最大值的例子来对Pregel进行具体的说明。

图中的实线箭头表明了图的链接关系，而图中节点内的数值代表了节点的当前数值；每个节点有两种状态，活跃与不活跃；迭代是从通讯阶段（一个超级步转到下一个超级步）开始的。

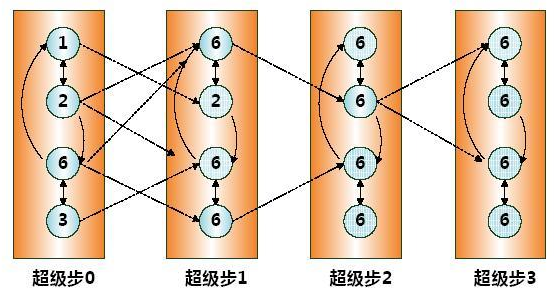


图15 pregel计算最大值

Fig.14 Find the Max Value Using Pregel

开始，超级步0沿着图的有向边将顶点数据传递给相邻的顶点，然后进入超级步1。接收到消息后超级步1开始启动同步阶段。每个定点合并所有接收到的消息并产生一条最终的消息（本例最终消息中为所有接收到消息的最大值）。之后超级步会进入计算阶段，对顶点的信息进行更新修改等。本例中会比较消息的值和顶点本身值的大小关系后选择性的更新替换顶点信息。更新完顶点信息后，每个顶点会将自身的值作为消息发送到相邻的顶点。到此超级步1结束，接收到消息的超级步2启动。

未接收到任何消息的节点会进去不活跃的状态。Pregel的迭代过程会在所有节点都进入不活跃状态或者迭代次数到达规定的最大值时结束。

# 4 CGP在GraphX中的实现

CGP与一般的进化程序不同。CGP节点间的关系以有向无环图而不是树的方式表示。这使得程序的输出可以有多个而不是单一的。另外，CGP还会重用中间结果。这些特性都使得CGP十分适合用作逻辑电路的设计，如加法电路、乘法电路、数字过滤等。本文便是基于这个特性在Spark上实现了一个11位多路选择电路以及一个1位的加法电路的分布式化演化程序。

1位全加器一共有3个输入位和2个输出位。输入由两个1位的加数和一个1位的进位构成。输出一个1位的结果和一个1位的输出进位构成。因此位加法器的真值表一共有=8个项，=16个输出。后面程序会根据这16个输出来计算每个个体的适应度。

11位多路选择器一共有11个输入位和1个输出位。11位加法器的真值表一共有=2048个项，=2048个输出。后面程序会根据这2048个输出来计算每个个体的适应度。

CGP中为了避免产生更加可怕、庞大的真值表，我们采用了一种压缩表的形式来表示真值表。表压缩的方式是在把一列中每32位看做32位的2进制数，然后将其换算成十进制来表示。比如，把S2列的前32项组合起来可以得到一个32位二进制的数00000000000000000000000011101000，将其换算成十进制可以得到一个数232。这个数便是S2在压缩真值表中的第一列。依照这种方法可以将1位全加器的真值表压缩成如下表3所示形式：

表3 1位加法器真值表

Table 3 Truth Table of 1 bit summator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入 | | | 输出 | |
| A1 | B1 | C | S1 | C0 |
| 240 | 204 | 170 | 232 | 150 |

同样，将11位选择器真值表的2048行11列按照每32行列压缩，可以得到一个2048/32=64行11列的以32位十进制数表示的压缩真值表。压缩表如下图17所示：

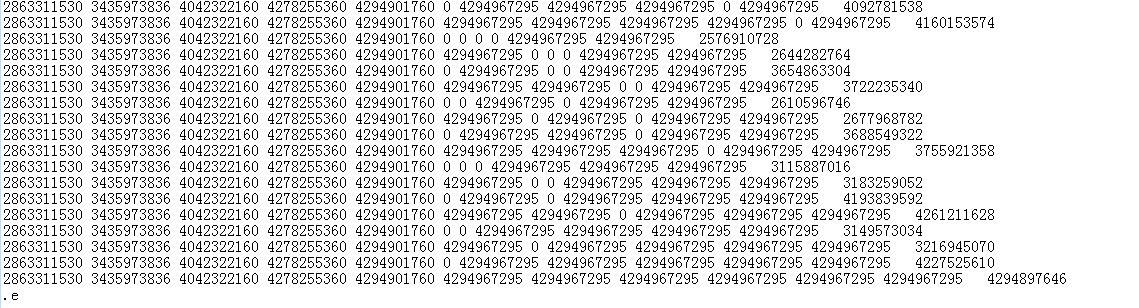
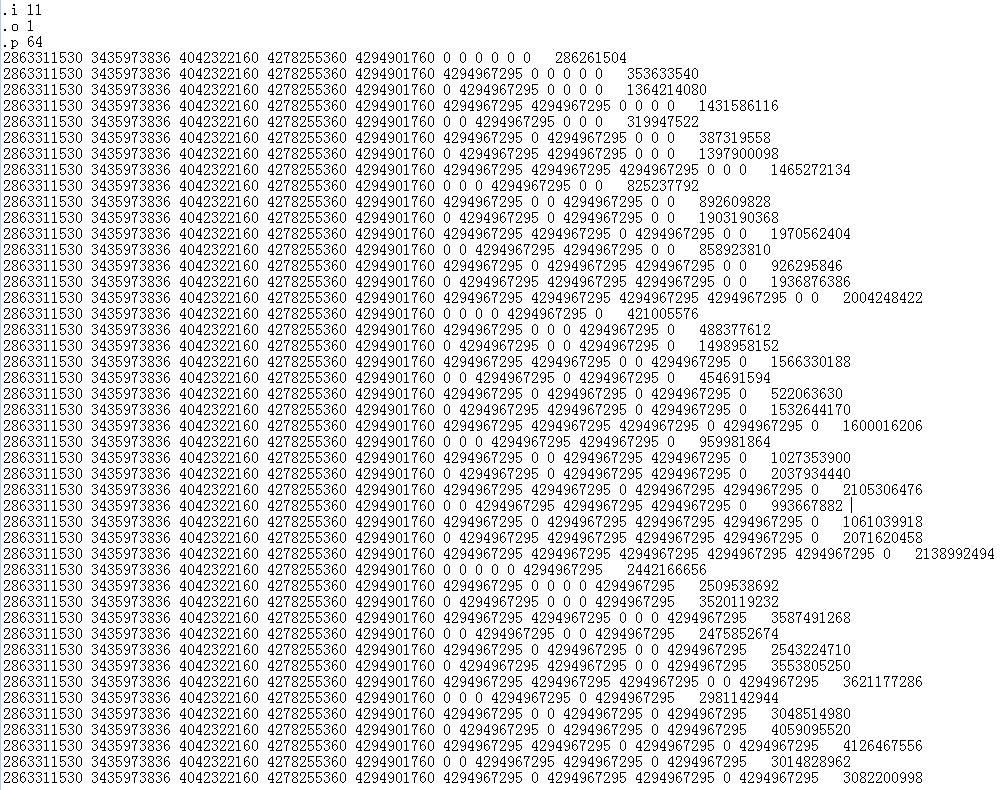


图 16 11位多路选择器真值表

Fig.16 Mux11 truth table

其中，每行前11个数代表选择器的输入，最后一个数代表选择器的输出。

## 4.1 程序采取的构图方式

本文实现的CGP逻辑在GraphX中是以图来表示的。图的结构如图17所示：

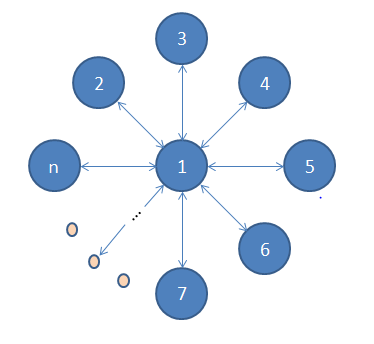


图 17 CGP图结构

Fig.17 CGP Graph Structure

图上每个节点都表示一个大小为5的种群。节点之间传递的消息为当前最优秀个体的信息。

按照Pregel操作的迭代方法。每个节点内首先需要将收到的信息与节点内信息比较，并更新节点。之后，程序依照用户定义的计算、合并函数遍历整张图上的节点，并对结果进行整合。整合出来的结果会被当作消息发送到对应的节点上。我们以图节点数为5的例子来进行解释。具体流程如下图18所示：

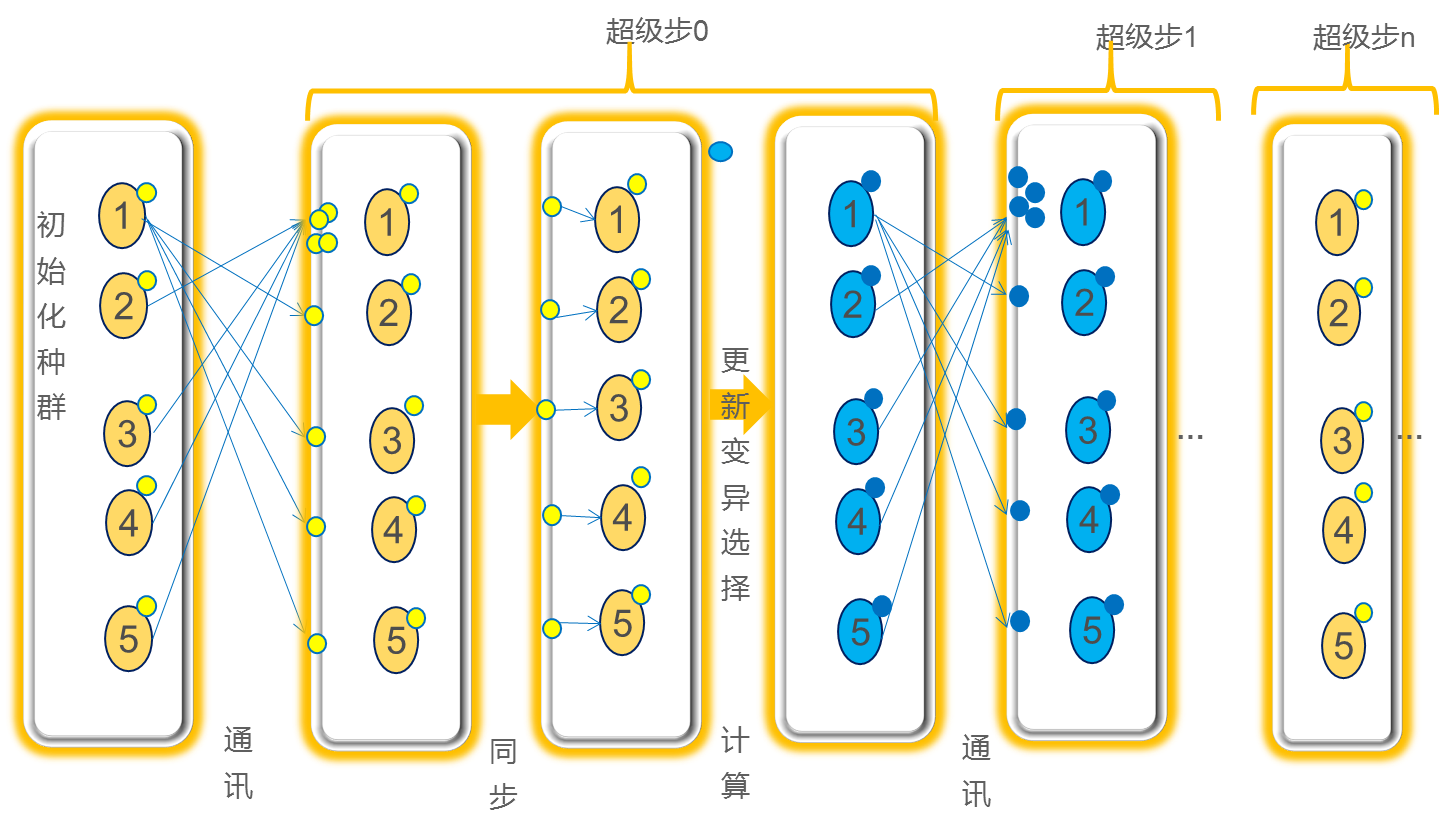


图 18 并行CGP的流程

Fig.18 the Procedure of Parallel CGP

与计算最大之的例子（图15）类似。图每个节点首先会初始化各自的种群，然后每个种群会向相邻种群发送消息（适应值最大的染色体），并进入超级步0。接收到消息的超级步0首先会进入同步阶段，即分别为每个节点的消息队列以聚集函数（mergeMsg）的逻辑进行聚集（比较所有消息染色体，仅留下适应值最大的那条），并产生出一条最终的消息。之后，超级步0的计算阶段开始启动。在计算阶段中，每个顶点会根据接收到的染色体来选择性的更新自己种群的信息。本例中是将消息染色体与种群最优染色体进行比较，用两者较优者替换种群的最优染色体。更新完成之后计算阶段开始对种群进行变异，即对最优的染色体进行单点变异生成余下的染色体。在这个过程中，种群之前的所有非最优染色体都会被替换。之后，每个节点都会对自己的新一代种群进行筛选，找出更新变异后的种群的最优的染色体。到这里为止，超级步0就接近了尾声。超级步0继续将顶点的信息按照边的连接关系发往相邻的顶点，然后程序结束超级步0转入超级步1。

当整张图中所有节点种群的最优个体的适应值都达到最大的时候，所有节点间都不会发送消息，即所有节点都进入不活跃状态。此时Pregel结束迭代，程序退出Pregel，将结果以图的形式返回。图中每一个顶点种群的最优的染色体都是CGP程序的结果。

## 4.2 程序的参数设置

程序的初始化参数保存在Data文件夹下的cgp.par的文件中。其样式如图19所示：

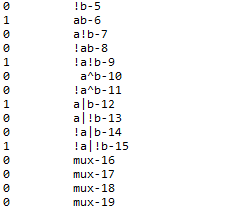
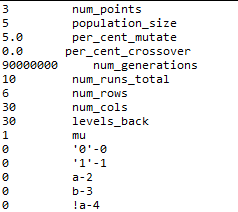


图 19 cgp.par文件

Fig.19 cgp.par file

重要程序参数含义：

num\_points: 图的节点数目

population：种群的规模

per\_cent\_mutate：基因变异的概率

num\_generation：种群培养的最大代数

num\_runs\_total：进化算法运行的次数

num\_rows：反馈阶段计算节点网格的行数

num\_cols：反馈阶段计算节点网格的列数

levels\_back：计算节点能够向前连接的最大列数

mu：用于生成子代的父母数目

另外，还需要获取2位全加器和11位多路选择器的压缩真值表。两张压缩表分表保存在Data文件夹下面名为add1c.plu和mux11.plu的文件中。内容见图17。

## 4.3 CGP代码框架

代码框架大致如下：

（1）配置程序参数（种群规模、变异概率、基因反馈阶段网络行列规模、操作基因功能列表、实现逻辑的真值表等）。

（2）生成图结构，并初始化每个节点的种群信息。

（3）进入Pregel(其中每一步都是分布执行的)：

1）接收：接收一共分为三步进行，更新、变异、选择。

更新：将接收到的个体（第一次迭代接收到的个体为默认的个体）与

种群最优个体比较，如果接收到的个体适应度更高，则用该个体替换

种群当前最优个体。

变异：根据事先设定好的变异概率（per\_cent\_mutate）对最优个体进

行变异，并产生一定数量的新个体。用这些新个体替换掉种群原有的

相应数量的劣势个体

选择：计算种群所有个体的适应度，比较后选出当前最优秀的个体。

2）通信：将最优个体的信息发送给相邻的所有种群。

3）合并：将每个种群收到的所有个体进行比较，筛选出适应度最高的个体并将该个体作为消息发送给对应的种群。

4）迭代：如果计数器未达到最大迭代次数，继续从步骤A)开始新的一轮迭代。否则Pregel结束。

## 4.4 实验结果与分析

1）与本地串行CGP(串行c程序)相比，当迭代次数较少，处理的数据量较低的时候分布式CGP(分布式spark/scala程序)的运行时间会相较慢很多。对比最大适应值只有16的一位全加器的串行CGP和分布式CGP的运行结果我们可以很清楚地得出这个结论。其中串行程序的种群规模为5，个体变异概率为1%；分布式程序图顶点个数为50，种群规模为5，个体变异概率为1%，种群间每1000代通信一次。两个程序的运行结果如下图20、21所示：

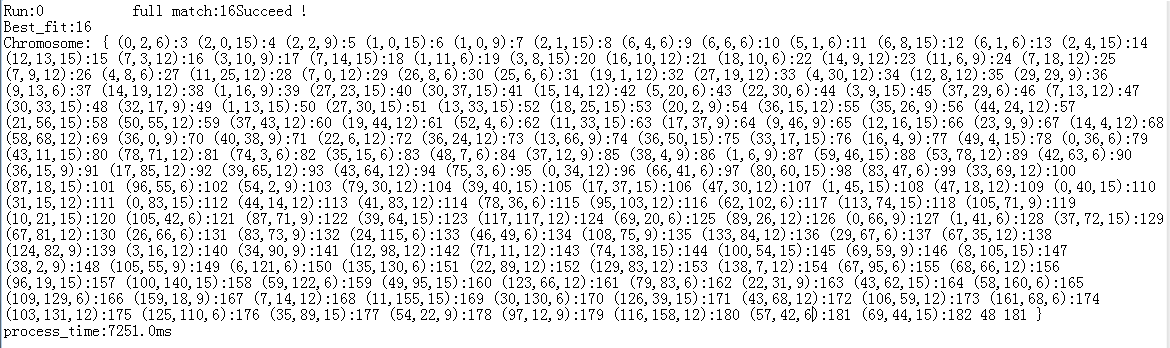


图 20 1位加法器分布式CGP运行结果

Fig.20 The result of add1 parallel CGP

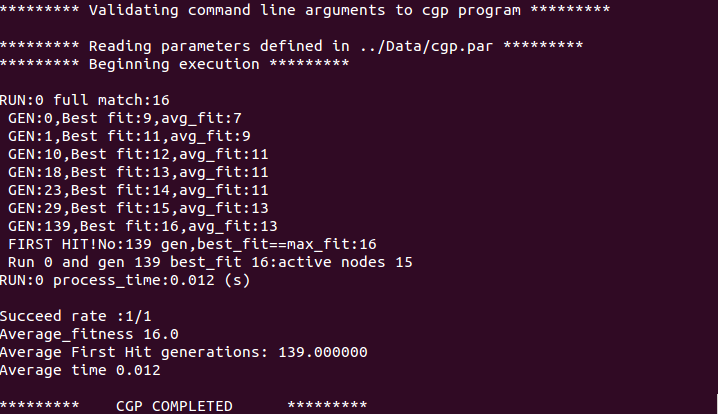


图 21 1位加法器串行CGP运行结果

Fig.21 The result of add1 serial CGP

图20是并行CGP设计的一个1位加法器的结果。这个结果向我们展示了两个程序的重要输出。第一个是程序搜寻到的解的构造（图中Chromosome：后面的基因序列），即满足1位加法器的计算逻辑的染色体的具体基因。第二个是程序的运行时间，即CGP找到正确解所花费的时间。

图21是串行CGP设计1位加法器的结果信息输出。因为本文的主要研究问题是如何通过分布式提高CGP的速度，因此在图21中我们主要关注程序的运行时间（图中的process\_time所示的时间）。

对比图20和图21后可以看出。在处理小规模如1位加法器的演化问题上，串行CGP（0.012秒）要远远快于分布式CGP（7.251秒）。

2)但是在规模极大，迭代次数极多的问题上。分布式CGP的速度就要远远高于串行的CGP，甚至用串行的CGP就根本无法解决。串行和分布式的11位选择电路的演化便是一个很好的例子。串行CGP的种群规模为5，变异概率为1%；分布式CGP图顶点为1000，种群规模为5，变异概率为1%，种群间每10000代通讯一次。程序的运行结果如图22、23所示：

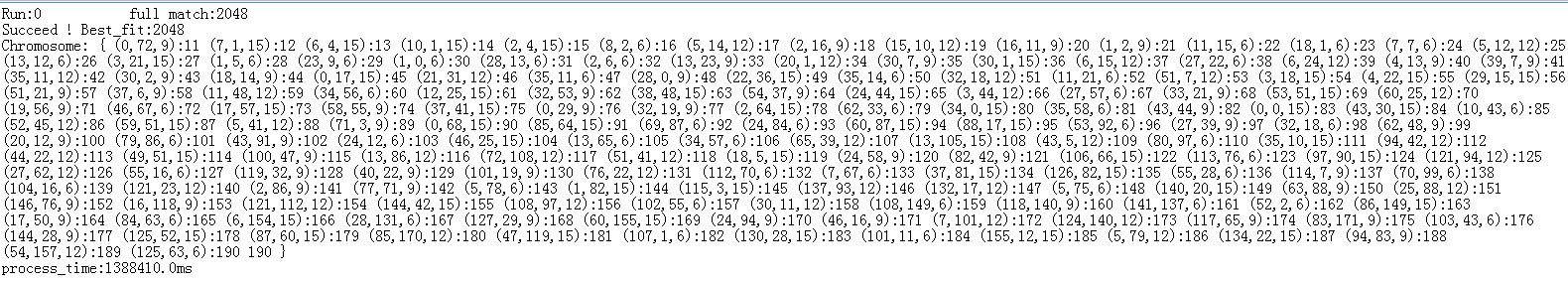


图 22 11位多路选择器分布式CGP结果输出

Fig.22 The result of mux11 parallel CGP

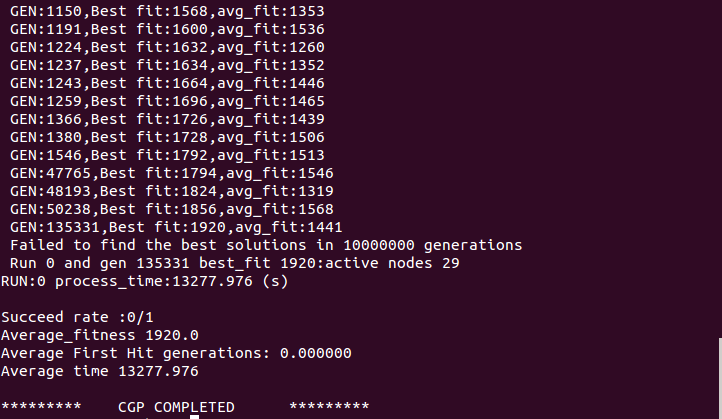


图 23 11位多路选择器串行CGP结果输出

Fig.23 The result of mux11 serial CGP

从11位多路选择电路的两个CGP运行结果可以看出。分布式CGP花了1388410ms=23.14min（图22所示）找出了最大适应度（2048）的解，但串行CGP花了13277.976s=3.5h（图23所示），变异了10000000代只找到了适应度为1920的解。由此我们可以得出，在处理大规模，迭代层次极深的问题时，分布式CGP的效率要远远高于串行的CGP。

3)通过改变节点间通讯的频率可以显著提高分布式程序效率。因为如果每产生一代新的种群不同种群间就进行信息交换的话会显著增加程序用户节点间的通信成本。为了测试因为种群通讯频率而带来的运行负担。我们将种群间每1000代通讯和每隔10000代通讯的情况进行对比，并保持两个并行程序图的顶点数都为1000，变异概率为1%，种群规模也斗为5。程序结果如图24、25所示：

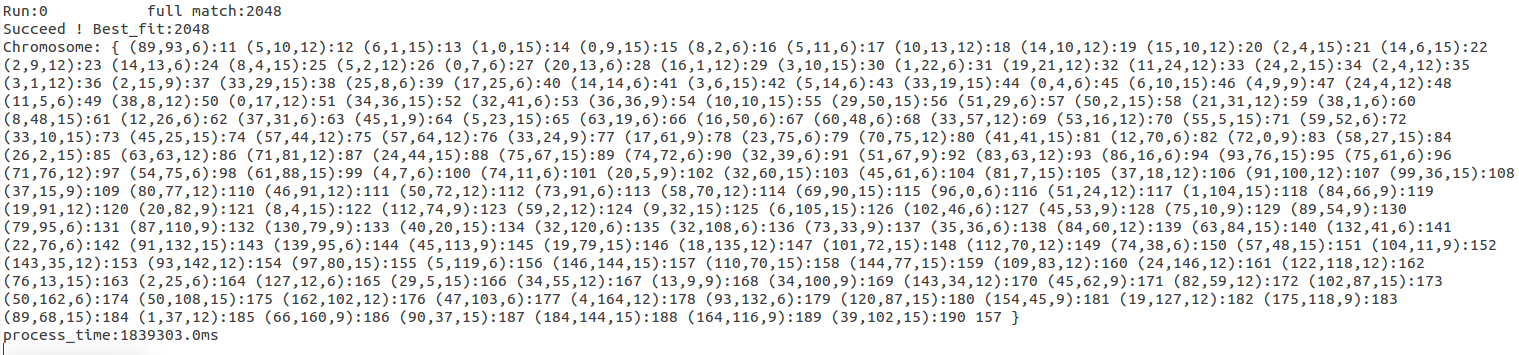


图 24 每1000代通讯的结果

Fig.24 The Result of communicating every 1000 generations

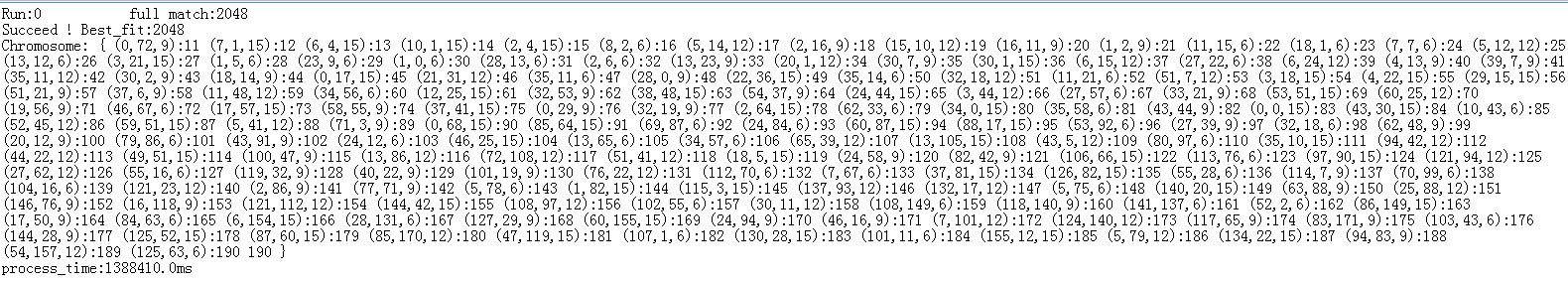


图 25 每隔10000代通讯的结果

Fig.25 The Result of communicating every 10000generations

从输出的结果我们可以看到。每隔1000代通讯花费1839383ms=30.5min（图24所示）找到解，每隔10000代通讯花费1388410=23.14min（图25所示）找到解。实验结果对比如表4所示。

表4 串行及并行CGP的结果对比

Table 4 Comparisions between the results of serial CGP and parallel CGP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 电路 | CGP | 种群  规模 | 图顶点数 | 变异  概率 | 通讯  代数 | 最佳适应度 | 最大适应度 | 运行时间 |
| 1位加法器 | 串行 | 5 | — | 1% | — | 16 | 16 | 0.012s |
| 并行 | 5 | 50 | 1% | 1000 | 16 | 16 | 7.251s |
| 11位多路选择器 | 串行 | 5 | — | 1% | — | 1920 | 2048 | 3.5h |
| 并行 | 5 | 1000 | 1% | 10000 | 2048 | 2048 | 23min |
| 并行 | 5 | 1000 | 1% | 1000 | 2048 | 2048 | 30min |

# 5 总结与展望

## 总结

本文的主要研究内容及创新之处如下：

（1）基于pregel模型在GraphX下首创实现了分布式CGP

（2）针对分布式的进化算法设计了星型的通信拓扑结构

（3）本文提出的分布式CGP有效地克服了进化算法后期容易陷入局部最优的问题。

## 5.2 展望

（1）本文只尝试了对笛卡尔遗传规划算法在Spark平台上使用GraphX框架

进行分布式处理。但是Spark平台还提供了许多其他的分布式处理方式，比如流式处理、机器学习等方式来更高效地处理数据。这些方法都值得深入研究来提供更多、更有效的分布式思路。

（2）本文中的分布式CGP中使用了星型的计算图模型。使用不同的图结构

对分布式程序会产生不同的影响，效率会有很大差别。因此研究如何针对特定的问题构造高效的图结构也将十分有价值。

（3）CGP广泛应用于机器学习，神经网络，人工智能，数据挖掘，金融预

测，函数优化，分类，电子电路设计等领域。使用Spark平台将这些大规模数据的串行程序分布化对提升这些领域工作的能力有很大的潜力。

# 参考文献

1. 柏磊. 数字电路进化设计算法研究[D].南京理工大学,2012.
2. 陈鹏. 基于BSP模型的图计算预处理研究[D].电子科技大学,2015.
3. 段剑峰. 基于Spark的大规模图数据分布式计算研究[J]. 现代计算机(专业版),2016,07:44-46+64.
4. 丁鑫. 基于BSP模型的分布式图计算系统性能优化研究[D].复旦大学,2014.
5. 李爱婷. 基于CGP的数字电路进化算法研究[D].河北师范大学,2014.
6. 李爽. 基于Spark的数据处理分析系统的设计与实现[D].北京交通大学,2015.
7. 李文栋. 基于Spark的大数据挖掘技术的研究与实现[D].山东大学,2015.
8. 梁彦. 基于分布式平台Spark和YARN的数据挖掘算法的分布式化研究[D].中山大学,2014
9. 唐振坤. 基于Spark的机器学习平台设计与实现[D].厦门大学,2014.
10. 张晓琳,郭彦磊,王静宇,张臣,张文超. 一种基于Pregel-like的社会网络隐私保护方法[J]. 计算机应用研究,2016,09:1-6.
11. Akidau T, Balikov A, Bekiro06lu K, et al. MillWheel: faulttolerant stream processing at Internet scale[J]. In Proceedings of the 39th International Conference on Very Large Data Bases (VLDB, 2013, 6(11).
12. Bryan Perozzi,Christopher McCubbin,J. T. Halbert. Scalable graph clustering with parallel approximate PageRank[J]. Social Network Analysis and Mining,2014,41:.
13. Branislav Kadlic,Ivan Sekaj,Daniel Pernecký. Design of Continuous-time Controllers using Cartesian Genetic Programming[J]. IFAC Proceedings Volumes,2014,473:.
14. Dean J. and S.Ghemawat, MapReduce:simplified data processing on large clusters[J]. Communications of the Acm, 2008, 51(1):147–152.
15. Hrbacek R, Sekanina L. Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation[J]. genetic programming, 2014:1015-1022.
16. Liu Y, Tempesti G, Walker J A, et al. A Self-scaling Instruction Generator Using Cartesian Genetic Programming[J]. Lecture Notes in Computer Science, 2011:298-309.
17. Low Y, Gonzalez J, Kyrola A, et al. Distributed GraphLab: A framework for machine learning and data mining[J]. in the cloud,” Proceedings of the VLDB Endowment, 2012:716--727.
18. James Alfred Walker,Katharina Völk,Stephen L. Smith,Julian Francis Miller. Parallel evolution using multi-chromosome cartesian genetic programming[J]. Genetic Programming and Evolvable Machines,2009,104:.
19. Miller J F, Thomson P. Cartesian Genetic Programming[J]. IEEE Transactions on Evolutionary Computation, 2008, 10(2):2701-2726.
20. Mauro N D, Basile T M A, Ferilli S, et al. An Exhaustive Matching Procedure for the Improvement of Learning Efficiency[J]. Lecture Notes in Computer Science, 2003:112--129.
21. Mihai Oltean. Evolving Evolutionary Algorithms with Patterns[J]. Soft Computing,2007,116:.
22. Yu T, Miller J. Finding Needles in Haystacks Is Not Hard with Neutrality[J]. Lecture Notes in Computer Science, 2002:13--25.

# 致谢

首先我需要感谢倪福川老师。在整个毕业设计的过程中，倪老师给我提供了许多的学习资源，让我了解Spark分布式程序、算法原理、graphx等。此外，我还需要感谢同组的李诺同学。我俩研究方向相近，经常可以一起讨论实验有关的概念和算法的难点。通过这次毕设我接触到了spark，并由此对分布式产生的浓厚的兴趣。也因为这次经历，让我更加坚定地在分布式的道路上一直走下去。

# 附录

本文使用的是搭载在Hadoop2.6.0的hdfs上的Spark1.2.0的分布式集群。集群的部署因此主要分为两步：搭建Hadoop2.6.0；搭建基于Hadoop的Spark分布式平台。

（1）Hadoop集群搭建:

1. 集群环境
2. 机器：六台物理机
3. linux版本：Ubuntu14.04 LTS
4. jdk版本：执行命令 pc@master:~$ java –version 查看。如图26所示：

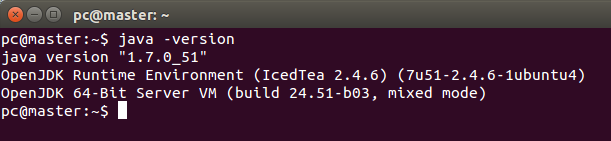


图 26 查看jdk版本

Fig.26 jdk version

1. 集群节点：一个Master(master 172.16.124.13)

五个Woker(slave1 172.16.124.14

slave2 172.16.124.15

slave3 172.16.124.16

slave4 172.16.124.17

slave5 172.16.124.18)

1. 准备工作
2. 安装Java jdk和scala
3. ssh免密码登录：
4. 下载Hadoop2.6.0（<http://mirror.bit.edu.cn/apache/hadoop/common/>）

三、安装Hadoop

1、解压Hadoop压缩文件到指定目录（本文解压到~/spark/Hadoop2.6.0目录下）

2、按照官方文档介绍配置关键Hadoop配置文件。修改用户环境变量/etc/profile.如图27所示：

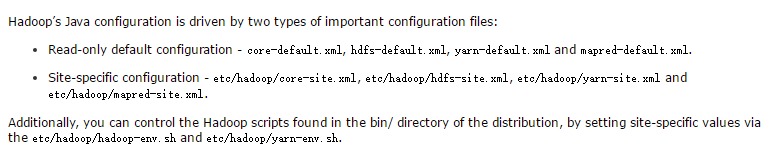


图 27 Hadoop集群搭建需要配置的文件

Fig.27 Hadoop’s important configuration files

3、写完配置文件后将Hadoop（~/spark/Hadoop2.6.0）整个文件拷贝到所有Worker节点上并保证路径相同。

4、第一次启动Hadoop的时候要在每台机器上格式化namdenode。如图28所示：

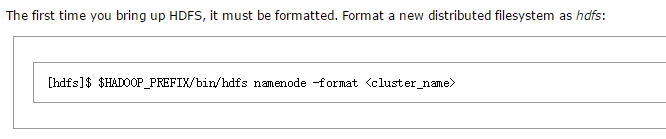


图 28格式化namenode

Fig.28 namenode format

注意：每次重新配置Hadoop集群的时候都需要格式化namenode,并且在格式化namenode之前需要删掉所有机器上datanode,namenode,tmp文件夹下的内容。

1. 启动集群namenodes和datanodes。如图29所示：

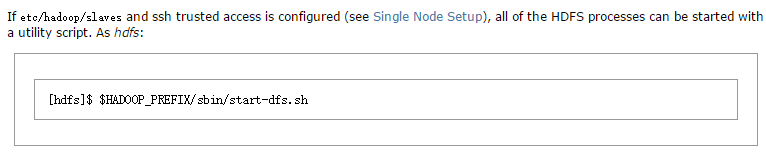
图 29使用start-dfs shell脚本启动集群namenodes和datanodes

Fig.29 using start-dfs.sh to start namenodes and datanodes

注意：使用start-dfs启动dfs需要事先配置好无密码ssh登录。每次机器关机重新启动之后必须先source /etc/profile, 并将/etc/hosts里面的最后一条本地127.0.1.1的信息删除。否则会出现slave节点上的datanode无法连接到master的问题。

1. 启动yarn资源管理器。如图30所示：

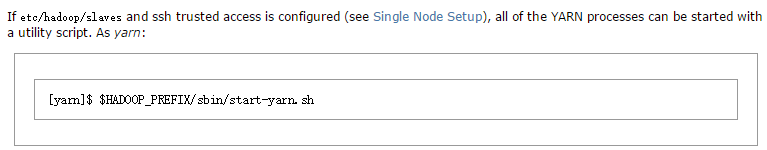


图 30启动yarn资源管理器

Fig.30 start yarn

注意：Start-yarn同样需要实现配好无密码ssh登录，否则会报错master无法连接。

1. 查看hadoop集群启动情况：

打开浏览器输入： master:8080查看集群。如图31所示：

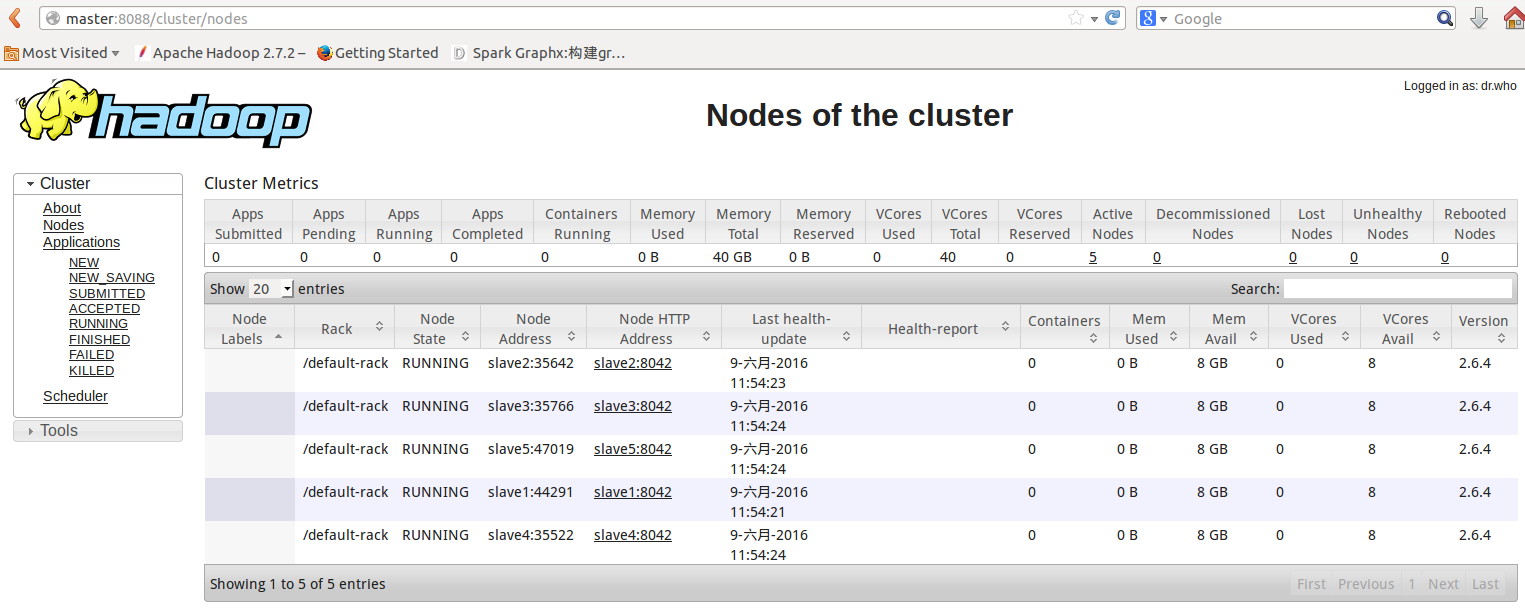


图 31查看Hadoop集群

Fig.31 Chek Hadoop

打开浏览器输入：master:50070查看namenode节点以及相应的datanode节点。如图32、图33所示：

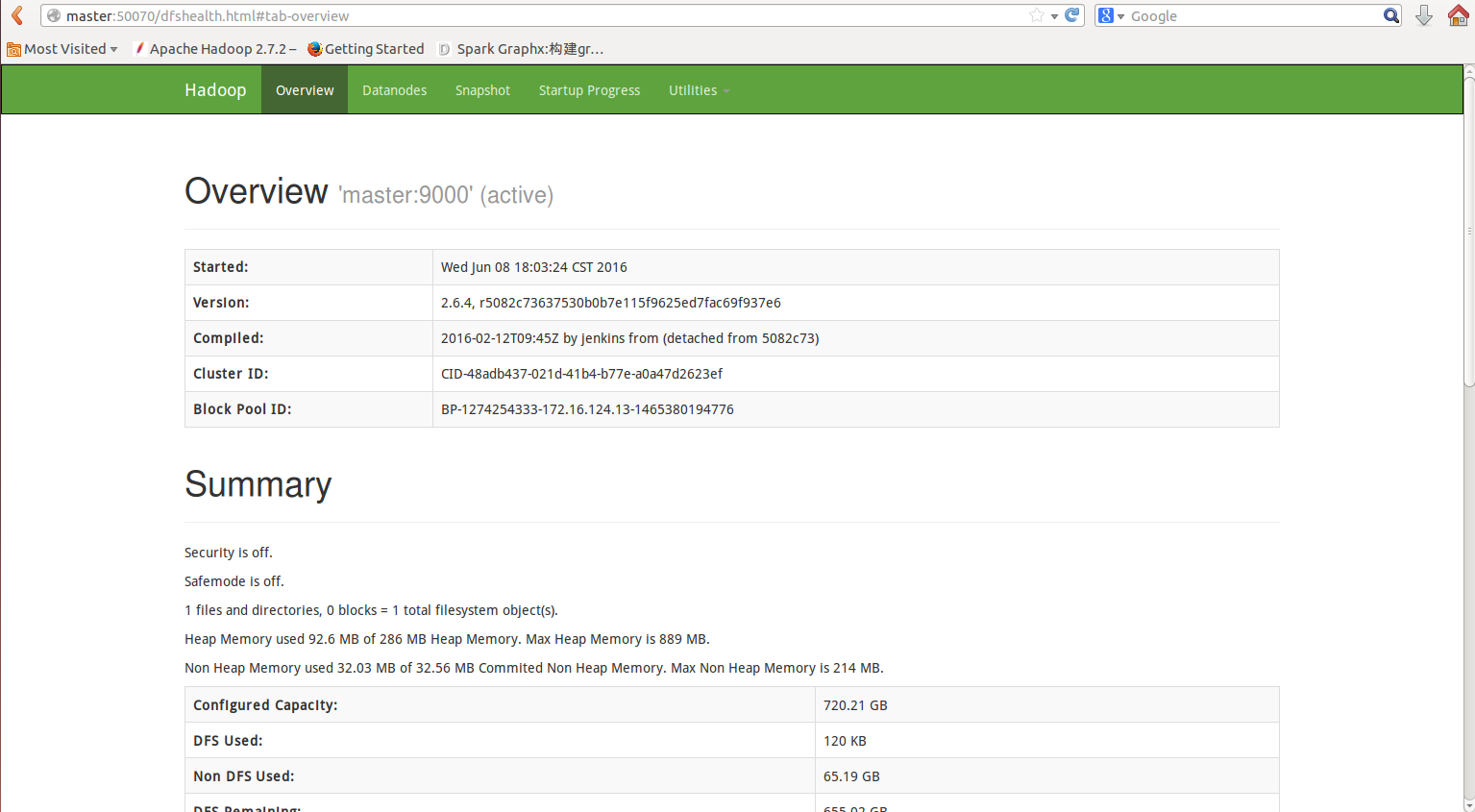


图 32查看namenodes节点

Fig.32 Check namenodes

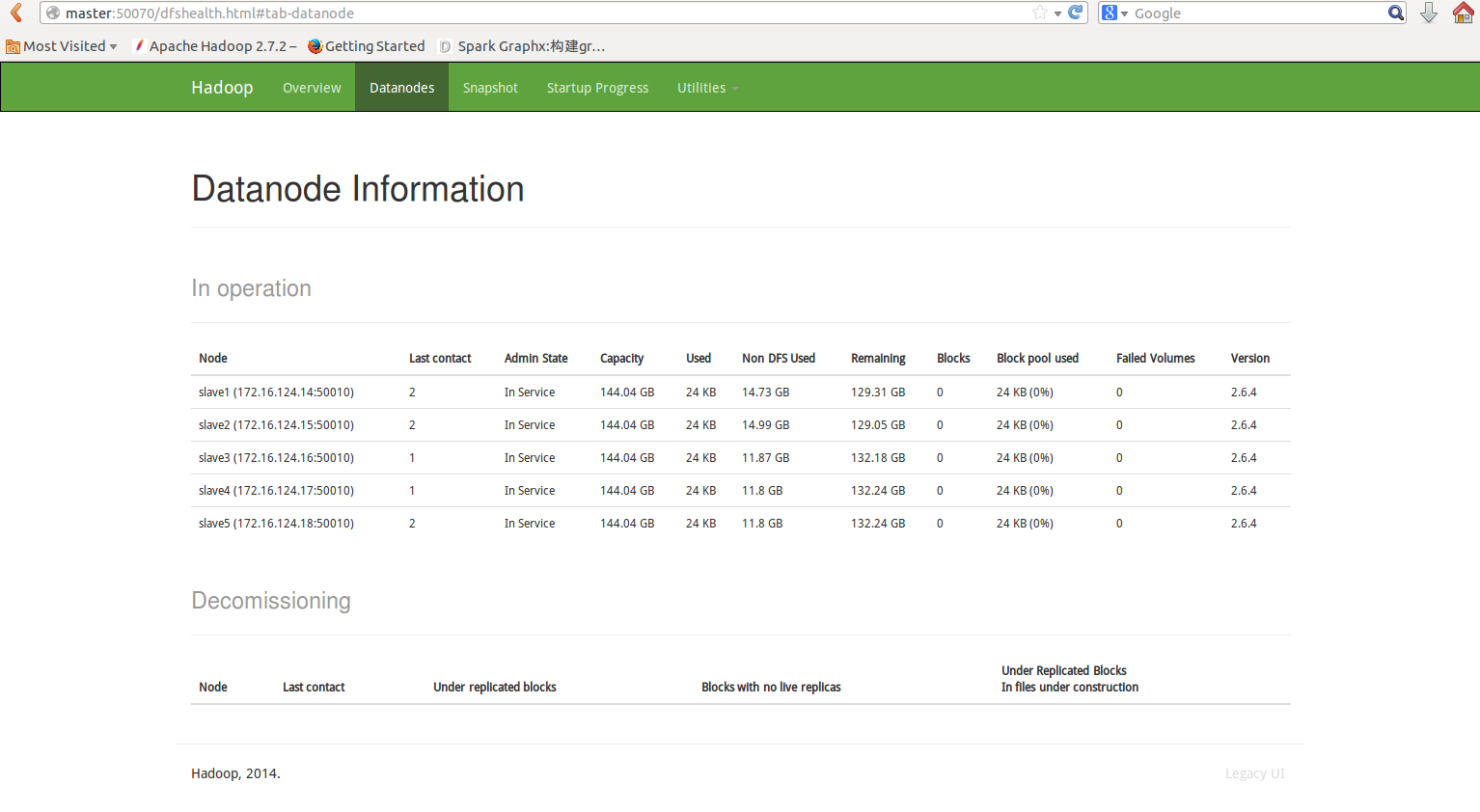


图 33查看datanodes节点

Fig.33 Check datanodes

8、使用Hadoop hdfs输入shell命令：./bin/hadoop fs –put <文件本地路径><文件上传路径>可以上传文件到hdfs。上传后效果如图34所示：

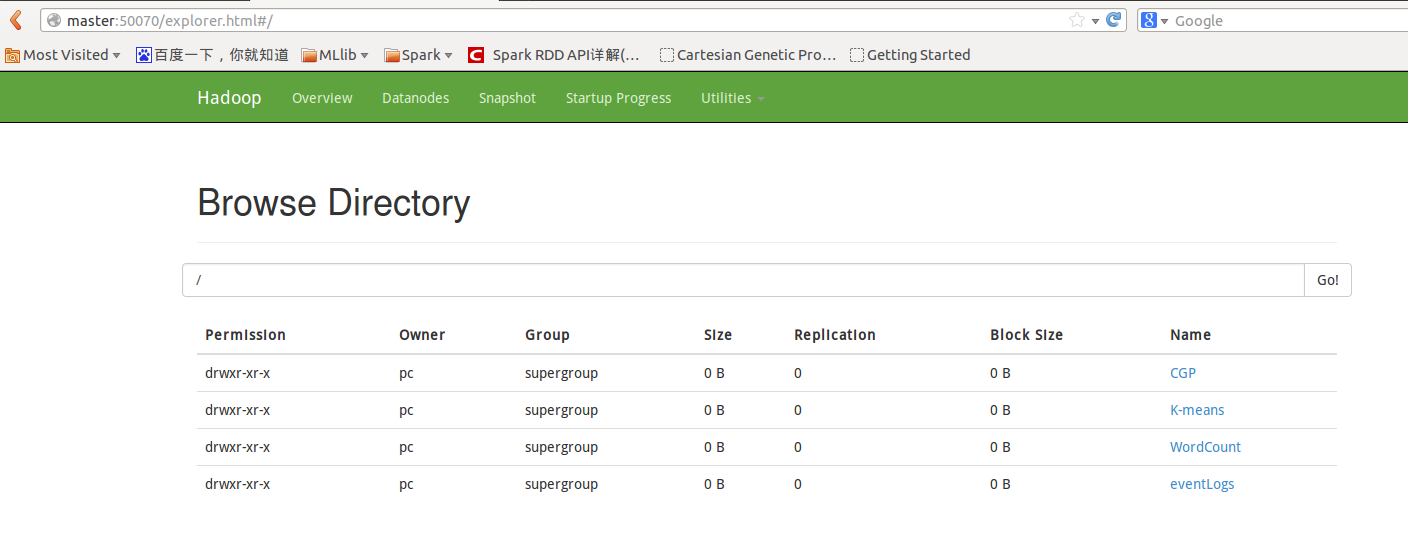


图 34 上传文件到hdfs

Fig.34 upload files to hdfs

（2）Spark集群搭建：

搭建好并启动Hadoop集群后便可以开始配置Spark。

1. 准备工作
2. 安装scala2.10.6：安装成功之后查看scala版本pc@master:~$ scala –version，显示结果如图35所示：

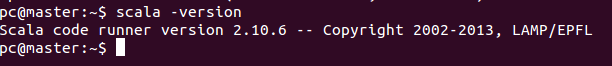


图 35查看scala版本

Fig.35 scala version

1. 下载spark-1.2.0-bin-hadoop2.4.tgz
2. 安装Spark
3. 解压spark-1.2.0-bin-hadoop2.4.tgz压缩文件到指定目录（本文解压到~/spark/spark-1.2.0-bin-hadoop2.4目录下）。
4. 修改用户环境变量/etc/profile；在spark-env.sh中配置spark环境变量；向slave文件中添加slave节点部署机器的IP地址。
5. 执行shell命令 ./sbin/start-all.sh 启动Spark集群。注意，在启动Spark之前，Hadoop的hdfs得先启动。
6. 页面查看集群启动情况：

打开浏览器输入：master:8080（Spark WebUI）。查看spark集群。效果如图

36所示：

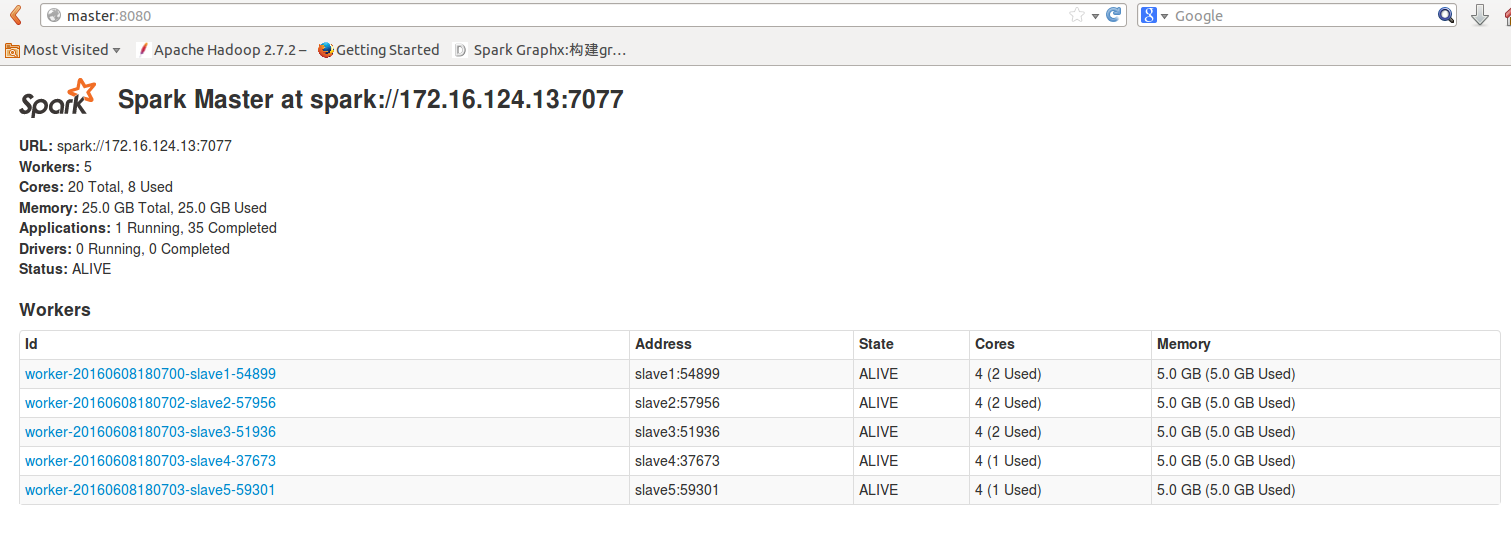


图 36 查看Spark集群的master 和 workers节点

Fig.36 Check master and workers

1. 启动spark-shell进入REPL模式。开始Spark/scala程序的编写。

输入shell命令启动spark-shell: pc@master:~$ ./bin/spark-shell。效果如图37所示：

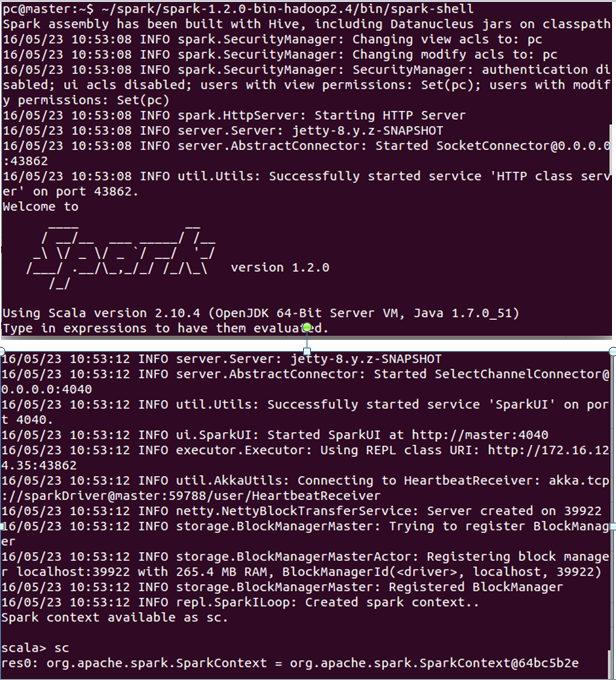


图 37 启动spark-shell开始Spark/scala编程

Fig.37 launch spark-shell