MapReduce 下连续泊位分配系统①

贾理国, 杨智应

(上海海事大学 信息工程学院, 上海 201306)

摘 要: 针对传统串行环境下码头连续泊位分配算法在船数大于七条时, 算法的执行效率明显降低、资源占用量显著增加. 首次设计了基于分布式环境下连续泊位分配系统总体架构和软件架构; 提出了基于 Hadoop 平台下连续泊位分配系统开发环境的搭建, 分布式环境下码头泊位分配系统组件设计与部署以及分布式环境下基于 MapReduce 改进连续泊位分配算法的关键技术; 最后实现了分布式环境下连续泊位分配系统. 实验结果表明, 该系统可以有效的提高连续泊位分配问题的执行效率.

关键词: 泊位分配; MapReduce; 并行计算; 单纯形算法

System for Continuous Berth Allocation in MapReduce

JIA Li-Guo, YANG Zhi-Ying

(Information Engineering College, Shanghai Maritime University, Shanghai 201306, China)

Abstract: According to the consecutive berth allocation algorithm of container terminals in the traditional serial environment, the efficiency of the algorithm is significantly reduced and the resource consumption is greatly increased when the number of ships greater than seven. The paper designs the overall system architecture and software architecture of the continuous berth allocation system of container terminals for the first time based on the distributed environment; puts forward the development environment construction of the continuous berth allocation system of container terminals based on the Hadoop platform, its component design and deployment as well as the key techniques of improved continuous berth allocation algorithm in distributed environment based on MapReduce. In the end, the continuous berth allocation system of container terminals in distributed environment is realized. The experimental results show that the system can effectively improve the efficiency of the continuous berth allocation.

Key words: berth allocation; MapReduce; parallel computing; simplex algorithm

Hadoop 提供一种对海量数据进行分布式并行处理的软件框架, 包括 Hadoop 分布式文件系统(Hadoop Distributed File System, HDFS)模块和 MapReduce 模块两大核心模块, 具有高可靠性、高扩展性、高容错性、低成本等良好特性[1,2]. 企业和用户通过 Hadoop 中的 MapReduce 分布式程序设计, 有效提高并行环境下算法的执行效率.

集装箱码头业务的快速发展使得码头连续泊位分配问题更加复杂化. 集装箱码头连续泊位分配是指码头调度人员根据优化策略以及泊位和岸桥是否空闲等约束条件为到港的船舶指定靠泊位置并最小化船舶在港时间. 但是随着航运业务的迅速发展, 到港船舶数量的增加, 码头吞吐量剧增, 传统泊位调度优化策略在解决大于七条船时计算效率明显降低, 需要一个提升泊位调度策略算法计算效率并且提供高效、稳定服务的技术平台. Hadoop 平台下 Mapreduce 分布式计算技术求解泊位分配调度策略算法成为必然的趋势.

国内外学者首先针对传统码头连续泊位分配进行了相关研究, Lim[3]基于船舶从靠泊到离开期间靠泊位置保持不变的假定, 定义了以任意时间码头泊位利用

① 基金项目:国家自然科学基金(61202021)

收稿时间:2016-02-27;收到修改稿时间:2016-03-31 [doi:10.15888/j.cnki.csa.005419]

率最大为目标的最优化码头泊位计划问题(Optimization Berth Planning Problem, OBPP), 通过将连续泊位方式规约到二维装箱问题, 证明其为 NP-难问题, 并提出启发式算法, 但当船舶数量不断增大时该算法无法提供高效的泊位分配计算. Guan[4]基于启发式算法提出与二叉树相结合的集装箱码头泊位分配算法, 以最小化带权重模型计算泊位分配完成时间为目标函数. 但对具体船舶作业分配细节没有有效描述, 尤其是泊位分配约束条件的约定. 杨春霞[5]建立了最小化船舶在港时间和集装箱码头生产成本的优化模型, 采用 Pareto[6]分级评价适应度, 并提出多目标遗传算法求解该模型的策略. 基于上述策略, 陈雪莲[7]提出兄弟-儿子方法对船舶位置进行调整, 并验证了泊位分配模型和算法的有效性. 刘莉莉[8]采用 Java 语言实现了基于单纯形算法的分支定界算法求解连续泊位分配的系统, 该系统可以获得连续泊位分配的最优解. 但当船舶数量大于七条时, 系统存在计算效率明显降低, 资源消耗增加等性能缺陷.

随着到泊船舶数量的增加, 系统出现算法处理能力差、运营成本高、负载均衡能力差, 系统易宕机等缺点更加凸显, 因此 Hadoop 平台下的 MapReduce 分布式计算技术逐渐被引入到求解泊位分配系统的设计与研发. Tsutsui[9]提出了蚁群优化算法[10](Ant Colony Optimization, ACO)的并行化计算平台, 采用多核处理器快速找到可接受的解决方案. Wu[11]在 Tsutsui 的基础上提出基于 MapReduce 的方法, 采用基于 ACO 算法和可建模的 MapReduce 框架, 描述了在 Hadoop 上的算法设计. 王诏远[12]提出运用云计算[13]下 MapReduce 并行计算过程来保证对硬件故障容错的能力, 改进 MapReduce 并行计算框架, 并在此基础上实现云计算下 MapReduce 蚁群优化算法. 随着 MapReduce 并行计算框架的发展, MapReduce 框架下 map()和 reduce()[14]函数极大的减少传统分布式程序设计中用户需要关注的数据分片、节点通信、数据传输等细节, 从而快速实现分布式程序设计, 为 Hadoop 平台下应用程序的开发提供硬件支持和负载均衡策略. 李建锋[15]以 MapReduce 为计算框架, 设计并实现了云计算下基于改进遗传算法的任务调度算法. Abhishek[16]通过将 MapReduce 框架整合到 Java 平台上, 实现了 eclipse 环境下使用 MapReduce 实现遗传算法.

目前连续泊位分配问题都是针对传统串行环境下构建的系统, 对云计算下基于分布式环境的连续泊位分配系统的研究仅给出了概念模型, 并未涉及具体的设计与实现. 针对以上不足, 本文将连续泊位分配系统与 MapReduce 并行计算框架结合, 设计了分布式环境下连续泊位分配系统的总体架构和软件架构; 提出基于 Hadoop 平台下连续泊位分配系统的搭建, 分布式环境下连续泊位分配系统组件设计与部署和分布式环境下基于 MapReduce 改进连续泊位分配算法的关键技术; 最后实现分布式环境下连续泊位分配系统的数据建模和模型求解.

1　分布式连续泊位分配系统架构设计

1.1 设计思想

本文的设计思想是将港口泊位分配系统部署在 Hadoop 分布式平台上, 使得港口泊位分配系统通过 Hadoop 平台下 MapReduce 框架的分布式计算和存储能力, 及时对连续泊位分配模型求解从而获取泊位分配方案, 通过分布式平台下多台 Linux 系统计算泊位分配模型, 有效降低单机求解泊位分配模型的压力, 提高算法的即时性. 本文将 Hadoop 平台下连续泊位分配系统的总体架构分为四层: 系统资源提供层, MapReduce 数据处理层, 算法计算层, 泊位系统应用层. 其中, 系统资源提供层为基于 Windows 平台下构建于 VMware 虚拟机上的三台 Linux 系统所构成的分布式系统环境; MapReduce 数据处理层架构在分布式资源提供层上, 可以有效的提供分布式系统所需的计算和存储能力; 算法计算层通过 MapReduce 数据处理层的数据计算实现了基于单纯形算法的分支定界法的分布式求解; 泊位系统界面应用层通过与算法计算层交互显示泊位分配模型求解结果. 分布式环境下港口泊位分配系统的总体架构设计如图 1 所示.

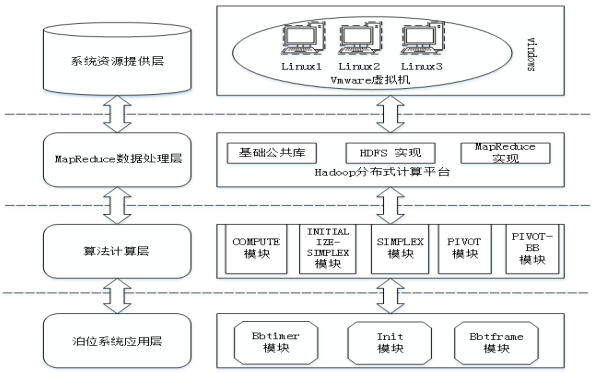

图 1　分布式环境下连续泊位分配系统的总体架构

在图 1 中, 第一层, 系统资源提供层为分布式连续泊位分配系统的开发提供硬件支持. 第二层, MapReduce 数据处理层提供基础公共库、HDFS 实现和 MapReduce 实现; 第三层, 算法计算层分为 5 个模块, COMPUTE 模块实现泊位分配模型构建线性规划方程, INITIALIZE-SIMPLEX 模块实现线性规划的一个初始可行解, SIMPLEX 模块描述线性规划的一个最优解, PIVOT 模块提供一次转动形式化主元的分布式计算过程, PIVOT-BB 模块实现基于单纯形算法的分支定界法; 第四层, 泊位系统应用层设计了 Bbtimer 模块用于计算算法的运行时间, Init 模块实现泊位分配系统数据读入,Bbtframe 模块提供泊位分配系统的结果展示界面.

通过以上设计, 可以将泊位分配系统部署在 Hadoop 平台上, 使用 MapReduce 对泊位分配算法进行分布式计算, 能有效提高连续泊位分配系统的计算和存储能力, 更好的处理大于七条船的泊位分配情况, 降低系统对硬件设备的依赖.

1.2 分布式连续泊位分配系统的软件架构设计

基于上述设计思想, 本文将连续泊位分配系统部署在 Hadoop 分布式平台上, 通过 Hadoop 平台提供的基础公共库、HDFS 分布式文件存储以及 MapReduce 并行计算框架, 实现分布式环境下连续泊位分配系统.

MapReduce 分布式计算框架, 提供分布式编程模型和运行时环境. 用户通过调用编程模型提供的编程接口实现一个分布式程序, 而节点间通信、数据切分由 MapReduce 运行时环境来完成. 本文使用 Eclipse 作为开发工具搭建开发环境, 设计并实现基于 Hadoop 的 MapReduce+Javase+Javaswing 的分布式连续泊位分配系统. 其中, MapReduce 实现泊位分配模型求解中单纯形算法的分布式计算, Javaswing 实现系统界面的设计以及结果展示, Javase 实现系统模型以及算法计算. 基于 Hadoop 分布式连续泊位分配系统的软件架构设计如图 2 所示.

在图 2 中, 基于 Hadoop 分布式连续泊位分配系统软件架构包括泊位系统界面应用层, 逻辑业务层, MapReduce 并行计算层和分布式平台硬件提供层. 在软件架构中, 泊位数据在用户层获得解析, 泊位分配结果的响应通过前端接口与逻辑业务层进行交互. 逻辑业务层通过实现基于单纯形算法的分支定界法对泊位分配模型求解, MapReduce 并行计算层使用 MapReduce 并行计算框架对逻辑业务层中单纯形算法进行分布式求解, 分布式计算所需的硬件则由硬件平台层提供的三台 Linux 虚拟机所提供. 基于 Hadoop 分布式港口泊位分配系统的软件架构设计分为 4 层, 如图 2.

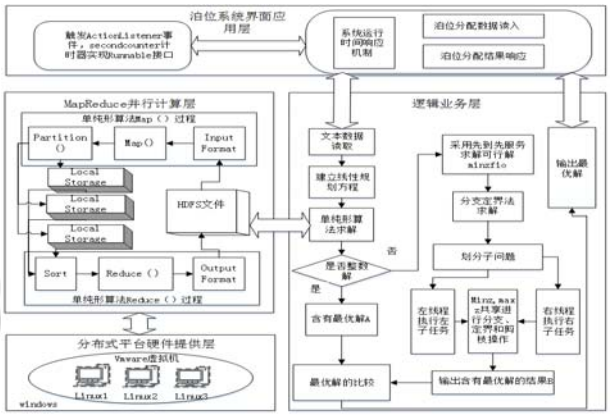

图 2 基于 Hadoop 分布式泊位分配系统软件架构设计

(1) 泊位系统界面应用层

泊位系统界面应用层提供 Init 类生成程序运行的界面, 并显示船舶的作业量、船长、到港时间等信息, 同时触发 ActionListener 事件创建 Timecounter 对象进行计时. Paint 方法提供坐标的绘制, Repaint 方法实现图形化实验结果.

(2) 逻辑业务层

逻辑业务层在 Compute 类中 Plan 方法实现泊位分配的模型的线性规划. Btime 类实现了单纯形算法, 具体包含 Pivot 过程的分布式计算方法、Initialile_Simplex 方法、Simplex 方法. Branch 类中 Simplex_BB 方法实现了基于单纯形算法的多线程分支定界算法, Simplex_BB 方法中采用先到先服务的原则计算全为整数的 minz 值, 通过一次分支定界法, 分出左、右两个子问题, 每个子问题交给不同的线程进行处理, 通过线程阻塞更新共有区域的 minz,maxz 值. 比较子问题的最优目标函数, 进行剪枝操作. 循环迭代, 最终求出最有整数解.

(3) MapReduce 并行计算层

MapReduce 并行计算层使用 MapReduce 并行计算框架对泊位分配问题中的单纯形算法进行分布式计算, 具体包括单纯形算法 Map 和单纯形算法 Reduce 过程. 在 Map 中, 先将单纯形算法分割并解析成对应的 key/value 对, 依次调用基于单纯形算法的自定义 map()

函数处理并等待 Reduce 的处理; 在 Reduce 中, 读取 Map 的中间数据, 按 key 对 key/value 对排序, 通过读取排序后的<key, value list>并调用基于单纯形算法的自定义 reduce()函数处理, 最终结果存放 HDFS 中.

(4) 分布式平台硬件提供层

分布式平台硬件提供层中包含一个基于 Windows 平台下的 VMware 虚拟机环境, 在 VMware 中安装 Linux1、Linux2 和 Linux3 三台虚拟 Linux 环境. 通过配置 ssh 无密钥访问以及 Hadoop 平台的文件配置, 从而搭建 Hadoop 分布式平台, 为 MapReduce 并行计算层供 MapReduce 并行计算框架和运行所需的环境.

2 分布式环境下泊位分配体统关键技术

2.1 基于 **Hadoop** 平台泊位分配系统开发环境搭建

由于 Hadoop 平台提供对数据进行分布式并行处理的软件框架, 具有高效性和有效的负载均衡机制且提供强大的分布式数据并行处理, 因此对于分布式环境下连续泊位分配系统的设计和实现显得尤为重要. Hadoop 平台下的 MapReduce 并行处理框架支持 Java 开发环境并为其提供不同功能的应用程序编程接口. 使用 Hadoop 平台下 MapReduce 并行处理框架开发基于 Java 分布式连续泊位分配系统如同开发传统基于串行环境下连续泊位分配系统应用程序一样, 需要安装基于 Hadoop 平台下 Eclipse 开发环境, 并且在 Eclipse 环境下添加 MapReduce 插件. 分布式连续泊位分配系统开发环境的搭建步骤如下:

步骤 1: Vmware 下安装 Linux. 首先在 Windows 环境下安装虚拟机 Vmware,在 Vmware 中安装三个 Linux 系统, 通过配置/etc/sysconfig/network 文件分别修改 Linux 的主机名为 namenode,datanode1,datanode2, 修改/etc/hosts 配置本地域名的映射关系并配置 jdk 环境.

步骤 2: ssh 无密钥访问配置. namenode 主机使用 ssh-keygen –t rsa 命令生成密钥, 在/root/.ssh 下生成两个文件 id_rsa.pub 和 id_rsa,将 id_rsa.pub 文件复制到 authorized_keys 中, 并将 authorized_keys 拷贝到 datanode1 和 datanode2 中, 使用 ssh 命令验证是否主机间可以互相访问.

步骤 3: hadoop 文件配置. Hadoop 压缩文件解压到 home 目录下的用户主目录中, 分别配置 hadoop-env.sh 、 core-site.xml 、 hdfs-site.xml 、 mapred-site.xml、masters 和 slaves 文件.

步骤 4: eclipse 下配置 MapReduce 插件. Namenode 主机上 home 目录下的用户主目录中安装 eclipse 开发环境, 在 Hadoop 安装目录 src/contrib/eclipse-plugin 下通过 ant 命令编译生成 eclipse 插件, 将生成的 eclipse 插件拷贝到 eclipse 安装目录 plugins 文件夹下实现 Mapreduce 插件的安装, 重启 Eclipse, 完成开发环境的搭建.

2.2 分布式环境下泊位分配系统组件设计与部署

为了方便系统的开发, 使得分布式连续泊位分配系统的开发过程更加简洁、模块化, 本文提出了分布式环境下连续泊位分配系统组件设计与部署. Hadoop 平台下分布式连续泊位分配系统的开发分成不同的子系统组件, 每个开发组件都有一个确定的任务定义, 基于 Hadoop 平台下连续泊位分配系统的具体组件设计与部署如图 3 所示.

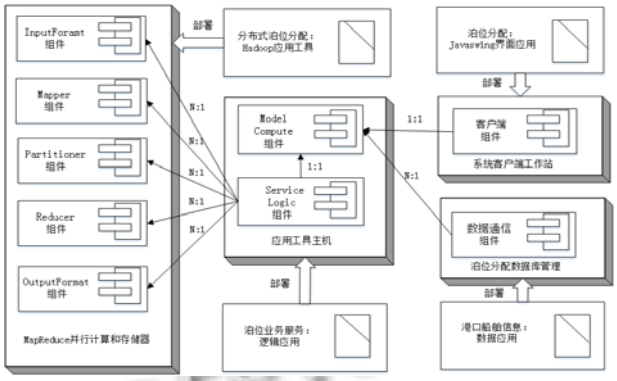

图 3 分布式环境下泊位分配系统组件设计与部署

在图 3 中, 泊位分配系统界面应用部署在客户端组件上. 船舶信息部署在数据通信组件上. 泊位应用工具组件上部署了泊位逻辑业务应用, 并同时管理 Model Compute 组件和 Service Logic 组件, Model Compute 组件中的数据获取和展现由客户端组件和数据通信组件提供. Service Logic 组件调用 MapReduce 并行计算和存储器中的组件进行分布式泊位信息业务逻辑处理, MapReduce 并行计算和存储器上通过部署 Hadoop 应用工具管理组件. 分布式环境下连续泊位分配系统各组件的具体功能设计如下:

(1) 客户端组件: 客户端组件负责泊位分配系统模型界面和可视化的泊位分配数据显示.

(2) 数据通信组件: 数据通信组件实现和 Model Compute 组件的交互, 提供泊位分配中船舶对应的泊

位数、桥吊、到港时间、作业量等船舶信息. 通过数据通信接口接入到泊位分配系统中, 用于保证泊位分配数据的完整性, 实现系统之间的数据通信.

(3) Model Compute 组件: Model Compute 组件实现泊位分配系统中船舶总在港时间最短的目标函数和泊位分配系统中岸线长度, 桥吊数等约束条件的构建.

(4) Service Logic 组件: Servie Logic 组件实现基于单纯形算法的分支定界法求解泊位分配模型. 在分支定界法中采用 Java 多线程思想提高系统运行效率. 在单纯算法中采用 MapReduce 并行计算和存储器上提供的组件进行分布式计算, 节省算法的计算时间.

(5) InputFormat 组件: InputFormat 组件用于将单纯形算法的一次转动 Pivot 过程的输入数据描述为 MapReduce 的输入数据的格式, 提供数据切分功能和 Mapper 组件的输入数据. 在数据切分中提供文件切分算法和 host 选择算法策略将数据切分为若干片段, 便于后续确定 MapTask 的个数. Mapper 组件将输入数据解析成 key/value 对.

(6) Mapper 和 Reducer 组件: Mapper 和 Reducer 组件中封装了单纯形算法的一次转动实现方法 Pivot 分布式计算的数据处理过程. InputFormat 组件中的 key/value 对作为输入交给 Mapper 和 Reducer 组件中的 map/reduce 函数处理, 产生对应的 key/value. MapReduce 框架提供 MapRunnable 接口, 在分布式泊位分配系统中通过实现该接口制定 Mapper 的处理方法和 key/value 的处理逻辑. MapReduce 框架提供多种 Mapper 和 Reducer 组件的实现方法, 本文采用支持链式作业的 ChainMapper/ChainReducer 实现 Hadoop 平台下的分布式泊位分配系统.

(7) Partitioner 组件: Partitioner 组件维护着 Reduce 组件上的负载均衡, 对 Mapper 的中间结果分片处理, 方便将同一 key/value 对应的数据交给同一 Reducer 处理.

(8) OutputFormat 组件: OutputFormat 组件用于描述单纯形算法的一次转动实现方法 Pivot 的输出数据的格式, 将 InputFormat 中解析的 key/value 对写入分布文件 HDFS 中.

在分布式连续泊位分配系统中, Service Logic 组件中通过基于单纯形算法的分支定界法对 Model Compute 组件中构造的泊位分配线性规划模型求解. 在分支定界法中采用多线程的设计思想, 通过一次分支定界分出左右子问题, 由于左右子问题存在数据交换而不是独立运行, MapReduce 并行计算框架并不适用于该情况的处理, 本文采用多线程处理, 通过分别调用 sleep()方法使左右线程进入阻塞状态, 然后调用 exchange()方法实现左右线程的数据传递. 多线程并行处理分布式泊位分配系统中分支定界法如表 1 所示. 在泊位分配问题的单纯形算法求解中, 由于单纯形算法的一次转动 Pivot 方法中存在对泊位分配的线性规划模型的目标函数和约束条件中的变量替换, 而这些替换之间是独立运行的不存在数据交换, 适用于 MapReduce 并行计算框架, 在上述组件中, 只需要编写 Mapper 和 Reducer 即可实现 Mapreduce 框架计算泊位分配问题中单纯形算法的一次转动 Pivot 方法的分布式求解. MapReduce 并行计算采用回调机制来设计接口, 在 MaprReduce 运行时环境会自动调用各个接口来处理数据. 例如, 在 Mapper 组件中, 泊位分配系统中通过将单纯形算法的一次转动逻辑实现为 PivotMapper 类中的方法 map(), 在 MapReduce 运行时环境中 Map Task 首先初始化并从输入数据中解析出 <key,value>,通过调用应用程序编写的 PivotMapper 类中的 map()方法迭代处理. MapReduce 求解分布式连续泊位分配系统中单纯形算法如表 2 所示.

表 1 并行处理分布式泊位分配系统中分支定界法

```
//在如果不全是整数的情况下进行分支定界
float minzleft,minzright;
executor=(ThreadPoolExecutor)Executors.newFixedThreadPool(2);//
创建左右两个线程池
leftthread left=new
leftthread(maxz,minz,j,lp,A1,T,M,l1,l2,la,lb,l5,TS,L,TW,N,B,a,b,c,v,x,
t1,t2);
rightthread right=new
rightthread(maxz,minz,j,rp,A1,T,M,r1,r2,ra,rb,r5,TS,L,TW,N,B,a,b,c,v,
x,t1,t2);
// leftthread ,rightthread 线程中分支定界的计算
Thread leftbranch =new Thread (left);
Thread rightbranch =new Thread (right);
try   { leftbranch.join(); rightbranch.join();}
 catch ( InterruptedException e ){ e.printStackTrace() }
minzleft = left.minz;   //左右线程取最小值
minzright = right.minz;
if (minzleft < minzright){ minz=minzleft;}
else{minz=minzright};
//在 leftthread ,rightthread 线程中通过 exchanger( )实现线程间数据
传递
Thread.sleep(4);
```

```
float  minzleft,minzright,maxleft,maxright;
maxin = exchanger.exchange(maxin);
minzleft=minz; maxleft=maxz;
minzright=maxin[1]; maxright=maxin[0];
float compare[]={minzleft,minzright,maxleft,mazright};
minz=minz(compare);
maxz=s.max(compare); //S 中主要为单纯形算法的实现
```

表 2　MapReduce 求解分布式泊位分配中单纯形算法

```
//Map 过程的初始化
public void configure(JobConf job) {  //配置初始化
    this.mapper=      ReflectionUtils.newInstance(job.getMapperClass(),
job);
    this.incrProcCount=
SkipBadRecords.getMapperMaxSkipRecords(job)>0&&
SkipBadRecords.getAutoIncrMapperProcCount(job); }
//迭代处理泊位分配中单纯形算法
public void run(RecordReader<K1, V1> input, OutputCollector<K2, V2>
output,Reporter reporter) throws IOException {
try {
// 根据分配的 key/value 调用单纯形算法的 Mapper
K1 key = input.createKey();
V1 value = input.createValue();
while ( input.next(key, value)) {mapper.map(key, value, output,
reporter);}
} finally { mapper.close ();} } //Map 过程运行结束
//用户编写的 Mapper 即 PivotMapper 类
Public class Myjob {
public static class PivotMapper implements Mapper<K,V,K,V>{
public void map(K key, V val, Output Collector<K,V> output, Reporter
report){
……//实现单纯形算法一次转动的分布式计算
output.collect(key,val);}}}
```

2.3 分布式环境下基于**MapReduce**改进泊位分配算法

分布式环境下连续泊位分配系统通过调用单纯形算法求解泊位分配线性规划模型, 基于 MapReduce 的单纯形算法分布式设计是分布式连续泊位分配系统的设计核心. 由于传统串行环境下连续泊位分配系统采用基于单纯形算法的分支定界法, 因此本文提出基于 MapReduce 的单纯形算法设计. 通过对单纯形算法的一次转动 Pivot 的 Mapper 设计, 使得 Pivot 的计算效率大大提高. MapReduce 分布式设计单纯形算法,使得泊位分配模型的求解时间缩短, 系统开销大大减小, 有利于分布式港口泊位分配系统的实现.

本文将泊位系统中单纯形算法进行分布式设计, 在过程 SIMPLEX 中确定 x_e 为最紧约束的前提下, 将单纯形算法的一次转动 Pivot 抽象为以下 4 个部分:

(1) x_e 重置于等式左边来计算 x_e 在新等式中的系数;

(2) 更新剩下等式, 每个 x_e 替换为等式右边;

(3) 对目标函数进行同样 x_e 替换;

(4) 更新基本变量集合和非基本变量集合并返回新松弛型;

由于上述四个部分, 前一个部分的输出结果需要作为下一个结果的输入, 为了解决该问题, 本文采用 MapReduce 框架中 ChainMapper/ChainReducer 接口技术, 通过在 Map 阶段设计多个 Mapper, 将前一个 Mapper 的输出结果重定向到下一个 Mapper 的输入中. ChainMapper/ChainReducer 中将原先需要写入到文件中的输出结果重定向到另一个 Mapper 中, 通过 OutputCollector 进行数据的重定向处理来管理输出. 分布式泊位分配系统开发过程中通过调用 addMapper 来添加一个 Mapper, 并指定对应的 JobConf, ChainMapper/ChainReducer 接口中将 Mapper 对应的 JobConf 进行对象序列化后保存于系统中唯一存在的 JobConf 中. 在 MapReduce 框架中给定元组所构造的一个松弛型对应 InputFormat 接口; x_e 重置于等式左边来计算 x_e 在新等式中的系数对应 Mapper1 接口; x_e 替换为等式右边更新剩下等式对应 Mapper2 接口; 目标函数进行同样的 x_e 替换对应 Mapper3 接口; 更新基本变量集合和非基本变量集合并返回新松弛型对应 Reducer 接口. MapReduce 中单纯形算法一次转动 Pivot 过程设计如图 4 所示.

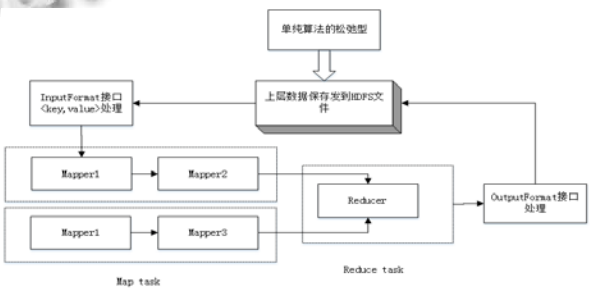

图 4　MapReduce 中单纯形算法一次转动 Pivot 设计

在图 4 中, InputFormat 接口通过获取 HDFS 中文件实现子过程 PIVOT 的松弛型数据构造, 并将构造好的数据传给 Mapper1 重置 x_e 在新等式中的系数, Mapper1 接口的处理数据分别传入 Mapper2 和 Mapper3 接口中进行约束条件剩余等式和目标函数中

x_e 的替换最后经 Reducer 进行算法中基本变量集合和非基本变量集合的更行, 并将更新的结果输出到 HDFS 文件中, 返回新的松弛型. 上述算法中接口之间的规则是根据 MapReduce 中以 key/value 传输数据的原理定义, InputFormat 接口负责处理原始输入松弛型的数据并将处理后的<key,value>对输入给 Mapper1 接口, Mapper1 接口将 InputFormat 接口传入的键值对处理后生成新的<key,value>对输入给 Mapper2 和 Mapper3 进行处理, Reducer 口接受 Mapper2 和 Mapper3 的<key,value>对输出新的松弛型. Map task 阶段输入的<key,value>对为当前松弛型所对应的元组; Map task 阶段输出<key,value>对为经过约束条件和目标函数中 x_e 替换的新的约束条件和目标函数值, Reduce task 阶段输入的<key,value>对为根据目标函数和约束条件更新后的基本变量集合和非基本变量集合; Reduce task 阶段输出的<key,value>对为 x_e 变量替换后的新的松弛型并经 OutputFormat 返回到 HDFS 文件中.

3 分布式泊位分配系统详细设计与实现

通过搭建 Hadoop 平台下连续泊位分配系统的开发环境, 设计分布式环境下连续泊位分配系统组件和基于 MapReduce 改进连续泊位分配算法. 分布式环境下连续泊位分配系统的设计和实现包括: 连续泊位分配模型设计和泊位分配模型求解流程. 下面进行详细的阐述.

3.1 数据模型设计

模型假设:

港口泊位水深必须满足所有到港船舶的吃水深度并且码头泊位区域是连续的. 作业船舶间的安全距离计算在船长里面, 模型中允许船舶之间没有距离间隔. 每条船舶只有一次靠泊机会. 为保证船舶的作业时间的连续性, 只有装卸作业完成后才可离港. 港口的船舶数、港口泊位数、计划作业时间和计划作业桥吊数是已知的. 每条船的到港时间、作业量、船长, 作业路数是已知量. 所有桥吊工作效率相同, 忽略桥吊迁移时间, 桥吊不可越轨跨界迁移. 由于船长和作业量的限制, 每条船分配的最小桥吊数为 1 并且存在最大桥吊数.

符号定义:

集合 LOC={m|m∈[0,M]∩m∈R},其中 m 表示船舶停靠的位置, M: 泊位岸线长度(米); 集合 SHIP={1,2,⋯ ,i,⋯,SHIPS}, 其中 i∈SHIPS, SHIPS 为到港总船舶数; x_i :船舶 i 的靠泊位置, 其中, $x_i \in$ LOC; l_i : 船舶 i 的长度, 其中, i∈SHIPS; T:计划作业总时间; C: 港口的总桥吊数; M:无穷大数; Q_i :船舶 i 的作业量; $Q_i^{\min}$: 船舶 i 最小可接受的桥吊数目; $Q_i^{\max}$:船舶 i 最大可接受的桥吊数目; t_i :船舶 i 的靠泊时刻, 其中, i∈SHIPS, $t_i \ge 0$, $x_i \in$ LOC; $r_{t_i,i}$:船舶 i 在 t_i 时刻分配的桥吊数; $T_{x_i,t_i,i}$:船舶 i 在 t_i 时刻停泊 x_i 处的作业时间, 其中,i∈SHIPS; a_i :船舶 i 的到港时刻, 其中, i∈SHIPS; $u_{x_i,i} \in \{0,1\}$:船舶 i 在岸线 x_i 处开始停泊时 $u_{x_i,i}$=1; 否则, 为 0; i∈SHIPS; $v_{x_i,i} \in \{0,1\}$:船舶 i 已经在岸线 x_i 处停泊时 $v_{x_i,i}$=1; 否则, 为 0; i∈SHIPS; $y_{t_i,i} \in \{0,1\}$:船舶 i 在 t_i 时刻开始停泊时 $y_{t_i,i}$=1; 否则, 为 0; i∈SHIPS; $z_{t_i,i} \in \{0,1\}$:船舶 i 在 t_i 时刻已经在泊时 $z_{t_i,i}$=1; 否则, 为 0; i∈SHIPS;

目标函数:

$$\min\left(\sum_{i=1}^{SHIPS}\sum_{t_i\ge a_i}^{T}(t_i * y_{t_i,i} - a_i) * z_{t_i,i} + \sum_{i=1}^{SHIPS} T_{x_i,t_i,i} * z_{t_i,i}\right) \quad (1)$$

约束条件:

$$u_{x_i,i} + v_{x_i,i} \le 1, i \ne j \quad (2)$$

$$y_{t_i,i} + z_{t_i,i} \le 1, i \ne j \quad (3)$$

$$v_{x_i,i} + z_{t_i,i} + v_{x_j,j} + z_{t_j,j} \le 3, i \ne j \quad (4)$$

$$\sum_{i=1}^{SHIPS} l_i * v_{x_i,i} * z_{t_i,i} \le L, \forall x_i \in LOC \quad (5)$$

$$T_{x_i,t_i,i} = \sum_{t_i\ge a_i}^{T} z_{t_i,i} \quad (6)$$

$$\sum_{t_i=1}^{T}(z_{t_i,i} * r_{t_i,i}) = Q_i \quad (7)$$

$$Q_i^{\min} \le r_{t_i,i} \le Q_i^{\max} \quad (8)$$

$$\sum_{i=1}^{SHIPS}(z_{t_i,i} * r_{t_i,i}) \le C \quad (9)$$

$$\sum_{t_i=1}^{T}\sum_{x_i=1}^{M-l_i+1} u_{x_i,i} * y_{t_i,i} = 1 \quad (10)$$

$$a_i - t_i \le M(1 - y_{t_i,i}) \quad (11)$$

$$1 - v_{x_i',i} \le M(1 - u_{x_i',i}), x_i' = 1,2,\ldots,M - l_i + 1, x_i^{*} = x_i', \ldots, x_i' + l_i - 1 \quad (12)$$

$$1 - z_{t_i',i} \le M(1 - y_{t_i',i}), t_i' = 1,2,\ldots,T - T_{x_i,t_i,i} + 1, t_i^{*} = t_i', \ldots, t_i' + T_{x_i,t_i,i} - 1 \quad (13)$$

$$z_{t_i^{*},i} = 0, t_i^{*} = 1,2,\ldots,a_i - 1 \quad (14)$$

$$y_{t_i^{*},i} = 0, t_i^{*} = 1,2,\ldots,a_i - 1 \quad (15)$$

$$u_{x_i^{*},i} = 0, x_i^{*} = M - l_i + 2, M - l_i + 3, \ldots, M \quad (16)$$

其中, 式(1)为目标函数, 式(1)~(16), 式(1)表示所有船舶在港总时间最短, 其中在港总时间包括船舶作业时间和等待时间. 式(2)、(3)和(4)表示任意两条到港船舶之间不可以交叉重叠; 式(5)表示任意时刻在港口停泊的总船长度不大于泊位岸线总长度; 式(6)表示每条船

舶的装卸作业时长; 式(7)表示每条船舶的作业量等于占用桥吊数和; 式(9)表示任意时刻港口总桥吊数不大于港口桥吊总数; 式(10)表示任意船舶 i 只有一次泊位的机会; 式(11)表示任意船舶只有到达后才可以泊位; 式(12)表示船舶时间上的连续性; 式(13)表示船舶在空间上的连续性; 式(14)、(15)和(16)确保船舶停泊后才可以作业.

从上述模型中可以看出, 所有在港船舶总时间最短描述为一个线性函数, 而资源的约束描述为线性不等式, 则上述问题即为一个线性规划问题. 所以连续泊位模型的求解可使用线性规划求最优解.

3.2 模型求解流程

通过搭建 Hadoop 平台下连续泊位分配系统开发环境, 设计分布式环境下基于 MapReduce 改进连续泊位分配算法, 本文实现了分布式环境下连续泊位分配系统的模型求解流程, 如图 5 所示.

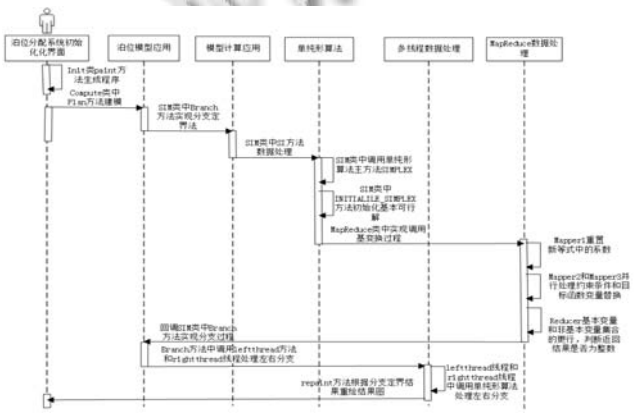

图 5　分布式泊位分配系统模型求解流程

在图 5 中, Init 类中 paint 方法采用 Javaswing 技术生成泊位分配系统界面, 并通过调用 Compute 类中 plan 方法对泊位分配系统建模,SIM 类中 Branch 类实现对泊位模型应用的分支定界法求解; 模型计算应用中通过调用的单纯形算法的求解过程, 包含 SIMPLEX 方法、INITIALLILE_SIMLPEX 方法、MapReduce 类中实现单纯形算法的一次转动; MapReduce 数据处理中通过 Mapper1 重置新等式中的系数, Mapper2 和 Mapper3 并行处理约束条件和目标函数中最紧约束变量的替换, Reducer 中实现基变量和非基变量的更新并判断返回结果是否为整数; 泊位模型应用中获得 MapReduce 数据处理的结果回调 Branch 分支定界法处理, Branch 方法中调用多线程处理左右分支; 实验结果通过 rapaint 方法以图形化的结果显示泊位分配的状态.

3.3 实验结果

基于上述分布式泊位分配系统的设计和实现, 给出十一条船的到港时间、船长等信息, 船舶编号分别为 1,2,3,4,5,6,7,8,9,10,11; 到港时间为 1,2,5,5,7,9, 10,12, 13,14,15; 作业量为 25,8,12,18,24,20,10,30,18,15,15; 船长为 13,18,20,15,21,14,16,22,21,20,20; 作业路数为 13,17,20,14, 21, 14,16,21,21,20,20.

系统针对给定数据求解泊位分配结果和运行时间, 下面给出传统串行环境下连续泊位分配系统求解十条船和十一条船的求解结果如图 6 和图 7 所示, 分布式环境下连续泊位分配系统求解十条船和十一条船的求解结果如图 8 和图 9 所示.

图 6　传统串行环境系统十条船求解时间

图 7　传统串行环境系统十一条船求解时间

图 8　分布式环境下系统十条船求解时间

图 9 分布式环境下系统十一条船求解时间

实验表明, 基于Hadoop平台分布式连续泊位分配系统能够通过 MapReduce 并行计算能力, 及时求解十条船舶和十一条船舶到港的问题, 相对于传统串行环境计算时间获得极大的缩小, 解决了传统串行环境下连续泊位分配系统在船数大于七条时计算效率下降的问题. 分布式环境下系统十条船舶和十一条船舶的分配结果分别如图 10 和图 11 所示.

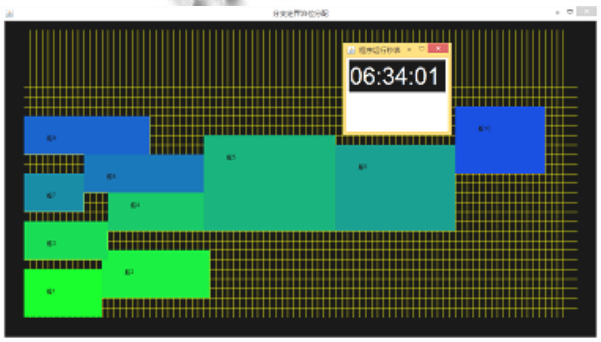

图 10 分布式环境下十条船泊位分配结果

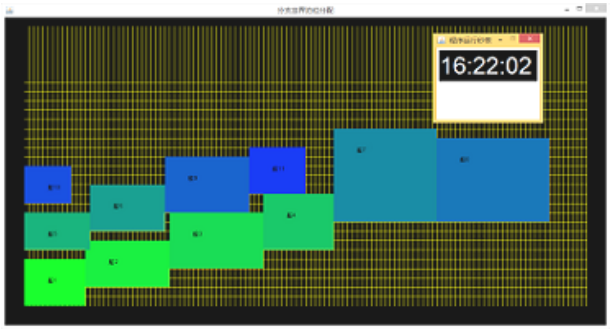

图 11 分布式环境下十一条船泊位分配结果

4 结语

传统串行环境下连续泊位分配系统面临着到港船舶数量的增加、岸线资源有限所造成的算法的执行效率明显降低、资源占用量显著增加等复杂问题, Hadoop平台下 MapReduce 并行框架的技术特性给传统串行环境下连续泊位分配系统出现的问题提供了有效的解决方案. 本文设计和实现了基于分布式环境连续泊位分配系统总体架构和软件架构, 提出了基于Hadoop平台下连续泊位系统开发环境的搭建, 分布式环境下连续泊位分配系统组件设计与部署, 分布式环境下基于 MapReduce 改进连续泊位分配算法算法的关键技术, 最后实现了分布式环境下码头泊位分配系统数据模型设计和模型求解流程. 经实验表明, 基于 Hadoop 平台分布式连续泊位分配系统能够通过 MapReduce 并行计算能力, 并且能够及时求解超过七条船舶到港的问题, 有效的降级硬件设备的压力, 解决了传统串行环境下箱码头泊位分配的问题. 下一步工作将针对JADE(JADE, Java Agent Development Framework)框架下的Java 多 Agent 技术, 提升泊位分配系统的可靠性.

参考文献

1 黄承真,王雷,等.Hadoop 任务分配策略的改进.计算机应用, 2013,33(8):2158–2162.

2 缪裕青,张锦杏,刘少兵,等.一种基于 Hadoop 平台的新聚类算法.计算机科学,2014,41(4):269–272.

3 Lim A. The berth planning problem. Operations Research Letters, 1998, 22(2-3): 105–110.

4 Guan Y, Xiao WQ, Cheung RK, Li C. A multiprocessor task scheduling model for berth allocation: Heuristic and worst-case analysis. Operations Research Letters, 2002, 30: 343–350.

5 杨春霞,王诺.基于多目标遗传算法的集装箱码头泊位—岸桥分配问题研究.计算机应用研究,2010,27(5):1720–1722.

6 Ewout V, Friedlander M. Probing the pareto frontier for basis pursuit solutions. Siam Journal on Scientific Computing, 2008, 31(2): 890–912.

7 陈雪莲,杨智应.桥吊可动态分配的连续泊位分配问题算法.计算机应用,2012,32(5):1453–1456.

8 刘莉莉.集装箱码头连续泊位[硕士学位论文].上海:上海海事大学,2010.

9 Tsutsui S, Fujimoto N. Parallel ant colony optimization algorithm on a multi-core processor swarm intelligence. International Conference on Swarm Intelligence. Springer Berlin Heidelberg. 2010. 488–495.

10 胡小兵,黄席樾.基于蚁群优化算法的 0-1 背包问题求解. 系统工程学报,2005,20(5):520–523.

11 Wu B, Wu G, Yang M. A MapReduce based ant colony optimization approach to combinatorial optimization problems. Knapsack Problem, 2012: 728–732.

12 王诏远,李天瑞,易修文.基于 MapReduce 的蚁群优化算法实现方法.计算机科学,2014,41(7):261–265.

13 陈全,邓倩妮.云计算及其关键技术.计算机应用,2009, 29(9):2562–2567.

14 Gatti MA, Vieira MR, Melo JPF, et al. Handling big data on agent-based modeling of online social networks with mapreduce. Proc. of the 2014 Winter Simulation Conference. 2014. 851–862.

15 Gatti MA, Vieira MR, Melo JPF, et al. Handling big data on agent-based modeling of online social networks with mapreduce. Proc. of the 2014 Winter Simulation Conference. 2014. 851–862.

16 Verma A, Llora X, Goldberg D, et al. Scaling genetic algorithms using MapReduce. 2009 Ninth International Conference on Intelligent Systems Design and Applications. IEEE Computer Society. 2010. 13–18.