# Hadoop云平台MapReduce模型优化研究

张 红[1,2],王晓明[1],曹 洁[2],马彦宏[3],郭义戎[1],王 憨[1]
ZHANG Hong[1,2], WANG Xiaoming[1], CAO Jie[2], MA Yanhong[3], GUO Yirong[1], WANG Min[1]

1.兰州理工大学 电气与信息工程学院,兰州 730050
2.兰州理工大学 计算机与通信学院,兰州 730050
3.国网甘肃省电力公司,兰州 730030
1.College of Electrical & Information Engineering, Lanzhou University of Technology, Lanzhou 730050, China
2.College of Computer & Communication, Lanzhou University of Technology, Lanzhou 730050, China
3.State Grid Gansu Electric Company, Lanzhou 730030, China

**ZHANG Hong, WANG Xiaoming, CAO Jie, et al. Research on optimized MapReduce model of Hadoop cloud platform. Computer Engineering and Applications, 2016, 52(22):22-25.**

**Abstract:** Sequential control of running mechanism of MapReduce model on Hadoop platform can lead to waste of computing resources. From the perspective of the fine-grained parallel data processing of each node, combined with multi-threads technique of Java shared memory, this paper optimizes MapReduce model and puts forward a MapReduce+OpenMP framework. This model is a distributed and parallel computing architecture based on Hadoop cloud platform, which combines computing resources of coarse and fine granularity. After programming and realizing on the GPS trajectory data of the taxi in the Hadoop distributed cluster environment, the results show that this distributed parallel computing model can really improve the computing efficiency of processing big data set, and it is an effective optimization and improvement to the MapReduce model of big data processing.
**Key words:** Hadoop; MapReduce; OpenMP; distributed; parallel

**摘 要:** 针对Hadoop平台MapReduce分布式计算模型运行机制中的顺序制约而产生的计算资源浪费问题,从提高平台中每个执行节点的细粒度并行数据处理角度出发,结合Java共享内存多线程编程技术,对该模型进行了优化,提出一种MapReduce+OpenMP粗细粒度相结合的分布式并行计算模型。并在由四个节点组成的Hadoop集群环境下对不同规模大小的出租车GPS轨迹数据分析处理,验证该模型的性能和效率,实验结果证明MapReduce+OpenMP分布式并行计算模型确实能够提高针对大数据集的计算效率,是对Hadoop平台大数据分析处理模型有效的完善和优化。
**关键词:** Hadoop; MapReduce; OpenMP; 分布式; 并行
**文献标志码:** A **中图分类号:** TP393 **doi:** 10.3778/j.issn.1002-8331.1604-0388

## 1 引言

传统的并行计算模型如MPI和OpenMP等抽象度不高,程序员需要了解底层的配置和并行实现细节。Apache开源社区Hadoop平台的MapReduce(缩写为MR)计算模型极大地简化了分布式并行程序的编写,降低了用户的使用门槛,其移动计算而不是数据的分布式设计理念,极大缓解了对大数据处理的I/O访问瓶颈问题,使得Hadoop平台成为大数据分析处理最流行的平台之一,被广泛应用于大数据的挖掘机器学习、文档聚类、统计机器翻译等领域[1]。基于Hadoop平台的MR计算模型将海量数据分布在大规模集群上进行分布式平行处理,极大加快了大数据分析处理效率。但是,该计算模型对同时具有数据密集和计算密集两个特点的大数据处理效率并不高[2],集群中每个节点的资源没有被充分

**基金项目:** 甘肃省自然科学基金(No.148RJZA019);甘肃省科技支撑计划基金(No.1304GKCA023)。
**作者简介:** 张红(1977—),女,博士研究生,讲师,研究领域为大数据,智能交通,E-mail:zhanghong@lut.cn;王晓明(1956—),男,教授,研究领域为智能交通;曹洁(1966—),女,教授,研究领域为智能信息处理,智能交通。
**收稿日期:** 2016-04-27 **修回日期:** 2016-06-17 **文章编号:** 1002-8331(2016)22-0022-04

利用，节点本身的计算性能有待加强[3]。

Yoo　R.[4]等提出了基于多核CPU实现的MapReduce框架Phoneix，实现了在多核多处理器上共享主存模式的MapReduce运行环境；Fang　Wei-bin[5]等基于NvidiaCUDA编程方法实现了在GPU上运行的MR框架Mars，解决GPU不能动态分配内存的问题；清华大学信息科学与技术国家实验室开发了同时支持GPU与CPU的MR框架MapCG；翟岩龙[6]研究了基于Hadoop平台的CPU、GPU协同计算方法。这些研究都是尝试实现高性能的MR计算框架，但是他们或者没有考虑大数据的处理问题或者没有基于目前最通用的Hadoop平台，对Hadoop平台的性能研究主要集中于内部执行过程的优化，并没有研究针对单个节点的处理性能问题。Wottrich　R.[7-8]提出了基于OpenMP和MR的分布式云计算模型OpenMR，通过将OpenMP中的循环迭代映射到Hadoop云平台，实现了分布式云环境中的循环迭代计算。然而，OpenMP用C/C++编写，Hadoop平台基于Java实现，要运行该模型，必须用Hadoop　Streaming进行接口转换，这限制了该模型的灵活性和可扩展性。因此，基于OpenMP技术，优化Hadoop平台的MapReduce计算模型，研究并开发应用Java多线程技术的分布式环境下的并行计算架构，以适应大数据环境下的计算密集型任务是亟待解决的问题。

本文在上述研究基础之上着重研究MapReduce+OpenMP（简写为MR+OMP）粗细粒度相结合的分布式并行计算模型，通过在map和reduce函数中引入Java共享内存多线程技术，实现Hadoop云平台集群中各个节点内部的线程级并行计算，提高节点的计算性能，解决MapReduce计算模型中顺序制约产生的资源浪费问题。另外，现在的PC机均为多核，这为扩展基于Hadoop的MR计算模型，在每个节点实现基于多线程的并行计算提供了便利。

## 2　Hadoop平台MR计算模型

MR执行流程如图1所示[9]。JobClient提交作业并复制作业资源；JobTracker将收到的新作业交给Job调度器，创建对象封装作业运行的任务、状态以及进度，为TaskTracker分配任务；Job调度器获取输入数据文件的分块信息，并创建map任务；TaskTracker复制有关本地化作业的JAR文件，新建TaskRunner实例运行任务。

从该任务执行流程中可以看到，JobTracker主要用来调度任务的执行，处理机器故障，管理与TaskTracker的通讯，监控Job和Task的执行情况；TaskTracker主要完成Map和Reduce任务，是真正的执行节点。

MR计算模型的任务执行过程可分为三个顺序阶段，即Map、Combiner和Reduce，其处理过程可用下面的式子简单描述，其中每个过程都以键/值对作为输入和输出：

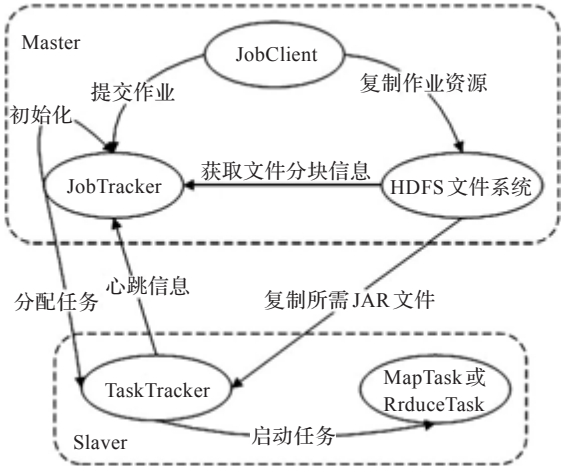


图1　MapReduce数据执行流程图

Map：(K1，V1)→list(K2，V2)

Combiner：list(K2，V2)→(K2，list(V))

Reduce：(K2，list(V))→list(K3，V3)

可以看出这三个阶段存在严重的顺序制约问题，Combiner和Reduce阶段必须等待Map阶段的所有任务完成后才能开始执行汇总，当Map　阶段的任务未完成时Reduce处于等待状态。如果　Map的任务划分不均衡或发生错误，Reduce就会一直等待，这样必然会浪费Reduce所在节点的计算资源。因此，设想将Map任务划分为多个线程，并让多个线程并行执行Map任务，这种细粒度的划分不仅可以提高Map的执行效率，还可以减少　Reduce的等待时间，因为假设有线程运行出了故障，其他的空线程会立即替代执行，这样可以大大减少云平台中各个节点计算资源的浪费，从而提高整个系统的计算效率。

## 3　MR+OMP计算模型

基于Hadoop平台的MR+OMP计算模型架构如图2，系统整体也是Master/Slaver主从式结构，JobTracker和

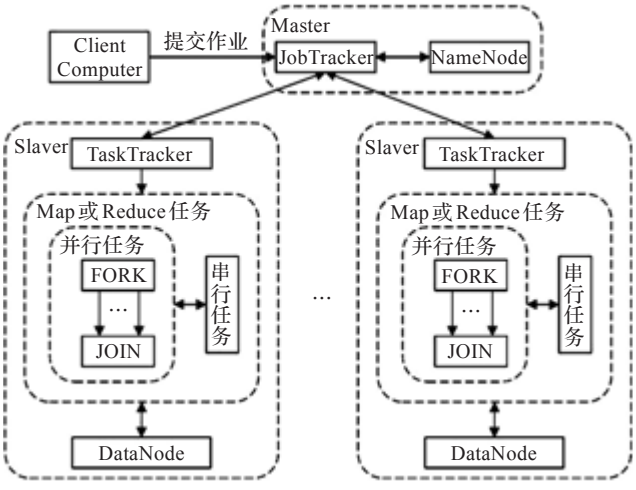


图2　MR+OMP架构图

TaskTracker间的交互、任务调度、文件分块等功能按图1的流程执行,在Slaver节点进行任务处理时,对Map或Reduce任务的处理引入Java OpenMP技术[10],充分利用共享内存的优点,用多个线程并行执行Map或Reduce任务,也就是在Slaver节点通过优化的多线程充分利用多核计算资源以实现节点内部的并行任务执行。

### 3.1 基于Java的OpenMP

Java中引入OpenMP借鉴了C++中引入OpenMP的机制,但是由于Java本身的特点,有几点与C++中引入OpenMP不同[11]。

(1)Java不支持条件编译,所以首先要实现条件编译机制,在Java源文件中定义以//omp开头的代码被认为是条件编译代码。

(2)Java中没有拷贝构造子,所有对象都是引用类型,无法实现在私有化的时候拷贝对象,这时规定所有shared的对象,拷贝的时候引用被拷贝,所有firstprivate和lastprivate的对象通过Object.clone()方法实现拷贝[12]。

(3)Java中很多循环都是针对集合对象和Iterator接口的,这点正好适合map和renduce函数针对大量文本及键值对的操作。

(4)Java本身支持多线程,在OpenMP的并行区域不能使用Java本身的线程调用,如Thread.currentThread()、synchronized()等。

根据Java以上特点,对Java源文件中需要并行执行的代码段前面加OpenMP编译指导语句[13],语句格式为见图3,具体实现细节、编译过程及参数说明文献[14]中有详细描述。

```
//omp parallel [if(<cond>)]
               [default (shared|none)]
               [shared(<vars>)]
               [private(<vars>)]
               [firstprivate(<vars>)]
               [reduction(<operation>:<vars>)]
<Java code block>
```

图3 java OpenMP编译指导语句格式

### 3.2 MR模型中的OpenMP实现

在Hadoop平台的MR计算模型中引入Java OpenMP技术,需要实现以下关键的几个步骤:

(1)在配置好的云平台环境中加载实现Java OpenMP的库文件和编译器,并设置库文件及编译器的路径。

(2)在map和reduce函数中,对要并行执行的部分按需增加如图3所示的编译指导指令,并调用Java OpenMP编译器进行预编译。

(3)编译器创建如图4所示的内部类,并实例化、对相应参数初始化,生成标准的java源文件。

(4)编译java源文件,生成JAR包文件,提交系统运行,实现Hadoop云平台的分布式并行作业执行。

与原计算模型相比,该模型需要预编译机制,而这种预编译机制只需要在本地化单机环境下调试运行,对云平台环境不会增加额外的负担,只需配置Java OpenMP的库文件和编译器环境。从程序开发的角度来看,只需在MR架构中的Map和Reduce函数中增添编译指导语句,图5为实验中的部分代码。因此该计算模型便于实现,且具有较强的实用性。

```
__omp_Class0 __omp_Object0 = new __omp_Class0();
__omp_Object0.n = n;// shared variables
__omp_Object0.args = args; // shared variables // shared variables
  try {
    jomp.runtime.OMP.doParallel(__omp_Object0);
  } catch(Throwable __omp_exception) {
    System.err.println("OMP Warning: Illegal thread exception ignored!");
    System.err.println(__omp_exception);
  }
  s += __omp_Object0._rd_s;  // reduction variables
  n = __omp_Object0.n;  // shared variables
  args = __omp_Object0.args;  // shared variables
```

图4 内部类的实例化及参数初始化

```
public class WCMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] words = line.split(" ");
        //omp parallel for
        for (int j=0;j< words.length; j++){
            float k= Float.valueOf(words[j]);
            if (k>0.0 && k <120.0) {
            context.write(new Text(words[j]), new LongWritable(1));
            }
        }
    }
}
```

图5 出租车GPS轨迹数据预处理map函数的实现

## 4 模型验证

本文构建了由4台普通PC(Intel® Core™ i5-3210M CPU@2.50 GHz 2.50 GHz,4 GB内存,Windows 7旗舰版64位)组建的集群,集群节点信息见表1所示,虚拟架构为VMWare Workstation 10,虚拟机操作系统为CentOS 6.0,Hadoop为1.2.1版本。分别对74 MB、1.2 GB和31.5 GB的出租车GPS轨迹数据(.plt文件)进行分布式并行预处理[15-16],过滤掉速度异常(大于80 km/h及小于等于0 km/h)的数据,每组大小不同的文件分别试验了8次。实验结果见图6,当处理74 MB的小文件时,三种方式即A(MR+OMP)、B(MR)、C(Single,单机串行方式)分别为1.1分钟、1.2分钟、0.8分钟;当处理1.2 GB的文件时,三种方式分别耗用时间为1.8分钟、2.7分钟、4.5分钟;当处理31 GB的文件时,分别耗用时间为3.6分钟、5.7分钟、12.8分钟。

表1 集群节点信息

| 机器名 | 作用 |
|---|---|
| Master | NameNode和JobTracker |
| Slave1 | DataNode和TaskTracker |
| Slave2 | DataNode和TaskTracker |
| Slave3 | DataNode和TaskTracker |

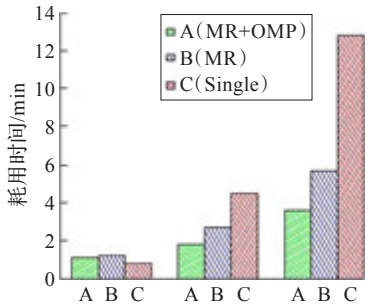


**图6　三种方式处理不同大小文件耗用时间**

从实验得出，处理小文件时，三种方式的平均耗用时间相差不大，甚至C方式耗用时间最少，没有显示出A和B方式的优越性，但是当文件增大到GB级别时，A和B方式的高效率逐渐体现出来，而C方式的运行性能越来越差。并且在每组实验中A方式的耗用时间始终比B方式的少，可见基于Hadoop的MR+OMP计算模式确实能够提高数据分析处理的效率，是对Hadoop平台MR计算模式的有效补充和改进，适合对大数据集的分析处理。

实验结果显示，数据集较小时，单机的串行方式下的耗用时间最少，性能更好，说明基于Hadoop的MR计算架构只有在分析大数据集时，才能显示其优越性，它并不适合处理小数据集的文件。基于Hadoop的MR+OMP计算模型能够充分利用本地机的资源，通过优化的多线程技术提高系统的整体性能，当数据集增加到TB或PB级时，其性能会更加优越，效果会更好。另外，当用MR+OMP计算模型时，首先要对带有OMP标注的文件条件编译，编译过程中要对Hadoop本身的类如Mapper、LongWritable等做import；该系统用Hadoop1.2.1，所用的关键jar包为hadoop-core-1.2.1.jar和commons-cli-1.2.jar，实验中的MR+OMP计算模型启用了4个线程。

## 5　结束语

在深入分析Hadoop平台MR计算模型的任务运行原理、计算流程的基础之上，本文重点分析了Map、Combiner和Reduce之间由于顺序制约问题造成的节点计算资源浪费问题，并引入Java OpenMP共享内存多线程并行计算技术，构建了Hadoop平台上的MR+OMP计算模型，提升了原 MR模型对计算密集型任务的的计算效率。

该模型满足集群环境下粗粒度和细粒度相结合的分布式并行执行的同时，节点内部的共享内存多线程编程模式提升了Map的执行效率，减少了Reduce的等待时间，从而提高了整个系统的数据分析处理效率，实现了Hadoop平台上并行计算的层次结构化，是对Hadoop平台上MR计算模型的有效优化及完善。在由4个节点构建的Hadoop集群上，对不同级别大小的文件进行了实验，实践证明该计算模式确实能够提高Hadoop平台的计算效率，并且数据集越大，相对耗用时间越少，性能越好，适合针对大数据集的分析处理，是对大数据背景下数据分析处理性能有效的理论探索及优化。

## 参考文献：

[1] 陈吉荣，乐嘉锦.基于Hadoop生态系统的大数据解决方案综述[J].计算机工程与科学，2013，35(10)：25-35.

[2] 李张永.基于Hadoop的MapReduce计算模型优化与应用研究[D].武汉：武汉科技大学，2015.

[3] 顾荣，严金双，杨晓亮，等.Hadoop MapReduce短作业执行性能优化[J].计算机研究与发展，2014，51(6)：1270-1280.

[4] Yoo R，Romano A，Kozyrakis C.Phoenix Rebirth：scalable MapReduce on a large-scale shared-memory system[C]// IEEE International Symposium on Workload Characterization，2009：198-207.

[5] Fang Wei-bin，He Bing-sheng，Luo Qiong，et al.Mars：accelerating MapReduce with graphics processors[J].IEEE Transactions on Parallel and Distributed Systems，2011，22(4)：608-620.

[6] 翟岩龙，罗壮，杨凯，等.基于Hadoop的高性能海量数据处理平台研究[J].计算机科学，2013，40(3)：100-103.

[7] Wottrich R，Azevedo R，Araujo G.Cloud-based OpenMP parallelization using a MapReduce runtime[C]//International Symposium on Computer Architecture and High Performance Computing，2014：334-341.

[8] Wottrich R，Araújo G.Loop parallelization in the cloud using OpenMP and MapReduce[J]. Biblioteca Digital Da Unicamp，2014.

[9] Chuck Lam.Hadoop in action[M].[S.l.]：Post & Telecom Press，2011.

[10] Klemm M，Bezold M，Veldema R，et al.JaMP：an implementation of OpenMP for a Java DSM[J].Concurrency & Computation：Practice & Experience，2007，19(18)：2333-2352.

[11] Bull J M，Kambites M E.JOMP-an OpenMP-like Interface for Java[C]//ACM 2000 Java Grande Conference，2000：1-10.

[12] Giacaman N，Sinnen O.Object-oriented parallelisation of Java desktop programs[J].IEEE Software，Software for the Multiprocessor Desktop：Applications，Environments，Platforms，2011，28(1)：32-38.

[13] Senghor A，Konate K.A Java hybrid compiler for shared memory parallel Programming[C]//13th International Conference on Parallel and Distributed Computing，Applications and Technologies，2012：131-136.

[14] Zhang Hong，Wang Xiaoming，Cao Jie，et al.Parallel computing of shared memory multiprocessors based on JOMP[C]// International Conference on Mechatronics，Electronic，Industrial and Control Engineering.2015，8：1628-1632.

[15] 孙卫真，王秀锦，徐远超.交通信息分布式处理中的Hadoop调度算法优化[J].计算机工程与设计，2014，35(4)：1269-1273.

[16] 丁鸿凯.基于Hadoop的交通视频异常事件检测系统的设计与实现[D].北京：北京邮电大学，2015.