Journal of Computer Applications
计算机应用,2015,35(1):43-47
ISSN 1001-9081
CODEN JYIIDU
2015-01-10
http://www.joca.cn

文章编号:1001-9081(2015)01-0043-05
doi:10.11772/j.issn.1001-9081.2015.01.0043

云平台下图数据处理技术

刘　超,唐郑望,姚　宏*,胡成玉,梁庆中
(中国地质大学(武汉) 计算机学院,武汉 430074)
(*通信作者电子邮箱 yaohong@cug.edu.cn)

摘　要:针对 Hadoop 云平台下 MapReduce 计算模型在处理图数据时效率低下的问题,提出了一种类似谷歌 Pregel 的图数据处理计算框架——MyBSP。首先,分析了 MapReduce 的运行机制及不足之处;其次,阐述了 MyBSP 框架的结构、工作流程及主要接口;最后,在分析 PageRank 图处理算法原理的基础上,设计并实现了基于 MyBSP 框架的 PageRank 算法。实验结果表明,基于 MyBSP 框架的图数据处理算法与基于 MapReduce 的算法相比,迭代处理的性能提升了 1.9~3 倍。MyBSP 算法的执行时间减少了 67%,能够满足图数据高效处理的应用前景。

关键词:图数据处理;云计算;MapReduce 计算模型;批量同步并行模型;PageRank 算法
中图分类号:TP391;TP311　　**文献标志码**:A

Graph data processing technology in cloud platform

LIU Chao, TANG Zhengwang, YAO Hong*, HU Chengyu, LIANG Qingzhong
(*School of Computer Science, China University of Geosciences, Wuhan Hubei* 430074, *China*)

Abstract: MapReduce computation model can not satisfy the efficiency requirement of graph data processing in the Hadoop cloud platform. In order to address the issue, a novel computation framework of graph data processing, called MyBSP (My Bulk Synchronous Parallel), was proposed. MyBSP is similar with Pregel developed from Google. Firstly, the running mechanism and shortcomings of MapReduce were analyzed. Secondly, the structure, workflow and principal interfaces of MyBSP framework were described. Finally, the principle of the PageRank algorithm for graph data processing was analyzed. Subsequently, the design and implementation of the PageRank algorithm for graph data processing were presented. The experimental results show that, the iteration processing performance of graph data processing algorithm based on the MyBSP framework is raised by 1.9-3 times compared with the algorithm based on MapReduce. Furthermore, the execution time of the MyBSP algorithm is reduced by 67% compared with MapReduce approach. Thus, MyBSP can efficiently meet the application prospect of graph data processing.

Key words: graph data processing; cloud computing; MapReduce computation model; Bulk Synchronous Parallel (BSP) model; PageRank algorithm

0　引言

近年来,随着互联网数据规模呈爆炸式增长,传统的分布式环境对大规模数据的处理能力受限,已经远远不能满足应用的需求[1]。在此背景下,云计算技术应运而生。MapReduce[2]是一种云平台下的计算模型,最早由谷歌公司在 2004 年提出,用来加速网页搜索业务数据的并行处理和分布式计算。Apache 软件基金会(Apache Software Foundation, ASF)随后成立大型项目 Hadoop[3],实现了 MapReduce 计算模型的开源版本。MapReduce 的优点主要包括两个方面:其一是简单易用,屏蔽了复杂的底层实现细节;其二是扩展性好,适合大规模计算。借鉴 MapReduce 编程思想的大型企业云平台包括微软的 Azure 云[4]、IBM 的蓝云(Blue Cloud)[5]和亚马逊的弹性计算云(Elastic Computing Cloud, EC2)[6]等。

随着信息检索、生物信息学、社交网络分析等应用的大量涌现,许多大规模计算问题本质上可以归结为图数据处理问题[7]。典型的图数据包括互联网 Web 图数据、社交网络图数据和基因谱系图数据等。图数据具有顶点数量多、顶点之间链接关系复杂等特点,涉及大量迭代式计算。MapReduce 计算模型和图数据节点之间链接关系的复杂特性存在不匹配问题,不能适应迭代式算法的求解[8-9];另外 MapReduce 在两次迭代计算之间需要重启作业,导致大量的 I/O 开销,也影响了计算效率。如何高效分析和处理图数据是学术界和工业界亟需解决的一个重大挑战[10]。

针对 MapReduce 计算模型存在的局限性,本文提出了一种适合图数据处理的 MyBSP 计算框架。其核心思想主要借鉴了谷歌公司开发的 Pregel 图处理系统[11],采用批量同步并行(Bulk Synchronous Parallel, BSP)计算框架进行设计[12]。MyBSP 框架的执行机制包括本地计算、全局通信和进程间同步等阶段。MyBSP 利用并行执行单元完成迭代处理,避免两次迭代之间的作业重启动,进而减少中间数据传输的网络开销,提高图数据处理的执行效率。本文选取 Web 图数据的典

收稿日期:2014-07-18;**修回日期**:2014-09-07。
基金项目:国家自然科学基金资助项目(61272470, 61305087);中央高校基本业务费专项资金资助项目(CUGL130233)。
作者简介:刘超(1979-),男,湖北武汉人,讲师,硕士,CCF 会员,主要研究方向:云计算、分布式系统; 唐郑望(1992-),男,湖北荆州人,硕士研究生,主要研究方向:大数据处理; 姚宏(1976-),男,河南许昌人,副教授,博士,主要研究方向:移动计算、物联网; 胡成玉(1978-),男,湖北襄阳人,副教授,博士,主要研究方向:云计算、水管网优化; 梁庆中(1979-),男,广西桂林人,讲师,硕士,主要研究方向:移动互联网与优化。

型算法 PageRank[13] 作为比较对象，分别给出了 Hadoop 云平台下基于 MapReduce 框架设计和基于 MyBSP 框架设计的思路，并通过实验验证了基于 MyBSP 框架的图数据处理算法相比 MapReduce 框架的图数据处理算法效率更高。

1 MapReduce 框架

MapReduce 采用分而治之的思想，将大规模的数据集分发给由一个主节点管理的若干个从节点进行分布式计算，然后将各个从节点的计算结果进行整合，得到最终结果。在 ASF 的顶级项目——开源云平台 Hadoop 中，每个 MapReduce 任务都被初始化为一个作业，每个作业又分为两个运行阶段：map 阶段和 reduce 阶段。在代码层面上这两个阶段分别用 map 函数和 reduce 函数表示：map 函数主要负责任务的分发，它接收一个〈key，value〉形式的输入，然后同样产生一个〈key，value〉形式的中间输出；reduce 函数负责结果汇总，它接收一个〈key，〈list of values〉〉形式的输入，然后对这个 value 集合进行整合处理，产生〈key，value〉形式的输出。

下面从运行机制层面分析 map 任务和 reduce 任务的执行流程，如图 1 所示。

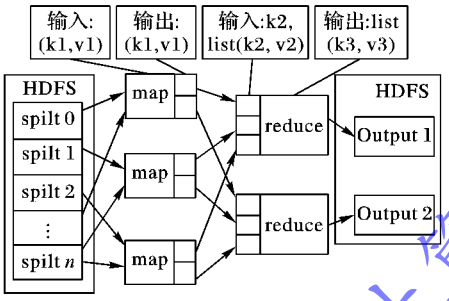

图 1 MapReduce 的运行机制

map 端的执行流程可以描述如下：

1）输入数据按照 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS）的块大小（缺省是 64 MB）进行分片，作业调度器为每个输入分片创建一个 map 任务来处理。按照数据就地计算的原则，map 任务计算完成的输出结果暂时放在本地的内存缓冲区。缓冲区满了，将缓冲区的数据写入本地创建的溢写文件（flush file）。

2）对 map 任务计算产生的多个溢写文件，还需要进行本地合并（combiner）、排序、分区操作，最后写入 HDFS 的本地磁盘。其中，分区数目对应于 reduce 任务的数目，使得一个 reduce 任务对应一个分区。

3）将 map 任务结束产生的分区结果数据，拷贝交换（shuffle）到需要它们的对应 reduce 任务，作为 reduce 的输入。通过向作业调度器请求 map 输出位置的信息，reduce 任务可以获取输入数据启动响应的处理。

reduce 端的执行流程可以描述如下：

1）reduce 端根据相应的键值接收 map 任务传来的有序数据。reduce 端尽量在内存中进行数据合并，若数据量超过限定的缓冲区大小，数据则会进行溢写操作，存储到本地磁盘中。同时，对于溢写文件，reduce 端后台线程会执行排序、合并操作。

2）当所有输入数据本地有效时，顺序读取文件，并把相同键值的键值对数据进行合并，合并后的值传给 reduce 方法函数，执行对应的算法。然后是计算下一个键，如此迭代执行下去。计算完成后将所得输出结果文件保存到分布式文件系统中。

MapReduce 在整个计算流程中需要使用 Hadoop 的底层分布式文件系统（HDFS）进行中间结果的存储，这样会带来较大的磁盘读写开销，从而降低数据处理的效率，因此不适合迭代式数据计算的应用场合。

针对 MapReduce 计算框架的不足，学术界和工业界提出了一些改进的思路。印地安纳大学的 Ekanayake 等[14] 提出一种增强型的 MapReduce 迭代框架——Twister，该框架采用发布/订阅消息架构用于通信和数据传输，并新增了广播和散播类型的数据传输接口；同时，支持迭代处理过程中静态数据的缓存操作，让 map 任务尽可能调度到本地计算。然而，Twister 在容错性和性能上还有待改进。美国华盛顿大学的 Bu 等[15] 提出了另外一种迭代数据处理框架——HaLoop，增加了缓存机制以支持循环感知的调度；同时修改了 Hadoop 的编程接口，支持一个作业内执行多个 map 和 reduce 任务对，实现作业内部的控制迭代。其主要缺陷在于迭代终止条件难以准确判定。卡内基梅隆大学的 Low 等[16] 采用顶点为中心的模式，提出了一个异步分布式共享内存的图数据处理框架——GraphLab，支持复杂、动态的机器学习和数据挖掘相关的图并行计算。为了高效处理谷歌产品的社交网络服务（Social Networking Service，SNS）中图计算问题，Malewicz 等[1] 基于 BSP 计算模型，专门针对图数据处理问题，提出了图数据处理系统 Pregel。该系统对图数据的计算、消息通信、容错和同步控制等方面提出了较为完善的解决方案。然而，Pregel 没有开放源代码，因此无法获知具体的实现细节。

2 MyBSP 迭代计算框架

谷歌开发的 Pregel 图数据处理框架的核心是实现了 BSP 计算模型。该模型在架构上采用一个主节点进行协调，负责分发图的顶点到工作节点，并由工作节点以同步方式执行计算任务。本文提出的 MyBSP 框架基于 Java 语言，实现了类似 Pregel 的 BSP 模型。

2.1 MyBSP 框架及工作流程

MyBSP 模型引入一个并行计算单元概念（也称为超级步骤（superstep））。一个 MyBSP 程序由一系列并行计算单元组成。一个并行计算单元包括多个处理进程，其工作步骤可以分为 3 个阶段：

1）本地计算阶段。每个处理进程对本地存储的数据进行相关计算，计算结束后转入第 2）阶段处理。

2）全局通信阶段。每个处理进程根据需要与相邻的进程以消息传递的方式进行通信并更新自身的状态，通信结束时，转入第 3）阶段。

3）进程间同步阶段。即一个并行计算单元内的多个处理进程等待所有通信活动结束，然后决定是否进入下一轮并行计算单元的处理。

支持迭代计算的图数据处理框架 MyBSP 的系统架构及其工作流程如图 2 所示。

从架构上来看，MyBSP 主要包括以下 4 个组成部分：主服务器 MyBSPMaster、应用服务器 AppServers、工作节点 MyBSPWorkers 和分布式协调器 Zookeeper[17]。其中：MyBSPMaster 负责对 AppServers 的任务调度，一个任务对应一个 AppServers，并通过周期性接收心跳信息维护 AppServers 的状态；应用服务器 AppServers 负责调用 MyBSPWorkers，可以看作是多个互不关联的进程，负责启动工作节点，以及执行 MyBSP 分派的计算任务；工作节点 MyBSPWorkers 用于执行

具体的任务;分布式协调器 Zookeeper 用于管理并行执行单元的同步操作,通过分布式锁机制控制并行执行单元的执行步骤,并提供系统的容错功能。

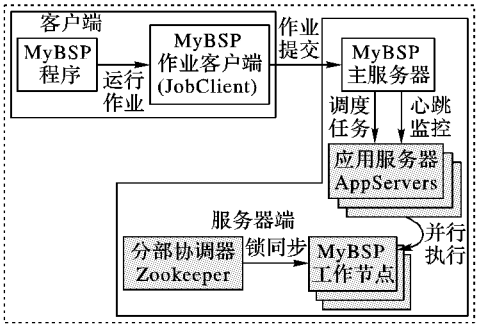

图 2　MyBSP 框架结构及工作流程

MyBSP 框架的工作流程分为以下 6 个步骤:

1)客户端的 MyBSP 程序本地运行一个作业,并提交给主服务器,进行作业注册。

2)主服务器在接受到提交的作业后生成 JobID,并根据其内部维护的计算节点列表,分配任务、调度任务到所有可用的应用服务器节点。

3)任务分配完毕后,主服务器把任务执行的信息广播到所有的应用服务器并开始执行任务,每个应用服务器节点启动一个工作节点进行相关任务的计算。

4)在计算的过程中,应用服务器会与主服务器通过心跳信息进行通信,用于维护应用服务器的状态,这些状态包含能处理任务的最大容量和可用的内存等信息。

5)MyBSP 框架通过 Zookeeper 来进行每一个并行执行单元的同步控制,并决定是否进行下一轮并行计算单元的处理。

6)计算结束后,所有应用服务器会将对应的本地计算结果返回到主服务器节点上,主服务器将所有子节点的信息进行汇总,得到最终的计算结果。

2.2　MyBSP 的接口函数

MyBSP 提供了类似谷歌 Pregel 系统的主要接口。分 3 类定义,如下所示。

1)输入和输出的接口函数。包括:setInputPath()、setInputFormat()、setOutputPath()、setOutputFormat(),用于定义输入输出的路径和格式。

2)通信和同步的接口函数。包括:send()用于节点之间发射消息;getMessage()获取对象的消息,包括工作节点的名称、IP 地址和工作状态等信息;getSuperStepCount()用于获取控制迭代次数的并行计算单元执行计数;sync()用于进程之间同步控制。

3)图计算专有接口函数。主要包括:VertexInterface()定义图的顶点、边和消息;Combiner()用于合并当前顶点消息,以减少通信开销;Aggreator()用于图信息的收集;compute()图计算函数,用于查询当前图顶点的消息以及与其他顶点的交互计算。

3　算法设计

3.1　PageRank 算法基本原理

文献[13]指出,PageRank 是一种根据网页之间的超链接来进行计算,并由此来确定页面等级的算法。该算法是谷歌公司搜索引擎的基本算法之一。其算法公式计算如下:

$$PR(A) = (1-d) + d \times \sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)} \tag{1}$$

其中:$PR(A)$ 表示网页 A 所对应的页面等级分值(PageRank value, PR);$PR(T_i)$ 表示网页 T_i 的 PR 值,并且 T_i 链接到网页 A;$C(T_i)$ 表示网页 T_i 向外链接的总数大小;d 被称为阻尼系数,表示用户到达某网页页面后并继续向后浏览的概率,通常情况下 d 的取值为 0.85。

PageRank 是迭代型的算法,所以在大规模的网络拓扑图里面,需要作多轮迭代使得最终结果 $|PR_k(A) - PR_{k-1}(A)|$ 收敛于一个很小的常数 μ,才能保证正确的结果,返回的 $PR_k(A)$ 值即为该网页的 PR 值。

根据式(1),PageRank 算法流程如图 3 所示。

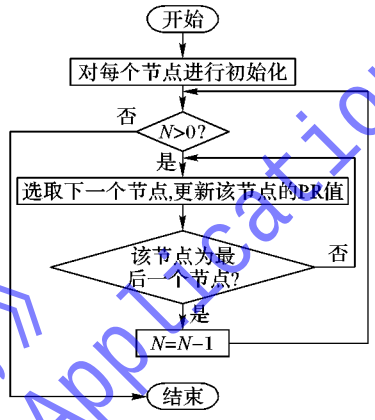

图 3　PageRank 算法流程

首先,对整个程序进行初始化操作,设置迭代次数 N 和阻尼系数,将每个节点的 PR 值初始化为 1.0。接着对于整个图数据进行遍历操作,统计每个节点的指入边的个数,并按照式(1)来更新其 PR 值。一轮遍历结束后会得到一组新的 PR 值,如此反复,在经过多轮迭代之后,得到一组收敛的结果,即为最终所求的 PR 值。

3.2　基于 MyBSP 迭代框架的 PageRank 算法设计

MyBSP 下 PageRank 算法设计从两个方面阐述。首先给出算法并行计算单元的执行流程示例,然后描述核心算法设计伪码。

3.2.1　PageRank 算法并行计算单元的执行流程

如 2.1 节所述,MyBSP 模型由一系列并行计算单元过程组成。采用 MyBSP 框架设计 PageRank 算法的并行执行步骤如图 4 所示。

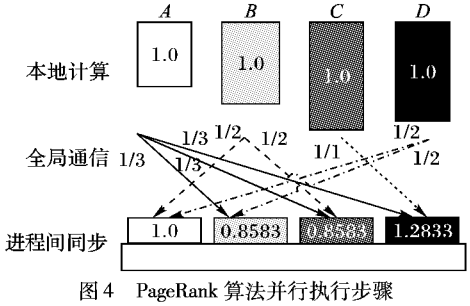

图 4　PageRank 算法并行执行步骤

假设 Web 页面图有 4 个顶点 A、B、C 和 D。其中:A 链接到 B、C 和 D;B 链接到 A 和 C;C 链接到 D;D 链接到 A 和 B。在此并行执行单元中,本地计算时 A、B、C 和 D 的初始 PR 值均为 1。接着,在全局通信阶段,各个顶点与其链接的顶点通信,并更新 PR 值,通过网页指出去的链接的总数将其对应的 PR 值进行均分。最后,在进程间同步阶段通过式(1)进行计算汇总 PR 值。例如对于网页 A,其经过一轮并行执行步骤后的 PR 值为 $PR_A = 0.15 + 0.85 \times (0.5_B + 0.5_D) = 0.15 + 0.85 \times 1 =$

1.0,网页 B、C、D 对应的值计算过程类似。

Hadoop 云平台下基于原生 MapReduce 设计的 PageRank 算法将图数据看作键值对处理[18]。通过与 MapReduce 算法执行流程比较,本文总结得出,MyBSP 优点在于:其全局通信阶段仅需要通过网络发送信息与邻近节点进行通信,节省了 map 端到 reduce 端传输中间数据的网络开销,也避免了网络拥塞可能导致的错误。

3.2.2 MyBSP 下 PageRank 核心算法设计

在 PageRank 算法设计中,核心是重载 Vertex 类。Vertex 类实现了 VertexInterface 抽象接口。该类的模板参数定义了 3 个值类型的参数,用于表示图数据的顶点、边和消息。Vertex 类提供 3 个主要的接口方法供用户调用或者重载,包括:用于查询顶点和边信息的 compute()方法、用于获取当前顶点值的 getValue()方法和用于来修改当前顶点值的 setValue()方法。基于 MyBSP 云平台实现 PageRank 的函数伪码如算法 1 所示。

算法 1 基于 MyBSP 实现 PageRank 算法的伪码。

```
public class PageRankMyBSPVertex extends Vertex {
  compute(messages) throws IOException {
  if (this.getSuperstepCount() ==0) {
    //在第 0 轮并行计算步骤中初始化节点 PR 值
    this.setValue();
    }
  else if (this.getSuperstepCount() >=1) {
    double sum =0;
    //对于邻接节点所贡献的 PR 值进行累加计算
    while (messages.hasNext()) {
      DoubleWritable msg =messages.next();
      sum +=msg.get();
      }
    double alpha =(1.0 -FACTOR)/this.getNumVertices();
    this.setValue(new DoubleWritable(alpha +(FACTOR *
      sum)));
    }
  else {
    //按照式(1)对给定的节点对 PR 值进行更新
    double alpha =1.0 -FACTOR)/this.getNumVertices();
    this.setValue(new DoubleWritable(alpha +(FACTOR *
      (sum +NodeContribution.get()/this.getNumVertices())));
    }
  if (this.getSuperstepCount() <this.getMaxIteration()) {
    int numEdges =this.getOutEdges().size();
    //在本地计算过程结束后与邻居节点进行通信
    sendMessageToNeighbors(new
      DoubleWritable(this.getValue().get()/numEdges));
    }}}
```

从以上算法描述可知,MyBSP 框架把 PageRank 处理的对象当作连通图,围绕图的顶点来控制循环迭代次数。在第一轮的并行执行过程中,所有的顶点初始是活动状态。没有计算任务的顶点则设置为休眠状态。如果顶点之间存在的链接关系,则通过消息传递激活顶点使之处于活动状态,并参与计算执行;当所有顶点都达到休眠状态,并且没有消息传送过程时,计算程序终止运行。

4 实验及结果分析

4.1 实验环境

本文在中国地质大学云计算平台(Cloud Computing Based Platform, CCBP)上搭建实验环境,进行实验对比分析。CCBP 由 5 个 Dell R720 高性能节点组成:1 个节点作为主控节点,4 个节点作为工作节点。每个节点的硬件配置如下:2 颗 Intel Xeon E5-2650 CPU,16 核,主频 2.0 GHz;内存为 64 GB DDR3;2 块 600 GB 的万转高速硬盘。软件配置如下:操作系统是 64 位的 Ubuntu 12.04 LTS;Hadoop 是 2013 年 8 月发布的 1.2.1 稳定版(参数为缺省配置);JDK 的版本是 Jdk1.7.0_7(x64);MyBSP 版本是 0.6.0。因为云平台是共享运行等原因,实际使用 2 个工作节点采用内置虚拟化软件虚拟出最多 8 个虚拟机,每个虚拟机配置为 4 个 CPU 核和 8 GB 的内存空间。

4.2 实验数据

实验采用的数据集是程序模拟生成的数据。数据是一行两列形式:(ID_A,ID_B),表示网页 ID_A 有一个超链接指向网页 ID_B。为了对比 Hadoop 云平台和 MyBSP 云平台下 PageRank 算法的性能,本文选取了 3 组小规模数据集和 2 组大规模数据集,分别是 13.8 MB、26.4 MB、53 MB、600 MB 和 1.28 GB 的数据集。在不考虑节点故障的前提下,分别记录程序执行时间。每个实验重复运行 3 次,最后取平均值。

4.3 实验结果及分析

为了比较 MyBSP 和 MapReduce 两种框架图数据处理的执行效率,实验通过调整数据规模、虚拟机节点数目和迭代次数控制参数,观察执行时间的变化情况。实验结果如图 5~7 所示。

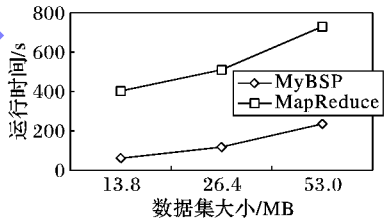

图 5 小规模数据集在云平台实验对比

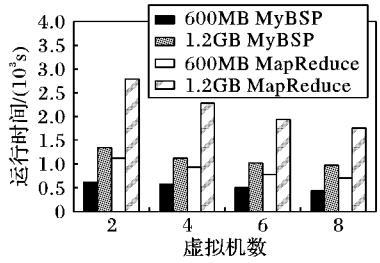

图 6 大规模数据集扩展性实验对比

图 5 显示了 8 个虚拟机节点组成的云环境中两种框架的执行时间随小规模数据集的变化情况。从图 5 中可以发现,Hadoop 和 MyBSP 两种框架执行时间随着数据文件的大小增加呈现近线性增长。当数据集是 53 MB 的时候,MyBSP 比 MapReduce 执行快 3.5 倍左右。主要原因在于 MapReduce 在每次迭代计算完成,会把计算中间结果写回磁盘,下次迭代时需要重启作业,再从磁盘读取中间结果数据,极其影响执行效率。而 MyBSP 避免了作业重启的开销,同时中间结果数据在内存中共享,也减少了访问磁盘时间。

图 6 给出了 MyBSP 和 MapReduce 框架下大规模数据集的扩展性实验。从图 6 中也可以看出,同一种数据集条件下,MyBSP 的执行时间低于 MapReduce 的执行时间。两种框架的执行时间都随着数据集大小增加同比例增长,同时随着虚拟机节点个数增加执行时间相应减少。原因在于,大规模数

据需要更多的节点计算资源,同时也更能发挥虚拟机节点的全部计算能力。

图 7 也选取了 8 个虚拟机节点的云平台执行实验,通过人工控制迭代次数的参数,观察两种计算框架的执行性能情况。

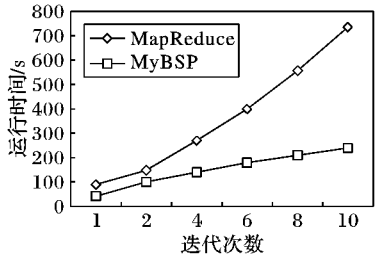

图 7　迭代次数与运行时间关系

从图 7 中可以看出 MapReduce 受迭代次数的影响较大。例如:当迭代次数设置为 1,MapReduce 的执行时间是 MyBSP 的 1.9 倍;当迭代次数设置为 10,性能差异扩大到 3 倍。这是因为迭代次数增加,作业重启动以及新一轮迭代 reduce 操作读取中间结果数据的磁盘开销也相应增加。MyBSP 仅在第一次执行迭代计算时产生磁盘开销,后续的迭代处理通过并行执行单元尽量本地读取内存的共享数据,因而对迭代次数参数不太敏感。

虽然 MyBSP 相比 MapReduce,在迭代式图数据处理过程中执行效率更高,但仍然存在一些问题。例如:目前的版本没有考虑容错功能,当数据集继续增加时,内存受限,性能会出现明显下降趋势,甚至发生运行时故障。

5　结语

本文针对 MapReduce 执行图数据处理算法效率低下的问题,提出了一种适合图数据处理的 MyBSP 框架。该框架采用类似 Pregel 的 BSP 模型,减少了每次迭代读取中间结果的磁盘 I/O 开销。分别讨论在 MapReduce 框架和 MyBSP 框架上图数据处理算法的设计和实现,并在单个节点的伪分布式环境和多个节点的云环境上作对比实验。实验结果表明,基于 MyBSP 框架的图数据处理算法具有良好的扩展性,有效地减少了算法的执行时间,验证了对图数据处理迭代计算的支持。今后,本文将设计其他的图数据处理算法,进一步验证 MyBSP 模型的适用性;并与其他的计算框架作比较,以提出 MyBSP 框架进一步改进的方案。

参考文献:

[1] GANTZ J, REINSEL D. Extracting value from chaos [EB/OL]. [2014-06-15]. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

[2] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107 – 113.

[3] The Apache Software Foundation. Apache Hadoop [EB/OL]. [2014-06-10]. http://hadoop.apache.org/.

[4] XIAO Q, WANG J, MA Y, *et al.* NOHAA: a novel framework for HPC analytics over Windows Azure [C]// Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems. Washington, DC: IEEE Computer Society, 2012: 448 – 455.

[5] WANG L, von LASZEWSKI G, YOUNGE A, *et al.* Cloud computing: a perspective study [J]. New Generation Computing, 2010, 28(2): 137 – 146.

[6] OSTERMANN S, IOSUP A, YIGITBASI N, *et al.* A performance analysis of EC2 cloud computing services for scientific computing [C]// CloudComp 2009: Proceedings of the First International Conference on Cloud Computing, LNCS 34. Berlin: Springer, 2010: 115 – 131.

[7] SALIHOGLU S, WIDOM J. GPS: a graph processing system [C]// Proceedings of the 25th International Conference on Scientific and Statistical Database Management. New York: ACM Press, 2013: 1 – 22.

[8] JIN W, WANG C. Iteration MapReduce framework for evolution algorithm [J]. Journal of Computer Applications, 2013, 33(12): 3591 – 3595. (金伟健, 王春枝. 适于进化算法的迭代式 MapReduce 框架[J]. 计算机应用, 2013, 33(12): 3591 – 3595.)

[9] LIANG Q, WU Y, FENG L. User ranking algorithm for microblog search based on MapReduce [J]. Journal of Computer Applications, 2012, 32(11): 2989 – 2993. (梁秋实, 吴一雷, 封磊. 基于 MapReduce 的微博用户搜索排名算法[J]. 计算机应用, 2012, 32(11): 2989 – 2993.)

[10] YU G, GU Y, BAO Y, *et al.* Large scale graph data processing on cloud computing environments [J]. Chinese Journal of Computers, 2011, 34(10): 1753 – 1767. (于戈, 谷峪, 鲍玉斌, 等. 云计算环境下的大规模图数据处理技术[J]. 计算机学报, 2011, 34(10): 1753 – 1767.)

[11] MALEWICZ G, AUSTERN M H, BIK A J C, *et al.* Pregel: a system for large-scale graph processing [C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2010: 135 – 146.

[12] SEO S, YOON E J, KIM J, *et al.* HAMA: an efficient matrix computation with the MapReduce framework [C]// Proceedings of the Second International Conference on Cloud Computing Technology and Science. Washington, DC: IEEE Computer Society, 2010: 721 – 726.

[13] CHEBOLU P, MELSTED P. PageRank and the random surfer model [C]// Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2008: 1010 – 1018.

[14] EKANAYAKE J, LI H, ZHANG B, *et al.* Twister: a runtime for iterative MapReduce [C]// Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. New York: ACM Press, 2010: 810 – 818.

[15] BU Y, HOWE B, BALAZINSKA M, *et al.* HaLoop: efficient iterative data processing on large clusters [J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 285 – 296.

[16] LOW Y, BICKSON D, GONZALEZ J, *et al.* Distributed GraphLab: a framework for machine learning and data mining in the cloud [J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716 – 727.

[17] HUNT P, KONAR M, JUNQUEIRA F P, *et al.* ZooKeeper: wait-free coordination for Internet-scale systems [C]// USENIXATC 2010: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference. Berkeley: USENIX Association, 2010: 11.

[18] KAMBATLA K, RAPOLU N, JAGANNATHAN S, *et al.* Asynchronous algorithms in MapReduce [C]// CLUSTER'10: Proceedings of the 2010 IEEE International Conference on Cluster Computing. Washington, DC: IEEE Computer Society, 2010: 245 – 254.