高技术通讯 2009 年 第 19 卷 第 10 期:991 ~ 997

doi:10.3772/j.issn.1002-0470.2009.10.001

基于集群系统的空间数据并行处理策略研究①

刘旭辉②*** 韩冀中③* 贺 劲* 韩承德*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院研究生院 北京 100190)

摘 要 为了解决单节点的 WebGIS 系统存储能力和计算能力受限的问题,提出了一种利用分布式文件系统(DFS)和 MapReduce 分布式计算框架在集群环境中并行处理空间数据的方法。还特别针对分布式文件系统,结合 WebGIS 的应用模式,提出了小文件优化策略。该策略的核心思想是通过将小文件合并为大文件来有效降低文件的数目。试验结果表明,在使用了小文件优化策略后,分布式文件系统中的节点平均内存占用率从 55.78% 降至 18.36%,文件的存储和读取性能分别提高了 63.3 倍和 2.0 倍。其次,基于经过优化后的分布式文件系统和 MapReduce 计算框架,设计并且实现了 HDWebGIS 原型系统,试验结果表明,使用了小文件优化策略后,HDWebGIS 系统性能比优化前提升了 78.11%。

关键词 网络地理信息系统(WebGIS),集群,分布式文件系统(DFS),MapReduce,小文件优化

0 引 言

地理信息系统(GIS)是一个为了采集、存储、分析、管理和发布与空间相关的数据及其属性而建立的计算机系统[1]。GIS 因具有直观与易用的特性而被广泛应用到各个领域——从 GPS 导航、电子地图,到 Google 的虚拟地球和虚拟宇宙。网络地理信息系统(WebGIS)[2]是伴随着 Internet 的快速发展而诞生的 GIS 技术,通过 WebGIS,不同地区使用不同平台的客户可以同时访问和管理最新的地理信息数据,真正实现地理信息的全球共享。随着技术的进步和发展,现代 GIS 出现了一个很重要的特征,即数据规模持续增长。当空间数据规模较小时,利用单个节点即可完成服务,但是随着数据规模的不断扩大,单节点在性能、可扩展性、可靠性三方面的缺点凸显了出来。

虽然采用独立磁盘冗余阵列(RAID)等技术可以在一定程度上缓解上述压力,但是仍然很难满足对计算能力和 I/O 能力的需求。克服上述三方面缺点的最直接有效的方法是采用集群来构建 WebGIS。文献[3,4]对分布式 WebGIS 系统进行了研究,并且提出了自己的解决办法,但这些办法是针对特定应用而设计专有的 WebGIS 系统,实现复杂,不适合推广到一般的 WebGIS 系统。文献[5]虽能巧妙地利用 P2P 的执行机制来避免集中式执行引擎带来的网络拥塞和单点失效问题,但需要由程序员来定义和控制在节点之间进行的消息传递,由此增加了系统的复杂度及开发难度。近年来 Google 公司推出的以 Google Earth 为代表的一系列互联网地理信息服务平台获得了广泛关注。这些服务平台的后端是基于 Google 集群的一组分布式计算与数据管理平台,由业务逻辑、分布式文件系统 Google File System[6]、分布式结构化数据管理系统 Bigtable[7]、锁服务器 Chubby[8]及 MapReduce[9]分布式计算框架组成。互联网巨头 YAHOO! 也在其计算平台上开发完成了开源版本的分布式存储计算框架 Hadoop[10]。MapReduce 框架一推出就受到了广泛关注,Paterson 将其称为数据中心的指令[11]。除了被应用在 Google 的搜索引擎中,MapReduce 分布式框架还被广泛应用于分布式查询、分布式排序、日志分析、文档归类及机器学习等应用中[12]。文献[13]扩展了 MapReduce,并用其来处理关系数据。此外,MapReduce 编程模型也被扩展到其他体系结构中,例如多核结构[14]和 Cell 结构[15]。但是至今还没有将

① 973 计划(2004CB318202)资助项目。

② 男,1979 年生,博士;研究方向:分布式存储,计算(Hadoop 系统以及 MapReduce 计算框架);E-mail: Mafish@gmail.com

③ 通讯作者,E-mail: hjz@ict.ac.cn

(收稿日期:2008-08-11)

MapReduce 计算框架和空间应用相结合的研究。

本文首次研究了 MapReduce 计算框架和空间应用相结合的问题,并提出了一种基于分布式文件系统和 MapReduce 分布式计算框架构建 WebGIS 的方法。该方法拟解决的问题是在一个集群系统中,如何充分利用集群系统中的计算能力和并行 I/O 能力分别满足 WebGIS 应用中对计算能力和 I/O 能力的需求。从 WebGIS 应用的特点来看,对于计算密集的入库阶段采用 MapReduce 分布式计算框架提升计算能力;对 I/O 密集的服务阶段采用分布式文件系统提升系统 I/O 性能以及可靠性。本文利用这一方法设计并实现了基于分布式存储计算框架的 WebGIS 原型系统 HDWebGIS (Hadoop Distributed WebGIS),并在该原型系统中利用 MapReduce 分布式计算框架进行矢量数据转栅格数据的并行计算。

1 分布式存储计算框架

近年来 Google 公司推出一系列空间信息相关服务,包括 Google Earth, Google Map, Google Sky 以及 Google Moon 等。Google 所提出的由分布式文件系统(Google file system, GFS)[6] 和 MapReduce[9] 分布式计算框架组成的分布式存储计算框架给了这些数据规模强大的应用很好的支撑。

1.1 GFS

GFS 是一个专门针对大文件进行优化的分布式文件系统,由主节点(Master),客户端(Client)和块服务器(Chunk server)组成。主节点是整个文件系统的核心,它记录了文件系统中的元数据。文件管理的核心是将大文件分成若干个块(Chunk)进行存储(块的默认大小是 64MB),一个块有多个副本,每个副本保存在一个块服务器上。主节点中记录了文件和块之间的映射关系,同时也负责为每个块分配服务器。客户端通过主节点对文件系统中的文件进行访问。主节点在系统中担负重要的角色,它的崩溃将会导致整个系统的崩溃。因此,在 Google 实际应用的系统中,采用了 1 个主节点加 4 个副本主节点的方式进行热备份,用以解决整个系统中可能存在的单点失效问题。

1.2 MapReduce 分布式计算框架

这是该系统的核心,它提供了一套方便的编写分布式应用程序的机制。它的工作原理非常简单,参与 MapReduce 计算的节点被分成两类:Master 和 Worker。在一个集群中,有一个 Master 和多个 Worker。Master 负责整个任务的协调分发,而 Worker 执行具体的计算任务。一个计算任务从输入会经历 Map、Partition/Comparison 和 Reduce 阶段,最后得到输出。与传统并行计算领域广泛应用的消息传递界面(MPI)/并行虚拟机(PVM)不同的是,MapReduce 隐藏了更多集群物理结构(拓扑结构与通信原语)以及节点/网络可用性方面的细节,向用户提供了一套类似于函数语言的数据处理接口,将开发人员从复杂细节中解放出来,使之将更多注意力放在应用逻辑本身的核心算法之上。框架系统则负责管理诸如任务分布、工作调度、容错以及节点通信等问题,最终起到简化整个并行程序开发的效果。Google 还专门为 MapReduce 开发了编程语言 Sawzall[16]。

此外,系统中还包括了用于管理结构化数据的类数据库的管理系统 Bigtable[7],以及锁服务器 Chubby[8]。

Hadoop 是分布式存储计算框架的一个实现,它提供了 Hadoop 分布式文件系统(Hadoop distributed file system, HDFS)和 MapReduce 分布式计算框架。

2 设计与实现

HDWebGIS 系统拓扑图如图 1 所示。底层采用了 Hadoop 系统,包括分布式文件系统 HDFS 和 MapReduce 分布式计算框架,它利用分布式文件系统管理栅格数据和 MapReduce 进行空间计算。在 HDWebGIS 的核心中,针对小文件进行了优化,同时实现了提升系统性能的缓存策略。在前端利用 Jetty 服务器对外提供 Http 服务。客户通过浏览器和服务器交互。

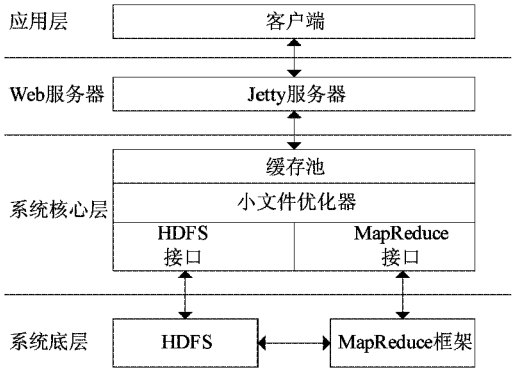

图 1 HDWebGIS 系统拓扑图

2.1 矢量数据栅格化

本文提出的地图浏览系统的数据源来自两种途径:(1)已有的栅格数据。这种数据经过小文件优化

方法处理,直接存入文件系统中。(2)Shapefile 栅格化。Shapefile 是在文件系统中矢量空间数据的一种保存格式,其本身不能直接展示给用户,需要经过栅格化后以图片的形式表现出来。因此,HDWebGIS 系统在将矢量数据存入系统之前首先进行预处理,按照不同的分辨率预先生成栅格图片,以便加速之后的访问速度。在这个过程中,使用了 Hadoop 提供的 MapReduce 分布式计算框架。

一个能用 MapReduce 解决的任务可以被切分为多个互相独立的小任务,因而,可将小任务交由若干 Worker 完成而无需在 Worker 之间进行同步。矢量数据栅格化的任务可被切分,因而可以利用 MapReduce 来完成。

MapReduce 处理过程如下:

(1)将要处理的一组 Shapefile 存入 HDFS 中。

(2)由 Master 将每个 Shapefile 分给一个 Worker。

(3)每个 Worker 拿到任务后,执行 Shapefile 到栅格数据的转换操作。执行完成后通知 Master,从 Master 处得到新的执行任务,直至所有的 Map 任务都执行完成。

(4)在本过程中,所有的 Map 任务执行完成后,整个处理即完成,不需要进行 Reduce 操作。

2.2 针对 HDFS 的小文件优化策略

2.2.1 HDFS 在处理小文件时存在的问题

HDFS 在存储大文件时能够体现出性能上的优势,而没有针对小文件做出优化,这一点在我们使用 HDFS 存储大量小文件的过程中得到了体现。在存储大约 55 万个 1KB 至 10KB 大小的小文件的过程中,我们观察到了两个现象:

(1)文件存储时间长:将上述文件存入到 HDFS 中需要花费大约 7.7h。

(2)为维持 HDFS 文件系统的运转,需要耗费大量的系统内存,在系统空闲时,集群中节点的平均内存占用率高达 55.87%。

造成文件存储时间过长的原因在于系统需要为每一个小文件保存元数据信息,并且由于存在多个副本,需要为其分配多个存储节点。大部分的时间都花费在了系统开销上,真正用于传输文件内容的时间所占的比例非常小,导致小文件数目过多时存储速度变慢。而造成集群内部节点内存占用率过高的问题的原因在于,不论是名字服务器还是数据服务器,都需要保存小文件的元数据信息,在 HDFS 的实现中,这部分信息是常驻内存的,因而当文件数目变得庞大时,所占用的内存也急剧增加。如果将这些文件元数据信息保存在磁盘中,那么可以预见,由于需要频繁地进行磁盘 I/O 访问,访问性能将急剧下降。为了解决上述问题,我们设计了专门针对 HDFS 的小文件优化策略。设计的基本思想是通过将小文件合并为大文件来减少文件的数目。

2.2.2 HDWebGIS 中文件的访问模式

首先我们对 HDWebGIS 中文件的访问模式进行分析,通过对访问模式的分析,对设计小文件的优化策略具有指导意义。

HDWebGIS 中对文件的访问具有如下特征:

(1)文件元数据中的一些属性是相同的或者相似的。对这些相同或者相似的属性,在优化时可以提取出来,不再需要对每个文件都保留相同的域,从而节省内存以及磁盘空间。

(2)在 HDWebGIS 中,一个文件被更新后,其老版本的文件依然具有历史价值,可以保留。因此在设计时就设计保留一个文件的最近 3 个历史版本。

2.2.3 文件的存储

小文件的存储过程就是将小文件合并为大的块的过程。存储过程中,会打开一个数据文件(Block),用于存储文件数据;一个索引(Map)文件,用于存储文件索引。整个策略的执行过程可以描述如下:

(1)策略接收的参数包括一个输入文件夹和一个输出文件夹。首先遍历输入文件夹,对该文件夹及其子文件夹下的每一个文件,执行以下操作:

(2)将文件内容以流的形式写入到当前打开的块中。写入完毕后将块的大小和给定的阈值进行比较,如果超过阈值,就将当前打开的块关闭,重新开启一个新块。

(3)在当前打开的索引文件中为小文件建立索引,索引的键值是小文件名,值是该文件存入的块号,块内偏移以及该文件的长度,对每一个文件,系统保存最近的 3 个版本,因而上述值有 3 个。加入索引条目后,检查当前索引文件的条目个数是否超过给定阈值,如果超过阈值,就将当前打开的索引文件关闭,重新打开一个索引文件。

(4)返回(1)开始执行,直至所有的文件都被存入。

存储完成后,在目标文件夹下可以看到如下的目录结构:

其中 indexes 中记录的是索引信息。索引文件记录的是小文件的索引,索引算法采用的是 Hash 树。索引的键是文件名,值是文件所在的块号以及在块中

```
output _ dir
`--    indexes
 |    |--map0
 |    |--…
 |    `--mapn
`--    blocks
      |--block0
      |--block1
      |--…
      `--blockm
```

的起始位置和文件的长度,每个文件都会保留 3 个历史版本。对每一个版本,还有一个标志位,用于标识这个文件是否已经被删除。在我们的实现中,映射文件的长度的阈值是 65536 个,占用的磁盘空间大约为 3MB。

而 Blocks 目录中记录的是文件数据,该文件记录的是经过合并后的大文件,HDFS 中将大文件分块存储,每个块的大小为 64MB,因而每个块文件长度不超过 64MB。

2.2.4 文件的更新和删除

对文件的更新采用追加更新的方式,更新的步骤如下:

(1)将文件内容追加至当前打开的块文件中。

(2)根据文件名在索引中查找,如果找到索引信息,说明该文件已经存在,如果文件版本不超过 3 个,更新其索引信息,文件更新完成;如果文件的版本已经超过 3 个,则首先删除最早版本的文件,然后用最新的文件信息替换被删除的文件信息。

文件的删除操作步骤如下:

(1)通过索引信息找到文件所在块,然后将块中该文件的有效位置变为无效。

(2)删除索引信息中该文件的信息。

(3)判断是否需要对块文件进行垃圾回收,垃圾回收的触发条件是块中被删除的文件数目超过文件总数的一半。垃圾回收的过程即将块中所有标为有效的文件重新分配新块的过程。

2.2.5 文件的读取

在小文件读取之前,系统有一个初始化的过程,将索引文件的内容读入到内存中。为避免占用过多系统内存,系统规定了读入的最大映射文件的个数,所有的索引文件被保存在一个队列中,采用 LRU 算法对内存中的以及外存上的索引文件进行管理。

初始化后,开始接收小文件的读请求。接收的输入是文件名。收到读请求后,首先通过计算文件所在组,然后依次读入并返回组内的所有文件。

2.2.6 数据的安全性和一致性

该策略是建立在分布式文件系统之上的,因而数据的安全性以及在分布式系统中的数据一致性将由底层的文件系统保证。

2.3 前端设计

我们利用 Openlayers[17]和 Jetty[18]容器设计实现了一套前端显示系统。Jetty 是一个轻量级的 Servlet 容器,其的功能是对外提供超文本传输协议(HTTP)服务。Openlayers 是一套 Javascript 库,其功能是根据用户的动作向服务器端发送 HTTP 请求,然后在客户端对从服务器端获得的图片进行组织并且显示。

3 试验结果

3.1 测试环境

测试平台是由 5 台部分异构的节点组成的集群。节点的型号都是 Dell Power Edge SC430,CPU 型号为 Intel Pentuim 4,2.8GHz,内存容量为 1GB 或 2GB,硬盘为 80 ~ 160GB 容量的 SATA 盘。操作系统为 RedHat AS4.4,Hadoop 的版本为 0.16.1,Java 虚拟机版本为 1.6.0。

3.2 矢量数据栅格化

选取的测试数据集合是一组包含 95 个 Shapefile 文件的集合,文件的大小从 156 个字节到 8MB 不等,其中包含的几何体的数目从 1 个到 8000 个不等。在测试过程中,每一个 Shapefile 被栅格化为 16 × 16 个 256 × 256 像素的文件。测试的项目包括单机串行执行栅格化的时间,分别利用 1 至 4 个节点进行 MapReduce 执行的时间。

执行结果如图 2 所示,同一个节点上可能同时运行多个 Map 任务。表 1 显示了执行所需时间以及

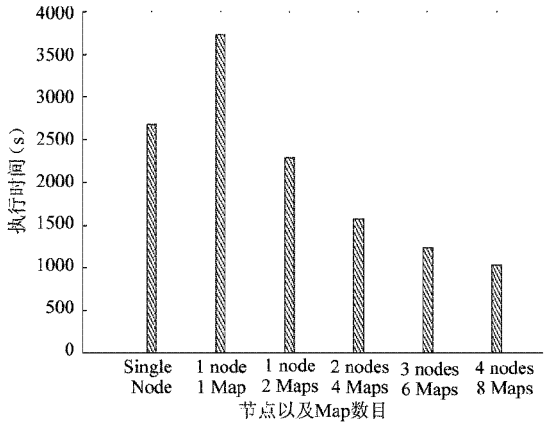

图 2 MapReduce 执行时间

表 1　Map/Reduce 执行时间及加速比

节点数	时间(s)	加速比	效率
单节点	2681	-	-
1(1 Map)	3728	-	-
1(2 Map)	2297	-	-
2	1567	1.71	0.85
3	1234	2.17	0.72
4	1034	2.59	0.64

加速比和效率。从上述结果可以,随着参与计算的节点数目的增多,系统开销增加,效率有所下降,但是加速比随着节点的增加而增加。

3.3　针对 HDFS 的小文件优化策略

本试验测试平台为一个 5 台节点组成的集群,其中 1 台节点上运行 Namenode,其余 4 台机器上运行 Datanode。试验主要测试的项目包括:

▷ 写入测试。将相同数目的文件直接存入 HDFS 和经过优化后存入本地文件系统,以及经过优化后存入 HDFS 的时间对比。

▷ 读出测试。根据需要生成一个访问列表,在上述三种情况下,分别测试访问列表中的文件所需的时间。

在记录测试时间的同时,还会记录在测试的过程中,各个节点的 CPU 以及内存的占用情况。

3.3.1　写入测试

测试用到的文件集合一共 558726 个文件,总量为 3.6GB,图 3 中展示了文件大小的分布情况,其中,小于 16KB 的文件占了总数的 96.19%。将这些文件直接存入 HDFS 耗时 27719s,经过优化后存入本地文件系统耗时 662s,经过优化后存入 HDFS 耗时 431s。图 4 展示了在存储过程中以及在存储完成后,HDFS 处于空闲状态时,各个节点上的内存占用率情况,将文件直接存入 HDFS 的过程中,系统的平

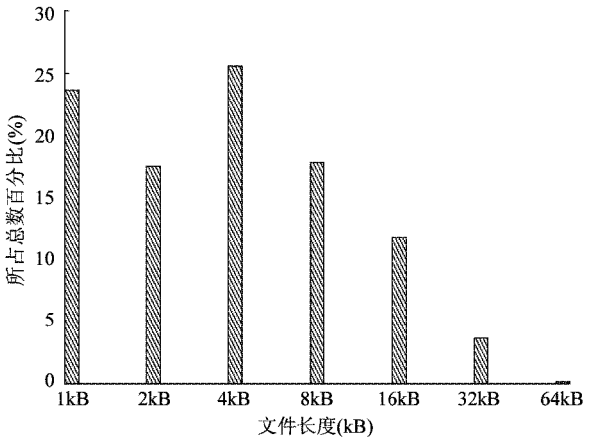

图 3　文件大小分布

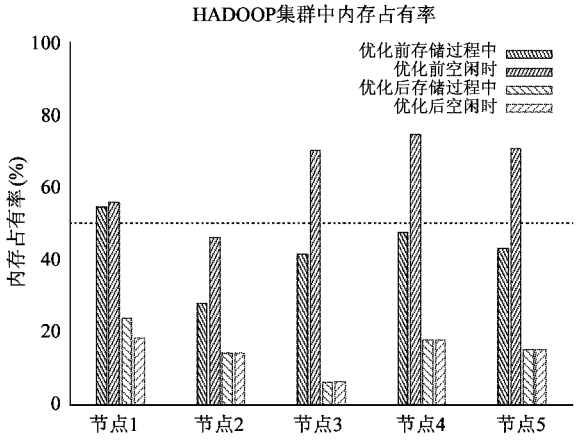

图 4　内存占用率

均内存占用率为 54.46%,而文件系统处于空闲状态时的内存占用率为 55.78%。经过优化后,将文件存入 HDFS 的过程中,系统的平均内存占用率为 24.00%,空闲时的内存占用率为 18.39%。

3.3.2　读出测试

为了模拟不同的访问模式,生成了三个访问文件列表(1) 50000 个不重复的随机文件。(2) 5000 组随机文件,每组 10 个连续文件。组与组不同,但是组内的文件有可能重复。(3) 500 组随机文件,每组 100 个连续文件。组与组不同,但是组内的文件有可能重复。

接下来,分别在四种情况下进行了对文件列表中的文件的读测试,分别为(1) 未经优化的 ext3 文件系统;(2) 经过优化的 ext3 文件系统;(3) 未经优化的 HDFS 文件系统;(4) 经过优化的 HDFS 文件系统。

测试的结果如图 5 所示。从图中可以看出,经过优化的 HDFS 性能最优。

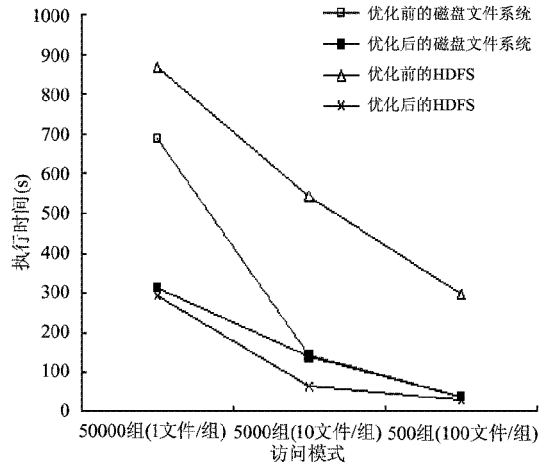

图 5　访问文件所需时间对比

3.4 系统测试

测试的目标系统有两个:(1) 基于原始 HDFS 的 HDWebGIS 系统,未对小文件进行优化。(2) 采用了小文件优化策略的 HDWebGIS 系统。

测试所用到的测试程序记录了 100 个大城市的坐标,在测试时,可以模拟用户浏览时的动作,首先从城市文件中选取需要访问的城市坐标,然后向服务器发出请求,按照图层分辨率一层一层放大(一共 17 级),在放大到最高分辨率后,再进行左右拖拽浏览操作。一个城市的访问操作需要发出 350 个文件的读请求。根据需要,程序可以模拟多用户并发访问。特别地,模拟程序中发出请求是大量连续的,而系统实际运行时,真实用户发出请求的密度将小得多。

在测试的过程中,分别模拟了 1、2、4、8、16、32 个客户端同时发出访问请求,每次请求会选择 10 个城市进行访问,记录访问时间。表 2 显示了分别对两个目标系统进行测试时所需的时间,因为访问每个城市所请求的图片数量大约在 350 幅左右,因而很容易算出系统每秒能响应的请求数目。图 6 展示系统每秒能响应的请求数目。从结果可以看出,经过优化的 HDWebGIS 系统每秒响应请求的能力比未经优化的 HDWebGIS 系统提高了 78.11%。

表 2 访问 10 个城市所需时间

客户端数目	HDWebGIS(优化前)	HDWebGIS(优化)
1	34.43	14.91
2	38.63	24.92
4	57.52	37.24
8	93.73	54.90
16	182.28	116.81
32	420.09	247.99

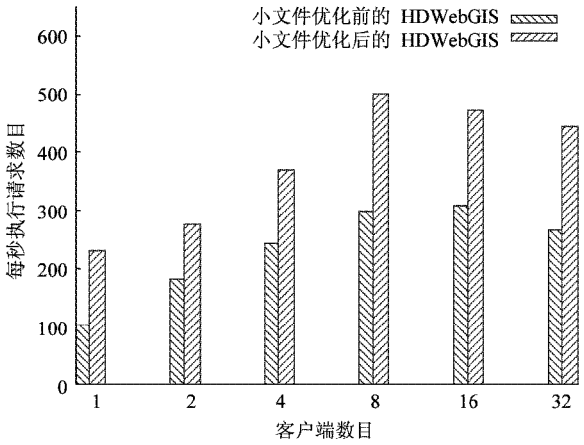

图 6 每秒执行的请求数目

4 结 论

本文提出了一种在分布式存储计算框架下并行处理空间数据的方法,利用该方法可以方便地构建 WebGIS 系统。该方法通过利用 MapReduce 分布式计算框架来开发集群中的计算能力,可满足 WebGIS 对计算能力的需求。通过利用分布式文件系统开发集群中的 I/O 能力,可满足 WebGIS 应用对于 I/O 的需求。但是分布式存储计算框架中提供的分布式文件系统是专门针对大文件进行优化的,对于 WebGIS 这类小文件数目众多的应用,直接利用其管理小文件会带来诸多性能问题,为此,文中还专门分析了 WebGIS 对小文件的访问模式,提出了针对分布式文件系统的小文件优化策略。测试结果表明,在利用了小文件优化策略后,集群中节点的平均内存占用率从优化前的 55.78%降至 18.36%,存储速度 55 万个左右文件所需时间由原来的 27719s 降至 431s,性能提高了 63.3 倍,而随机读取 50000 个文件的时间由原来的 867s 降至 292s,性能提高了 2.0 倍。

基于上述方法和小文件优化策略,本文还设计并且实现了 HDWebGIS 原型系统。试验结果表明,利用 5 个节点构建的使用了小文件优化策略的 HDWebGIS 系统比未使用小文件优化策略的 HDWebGIS 提高了 78.11%。

随着 GIS 应用的发展,利用集群系统来构建 WebGIS 系统逐渐成为一种发展趋势。本文提出的利用分布式文件系统和 MapReduce 技术在集群系统中并行处理空间数据的策略对构建一个高性能、高可靠性、高可扩展性的 WebGIS 系统具有参考意义。

参考文献

[1] Bolstad P V. GIS Fundamentals: A First Textbook on Geographic Information Systems. 3rd edition. Ashland: Bookmasters Dist, 2008.15-30

[2] 刘仁义,刘南. WebGIS 原理和应用. 北京: 科学出版社,2005.20-35

[3] 王宇翔. 分布式网络地理信息系统研究:[博士学位论文]. 北京:中国科学院研究生院,遥感应用研究所,2002

[4] 钱贞国. 面向互操作的分布式网络地理信息系统研究:[博士学位论文]. 北京:中国科学院研究生院,遥感应用研究所, 2004

[5] 马修军, 陈斌, 方裕等. 基于 P2P 的空间信息服务组合执行引擎. 高技术通讯, 2006, 16(5): 446-451

[6] Ghemawat S, Gobioff H, Leung S T. The google file system. In: Proceedings of the 19th ACM symposium on Operating Systems Principles. New York: ACM Press, 2003. 29-43

[7] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. In: Proceedings of the 7th Usenix Symposium on Operating Systems Design and Implementation, Seattle, WA, USA, 2006. 205-218

[8] Burrows M. The chubby lock service for looselycoupled distributed systems. In: Proceedings of the 7th Usenix Symposium on Operating Systems Design and Implementation, Seattle, WA, USA, 2006. 335-350

[9] Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 2004, 51(1), 137-150

[10] The Apache Software Foundation. Welcome to Apache Hadoop! http://hadoop.apache.org/core/: Apache, 2008

[11] Patterson D A. Technical perspective: the data center is the computer. *Communications of the ACM*, 2008, 51(1): 105

[12] Wikipedia. MapReduce. http://en.wikipedia.org/wiki/MapReduce: Wikipedia, 2008

[13] Yang H C, Dasdan A, Hsiao R L, et al. Map-reduce-merge: simplified relational data processing on large clusters. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2007. 1029 1040

[14] Ranger C, Raghuraman R, Penmetsa A, et al. Evaluating mapreduce for multi-core and multiprocessor systems. In: Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture. Washington D. C.: IEEE Computer Society, 2007. 13-24

[15] Kruijf M, Sankaralingam. K. MapReduce for the Cell B.E. Architecture: [technical report]. Madison, WI: Department of Computer Sciences, the University of Wisconsin-Madison, 2007

[16] Pike R, Dorward S, Griesemer R, et al. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 2005, 13(4):277-298

[17] MetaCarta. Openlayers. http://OpenLayers.org: MetaCarta, 2008

[18] Bay M. Jetty. http://www.mortbay.org: Codehaus Foundation, 2008

An approach to parallel processing of spatial data on clusters

Liu Xuhui[*,**], Han Jizhong[*], He Jin[*], Han Chengde[*]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] Graduate University of Chinese Academy of Sciences, Beijing 100190)

Abstract

This paper proposes a new approach to parallel processing of spatial data on clusters. This approach uses the distributed file system (DFS) and the MapReduce framework to exploit I/O and computing capability on clusters. Especially, the paper proposes an optimizing schema for DFS to efficiently manage massive amount of small files. The main idea of the schema is to reduce the number of files by merging a group of small files to big ones. The experimental results show that after adopting the schema in DFS, the writing performance and the reading performance can be improved by 6330% and 200% respectively, while the average memory usage ratio in cluster nodes can be reduced from 55.78% to 18.36% after adopting the schema. Furthermore, a prototype WebGIS system called Hadoop Distributed WebGIS (HDWebGIS) was designed and implemented based on DFS and MapReduce framework to evaluate effectiveness of the schema from system's perspective, the experiment results also show that HDWebGIS has a performance promotion of 78.11% after using the schema.

Key words: WebGIS, clusters, distributed file system (DFS), MapReduce, tuning for small files